§1. UVOD

1.1. Što je algoritam?

Sadržaj ovog kolegija su: algoritmi i kompleksnost.

Oba ova termina se mogu **precizno** definirati u okviru matematičke logike. Dio puta prema preciznoj formulaciji ovih pojmova ćemo napraviti na kraju kolegija, kad ćemo govoriti o NP-potpunosti.

Za početak, zadovoljimo se grubim opisom ovih pojmova.

Algoritam = metoda, proces, **postupak** ili **pravilo** za rješavanje neke klase problema (obično na računalu).

Kompleksnost algoritma = cijena korištenja tog algoritma za rješenje **jednog** od tih problema iz klase. Obično se mjeri u vremenu izvođenja ili potrebnoj memoriji.

Općenito oba ova pojma **nešto** detaljnije.

Algoritam - postupak se **sastoji** od niza koraka - instrukcija ili naredbi.
Izvođenje tog niza naredbi mora biti objektivan proces koji se, u načelu, mora moći reproducirati.
Također, izvođenje ne smije zahtijevati neku intuiciju ili kreativnost.
Kasnije ćemo vidjeti da **postoji** i važna klasa algoritama koja je izuzetak od ovih zahtjeva - tzv. probabilistički algoritmi.

Zahtjevi na postupak - niz koraka su slijedeći:

1. Postoji jednoznačno određeni početni korak (s kojim počinje izvođenje)
2. Po završetku nekog koraka, točno je određen slijedeći korak koji treba obaviti.
   (Ovo je determiniranost algoritma.
   Postoje i nedeterministički algoritmi!)
3. Postupak uvijek završava u konačno mnogo koraka, za bilo koji problem iz neke klase.
4. Postupak se sastoji od konačnog broja osnovnih (elementarnih, primitivnih) instrukcija.
   Značenje tih osnovnih instrukcija je jednoznačno definirano (i efektivno!) za izvršioca postupka (stroj ili osoba).

Osnovna svojstva algoritma:

1. Ulaz – svaki algoritam ima $\emptyset$ ili više (ali konačno mnogo) ulaznih međurezultata - objekata.
Ti objekti jednoznačno određuju pojedinačni problem koji treba rješiti - to je element u klasi problema koju algoritam rješava
Ulaz definira konkretan problem - zadatak, u toj klasi problema.
Primjer – koeficijenti kvadratne jednadžbe za algoritam koji rješava kvadratnu jednadžbu.

2. Izlaz – svaki algoritam mora imati barem 1 izlaz.
To je rezultat, rješenje zadaće - konkretnog problema.

3. Konačnost – algoritam mora završiti u konačno mnogo koraka za svaki ulaz, tj. za svaku konkretnu zadaću tog problema.
Katkad je potrebno ograničiti ulaz da zadaća bude korektno zadana.

Dali je duo alg. opet alg. (procesi – bez izlaza → signal da je završio). Pretpostavljamo, u načelu, da su zadaće korektno zadane.

Interrupt controled processes. (Oper. sustav)

Programska realizacija algoritma može imati dodatne provjere korektnosti ulaza.
Na pr. u kvadratnoj jednadžbi

$$ax^2+bx+c=0$$

pretpostavljamo da je $a \neq 0$, inače to nije kvadratna jednadžba.
Program može to provjeravati, ali na nivou algoritma za rješenje kvadratne jednadžbe, smatramo da je ulaz korektan.

4. Definiranost i nedvosmislenost.
Svaka osnovna instrukcija mora biti jednoznačno definirana – bar za izvršioca.
Što su osnovne instrukcije – to je pitanje dogovora (tj. ovisi o onom stroju kojeg koristimo).
Taj dogovor utječe i na zapis - reprezentaciju algoritma.
Za opis algoritama koristimo jezik vrlo sličan Pascalu. Daljnje diskusije na temu osnovnih instrukcija – kasnije, kad budemo analizirali njihov utjecaj na kompleksnost.

(praktična složenost osn. instr., primjena na Pascal – na pr. petlje)

Napomena: Algoritmi koje ćemo promatrati, u principu rade za bilo koji ulaz.
U praksi, ovdje treba biti oprezan, jer računala ne mogu realizirati bilo koji apstraktni ulaz, već imaju ograničen skup podataka koje mogu obrađivati, a, također, i ti podaci imaju ograničen skup mogućnosti.

Algoritam na prirodnim brojevima, u principu, radi za sve prirodne brojeve. Računala, međutim, mogu reprezentirati samo konačni podskup skupa prirodnih brojeva.
Isto vrijedi i za realne brojeve, na pr. u kvadratnoj jednadžbi.

Na nekim primjerima ćemo pokazati kako ovo utječe na realizaciju algoritma u terminu raspoloživih instrukcija i tipova podataka u računalu.

- Za praksu je potrebno dodati i slijedeći zahtjev na algoritam:

5. Efikasnost – algoritam mora završiti u "razumnom" vremenu, uz "razumnu" potrebnu prostor.

Što znači razumno" – naravno ovisi o snazi mašine.

Time smo došli do pojma efikasnosti algoritma.

1.2. Efikasnost algoritma

Rješavanje problema (izračunavanje - u širem smislu) traje – troši vrijeme.
Neki problemi zahtijevaju vrlo mnogo vremena, a neki ne.
Za neke probleme nam se čini da troše mnogo vremena, a onda netko nađe brzi način za njihovo rješavanje (brzi algoritam).
Za određeni problem postoji više (ili mnogo) algoritama za rješenje. Treba se odlučiti kojeg ćemo upotrijebiti = najboljeg, naravno. Ali po kojim kriterijima.

Ova pitanja spadaju u granu / domenu složenosti izračunavanja (Computational complexity).
Predmet proučavanja – količina resursa izračunavanja koji su potrebni za određene vrste izračunavanja.

Tu količinu potrebnih resursa zovemo kompleksnost algoritma.

Najčešće nas zanimaju dvije vrste potrebnih resursa za neki računski postupak:

- vrijeme
- prostor,

pa obično promatramo dvije vrste kompleksnosti:

(a) <u>vremenska kompleksnost</u> = potrebno vrijeme za izvođenje algoritma, u nekim osnovnim jedinicama. Najčešće jedinice su ili standardne vremenske (sekunde, minute i sl.) ili strojne (osnovni ciklus računala ili prosječno trajanje osnovnih instrukcija).

Mjeru (jedinice) vremenske kompleksnosti treba odabrati tako da možemo objektivno uspoređivati <u>algoritme</u>, a ne strojeve. O tome malo kasnije.

[Spomeni "antim. slož."]

(b) <u>prostorna kompleksnost</u> = potrebna memorija za izvođenje algoritma, u nekim osnovnim jedinicama. Objektivna jedinica za mjerenje prostorne kompleksnosti je <u>bit</u>, ali je relativno nepraktična, zbog različitosti arhitektura. Korisnije jedinice su riječ (= 1 integer ili real) ili, još apstraktnije, broj. Naravno, i ovdje treba biti oprezan, kao što ćemo vidjeti.

Prostorna kompleksnost, osim ulaza, uključuje i sve međurezultate koji se javljaju prilikom izvođenja algoritma. Izlaz je također uključen (možemo ga smatrati finalnim međurezultatom).

Pri tome ne mjerimo <u>ukupni</u> izlaz, ako se on producira jedan po jedan!

Tj. dozvoljeno je više puta koristiti istu memoriju za razne podatke u razna vremena!

Prostorna kompleksnost je uvijek bitno manja od vremenske, tj. prostor predstavlja bitno manje ograničenje na primjenjivost algoritma od vremena.

To se može i dokazati uz odgovarajući model izračunavanja.

Umjesto dokaza, ilustrirajmo to jednim primjerom.

Za paralelno rač – još i broj procesora!

Primjer 1. Za zadani $n \in \mathbb{N}$ treba štampati sve permutacije skupa $\{1, 2, \dots, n\}$.

Bilo koji smisleni algoritam za ovaj problem će redom generirati permutacije - jednu po jednu. Čim generira permutaciju - odmah ju štampa, tj. dovoljno je pamtiti jednu ili dvije (stari i novu) permutaciju istovremeno.

Dakle, algoritam troši konstantan prostor za sebe i najviše još 2 permutacije.

Broj permutacija je, međutim, $n!$ i algoritam mora trošiti vrijeme koje je proporcionalno $\geq n!$ (jer je to i veličina traženog izlaza).

Dakle: prostor $\sim$ 1 ili 2 permutacije
vrijeme $\sim$ $n!$ permutacija.

[Smatramo sekvencijalno izvođenje, ali ni paralelno ne bi pomoglo, jer je izlaz veličine $n!$ permutacija]

Izbor jedinica za prikaz kompleksnosti očito ne utiče bitno na usporedbu. [Namjerno je neprecizan, bez detalja].

→ {Prog. koji koristi veliki prostor, treba bar toliko vrem. jedinica koliko i prostornih (da bar pogleda tai prostor).

- Zbog toga ćemo obično promatrati vremensku kompleksnost kao bitno značajniji faktor efikasnosti algoritma.

Zasad smo opisali što ćemo uzeti kao mjeru efikasnosti algoritma (bez detalja o jedinicama u kojima mjerimo).

Nismo, međutim, rekli kako ćemo mjeriti kompleksnost i o čemu ona ovisi.

- O čemu ovisi kompleksnost?

→ Alg.: info o alg. koji onog. usporedi alg. (NE izvršitelja!)

Pretpostavimo da imamo dva algoritma koji rješavaju isti problem - tj. imaju isti skup zadataka koje mogu riješiti. Kako ćemo ih usporediti?

Precizno govoreći, trebalo bi "izmjeriti" kompleksnost tih algoritama za svaku zadaću tog problema.

Dakle, kompleksnost bi tada bila funkcija oblika

zadaća $\longmapsto$ broj

problem $\longrightarrow$ $\mathbb{R}^+$ (ili pr. ili $\mathbb{N}$)

Zadan je poligon (ne nužno pravilni i ne nužno konveksni) i imamo beskonačno takvih pločica. Može li se njima popločati zadana ravnina?

ALG. NEPOZNATO. (Nema alg. koji daje odg. DA/NE!)

Ovo je vrlo objektivno, ali, ozato, malo neprakticno. Trebalo bi efektivno rijesiti sve zadaće tog problema. No, broj zadaća u problemu je, ocito, beskonacan.

U primjeru s permutacijama, zadaća je (ulazni) broj $n \in \mathbb{N}$, a skup $\mathbb{N}$ je beskonacan.

Dakle, nemoguće je rijesiti sve zadaće. Osim toga, kad bismo ih jednom sve rijesili i zapamtili rjesenja, kasnije bi samo trebalo pronaći i procitati rjesenje.

— Ovakav pristup je los. Moramo smanjiti domenu za kompleksnost. (Particija!)

Prirodno je ocekivati da proracun s ulazom od nekoliko milijuna bita, vjerojatno, traje dulje od proracuna s ulazom od nekoliko bitova. (To ne mora uvijek biti tako!)

Zbog toga, kompleksnost algoritma izrazavamo kao funkciju veličine zadaće koju treba rijesiti.

Velicina zadaće je, naravno, neprecizan pojam. To je neka mjera količine podataka koja je potrebna za opis zadaće na ulazu u algoritam.

Velicina zadaće x, u oznaci $|x|$, je formalno govoreći: broj bitova potreban za reprezentaciju zadaće x na ulazu algoritma, uz koristenje jednoznacno definiranog i razumno kompaktnog nacina kôdiranja.

Može se pokazati su razumno kompaktni nacini kodiranja medusobno ekvivalentni u smislu da su gotovo proporcionalni.

Ako je $|x|_1$ duljina zadaće x u kôdu 1, a $|x|_2$ duljina zadaće x u kodu 2, onda postoje konstante $c_1, c_2 > 0$ takve da za svaku zadaću x mjedi

$$c_1 |x|_1 \leq |x|_2 \leq c_2 |x|_1 .$$

Ilustrirajmo to na primjeru s permutacijama.
Ulaz je ulaz = prirodni broj n.
Sto je razumno kompaktno kodiranje?

Brojeve prikazujemo u pozicionom zapisu u nekoj bazi b. Obično je $b = 2$ za računala:

$$n = a_k b^k + \dots + a_1 b + a_0$$

gdje su a_i, $i = 0, \to k$ znamenke. Zapis smatramo normaliziranim, tj

$$a_i \in \{0, \to b-1\} \quad , \; i = 0, \to k$$

i vodeća znamenka je pozitivna

$$a_k > 0 .$$

Dakle, ulaz je niz znamenki $a_0, \to a_k$. Duljina ulaza je broj znamenki. U ovom slučaju, za zadaću $x = \{n\}$, kôdiranu optimalnim normaliziranim pozicionim zapisom u bazi b, $b \in \mathbb{N}$, $b \geq 2$, imamo

$$|x|_b = k+1 = \lfloor \log_b n \rfloor + 1 .$$

$\lfloor \dots \rfloor$ je funkcija "najveće cijelo manje ili jednako od ...".

<u>Zadatak 1.</u> Dokaži da su duljine zapisa u raznim bazama gotovo proporcionalne. Da li je to točno? → v. (7-1) (7-2)

U bazi $b = 2$, uz dogovornu oznaku $\log_2 = \lg$ je:

$$|x| = \lfloor \lg n \rfloor + 1 .$$

- Pokažimo i jedan primjer nekompaktnog ili opširnog kodiranja.

  Broj n prikažemo u "bazi 1", tj zapišemo ga kao n znakova $\emptyset$ (ili 1) na ulazu. Tada je

$$|x|_1 = n$$

što je očito nerazumno kôdiranje.

— · —

U ovom primjeru s permutacijama, uvođenje duljine ulaza očito doprinosi dobitka dijela informacije, jer iz brojeva ima istu duljinu zapisa.

Korisnije je kompleksnost izražavati preko n, umjesto u duljini zapisa broja n.

<u>Zadatak 1</u> – odgovor (v. Saša HR, str. 48-49)

Dvojne zadaće – brojeva n u bazama $b_1, b_2 \geq 2$ su:

$$|x|_{b_1} = \lfloor \log_{b_1} n \rfloor + 1$$

$$|x|_{b_2} = \lfloor \log_{b_2} n \rfloor + 1$$

Obje ove funkcije su pozitivne i rastuće (riječ <u>ne</u> strogo). Promatramo omjer:

$$f(n) = \frac{|x|_{b_2}}{|x|_{b_1}} = \frac{\lfloor \log_{b_2} n \rfloor + 1}{\lfloor \log_{b_1} n \rfloor + 1}$$

Ocjenjujemo taj omjer odozgo i odozdo, koristeći relaciju

$$u - 1 < \lfloor u \rfloor \leq u, \quad \forall u \in \mathbb{R}$$

za brojnik i za nazivnik.

– Ocjena odozgo (brojnik odozgo, nazivnik odozdo):

$$f(u) < \frac{\log_{b_2} n + 1}{\underbrace{\log_{b_1} n}_{\neq 0, \text{ za } n > 1}} = \log_{b_2} b_1 + \frac{1}{\log_{b_1} n}.$$

Ovo vrijedi za $n > 1$. Za $n \geq b_1$ (pa je sigurno $n > 1$, zbog $b_1 \geq 2$) je $\log_{b_1} n \geq 1$ tj.

$$f(n) < \log_{b_2} b_1 + 1.$$

Ova relacija vrijedi i za $n < b_1$, jer je tada $|x|_{b_1} = 1$ (jednoznamenkasti brojevi u bazi b_1), pa izravno iz definicije $f(u)$ izlazi

$$f(u) = |x|_{b_2} \leq \log_{b_2} n + 1 < \log_{b_2} b_1 + 1.$$

$\downarrow$ ($n < b_1$)

– Ocjena odozdo (brojnik odozdo, nazivnik odozgo) dalje:

$$f(n) > \frac{\log_{b_2} n}{\log_{b_1} n + 1} = \log_{b_2} b_1 \cdot \frac{\log_{b_1} n}{\log_{b_1} n + 1}.$$

Dakle, dobili smo ocjenu:

$$\log_{b_2} b_1 \cdot \underbrace{\frac{\log_{b_1} n}{\log_{b_1} n + 1}}_{\text{ovo još ovisi o } n.} < f(n) < \log_{b_2} b_1 + 1$$

Za bilo koju ocjenu tog dijela odozdo, treba postaviti neku donju ograđu na n, odn. njegovu duljinu u bazi b_1. Na pr. za $n \geq b_1$ je $\log_{b_1} n \geq 1$, pa je

$$\frac{\log_{b_1} n}{\log_{b_1} n + 1} \geq \frac{1}{2}.$$

Dakle, za $n \geq b_1$ vrijedi:

$$\underbrace{\frac{1}{2} \log_{b_2} b_1}_{c_1} < f(n) < \underbrace{\log_{b_2} b_1 + 1}_{c_2}.$$

Dokazali smo da su duljine zapisa u raznim bazama gotovo proporcionalne, ali tek uz neko ograničenje da n bude dovoljno velik. Ovo nije veliko ograničenje (bar 2 znamenke u bazi b_1), ali bez tog ne vrijedi.

Promatrajmo li omjer $f(n)$ za vrlo velike zadaće, kad brojevi i njihove duljine neograničeno rastu:

$$n \to \infty \Rightarrow \log n \to \infty \quad \text{(u bilo kojoj bazi)}$$

onda (zbog $n \geq b_1$) izlazi:

$$\log_{b_2} b_1 \cdot \underbrace{\frac{\log_{b_1} n}{\log_{b_1} n + 1}}_{\to 1} < f(n) < \log_{b_2} b_1 + \underbrace{\frac{1}{\log_{b_1} n}}_{\to 0}$$

$$\Rightarrow \lim_{n \to \infty} f(n) = \log_{b_2} b_1.$$

Tj: za velike zadaće, konstante c_1, c_2 postaju sve bliže u "gotovo proporcionalne" i na limesu dolazimo baš proporcionalnost.

Kasnije ćemo to prirodno nazvati kodominantnost, odn. asimptotsku proporcionalnost duljina zapisa u raznim bazama (v. 1.4.).

(više i ne trebamo za usporedbu složž – jer to radimo za velike zadaće)

U ovom slučaju, kompleksnost fundamentalno ovisi baš o mjerodavnosti ulaznog broja - njegovoj veličini (a ne o $n!$. duljini zapisa). $n \to n!$ je ljepše od $\underbrace{\lfloor \log n \rfloor + 1}_{=|x|} \to \ldots$ prema $|x|$

Kod niza numeričkih problema, efikasnost algoritama (kompleksnost) također bitno ovisi baš o veličini - mjerodavnosti brojeva u zadaći, a manje o broju bitova potrebnih za njihov prikaz. Muogo važniji faktor u praksi je relativna točnost prikaza i aritmetike (famozni ε) koji dodatno ulazi u kompleksnost, kad tražimo rezultate na zadanu točnost.
(Na pr. Newtonova metoda za ryeš. jednadžbi).
[Većina metoda koja daje ryeš. kao LIMES, nije alg.; bez dodatnog zahtjeva točnosti]

U nizu drugih problema, veličina ulaznih brojeva (podataka) nije bitna, već je dominantan njihov broj.

Primjer 2. Standardni algoritam za množenje matrica reda n zahtijeva n^3 množenja. (Vrpce ne prijeđu kako se kodiraju elementi!)
Ovdje je broj ulaznih podataka $2n^2$, a njihova veličina nije posebno bitna (bar za broj apstraktnih aritmetičkih operacija). No, red matrice je najprirodnija mjera veličine problema.

— Zbog toga ćemo veličinu ulaza relativno slobodno interpretirati, na načinu koji prirodno odgovara problemu koji promatramo.
Ta veličina je neki prirodni broj koji prirodno opisuje veličinu zadaće.

— Tek kad budemo precizno kategorizirali algoritme i probleme po kompleksnosti, onda ćemo i precizno voditi računa o veličini ulaza.

Dakle, najčešće je kompleksnost funkcija oblika:

veličina zadaće $\longmapsto$ broj ($\in \mathbb{N}$ ili, rijeđe, $\in \mathbb{R}^+$)

(partnaja problema po veličini zadaća)

$\updownarrow$ → ovo su najčešće prirodni brojevi ("broj bitova")

$x \xrightarrow{1:1} |x| \longmapsto \text{compl}(x)$

Uočimo da sada postoji više zadaća s istom veličinom.

Općenito, nema nikakve garancije, da sve te zadaće imaju istu kompleksnost.

Kako onda veličini zadaće pridružiti broj?

Dva su pitanja:

- po kojem principu pridružujemo "broj" zadaćama iz iste klase – s istom veličinom,
- kako stvarno dolazimo do tog "broja". ([empirija] [praksa] teorija)

("brojanje osn. instr.")

Prvo pitanje je zapravo pitanje svrhe – što treba reprezentirati kompleksnost.

Na pr. u nekom problemu s ulazom od B bitova, algoritam, za razne zadaće iste duljine B, može trajati sasvim različito.

Za neke zadaće može trajati 6B vremenskih jedinica, za neke druge na pr. $10B \log B$, a neke čak i $5B^2$ jedinica. [Primjer – kasnije = QuickSort]

Koji od tih brojeva ćemo koristiti?

Postoje dva pristupa – maksimalni i prosječni. Rijetko se koristi drugi ekstrem – minimalni.

Ta 3 pristupa imaju i svoja imena:

- <u>najgora kompleksnost</u> (worst-case complexity)

  za danu veličinu problema uzimamo najveće moguće vrijeme (tj. vrijednost druge), za sve zadaće te veličine.

  To je tzv. najgori mogući scenarij tj. <u>garancija</u> da proračun ne traje više od toga.

  Ova vrsta kompleksnosti se najčešće koristi, jer se do nje „obično" najlakše dolazi. Praktična korist se u primjenama kod kojih je vrijeme kritično (real-time obrade).

- <u>prosječna kompleksnost</u> (average complexity)

  Ovdje uzimamo prosječno trajanje (tj. vrijednost druge) po svim zadaćama iste veličine.

  To je mnogo realističnija i korisnija mjera, pogotovo u primjenama gdje se algoritam često koristi na raznim zadaćama. Na velikom broju zadaća, trajanje će se, u prosjeku, tako ponašati.

Nažalost, barem teoretski, prosječna kompleksnost je mnogo teži problem od najgore.
(Ilustrirat ćemo to kod sortiranja).

1.3. Analiza efikasnosti algoritma

- Kako nalazimo kompleksnost (onu koju želimo - najgoru, prosječnu).

Opet imamo 2 pristupa - metodologije gledamo.

- empirijski (a posteriori - naknadni)
- teorijski (a priori - prethodni)

(Pojavit će se i treći - hibridni, to je kombinacija ova dva pristupa).

- Empirijski pristup - programiramo algoritam kojeg želimo izanalizirati - u nekom jeziku, na nekom računalu. Taj program izvodimo na nekom odabranom uzorku zadataka i mjerimo potrebne resurse (mjere).

Zvuči jednostavno, ali nije sasvim tako.
Ovakvo mjerenje opet daje "gole" brojeve, a ne "funkciju" veličine problema.
To je zgodan i jednostavan pristup, ako uspoređujemo nekoliko algoritama, pa nas zanima njihov relativan odnos na odabranom uzorku.
No, ako želimo dobiti "apsolutnu" informaciju o jednom algoritmu, onda dobivene brojeve treba statistički obraditi i komprimirati u neku prihvatljivu formu - najčešće je to neki poznati oblik funkcije s nekoliko parametara.

- Kako doći do oblika funkcije?
  Ili eksperimentiranjem - izaberemo par f-a i gledamo koja funkcija daje bolje rezultate,
  ili a priori analizom - tj. teorijom.

Druga metoda je očito poželjnija - pa dobivamo tzv. hibridni pristup.

- Potrebni param. se odreduju stat. analizom - na pr. metodom najmanjih kvadrata.

- Da bi statistika dala reprezentativne rezultate, treba pažljivo odabrati uzorak.
- Ako mjerimo najgoru kompleksnost - potrebno je znati oblike zadaća koji daju najgoru kompl. (⇒ teorija), a ne samo uzeti najveća vremena. (U poč. faci eksperimenta, može i obratno - da se otvore svojstva najgorih zadaća).
- Za prosj. kompl. - uzorak mora biti statistički reprezentativan (za primjenu ili za opći problem) - "slučajno" generiran, ali s pravom distribucijom. (⇒ teorija).

- Teorijski pristup - matematičkom analizom algoritma odredujemo funkciju kompleksnosti u ovisnosti o veličini problema. zadaće,
  Da to možemo napraviti, moramo izabrati neki model izvršavanja alg. i dogovoriti se o kompleksnosti osnovnih operacija u tom modelu
- Jednostavnije - treba odrediti/izabrati osnovne instrukcije - elementarne operacije u tom modelu i svaku od njih dodijeliti neku osnovnu kompleksnost.
- Ovo nije uvijek jednostavno, pogotovo ako želimo da model bude realan.

Na pr. u primjeru 2 za množenje matrica, brojimo samo množenja skalara, a sve ostalo ignoriramo (iako možemo brojati i zbrajanja skalara - posebno).
No, bitna pretp. u tom primjeru je da množenje bilo koja dva skalara traje jednako, neovisno o veličini. To, naravno, nije realistično. Trajanje množenja će gotovo konstantno samo u prikazivom rasponu, a van njega itekako ovisi o veličini (duljini) brojeva.
- Isti problem ćemo ilustrirati nešto kasnije - kod Fib. br.

Problem modela, izbora i kompleksnosti elementarnih operacija možemo zaobići manom teoretskog pristupa

No, to je i prednost - jer možemo birati i pažnju usmjeriti samo na pojedine vrste operacija.
(OK - sve ovo ima bar malo veze s realnim svijetom).

Koje su prednosti teoretskog pristupa?

- On eliminira specifičnosti empirijskog pristupa, tj. ne ovisi o računalu koje koristimo, programskom jeziku u kom pišemo ili čak o vještini programera.
- Štedimo vrijeme i programera i stroja - za programiranje i izvođenje nekog neefikasnog algoritma.
  Vrlo često se elementarnom analizom mogu eliminirati neki alg. u usporedbi s drugima.
- Na kraju i najvažnije - dopušta nam analizu efikasnosti algoritma na zadacima bilo koje veličine.
  U praksi, često nam ograničeni resursi dozvoljavaju testiranje alg. samo za zadaće malih ili umjerenih veličina. (Npr. matrica reda 100000.).
- Ono je još vrlo važno. Mnogi nedavno otkriveni algoritmi postaju efikasniji od svojih prethodnika tek kad se koriste na velikim zadaćama.
  Danas, možda, tako velike zadaće nisu realno izvodive, ali ubrzo to mogu postati - razvojem tehnologije i potreba u primjeni.
- Teorijski pristup daje oblik funkcije kompleksnosti - u ovisnosti o veličini zadaće. Parametre možemo utvrditi regresijom - empirijski, za odabranu implementaciju.
  Taj hibrid dozvoljava i ekstrapolaciju - predviđanje potrebnog vremena za zadaće mnogo veće od onih u testu. To je korisno - za utvrđivanje domene primjenjivosti - granica primjenjivosti neke implementacije alg.
  Pri tome treba biti oprezan!
  Ekstrapolacija, samo na bazi empirijskog testa, bez teorijskog razmatranja, može biti neprecizna ili čak potpuno pogrešna. (Da li je npr. trend n^5 ili 5^n za male n?)
- U teorijskom pristupu ostaje još jedan problem:
  - U kojim jedinicama ćemo izražavati (vremensku) kompleksnost algoritma?

  Sekunde (ili sličnu jedinicu) odmah ne dolaze u obzir! Trebao bi nam "standardni" kompjuter kao referentna točka, po kojoj "baždarimo" sva mjerenja.

To je moguće učiniti!
Možemo odabrati jedan od standardnih modela izračunavanja - na pr. Turingov stroj, i sve mjeriti u broju osnovnih operacija tog stroja.

- To je, međutim, neprakticno, za većinu naših potreba. Strogi i precizni model misli iz nepotrebnih detalja koji strahovito kompliciraju analizu (v. Cooke-ov teorem u §7). Za usporedbu je dovoljno uzeti iste ili slične (proporc.) jed.
— Nama treba jednostavni, pa makar i pomalo neprecizan, način za usporedbu algoritama.
Teoriji gledano, obrnuto, nama je globalna informacija o kompleksnosti alg.
Što je ta globalna informacija? U kojoj domeni veličina ona mora biti reprezentativna?

Odgovor: Ponašanje alg. za velike zadaće!

Dakle, globalna inf. o alg. je njegova kompleksnost za dovoljno (proizvoljno) velike zadaće.

Zašto?
- (Okto, bar teoriji) "većina zadaća je velika" (vrlo grubo rečeno).
- Tehnologija napreduje - omogućava rješavanje sve većih problema, pa treba imati prave, dobre algoritme za te probleme.

Danas postoje neki alg. koji postaju efikasniji od ostalih tek za ogromne probleme! - daleko preko snage današnjih strojeva. [Strassen-Schönhage alg. za množenje velikih brojeva]. No, uskoro to može biti drugačije.

To pokazuje da treba imati i alg. za "male" probleme, tj. da alg. treba uspoređivati na čitavom rasponu veličine problema. Za odgovarajuću veličinu problema treba koristiti pravi alg., pa treba znati i kada je koji alg. bolji.
Dakle, hibridni pristup ima veliku praktičnu korist.

- No, većina zadaća je velika!
Kako uspoređivati alg. za velike zadaće?

[IAKO JE U ~~RAČ~~ RAČUNALU, FIKSNO OGRNIČ.]

Funkcije kompleksnosti algoritama mogu biti i vrlo komplicirane. Za velike zadaće, zanima nas samo onaj dio te funkcije koji dominantno utiče na njeno ponašanje za velike argumente (veličine problema) To je tzv. vodeći ili dominantni član.

Ilustrirajmo primjerom njegovu ulogu.

Primjer 3. Standardni alg. za množenje matrica reda n zahtjeva n^3 množenja.
Na nekoj mašini, pripadni program traži recimo $3\emptyset n^3$ mikrosekundi.
Za matricu reda $n=100$, to znači 30 sekundi. Za duplo veću matricu, reda $n=200$, potrebno vrijeme je $2^3=8$ puta veće, tj. 240 sekundi.

Bitna informacija u ovom je da 2 puta veći problem zahtjeva $2^3=8$ puta više vremena.
Tj. bitan je eksponent 3, a ne konstanta $3\emptyset$.
Na nekoj brzoj mašini, potrebno vrijeme može biti $\emptyset.5n^3$ mikrosekundi, ali još uvijek 2 puta veći problem traži $2^3=8$ puta više vremena.

- Uočimo da je oblja cn^3 (c konstanta) i jedini član (pa je on i dominantan) u funkciji kompleksnosti.
- Ako je točno vrijeme zapravo
$$3\emptyset n^3 + 6\emptyset n^2,$$
još uvijek, za malo veće n, 2 puta veći problem traži $2^3=8$ puta više vremena.
- Ovo ilustrira što je dominantni dio funkcije kompleksnosti. Pri tome nam konstanta c ($=3\emptyset$ ili $\emptyset.5$) nije fundamentalno bitna za ponašanje algoritma. Ona je naravno važna u primjeni.
Kažemo da je potrebno vrijeme za ovaj algoritam "reda veličine n^3", ili kubno u n. (propor. s n^3)
[Preciznа definicija malo kasnije]. [preciznа razl. kasnije]

Za umnoženje matrica reda n postoji i drugačiji algoritam - tzv. Strassenov algoritam, čije potrebno vrijeme je "reda veličine $n^{\log_2 7}$" tj. $n^{2.81}$.

Stvarno potrebno vrijeme je možda

$$350\, n^{2.81} + 500\, n^2 \quad \text{(na pr.)}.$$

Iako je, očito, standardni alg. brži za male n, jer tu dominiraju multiplikativne konstante i preostali članovi, ipak, za dovoljno velike n je Strassenov alg. brži!

- To je željeni globalni kriterij za uspoređivanje alg.

potencija (eksponent)

Stand. množenje $\log_2 8 = 3$ > Strassen-ov alg $\log_2 7 = 2.81\ldots$

- Nažalost, postoje alg. kod kojih usporedivanje ne može tako jednostavno, jer vodeći član nije "gola" potencija od n (veličina problema).
- Na pr. standard. alg. za množenje polinoma stupnja n traži "reda veličine n^2" operacija, a postoji i algoritam koji zahtijeva "reda veličine" $n \log n$ operacija.
- Dakle, trebamo uvesti pojmove koji omogućavaju uspoređivanje funkcija za velike argumente.
- Prije toga, objasnimo još malo detaljnije, zašto nam nije bitna ni multiplikativna konstanta u vodećem članu.

  Već smo rekli da potrebno vrijeme za neki alg. može biti $30u^2$ na jednoj mašini, a $0.5u^2$ na drugoj mašini.

  Usporedba ta dva vremena je uspoređivanje strojeva (tj. implementacija), a ne uspoređivanje algoritama.

- Razumno je pretpostaviti da za različite implementacije istog algoritma mijenja slijedeći princip invarijantnosti:

  { Različite implementacije istog algoritma, razlikuju se u efikasnosti najviše do na multiplikativnu konstantu.

Precizuje, one su gotovo proporcionalne - bar za velike zadaće. [Malo kasnije - sasvim precizno].

Ovaj princip unijeli za sve strojeve s principijelno istom arhitekturom (tj. ne ovisi o detaljima arhitekt.) i ne ovisi o prog. jeziku, vještini programera i sl. (Cuvamo - za isti alg.).

Napomena: bitna promjena arhitekture, može lokalno promijeniti kompleksnost! (for petlja seq → Σ, paral → max)

Dakle, verzija:

Želimo uvesti uredaj - usporedivanje među funkcijama kompleksnosti; i to prema njihovom ponašanju za velike argumente - tj. asimptotski u beskonačnosti.

1.4. Asimptotsko ponašanje funkcija

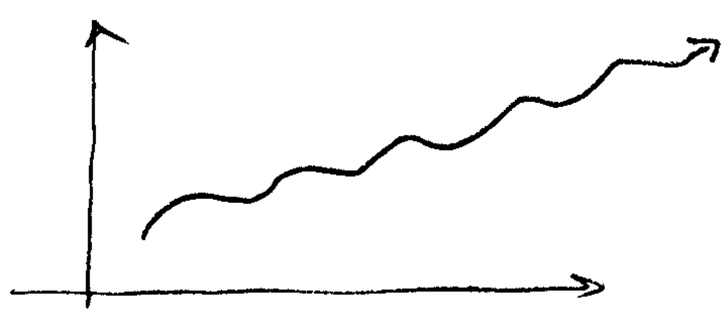
Kompleksnost je funkcija veličine zadaće nekog problema. U principu, pretpostavljamo da je skup svih zadaća beskonačan. Čak i ako je skup veličina svih zadaća neograničen podskup na pr. u $\mathbb{R}$ ili $\mathbb{R}^+$ ili $\mathbb{N}$, jer nas interesira ponašanje za dovoljno velike zadaće.

Zbog toga pretpostavljamo da je kompleksnost f nekog problema, funkcija oblika

$$f : D \to \mathbb{R}$$

gdje je D odozgo neograničen podskup, na pr. u $\mathbb{R}$. (Obično je $D \subseteq \mathbb{R}_0^+$ ili $\mathbb{N}_0$ ili $\mathbb{R}^+$ ili $\mathbb{N}$.
Analogno, obično je $f(D) \subseteq \mathbb{R}_0^+$ ili $\mathbb{N}_0$ ili $\mathbb{R}^+$ ili $\mathbb{N}$).
U praksi, očekujemo da je veća zadaća ujedno i teža. To bi značilo da je f monotono rastuća funkcija.

- Međutim, to nećemo općenito pretpostavljati, jer postoje algoritmi (FFT) kod kojih to ne mora biti tako. Kompleksnost raste "u globalu", ali ne mora lokalno:



Zbog toga nema dodatnih pretpostavki na f.

Usporedivanje kompleksnosti uvodimo sljedećom definicijom:

Definicija 1

Neka su $f, g : D \to \mathbb{R}$ dvije funkcije na odozgo neograničenom podskupu $D \subseteq \mathbb{R}$.

(a) f je manjeg reda veličine od g, ili f raste sporije od g (bitno sporije $\prec$), u oznaci

$$f(x) \overset{\in}{=} o(g(x)) \quad (x \to \infty) \qquad \text{"}o \to \text{malo } o\text{"}$$

ako postoji limes $\qquad f(x) \in o(g(x))$

$$\lim_{x \to \infty} \frac{f(x)}{g(x)}$$

i jednak je 0.

(b) f nije većeg reda veličine od g, ili f ne raste brže od g ($\preceq$), u oznaci

$$f(x) \overset{\in}{=} \mathcal{O}(g(x)) \quad (x \to \infty) \qquad \text{"}\mathcal{O} \to \text{veliko } o\text{"}$$

ako $\exists C \in \mathbb{R}$ i $\exists x_0 \in D$ takvi da je

$$\forall x > x_0, \quad |f(x)| < C\,|g(x)|. \qquad (\text{često je } C > 0)$$

(c) f je istog reda veličine kao i g, ili f raste istom brzinom kao i g, u oznaci

$$f(x) \overset{\in}{=} \Theta(g(x)) \quad (x \to \infty) \qquad \text{"}\Theta \to \text{veliko theta ili samo theta"}$$

ako $\exists c_1, c_2 > 0$ $(c_1, c_2 \in \mathbb{R})$ i $\exists x_0 \in D$ takvi da je

$$\forall x > x_0, \quad c_1\,|g(x)| < |f(x)| < c_2\,|g(x)|.$$

(Ovo bi odgovaralo pojmu asimptotski gotovo proporcionalne. Bitni dio koji smo dodali je asimptotski, tj. za sve dovoljno velike x.)
"Gotovo" prop - znači "stisnuti" između 2 konstante.

(d) [Ovo je još precizuje od Θ]

f i g su <u>asimptotski jednake</u>, (i konstante rasta su iste), u oznaci

$$f(x) \sim g(x) \qquad (x \to \infty)$$

ako postoji $\lim_{x\to\infty} f(x)/g(x)$ i vrijedi $\in$ nema smisla

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = 1 .$$

Dakle, vodeći članovi dominantni dijelovi u f i g su isti.

(e) f <u>niže manjeg reda veličine</u> od g, ili f <u>raste</u> <u>barem jednako brzo</u> kao i g, ($\geqslant$), u oznaci

$$f(x) \overset{\in}{=} \Omega(g(x)) \qquad (x\to\infty)$$ "Ω - veliko omega"

ako <u>ne</u> vrijedi $f(x) = o(g(x)) \quad (x\to\infty)$.

Ovo smo definirali kao negaciju "malo o".
U afirmativnom obliku, to znači (uz pretp. $g(x)\neq 0$ za dov. velike x)

da $\exists \varepsilon > 0$ i $\exists$ niz $x_n \in D$, $n\in\mathbb{N}$, $x_n \to \infty$,
takvi da je

$$\forall n\in\mathbb{N}, \quad |f(x_n)| > \varepsilon \cdot |g(x_n)| .$$

(f) f <u>je većeg reda veličine</u> od g, ili f <u>raste brže od</u> g ($>$), u oznaci

$$f(x) \overset{\in}{=} \omega(g(x)) \qquad (x\to\infty)$$ "ω - malo omega"

ako <u>ne</u> vrijedi $f(x) = O(g(x))$, $(x\to\infty)$.
To znači da $\exists$ nizovi $\varepsilon_n > 0$, $x_n \in D$, $n\in\mathbb{N}$, takvi

da $\varepsilon_n \to \infty$, $x_n \to \infty$

i $\forall n\in\mathbb{N}$ $\quad |f(x_n)| > \varepsilon_n |g(x_n)|$.

■

Nap: poćer od x_0 se može prebaciti u <u>za sve</u> (podesi konst).

<u>Napomena</u> Nas zanima ponašanje kompleksnosti algoritama za velike zadaće. Zbog toga smo definirali relacije asimptotskog ponašanja funkcija u okolini točke ∞. Definicija se, naravno, može poopćiti na bilo koje gomilište domene.

Kod navođenja asimptotskih relacija, oznaku $(x \to \infty)$ najčešće izpuštamo, s tim da se ona podrazumijeva.

Također funkcije kompleksnosti su uvijek nenegativne, pa apsolutnu vrijednost u def. 1.1. uvijek možemo izpustiti. Tj. relacije asimpt. ponašanja vezu funkcije kompleksnosti, a ne samo njihove apsolutne vrijednosti.

<u>Primjer 4.</u> Neki primjeri asimptotskog ponašanja i relacije među funkcijama za velike argumente

(a) relacija "o":

1. $x^2 = o(x^5)$, jače je $x^2 = o(x^{2+\varepsilon})$, $\forall \varepsilon > 0$
2. $\sin x = o(x)$
3. $15\sqrt{x} = o\left(\frac{x}{2} + \underbrace{7\cos x}\right)$
   ovo je nepotrebno, jer $7\cos x = o\left(\frac{x}{2}\right)$
   pa je dovoljno
   $15\sqrt{x} = o\left(\frac{x}{2}\right)$
4. $\frac{1}{x} = o(1)$
5. $23 \log x = o(x^{0.02})$, opet jače je $23 \log x = o(x^{\varepsilon})$, $\forall \varepsilon > 0$

Upotreba: ako su f, g kompleksnosti dva algoritma i ako je
$$f(x) = o(g(x))$$
onda je prvi algoritam <u>bitno</u> bolji za sve dovoljno velike zadaće.

Tako je $350\, n^{2.81} + 500\, n^2 = o(30u^3 + 60u^2)$
$\searrow = o(u^3)$

za razuzi primjer 3.

(b) relacija "σ".

1. $x^3 + 5x^2 + 2\cos x = \sigma(x^5)$, jače $= o(x^5)$
   $\searrow = \sigma(x^3)$ i tu <u>ne</u> može $o(x^3)$

Dakle σ lijepo izolira vodeći član, ako je funkcija g optimalno izabrana (bar za oblik f).

To pokazuje ola u usporedbi, funkciju g treba što bolje izabrati.

2. $\sin x = O(x)$ – što je slabije od ranijeg primjera $\sin x = \sigma(x)$

$\sin x = O(1)$ i to je najpače moguće.

3. $\frac{1}{1+x^2} = O(1)$, ali još jače je $\frac{1}{1+x^2} = o(1)$

Ovo pokazuje da je malo o jača i preciznija informacija od velikog O (naravno uz isti g).

U ovom primjeru to kaže, ne samo da je $\frac{1}{1+x^2}$ ograničena za velike x (to je $O(1)$), nego da ta funkcija teži prema 0, za velike x (to je $o(1)$).

– Kod uspoređivanja algoritama, relacija O je obično dovoljna. (oprez s konstantom c iz def. za O – da li je $c<1$ ili $c>1$).

(c) relacija "Θ" – daje precizniju informaciju o redu veličine i rastu funkcije.

1. $(x+1)^2 = \Theta(3x^2)$ → beznačajno, jer ovdje može biti bilo koja konst. $\neq 0$. Zbog toga se ta konst. obično NE PIŠE.
   $= \Theta(x^2)$

2. $\frac{x^2+5x+7}{5x^3+7x^2+2} = \Theta(\frac{1}{x})$

3. $\sqrt{3+\sqrt{2+x}} = \Theta(x^{1/4})$

4. $(1+\frac{3}{x})^x = \Theta(1)$ (limes je e^3)

Ova relacija zapravo točno izdvaja "vodeći" ili "dominantni" član. To je prava informacija o pojedinom alg.
(o, O su za uspoređivanje. Kad je Θ u usporedbi – onda imamo 2 podjednaka alg. – i treba precizno analizirati njihovo ponašanje – uključivo i multiplikativne konstante, za usporedbu).

(d) relacija "$\sim$" - precizno izoliva red veličine, <u>uključivo</u> i "vodeću" <u>multiplikativnu konstantu</u>!

1. $x^2 + x \sim x^2$
2. $(3x+1)^4 \sim 81x^4$ (ovdje g mora imati pripadnu konstantu)
3. $\sin\frac{1}{x} \sim \frac{1}{x}$ (jer $\sin x \sim x$ u okolini točke 0)
4. $\frac{2x^3+5x+7}{x^2+4} \sim 2x$
5. $2^x + 7x^3 \log x + \cos x \sim 2^x$.

Relacije Ω i ω se očito NE koriste za algoritme. Donje ograde kompleksnosti alg. nisu interesantne. Međutim, za kompleksnost - tj. za PROBLEME, bez obzira na alg., ove relacije su itekako korisne - "problem je bar toliko težak" (v. §7). ■

- U primjeru (d2), vidjeli smo da je $g(x) = 81x^4$, tj. g mora sadržavati pripadnu konstantu za asimptotsku jednakost. Katkad nam ta konstanta nije bitna, već je dovoljno znati da ona postoji.
Ako nam je dovoljna takva informacija, to znači da nas zadovoljava da su $(3x+1)^4$ i x^4 asimptotski proporcionalne.
Pripadna definicija je:

<u>Definicija 2</u>. U 2 iste pretpostavke kao u def. 1, kažemo:

(g) funkcije f i g su <u>asimptotski proporcionalne</u> ako postoji konstanta $c \neq 0$, takva da je

$$f(x) \sim c\, g(x).$$

← točno tako ćemo i pisati, ne specificirajući c. ■

Drugim riječima, to znači

$$\lim_{x\to\infty} \frac{f(x)}{c \cdot g(x)} = 1,$$ tj., zbog $c \neq 0$,

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = c.$$

Elementarna svojstva ovih relacija dana su sljedećom propozicijom koju ne dokazujemo (barem ne cijelu).

Propozicija 1.

(a) Relacije asimptotskog ponašanja $\mathcal{O}$ i $\sim$ su relacije ekvivalencije na $\mathbb{R}^D$ – skupu svih funkcija $f: D \to \mathbb{R}$.
[To opravdava njezin "istog reda veličine" i "iste" iz definicije].

(b) Na skupu klasa ekvivalencije $\mathbb{R}^D/\Theta$, relacije o i $\mathcal{O}$ su relacije parcijalnog uređaja.
[Te relacije odgovaraju relacijama "manje" odnosno "manje ili jednako"].

{Zašto baš na $\mathbb{R}^D/\Theta$? Da se eliminira utjecaj konstanti, koje inače bitne za o, $\mathcal{O}$.}

(c) Potrebno, jer je Θ relacija ekvivalencije, to je Θ i simetrična, tj:

Ako je $f(x) = \Theta(g(x))$, onda je i $g(x) = \Theta(f(x))$.

Dz: Zbog $c_1, c_2 > 0$, iz definicije je:

$$c_1 |g(x)| < |f(x)| < c_2 |g(x)| \quad \text{za } x > x_0 \in D$$

pa je

$$\frac{1}{c_2} |f(x)| < |g(x)| < \frac{1}{c_1} |f(x)| \quad \text{– ” –}.$$

(d) Potrebno, relacija $\mathcal{O}$ je antisimetrična na $\mathbb{R}^D/\Theta$, tj.:

Ako je $f(x) = \mathcal{O}(g(x))$ i $g(x) = \mathcal{O}(f(x))$, onda je

$$f(x) = \Theta(g(x))$$

i

$$g(x) = \Theta(f(x)) \quad (\text{po (c)}).$$

Dz: Po def. 1., postoje konstante $C_1, C_2 > 0$ i postoje $x_1, x_2 \in D$ takvi da je

$$|f(x)| < C_1 |g(x)|, \quad \forall x > x_1$$
$$|g(x)| < C_2 |f(x)|, \quad \forall x > x_2.$$

Definiramo $x_0 = \max\{x_1, x_2\}$. Dijeljenjem druge relacije s $C_2 > 0$ slijedi

$$\forall x > x_0 \,, \quad \frac{1}{C_2} |g(x)| < |f(x)| < C_1 |g(x)| \,. \blacksquare$$

Funkcije kompleksnosti su "po prirodi" nenegativne. Tada možemo izpustiti apsolutne vrijednosti i u prethodnoj definiciji 1.

Relacija Θ tada postaje dobiva na snazi. Prethodna definicija omogućuje ove definicije

Definicija 3.

Ako su f i g nenegativne funkcije i ako je

$$f(x) = \Theta(g(x))$$

onda kažemo da su f i g kodominantne $\blacksquare$

(Precizuje bi bilo "asimptotski kodominantne").

Vrijedi reći da funkcije kompleksnosti "u globalu" rastu, ili, barem, "u globalu" ne padaju. (Veća zadaća → veća kompleksnost).

U primjenama, za one vrlo složene mjere kompleksnosti mjerodavni

$$\lim_{x \to \infty} f(x) = +\infty$$

tj. kompleksnost neograničeno raste s veličinom problema.

Naš sljedeći zadatak je klasifikacija funkcija kompleksnosti prema "brzini rasta" – s ciljem da dobijemo jednostavnu informaciju o kvaliteti algoritma.

Suprotno od rasta je pad. Funkcije koje dijele "rast" od "pada" su konstante oblika

$$g(x) = c$$

uz $c > 0$.

Najblaža korisna pretpostavka za funkciju kompleksnosti je da ona "raste bar istom brzinom kao i proizvoljna konstanta".

Definicija 4.

Neka je $f: D \to \mathbb{R}_0^+$ nenegativna funkcija na odozgo neograničenom podskupu $D \subseteq \mathbb{R}_0^+$. f je funkcija kompleksnosti ako vrijedi

$$f(x) \in \Omega(c)$$

(jasno je $f(x) = \omega(c)$, što je već "blizu" $f(x) \to \infty$ za $x \to \infty$).

gdje je $c > 0$ neka konstanta. ■

(Ova def. ne ovisi o izboru konstante c, pa uvijek uzeti $c = 1$)

Sljedeća definicija uvodi klasifikaciju funkcija kompleksnosti prema asimptotskom ponašanju (rastu) za velike argumente.

Definicija 5.

Neka je f funkcija kompleksnosti.

(a) f blago (ili sporo) raste, ako f raste sporije od bilo koje pozitivne potencije x^ε.

$$\forall \varepsilon > 0, \quad f(x) \in o(x^\varepsilon)$$

(b) f polinomno raste, ako f raste "skromno" brzinom kao i neka pozitivna potencija. Preciznije, f raste barem tako brzo kao neka potencija, ali ne brže od neke potencije. Tj. postoje $a_1, a_2 > 0$, takvi da je

$$f(x) \in \Omega(x^{a_1}) \quad i \quad f(x) \in O(x^{a_2}).$$

(c) f blago eksponencijalno raste, ako f raste brže od bilo koje potencije, ali raste sporije od bilo koje eksponencijalne funkcije oblika c^x, uz $c > 1$. Tj:

$$\forall a > 0, \quad f(x) \in \Omega(x^a)$$

i

$$\forall \varepsilon > 0, \quad f(x) \in o((1+\varepsilon)^x).$$

(d) f <u>eksponencijalno</u> raste, ako postoje $c_1, c_2 > 1$ takvi da je

$$f(x) \overset{\in}{=} \Omega(c_1^x) \quad i \quad f(x) \overset{\in}{=} O(c_2^x).$$

(e) f <u>neeksponencijalno</u> raste, ako f raste brže od bilo koje eksponencijalne funkcije, tj.

$$\forall c > 1, \quad f(x) \overset{\in}{=} \Omega(c^x). \blacksquare$$

<u>Napomene</u>:

U oba definicija, svagdje gdje stoji O, može stajati o i obratno. Isto tako, svagdje gdje stoji Ω, može stajati ω i obratno.

$$O \leftrightarrow o$$
$$\Omega \leftrightarrow \omega.$$

Navamo, ono se može definirati da mjedi i za funkcije oblika $f(x) = O(1)$ (ograničene) ili $f(x) = o(1)$ (po volji male) za velike argumente, ali takve funkcije nisu zanimljive u analizi kompleksnosti.
Za takve funkcije može se definirati analogna klasifikacija pada za velike argumente. Dobiva se direktno iz one, primjenom na funkciju $1/f$ (uz $f(x) \neq 0$ za dovoljno velike x).

- Za praksu su najvažniji algoritmi najniže eksponencijalne kompleksnosti.
  Eksponencijalni algoritam dozvoljava, obično, vješavanje zadaća umjerenih veličina ($|x| \sim 10, 20, 30$), koje zadovoljavaju standardne potrebe. Vau toga – ne zahtjeva se više egzaktno, već približno rješenje (TSP).
  U teoriji, posebnu ulogu imaju polinomni algoritmi, bez obzira na veličinu a_2.
  - Navamo, <u>efikasnim</u> ili <u>brzim</u>, možemo smatrati samo one algoritme čija kompleksnost blago raste, ili raste polinomno i to s malom potencijom a_2 (otprilike $a_2 \leq 4$) – u ovisnosti o veličini problema.
  - Većina algoritama u praksi ne prelazi $a_2 = 3$. (linearni, kvadratni, kubni).

Primjer 5. Neki karakteristični netrivijalni primjeri klasifikacije rasta:

(a) 1. $f(x) = \log x$ blago rasta.

2. Još sporije rasta $f(x) = \log\log x$.

Nastavimo li induktivno dalje, za $n \in \mathbb{N}$

$$f(x) = \underbrace{\log \dots \log}_{n \text{ puta}} x$$

dobivamo monotono rastuće funkcije (nadite domenu!) koje su sve sporije rastuće.
Ovo pokazuje da nema "najsporije" rastuće funkcije.

(b) 1. $f(x) = x \log x$ polinomno rasta

2. $f(x) = x^2 / \log x$ — " —

(c) 1. $f(x) = x^{\log x} = e^{\log^2 x}$ blago eksponencijalno rasta.

Na pr. $x^{\log x} > x^{1000}$ za $\log x > 1000$
tj. $x > e^{1000}$.

2. $f(x) = e^{\sqrt{x}}$

(d) 1. $f(x) = x 2^x$ eksponencijalno rasta

2. $f(x) = x^{\log x} 3^x$ — " —

2. $\longrightarrow$ 3. $f(x) = \dfrac{e^x}{e^{\sqrt{x}} + x^{50}}$

(e) 1. $f(x) = \Gamma(x+1)$ (odn. $f(n) = n!$) nadeksponencijalno rasta

2. $f(x) = 2^{x^2}$ rasta još brže.

3. $f(x) = x^x$ nadeksponencijalno, isto brže od Γ

■

Primjer 6. Ovdje su potencije oblika $f(x)=x^a$, $a>0$, trivijalni primjeri funkcija polinomnog rasta. Međutim, polinomni rast funkcije ne znači da postoji $a>0$ takav da je

$$f(x) \overset{\in}{=} \Theta(x^a).$$

Na primjer, za funkciju

$$f(x) = x^a \log x \quad , \quad a>0$$

vrijedi:

$$\forall a_1 \leq a, \quad f(x) \overset{\in}{=} \omega(x^{a_1}) \qquad (>)$$

i

$$\forall a_2 > a, \quad f(x) \overset{\in}{=} o(x^{a_2}) \qquad (<).$$

Dokaz:

Za $a_1 \leq a$ je $\dfrac{f(x)}{x^{a_1}} = x^{\underbrace{a-a_1}_{\geq 0}} \log x \to \infty$, za $x \to \infty$,

a za $a_2 < a$ je $\dfrac{f(x)}{x^{a_2}} = \dfrac{\log x}{x^{\underbrace{a_2-a}_{>0}}} \to 0$, za $x \to \infty$.

Dakle, $f(x) = x^a \log x$ polinomno raste, ali ne postoji potencija s kojom bi f bila kodominantna ■

Slijedeći primjer je generalizacija ovog.

Primjer 7. Neka je $n \in \mathbb{N}_0$ fiksan i neka je $f: D \to \mathbb{R}$ funkcija oblika

$$f(x) = x^{a_0} (\log x)^{a_1} (\log\log x)^{a_2} \cdot \ldots \cdot (\underbrace{\log \ldots \log}_{n \text{ puta}} x)^{a_n}$$

gdje su $a_0, a_1, \ldots, a_n \in \mathbb{R}$, i bar jedan od tih brojeva je različit od 0, s tim da je

$$a_k > 0 \quad \text{za} \quad k = \min\{i \mid a_i \neq 0\}.$$

Ova je monotono rastuća funkcija na prirodno definiranoj domeni.

Kao u prošlom primjeru, lako se dokazuje da je:

$$\forall \varepsilon > 0, \quad f(x) \overset{\in}{=} o(x^{a_0+\varepsilon})$$
$$\forall \varepsilon > 0, \quad f(x) \overset{\in}{=} \omega(x^{a_0-\varepsilon}).$$

(Može, naravno, još precizirati, osim o $a_1, \ldots$).

Dakle, za $a_0 > 0$, f polinomno raste, a za $a_0 = 0$ f blago raste.

Za $a_0 = 0$, slična karakterizacija vrijedi za najmanji $a_k \neq 0$ (sa $\pm \varepsilon$) ■

Funkcije ovog oblika najčešće se koriste za karakterizaciju kompleksnosti polinomnih algoritama ("vodeći član" ili red veličine).

1.5. Rekurentne (rekurzivne) relacije

Analiza kompleksnosti algoritma vrlo često vodi na rekurzivne relacije.

Petlja ili rekurzija u algoritmu omogućava opis rješenja zadaće veličine 1×1, preko rješenja neke manje zadaće - istog ili sličnog tipa.

Na isti način možemo kompleksnost algoritma za tu zadaću x, izraziti preko manje zadaće (opet - ne nužno iste).

Tada za kompleksnost dobivamo rekurzivnu relaciju.

Ta relacija može biti jednadžba. No, često nas zanima samo red veličine kompleksnosti - pa koristimo nejednadžbe.

Zbog toga, ćemo prvo pokušati zainteresirati o rješavanju rekurzivnih jednadžbi. Zatim ćemo pokazati neke načine za procjenu ili ocjenu (odozgo) za rješenja rekurzivnih nejednadžbi.

Počinjemo s rekurzivnim jednadžbama, prvog reda - slijedeća vrijednost u nizu ovisi samo o jednoj prethodnoj - i to točno o prethodnom članu.

Oznake:

Smatramo da su, ovdje, veličine zadaća 1×1 prirodni brojevi - oznaka je $n \in \mathbb{N}_0$. Kompleksnost je najčešće vremenska - oznaka je, kao funkcija:

$$T(1 \times 1) = T(n) = t_n$$

jer vrijednosti funkcije T možemo interpretirati i kao niz.

Oblik relacije je:

$$T(n) = t_n \overset{=}{\leq} f_n(t_{n-1}, \ldots, t_0), \quad n \in \mathbb{N}$$

jer dozvoljavamo da f_n ovisi o n.

Opći oblik rekurzivne (diferencijske) jednadžbe prvog reda je:

$$t_n = f_n(t_{n-1}) \quad ; \quad n \in \mathbb{N}.$$

Ako znamo sve funkcije f_n, $n \in \mathbb{N}$, dovoljno je zadati početnu vrijednost t_0, da citav niz t_n bude jednoznačno odredjen (ako su f_n definirane korektno).

Za početak, pretpostavimo da je f_n linearna (afina) funkcija:

$$f_n(t) = b_n t + c_n \quad , \quad n \in \mathbb{N}$$

gdje su b_n, c_n zadani.

Primjer 8. Najjednostavniji oblik je $f_n(t) = bt$, tj. svi b_n su isti, a svi c_n su 0.
Relacija ima oblik

$$t_n = b \cdot t_{n-1} \quad , \quad n \in \mathbb{N}.$$

Za zadani t_0, direktno dolivamo sve članove uzastopnim uvrštavanjem:

$$t_1 = b \cdot t_0$$
$$t_2 = b \cdot t_1 = b^2 t_0$$
$$\vdots$$
$$t_n = b \cdot t_{n-1} = \text{indukcija} = b^n \cdot t_0 \quad \blacksquare$$

Preciznije, uvrštavanje naslućuje odgovor, a indukcija ga dokazuje.

Primjer 9. Nešto kompliciranji oblik je $f_n(t) = b_n t$, tj. f_n je linearna (homogena, a ne afina) funkcija.
Za zadani t_0, rješenje relacije

$$t_n = b_n \cdot t_{n-1}$$

ide istim putem:

$$t_1 = b_1 \cdot t_0$$
$$t_2 = b_2 \cdot t_1 = b_2 \cdot b_1 \cdot t_0$$
$$\vdots$$
$$t_{n-1} = b_{n-1} \cdot t_{n-2} = b_{n-1} \cdot \ldots \cdot b_1 \cdot t_0$$
$$t_n = b_n \cdot t_{n-1} = b_n \cdot b_{n-1} \cdot \_ \cdot b_1 \cdot t_0$$

tj.

$$t_n = \left(\prod_{i=1}^{n} b_i \right) \cdot t_0 \quad \blacksquare$$

Napadnimo sada ozbiljni problem.

Primjer 10. Opća linearna, nehomogena rekurzija prvog reda je:

$$t_n = b_n t_{n-1} + c_n \,, \quad n \in \mathbb{N}.$$

Uz zadani t_0, ista strategija daje:

$$t_1 = b_1 t_0 + c_1$$

$$t_2 = b_2 t_1 + c_2 = b_2 b_1 t_0 + b_2 c_1 + c_2$$

$$\vdots$$

što ubrzo postaje zamorno, iako vodi do rješenja.
Sistematski pristup, ali metoda za rješenje je:

(a) Uvodimo supstituciju koja ima oblik rješenja pripadne homogene jednadžbe:

$$t_n = b_n \cdot \ldots \cdot b_1 \cdot u_n \,, \quad u \in \mathbb{N}.$$

Dobivamo novi nepoznati niz u_n, uz dogovor istog starta $u_0 = t_0$.
Ova zamjena vrijedi u relaciji daje:

$$\underbrace{b_1 \cdot \ldots \cdot b_n}_{\text{isto}} u_n = \underbrace{b_n \cdot b_1 \cdot \ldots \cdot b_{n-1}}_{\text{isto}} u_{n-1} + c_n \,, \quad n \in \mathbb{N}$$

Koeficijenti uz u_n, u_{n-1} su isti, pa podijelimo relaciju s tim koef. Izlazi:

$$u_n = u_{n-1} + d_n \,, \quad n \in \mathbb{N}$$

uz oznaku:

$$d_n = \frac{c_n}{b_1 \cdot \ldots \cdot b_n} \,, \quad n \in \mathbb{N}.$$

Uočimo da su d_n poznati (b-oni i c-oni su zadani).

(b) Ostaje još izvesti ovu relaciju. No, ova je trivijalna, jer se svodi na zbrajanje prethodnika:

$$u_n = u_0 + \sum_{j=1}^{n} d_j \,, \quad n \in \mathbb{N}.$$

(Može istom tehnikom uvrštavanja pa sređivanja i indukcija).

(c) Povratak na polazni niz: $t_n = b_1 \cdot \ldots \cdot b_n u_n \Rightarrow$

$$t_n = (b_1 \cdot \ldots \cdot b_n) \cdot \left(t_0 + \sum_{j=1}^{n} d_j\right), \quad n \in \mathbb{N}.$$

Možemo još i d-ove vratiti u b, c-ove: $d_j = \frac{c_j}{b_1 \cdot \ldots \cdot b_j}$

$$t_n = b_1 \cdot \ldots \cdot b_n t_0 + \sum_{j=1}^{n} b_{j+1} \cdot \ldots \cdot b_n c_j \quad \blacksquare$$

Zadatak 2. Riješite relaciju

$$t_n = 3t_{n-1} + n, \quad n \in \mathbb{N}$$

uz početni uvjet $t_0 = 0$.

Rješenje: $b_n = 3$, $c_n = n$. Supstitucija je

$$t_n = 3^n u_n$$

pa dobivamo:

$$u_n = u_{n-1} + \frac{n}{3^n}.$$

Sumacija daje:

$$u_n = \sum_{j=1}^{n} \frac{j}{3^j} \quad (\text{jer } u_0 = t_0 = 0)$$

i na kraju je:

$$t_n = 3^n \cdot \sum_{j=1}^{n} \frac{j}{3^j} = \sum_{j=1}^{n} j \cdot 3^{n-j} \quad \blacksquare$$

Generalizacija jednadžbi prvog reda su jednadžbe višeg reda - na pr. k-tog. Slijedeća nejednost t_n u nizu ovisi o k prethodnih.

Homogene rekurzije:

Najjednostavniji slučaj takvih rekurzivnih jednadžbi su linearne, homogene jednadžbe, s konstantnim koeficijentima.

Oblik je:

$$a_0^{(k)} t_n + a_1^{(k-1)} t_{n-1} + \ldots + a_k^{(0)} t_{n-k} = 0 \qquad (1)$$

(Cako sve članove niza prebacimo na lijevu stranu).

Koeficijenti $a_0^{(k)} (\neq 0), a_1^{(k-1)}, \ldots, a_k^{(0)} (\neq 0)$ su konstante (tj. ne ovise o n).

Linearnost - sadrži samo linearnu kombinaciju t_i-ova. Nema članova oblika $t_i t_j$ ili t_i^2 i sl.

Homogenost - linearna kombinacija t_i je jednaka nuli, a ne nekom drugom.

Prethodni primjer sugerira traženje rješenja u obliku

$$t_n = x^n$$

gdje je x nepoznata konstanta. (Po analogiji s diferencijalnim jednadžbama $y(t) = e^{xt}$).
Uvrštavanjem ovog oblika u jednadžbu (1) dobivamo jednadžbu za x:

$$a_0 x^n + a_1 x^{n-1} + \dots + a_k x^{n-k} = 0$$

Za $n > k$ ova jednadžba sigurno ima rješenje $x = 0$.
No, to vodi na $t_n = 0$, što je očito trivijalno rješenje homogene jednadžbe (1). To rješenje nas ne zanima (komplement = $\emptyset$), pa dijeljenjem sa x^{n-k} (x inače ne bi smio ovisiti o n) izlazi:

$$P_k(x) = a_0 x^k + a_1 x^{k-1} + \dots + a_k = 0 . \qquad (2)$$

Ova jednadžba je polinomna, stupnja k u x i zove se <u>karakteristična jednadžba</u> rekurzivne relacije (1).
(Stupanj je točno k, zbog $a_0 \neq 0$, i nema trivijalnih rješenja, zbog $a_k \neq 0$ – inače jednadžba i relacija imaju niži stupanj).

Oznaka x, umjesto uobičajenog λ, je ovdje ugodna za polinom!

Prije nastavka, uočimo da iz linearnosti jednadžbe slijedi da je linearna kombinacija bilo kojih rješenja, opet rješenje za (1). Tj. rješenja za (1) čine vektorski prostor. Znamo (more se dokazati) da je on dimenzije k, pa tražimo bazu u tom prostoru – linearno nezavisan skup od k rješenja.

– Karakteristična jednadžba (2) ima točno k rješenja

$$r_1, r_2, \dots, r_k \in \mathbb{C}$$

(ako su $a_i \in \mathbb{R}$ ili $\mathbb{C}$). Onda je

$$t_n = r_1^n, r_2^n, \dots, r_k^n \quad , n \in \mathbb{N}$$

jedan skup rješenja u prostoru rješenja. Taj skup ne mora biti baza – ako postoje bar 2 ista r-a, onda imamo najviše $k-1$ razl. rješenja, pa je taj skup linearno zavisan i fale rješenja do baze (ako eliminiramo takva ista – višestruka rješenja).

Bilo koja linearna kombinacija tih rješenja, oblika:

$$t_n = \sum_{i=1}^{k} c_i r_i^n \qquad (3)$$

s $c_1, \dots, c_k \in \mathbb{C}$ (ili $\mathbb{R}$) je također rješenje za (1).

(A) Ako su svi korijeni karakteristične jednadžbe različiti

$$i \neq j \Rightarrow r_i \neq r_j \quad , \forall i, j \in \{1, \dots, k\}$$

onda su funkcije r_i^n, $i = 1, \dots, k$ i linearno nezavisne, pa čine bazu prostora rješenja.

Sva rješenja za (1) su oblika (3). Konstante $c_1, \dots, c_k$ se odreduju iz k početnih uvjeta – treba zadati prvih k članova niza.

(B) Ako karakteristična jednadžba ima višestruke korijene, onda dobivamo linearno zavisan skup, tj. fale nam rješenja do baze.

Pretpostavimo da je r višestruki korijen karakteristične jednadžbe

$$P_k(x) = a_0 x^k + a_1 x^{k-1} + \dots + a_k = 0 .$$

Za bilo koji n, $n \geq k$, promatramo polinom h_n, stupnja n definiran s

$$h_n(x) = x \cdot [\underbrace{x^{n-k} P_k(x)}_{\substack{\text{stupnja } n \\ = \text{ono što orig.} \\ \text{dolazemo ulaz.} \\ t_n = r^n \\ \text{u jedn. (1)}}}]' = \quad \text{deriv. je stupnja } n-1, \text{ pa zato još } x \text{ vani.}$$

$$= a_0 n x^n + a_1 (n-1) x^{n-1} + \dots + a_k (n-k) x^{n-k} .$$

Ako je r barem dvostruki korijen karakt. jedn., onda postoji polinom Q takav da je

$$P_k(x) = (x-r)^2 \cdot Q(x)$$

Tada je:

$$h_n(x) = x\cdot\left[\underbrace{x^{n-k}}\cdot(x-r)^2\underbrace{Q(x)}\right]' = x\cdot\left[(x-r)^2\cdot(x^{n-k}Q(x))\right]'$$

$$= x\cdot\left[2\cdot\underbrace{(x-r)}\cdot x^{n-k}Q(x) + \underbrace{(x-r)^2}\cdot(x^{n-k}Q(x))'\right]$$

tj.

$$h_n(x) = (x-r)\cdot g_n(x).$$

Posebno je

$$h_n(r) = 0$$ (za sve dovoljno velike n, na pr. $n \geq k$)

tj.

$$a_0\underbrace{nr^n}_{k} + a_1\underbrace{(n-1)r^{n-1}}_{k-1} + \ldots + a_k\underbrace{(n-k)r^{n-k}}_{0} = 0$$

što pokazuje da je i

$$t_n = n\cdot r^n$$

također rješenje jednadžbe (1).

- Općenito, ako je r korijen karakterističke jednadžbe multipliciteta m, onda su sva pripadna linearno nezavisna rješenja rekurzivne jednadžbe dana sa:

$$t_n = r^n,\ nr^n,\ n^2r^n,\ \ldots,\ n^{m-1}r^n.$$

Opće rješenje je linearna kombinacija onih rješenja, po svim različitim korijenima karakteristične jednadžbe.

- Opet, konstante $c_1, \ldots, c_k$ se odreduju iz početnih uvjeta.

— · —

Prije nastavka, za analizu algoritama je važna sljedeća napomena:

Napomena: Obično nas ne zanima točno i potpuno rješenje rekurzivne relacije, već samo red veličine tog rješenja.

Većina budućih primjera i zadataka će ići u tom smjeru – kako što efikasnije odrediti red veličine rješenja, a tek ponekad ćemo gledati punu formu.

(Puna forma je korisna za empirijsko određivanje parametara – bar nekoliko dominantnih dijelova ili parametara, ali ne previše [da sum ne uprosječi rezultate])

<u>Primjer 11.</u> (Alg., Ex. 2.3.1 p. 66) Zadana je rekurzivna relacija

$$t_n - 3t_{n-1} - 4t_{n-2} = 0, \text{ za } n \geq 2$$

uz početne uvjete $t_0 = 0, t_1 = 1$.

<u>Rješenje</u>: Karakteristična jednadžba je

$$x^2 - 3x - 4$$

s rješenjima $r_1 = -1$, $r_2 = 4$. Opće rješenje relacije je

$$t_n = c_1 (-1)^n + c_2 4^n.$$

Oprez! Odavde još <u>ne</u> možemo zaključiti da je

$$t_n = \Theta(4^n) \text{ ili } t_n \sim c_2 4^n$$

jer može izaći $c_2 = 0$.

Iz početnih uvjeta dobivamo:

$$\begin{array}{lrcl} n=0: & c_1 + c_2 = 0 & \Rightarrow & c_1 = -c_2 \\ n=1: & -c_1 + 4c_2 = 1 & +\hookrightarrow \Rightarrow & 5c_2 = 1, \; c_2 = \frac{1}{5} \end{array}$$

Rješenje je:

$$c_1 = -\frac{1}{5}, \; c_2 = \frac{1}{5}.$$

Potpuno rješenje je:

$$\underline{t_n = \frac{1}{5}\left[4^n - (-1)^n\right]}, \; n \in \mathbb{N}_0.$$

Očito je

$$\underline{t_n \sim \frac{1}{5} 4^n}$$

i rješenje oscilira oko dominantnog člana, s amplitudom $\frac{1}{5}$. Dakle, rješenje se može samim jednostavno izračunati kao:

$$t_n = \left[\frac{1}{5} 4^n\right]$$

gdje $[.]$ označava funkciju "najbliže cijelo" ili <u>round</u> ■

– To je pravo – korektno zaokruživanje, a postoje razni dogovori koja se vrijednost uzima za $[x]$, ako je x oblika $n + \frac{1}{2}$ – točno na polovini između 2 cijela broja.

Varijante: $[n + \frac{1}{2}] = n + 1$ → "na gore"

$[n + \frac{1}{2}] = n$ → "na dole" (odbacivanje)

$$[n + \tfrac{1}{2}] = \begin{cases} n+1, & \text{za } n \text{ neparan} \\ n, & \text{za } n \text{ paran} \end{cases} \rightarrow$$ "simetrično" ili "perfektno" zaokr – na najbliži parni.

Rješenja - koreni karakteristične jednadžbe su, općenito, kompleksni brojevi.
Sljedeći primjer ilustrira postupak u tom slučaju.

Primjer 12. Zadana je rekurzivna relacija (Alg., Prob. 2.3.1 p. 67)

$$t_n = 2t_{n-1} - 2t_{n-2}\,, \quad n \geq 2$$

uz početne uvjete $t_0 = 0$, $t_1 = 1$.

Rješenje: U standardnom obliku, relacija glasi:

$$t_n - 2t_{n-1} + 2t_{n-2} = 0\,, \quad \text{za } n \geq 2\,.$$

Karakteristična jednadžba je

$$x^2 - 2x + 2 = 0\,.$$

Rješenja su:

$$r_{1,2} = \frac{2 \pm \sqrt{4-8}}{2} = \frac{2 \pm \sqrt{-4}}{2} = 1 \pm i\,.$$

Opći oblik rješenja je:

$$t_n = c_1 (1+i)^n + c_2 (1-i)^n\,.$$

Iz početnih uvjeta dobivamo:

$$n = 0: \quad c_1 + c_2 = 0 \quad \Rightarrow \quad c_1 = -c_2$$

$$n = 1: \quad \underbrace{c_1(1+i) + c_2(1-i)}_{c_1 + c_2 + i(c_1 - c_2)} = 1$$

$$c_1 + c_2 + i(c_1 - c_2) = 1 \quad \Rightarrow \quad c_1 - c_2 = -i$$

($c_1 + c_2 = 0$ iz prve jedn.)

Nakon supstituiranja c_2, izlazi: $2c_1 = -i$ tj.

$$c_1 = -\frac{1}{2}i\,, \quad c_2 = \frac{1}{2}i\,.$$

Rješenje naše rekurzije je:

$$t_n = \frac{1}{2}i \cdot \left[-(1+i)^n + (1-i)^n \right]\,, \quad n \in \mathbb{N}_0\,.$$

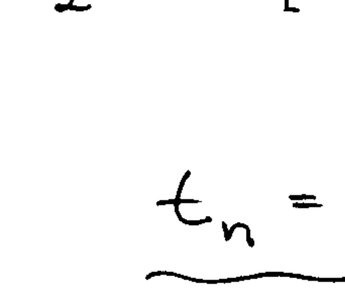
No, iz početnih uvjeta i rekurzije je očito da rješenja moraju biti cijeli brojevi. To se iz ove forme ne vidi odmah.

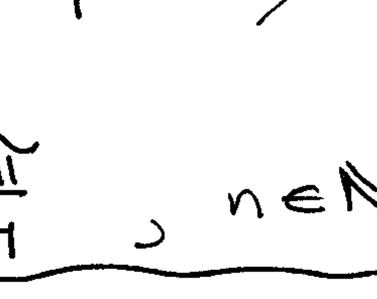
Također, iz ove forme nije moguće odrediti red veličine (asimptotsko ponašanje) rješenja za velike n.

"Jedan pristup je razvoj po binomnom teoremu i sređivanje u kartezijevom obliku.

Lakši pristup je prelaz na polarni oblik.

$$1+i = \sqrt{2}\, e^{i\cdot\frac{\pi}{4}}, \qquad 1-i = \sqrt{2}\, e^{i\cdot\frac{7\pi}{4}} = \sqrt{2}\, e^{-i\frac{\pi}{4}}$$





Onda je:

$$(1+i)^n = (\sqrt{2})^n e^{i\cdot n\frac{\pi}{4}} = (\sqrt{2})^n\left(\cos\frac{n\pi}{4} + i\sin\frac{n\pi}{4}\right) \quad \leftarrow \times(-i)$$

$$(1-i)^n = (\sqrt{2})^n e^{-in\frac{\pi}{4}} = (\sqrt{2})^n\left(\cos\frac{n\pi}{4} - i\sin\frac{n\pi}{4}\right). \quad \leftarrow \times(+i)$$

Oduzmemo izrazi:

$$t_n = \frac{1}{2}(\sqrt{2})^n\left[-i\cos\frac{n\pi}{4} + \sin\frac{n\pi}{4} + i\cos\frac{n\pi}{4} + \sin\frac{n\pi}{4}\right]$$

tj.

$$t_n = (\sqrt{2})^n \sin\frac{n\pi}{4}, \quad n\in\mathbb{N}_0$$

Dakle, rješenje jest realno, a lako se vidi da je i cjelobrojno ($|\sin\frac{n\pi}{4}| \in \{0, \frac{\sqrt{2}}{2}, 1\}$).

Vrijednosti za t_n osciliraju po predznaku, a amplituda raste.

Asimptotski gledano, otkud vijedi

$$t_n = \Theta(2^{n/2})$$

i to je najjače što možemo reći. Kodominantnost (Θ) tj. asimptotska proporcionalnost <u>ne</u> vijedi, jer u nizu ima nula (tj. ne možemo osigurati $c>0$ u Θ i $\sim$). ■

Slijedeći primjer ilustrira pojavu višestrukih korijena karakteristične jednadžbe.

<u>Primjer 13.</u> (Alg., Ex. 2.3.3, p. 67-68) Zadana je rekurzija

$$t_n = 5t_{n-1} - 8t_{n-2} + 4t_{n-3}, \quad n\geq 3$$

uz početne uvjete $t_0=0$, $t_1=1$, $t_2=2$.

(Rješenje, naravno, <u>nije</u> $t_n=n$.).

Rješenje: U standardnom obliku, rekurzija glasi:

$$t_n - 5t_{n-1} + 8t_{n-2} - 4t_{n-3} = 0.$$

Karakteristična jednadžba je:

$$x^3 - 5x^2 + 8x - 4 = 0$$

ili, faktorizirano (v. na pr. Cardanove formule)

$$(x-1)(x-2)^2 = 0.$$

Korijeni su: $r_1 = 1$, multipliciteta 1 i
$r_2 = 2$, multiplicitet 2.

Opće rješenje rekurzije je:

$$t_n = c_1 \cdot \underline{1^n} + c_2 \cdot \underline{2^n} + c_3 \cdot \underline{n \cdot 2^n}.$$

Iz početnih uvjeta dobivamo:

$$\begin{aligned} n=0: &\quad c_1 + c_2 = 0 && \Rightarrow c_1 = -c_2 \\ n=1: &\quad c_1 + 2c_2 + 2c_3 = 1 && (1)-(0): \; c_2 + 2c_3 = 1 \;\big|\cdot(-3)\big| + \\ n=2: &\quad c_1 + 4c_2 + 8c_3 = 2 && (2)-(0): \; 3c_2 + 8c_3 = 2 \\ & && \Rightarrow \quad 2c_3 = -1 \\ & && \Rightarrow \quad c_3 = -\tfrac{1}{2} \\ & && \Rightarrow (1)-(0): \; c_2 = 2 \\ & && \Rightarrow \quad c_1 = -2 \end{aligned}$$

Rješenja ovog sustava su:

$$c_1 = -2, \; c_2 = 2, \; c_3 = -\frac{1}{2}.$$

Potpuno rješenje rekurzije je:

$$t_n = -2 + 2 \cdot 2^n - \frac{1}{2} \cdot n 2^n \quad , \; n \in \mathbb{N}_0$$

ili, u sređenom obliku:

$$\begin{aligned} t_n &= 2^{n+1} - n \cdot 2^{n-1} - 2 \\ &= \underline{(4-n) \cdot 2^{n-1} - 2} \quad , \; n \in \mathbb{N}_0. \end{aligned}$$

Asimptotsko ponašanje rješenja je oblo:

$$\underline{t_n \sim -n 2^{n-1}}.$$

Nehomogene rekurzije

Linearna, nehomogena jednadžba s konstantnim koeficijentima ima isti opći oblik kao i (1), ali je desna strana, općenito, različita od $\emptyset$:

$$\underset{k}{a_0} t_n + \underset{k-1}{a_1} t_{n-1} + \ldots + \underset{0}{a_k} t_{n-k} = g(n), \quad n \geq k.$$

Bez dodatnih informacija o funkciji g, ništa se ne može reći o rješenju ovakve rekurzije.

Zbog toga promatramo neke specijalne slučajeve za g, bitne u analizi kompleksnosti algoritama.

Na početku, uzmimo da je g oblika

$$g(n) = b^n \cdot p_d(n)$$

gdje je b konstanta ($b > 0$) i p_d je polinom u n, stupnja $d \in \mathbb{N}_0$.

Ilustrirajmo primjenima opću metodu rješavanja ovakvih rekurzivnih jednadžbi.

Primjer 14. (Alg., p. 68-69) Rekurzivna relacija ima oblik:

$$t_n - 2t_{n-1} = 3^n, \quad n \geq 1.$$

U ovom slučaju je $b = 3$ i $p_0(n) = 1$ - polinom stupnja 0.

Osnovna ideja je: manipulacijom prevesti ovu rekurziju u ekvivalentnu homogenu
(ovaj trik se često koristi u diferencijalnim jednadžbama).

Napišimo ovu rekurziju, uz zamjenu n sa $n+1$:

$$t_{n+1} - 2t_n = 3^{n+1}.$$

Ako polaznu rekurziju pomnožimo s 3, dobivamo isti nehomogeni član:

$$3t_n - 6t_{n-1} = 3^{n+1}.$$

Oduzmemo te dvije jednadžbe:

$$t_{n+1} - 5t_n + 6t_{n-1} = 0.$$

Dobili smo homogenu rekurziju, ali drugog, a ne više prvog reda. Tj. red je skočio za 1.

Karakteristična jednadžba ove rekurzije je

$$x^2 - 5x + 6 = 0$$

ili u faktoriziranom obliku:

$$(x-2)(x-3) = 0 .$$

Polazna rekurzija ima karakterističnu jednadžbu

$$x - 2 = 0 .$$

Dakle, prvi faktor u jednadžbi za novu rekurziju potječe od stare rekurzije. Drugi faktor $x-3 = x-b$ potječe od naših manipulacija da se riješimo desne strane.

Ova rekurzija se lako rješava (sami!). Jedino nam fali početnih uvjeta. Polazna rekurzija zahtjeva samo jedan, a nova treba dva početna uvjeta.

Dodatni potrebni početni uvjet (za t_1) se naprosto izračuna iz polazne rekurzije! ■

U slijedećem primjeru, polinom p nije trivijalan.

Primjer 15. (Rg., p. 69) Rekurzivna relacija ima oblik

$$t_n - 2t_{n-1} = 3^n \cdot (n+5), \quad n \geq 1.$$

Rekurzija je ista kao u prošlom primjeru, s $b=3$, ali uz

$$p_1(n) = n + 5.$$

Ideja je ista - poreći red rekurzije da se eliminira nehomogeni član. No, sada, osim faktora 3^n, treba podesiti i linearni polinom tako da se ove skrati. Tj. treba podesiti 2 koeficijenta (uz n^1 i n^0), pa očekujemo da treba poreći red za 2, a ne za 1, kao u prošlom primjeru.

(Pokušajte sami - poreći jedna za 1 → neće ići).

Napišimo rekurzije koje dobivamo zamjenama n sa $n+2$ i $n+1$ i još polaznu rekurziju.

$$\begin{aligned}
n \to n+2 \qquad & t_{n+2} - 2t_{n+1} = 3^{n+2}(n+7) = 3^n \cdot 9(n+7) \\
n \to n+1 \qquad & t_{n+1} - 2t_n = 3^{n+1}(n+6) = 3^n \cdot 3(n+6) \\
n \qquad & t_n - 2t_{n-1} = 3^n (n+5) = 3^n \cdot \underbrace{(n+5)}_{\text{bitni nehomogeni dijelovi!}}
\end{aligned}$$

Ideja: drugu jednadžbu pomnožiti sa a, treću sa b i to zbrojiti, tako da dobijemo nehomogeni član prve jednadžbe:

$$g(n+7) = a\cdot[3(n+6)] + b\cdot[n+5].$$

(pišemo bez faktora 3^n).

[Napomena: uvjet dovoljno povećati red za 1, pa izabrati a i b tako da linearna kombo. nehomogenih dijelova bude nula, jer dobivamo homogeni linearni sustav za a i b - općenito regularan, $\Rightarrow a=b=0$!].

Dobivamo slijedeći linearni sustav za a i b, izjednačavanjem članova uz iste potencije od n (jer rekurzija mora vrijediti za $\forall n\in\mathbb{N}$)

$$\begin{aligned} 3a + b &= 9 \quad \mid -6 \mid + \\ 18a + 5b &= 63 \end{aligned}$$

$$\Rightarrow \quad 5b - 6b = 63 - 54$$

$$\underline{b = -9}$$

i iz prve jedn: $3a = 18$

$$\underline{a = 6},$$

Dakle, pomnožimo drugu jednadžbu s $\underline{-a=-6}$, treću jedn. s $\underline{-b=9}$ i dodamo proj - dobivamo homogenu rekurziju:

$$t_{n+2} - 2t_{n+1} - 6(t_{n+1} - 2t_n) + 9(t_n - 2t_{n-1}) = 0, \quad n \geq 1.$$

Ako ovo sredimo, izlazi

$$t_{n+2} - 8t_{n+1} + 21t_n - 18t_{n-1} = 0.$$

Međutim, karakterističnu jednadžbu je lakše napisati iz početnog oblika, jer ona čuva oblik polazne rekurzije, a time i potpadni faktor u karakterističnoj jednadžbi. Karakt. jedn. ove rekurzije je:

$$x^3 - 2x^2 - 6(x^2 - 2x) + 9(x-2) = 0$$

ili

$$(x-2)(x^2 - 6x + 9) = 0$$

$$(x-2)(x-3)^2 = 0.$$

Faktor $x-2$ odatle potiče od polazne rekurzije, a faktor $(x-3)^2 = (x-6)^{d+1}$ od homogenizacije poređavanjem reda rekurzivne jednadžbe ■

Generalizacija je ovdje:

Linearna, nehomogena rekurzija (s konstantnim koef), reda k, oblika

$$a_0^{(k)} t_n + a_1^{(k-1)} t_{n-1} + \dots + a_k^{(0)} t_{n-k} = b^n p_d(n), \quad n \geq k \tag{4}$$

gdje je $b \in \mathbb{C}$, $b \neq 0$ konstanta i p_d (općenito kompleksni) polinom stupnja točno d u varijabli n, može se prevesti u linearnu homogenu rekurziju reda $k+d+1$ s karakterističnom jednadžbom

$$(a_0^{(k)} x^k + a_1^{(k-1)} x^{k-1} + \dots + a_k^{(0)}) \cdot (x-b)^{d+1} = 0 \quad \blacksquare$$

Dakle, dovoljno je nizati polazni rekurziju i dodati član koji odgovara faktoru $(x-b)^{d+1}$, tj. linearnu kombinaciju članova

$$b^n, nb^n, \dots, n^d b^n.$$

— Ovo vrijedi samo ako b nije korijen homog. dijela.

Dodatni početni uvjeti ($d+1$ njih) za koeficijente uz ove dodatne članove mogu se izračunati iz rekurzije.
U stranu, kao što ćemo pokazati, ti koeficijenti se mogu odrediti direktno iz općeg oblika rekurzije (4), bez poznavanja njenih početnih uvjeta (tj. odredit ćemo opći oblik, a ne ovise o početnim uvjetima).

Prije primjera, kako se dokazuje ovaj rezultat?
Točno istim putem kao u prostom primjeru. Napiše se relacija (4) za $n, n+1, \dots, n+d+1$ i traže koeficijenti s kojima treba pomnožiti prvu i sve zbrojiti, tako da eliminiramo nehomogeni član.

Pripadni nehomogeni članovi su redom:

$$\begin{aligned}
n: &\quad b^n \cdot p_d(n) &&= b^n \cdot p_d(n) \\
n+1: &\quad b^{n+1} \cdot p_d(n+1) &&= b^n \cdot b \cdot p_d(n+1) \\
&\quad \vdots && \quad \vdots \\
n+d: &\quad b^{n+d} \cdot p_d(n+d) &&= b^n \cdot b^d \cdot p_d(n+d) \\
n+d+1: &\quad b^{n+d+1} \cdot p_d(n+d+1) &&= b^n \cdot \underbrace{b^{d+1} \cdot p_d(n+d+1)}_{\text{bitni nehomogeni dijelovi}}
\end{aligned}$$

Zatim se traže koeficijenti $\beta_1, -, \beta_{d+1}$ takvi da je

$$(*) \qquad b^{d+1} \cdot p_d(u+d+1) + \beta_1 \left[b^d \, p_d(u+d) \right] + \ldots + \beta_{d+1} \left[p_d(u) \right] = 0$$

i zatim se dokaže da je

$$(x-b)^{d+1} = x^{d+1} + \beta_1 x^d + \ldots + \beta_d x + \beta_{d+1}.$$

(Može i obratno - znajući se β_j iz zadnje relacije i dokaže se da vrijedi $(*)$ - što je poznata kombinatorna relacija za binomne koeficijente, ako se $p_d(n+i)$ razvije po Taylorovu oko n i dobro sredi.)

<u>Primjer 16</u>. (Hanojski tornjevi)
(Alg., Ex. 2.3.4, p. 69-70)

Problem Hanojskih tornjeva je klasični primjer rekurzivnog algoritma. (v. Pascal).

Na raspolaganju imamo 3 štapa (igle) [od dijamanata] i n (po legendi $n = 64$) prstenova s rupom u sredini [prstenovi su zlatni]. Svi n je različitih veličina i u početnoj poziciji su poslagani na prvoj igli, tako da je najveći na dnu, a najmanji na vrhu.

Zadatak je premjestiti sve prstenove na neki drugu iglu. Dozvoljena operacija pritom je:

- premještanje jednog jedinog prstena s jedne igle na drugu, ali tako da nikad ne stavimo veći prsten iznad manjeg.

Za rješenje općeg problema s n prstenova, dovoljno je uočiti sljedeće:

Operaciju "premjesti n najmanjih (najgornjih) prstenova s igle i na iglu j", realizirat ćemo tako da prvo prebacimo najmanjih $n-1$ prstenova na pomoćnu iglu (preostalu), zatim prebacimo n-ti prsten s i-te na j-tu, i, na kraju, vratimo najmanjih $n-1$ prstenova s pomoćne na j-tu iglu.

Igle označavamo brojevima od 1 do 3, tj.

$$i, j \in \{1, 2, 3\}, \quad i \neq j$$

a pomoćna igla je jedina preostala. Njen redni broj je

$$1 + 2 + 3 - i - j = 6 - i - j.$$

Pripadni algoritam za "prebaci n s i na j" je:

```
procedure Hanoi ( n, i, j : integer );
```

↳ valli samo da dipone param. nečemo uvijek precizno deklarirati.

```
{ prebaci n najmanjih prstenova s igle i na iglu j }
begin
  if n > 0 then                    ← sporiji oblik za if n > 1 then ovo else samo uvijetu.
  begin
    Hanoi ( n-1, i, 6-i-j );
    writeln ( i, "->", j );        { prebaci s i na j }
    Hanoi (n-1, 6-i-j, j)
  end
end;
```

Umjesto stvarnog prebacivanja prstena, mi pišemo redoslijed prebacivanja.

Za rješenje polaznog problema iz legende, dovoljno je pozvati ovaj potprogram s parametrima

(64, 1, 2).

prostor → sam (zadatak)

Potrebna količina posla, odn. trajanje algoritma, najprirodnije je opisati brojem poteza - prebacivanja. Tj. brojimo koliko puta se izvršava writeln naredba u ovoj proceduri.

(To je isto što i broj izvršavanja bloka od 3 naredbe u proceduri Hanoi).

Odgovor je, očito, funkcija samo od n - broja prstenova. (Ne ovisi o redoslijedu igala - numeraciji...).

Označka: t_n = broj poteza za n prstenova.

Rekurzivna relacija za t_n je očita iz algoritma:

$$t_n = \begin{cases} 0 & \text{za } n = 0 \\ 2t_{n-1} + 1, & \text{za } n > 0 \end{cases}$$

(prekratko ove naredbe, pa i nije)

jer 2 puta zovemo Hanoi, ali s n-1 prstenova i još jedan ispis u ovom bloku.

Ovo je nehomogena rekurzija oblika (4), s

$$b = 1, \quad p_0(n) = 1.$$

Karakteristična jednadžba je:

$$(x-2)(x-1) = 0.$$

Faktor $x-2$ dolazi iz homogenog dijela rekurzije na lijevoj strani, a faktor $x-1$ od nehomogenog dijela s desne strane.

Korijeni su očito $r_1 = 1$, $r_2 = 2$, a opće rješenje rekurzije je

$$t_n = c_1 1^n + c_2 2^n.$$

Početni uvjet je $t_0 = 0$. Fali nam još jedan početni uvjet, pa računamo t_1 iz rekurzije:

$$t_1 = 2t_0 + 1 = 1.$$

Za konstante c_1, c_2 dobivamo:

$$\begin{array}{lll} n=0: & c_1 + c_2 = 0 & \cdot(-1) \\ n=1: & c_1 + 2c_2 = 1 & + \end{array}$$

$$\Rightarrow \quad c_2 = 1$$

$$\Rightarrow \quad c_1 = -1 \quad \text{(iz prve)}.$$

Rješenje rekurzije, tj. broj premještanja prstenova u problemu s n prstenova je:

$$\underline{t_n = 2^n - 1}, \quad n \geq 0.$$

- Ako želimo dobiti samo red veličine za t_n, ne treba računati konstante c_1, c_2 u ovom primjeru.

Naime, za broj premještanja prstenova vrijedi

$$t_n \geq n$$

jer barem n komada treba prebaciti s i na j.

Čim znamo da je $t_n = c_1 1^n + c_2 2^n$

odavde slijedi da je $c_2 > 0$, tj.

$$\underline{t_n = \Theta(2^n)}.$$

Note: partik. rješ. nehom. rek. je $t_n = -1$ (bez PU)
pa ide: opće rješ. hom. + part. rj. nehom.

Hanoi model – precizniji

TRY "WRITELN" model do $n=15$.

Isto da $t_0 \neq 0$ nego const.

i trajanje writelna-a (odn. jednog poteza!)

c_0 = call izvana, ulaz (stack param.) sve do uključivo if (uklj. stack clean.)

c_m = move – poteg – writelna

Model [$\emptyset$]:
$$t_n = \begin{cases} c_0 & , \text{za } n=0 \\ c_0 + 2t_{n-1} + c_m & , n>0 \end{cases}$$

Model [1]:
$$t_n = \begin{cases} c_0 + c_m & , n=1 \\ c_0 + 2t_{n-1} + c_m & , n>1 \end{cases}$$

sto je ekvivalentno prvom modelu [$\emptyset$], ako "vatimo" drugu rekurziju za 1 dan unazad, uz $t_0=0$ (što odgovara pravom stanju stvari, jer poziva s $n=0$ zaista NEMA!)

Model [1]:
$$t_n = \begin{cases} 0 & , \text{za } n=0 \\ c_0 + 2t_{n-1} + c_m & , n>0 \end{cases}$$

Dakle, u oba modela imamo istu rekurziju
$$t_n = 2t_{n-1} + c_0 + c_m, \text{ za } n>0$$
a potrebni uvjeti su različiti:
[$\emptyset$]: $t_0 = c_0$
[1]: $t_0 = \emptyset$.

– Rješenje rekurzije: (oblik (4), uz $b=1$, $p_0(n) = c_0 + c_m$)
Kj homogenizirane rekurzije je $(x-2)(x-1)=0$.
Korijeni su $r_1=1$, $r_2=2$, a opće rješenje rekurzije je:
$$t_n = c_1 \cdot 1^n + c_2 \cdot 2^n$$
Uvrstimo ovo u polazni rekurziju da dobijemo konstantu c_1 uz dio koji odgovara nehomogenom članu:
$$c_1 + \cancel{c_2 \cdot 2^n} = 2(c_1 + \cancel{c_2 \cdot 2^{n-1}}) + c_0 + c_m$$
$$c_1 = 2c_1 + c_0 + c_m$$
$$\Rightarrow \underline{c_1 = -(c_0 + c_m)}$$
Dakle, opće rješenje polazne nehomogene rekurzije je:
$$t_n = c_2 \cdot 2^n - (c_0 + c_m)$$
a c_2 se računa iz početnog uvjeta t_0.

Dalje, $t_n = c_2 \cdot 2^n - (c_0 + c_m)$, c_2 iz t_0.

Model [$\emptyset$]: $t_0 = c_0$ $\quad c_2 \cdot 2^0 - (c_0 + c_m) = c_0$

$\Rightarrow$ $\boxed{c_2 = 2c_0 + c_m}$

Model [1]: $t_0 = 0$ $\quad c_2 \cdot 2^0 - (c_0 + c_m) = 0$

$\boxed{c_2 = c_0 + c_m}$

Dalje: [$\emptyset$]: $t_n = (2c_0 + c_m) \cdot 2^n - (c_0 + c_m)$

$= c_0 \cdot 2^n + (c_0 + c_m) \cdot (2^n - 1)$

[1]: $t_n = (c_0 + c_m) \cdot 2^n - (c_0 + c_m)$

$= (c_0 + c_m) \cdot (2^n - 1)$ $\Leftarrow$ "skalirani broj potera", jer piše "$c_0 + c_m$" u rekurziji, umjesto "1", uz isti PU.

To znači (uz pretp. iste c_0, c_m u oba modela), da je trajanje u Modelu [$\emptyset$] VEĆE za

$c_0 \cdot 2^n$

tj. imamo 2^n bespotrebnih rekurzivnih poziva s $n=0$.

- zaključak: $T_\emptyset(n) = (2c_0 + c_m) 2^n - (c_0 + c_m)$

$= c_0 \, 2^n + (c_0 + c_m)(2^n - 1)$

$T_1(n) = (c_0 + c_m) - (c_0 + c_m)$

$= (c_0 + c_m)(2^n - 1)$

Asimptotski, a onda i za vrlo velike n (zanemarimo 1 u $2^n - 1$, tj. $2^n - 1 \approx 2^n$)

$T_\emptyset(n) \sim (2c_0 + c_m) \cdot 2^n$

$T_1(n) \sim (c_0 + c_m) \cdot 2^n$

Ove druge konstante uz 2^n nije teško eksperimentalno odrediti (Napomena: MLS za PUNI oblik se NE ISPLATI jer imamo lošiju točnost mjerenja $T(n)$ za male n - tj. tamo gdje zbog uz "-1" još ima neki utjecaj)

Dakle - 1-param model je bitno točniji !!)

U stari, iz općeg rješenja rekurzije možemo dobiti malo više - ime konstante koje dolaze uz tzv. nehomogeni dio rješenja - onaj koji potiče od faktora koje donosi nehomogeni dio.

Uvrstimo opće rješenje u rekurziju:

$$t_n = 2t_{n-1} + 1$$

$$c_1 1^n + c_2 2^n = 2(c_1 \cdot 1^{n-1} + c_2 \cdot 2^{n-1}) + 1$$

$$c_1 + \cancel{2^n c_2} = 2c_1 + \cancel{2^n c_2} + 1$$

$$\underline{c_1 = -1}.$$

Dakle, iz početnog uvjeta $t_0 = 0$, treba samo odrediti c_2. To je za očekivati, jer smo dodatne početne uvjete računali iz rekurzije, tj. moramo ih i dobiti direktno iz rekurzije, uvrštavajući opće rješenje ■

<u>Primjer 17.</u> (Alg., Ex. 2.3.2, p. 71) Zadana je rekurzija oblika:

$$t_n = 2t_{n-1} + n, \quad n \geq 1.$$

Ovo je sličnog oblika kao u prošlom primjeru, osim što je nehomogeni član sada n, umjesto 1.

Rekurzija je oblika (4):

$$t_n - 2t_{n-1} = n, \quad n \geq 1$$

uz $b = 1$ i $p_1(n) = n$.

- Karakteristična jednadžba je:

$$\underbrace{(x-2)}_{\text{homog.}}\underbrace{(x-1)^2}_{\text{nehomog. dio}} = 0$$

  - s korijenima: $r_1 = 1$, multipliciteta $m_1 = 2$
    $r_2 = 2$, multipliciteta $m_2 = 1$.

- Opće rješenje rekurzije ima oblik:

$$t_n = c_1 \cdot 1^n + c_2 \cdot n \cdot 1^n + c_3 \cdot 2^n$$

$$= c_1 + c_2 n + c_3 \cdot 2^n.$$

Odredimo asimptotsko ponašanje ovog rješenja sto točnije, bez korištenja početnih uvjeta.

- Ako je t_n mjera (funkcija) kompleksnosti nekog problema, onda je sigurno

$$t_n \geq 0 \,, \quad n \in \mathbb{N}_0$$

pa je onda

$$\underline{t_n = \mathcal{O}(2^n)}$$

i to bez apsolutnih nejednakosti (u Def. 1).

- Slijedeći korak: uvrstimo opće rješenje u rekurziju da odredimo konstante uz "nehomogeni" dio:

$$t_n - 2t_{n-1} = n \,, \quad n \geq 1$$

$$c_1 + c_2 \cdot n + c_3 \cdot 2^n - 2(c_1 + c_2(n-1) + c_3 \cdot 2^{n-1}) = n$$

$$\underline{c_1} + c_2 n + \cancel{c_3 \cdot 2^n} - \underline{2c_1} - 2c_2 n + \underline{2c_2} - \cancel{c_3 \cdot 2^n} = n$$

$$(2c_2 - c_1) - c_2 n = n \,, \quad n \geq 1$$

Izjednačavanjem koeficijenata slijedi:

$$-c_2 = 1 \quad \Rightarrow \quad c_2 = -1$$
$$-c_1 + 2c_2 = 0 \quad \Rightarrow \quad c_1 = 2c_2 = -2$$

pa je opće rješenje zapravo:

$$\underline{t_n = c_3 2^n - n - 2} \,, \quad n \geq 0 \,.$$

- Ako je t_n kompleksnost, tj. $t_n \geq 0$, za $\forall n \in \mathbb{N}_0$, onda je nužno i

$$\underline{c_3 > 0}$$

iz čega direktno slijedi

$$\underline{t_n = \Theta(2^n)} \,.$$

- Dakle samo c_3 treba odrediti iz početnog uvjeta, i ako je $t_0 \geq 0$, onda je, iz

$$t_0 = c_3 - 2 \quad \text{tj.} \quad c_3 = t_0 + 2$$

da je $\underline{c_3 \geq 2}$. ■

Generalizacijom iste vrste razmatranja, možemo rijesiti nehomogene rekurzije oblika:

$$(5)\qquad \underset{k}{a_k} t_n + \underset{k-1}{a_{k-1}} t_{n-1} + \dots + \underset{0}{a_0} t_{n-k} = b_1^n \cdot P_{d_1}(n) + b_2^n \cdot P_{d_2}(n) + \dots + b_e^n \cdot P_{d_e}(n)$$

Nehomogeni dio je suma članova čiji imaju oblik nehomogenog dijela iz (4).

Ovdje su b_i međusobno različite konstante (riječ zbrojimo pripadne p-ove), a $P_{d_i}(n)$ su polinomi u n, stupnja točno d_i.

Ovakva rekurzija se može homogenizirati i dobivena homogena rekurzija ima karakterističnu jednadžbu:

$$\underbrace{(a_k x^k + a_{k-1} x^{k-1} + \dots + a_0)}_{\text{potiče od homogenog dijela u (5)}} \cdot \underbrace{(x-b_1)^{d_1+1} \cdot \dots \cdot (x-b_e)^{d_e+1}}_{\text{potiče iz homogenizacije nehomogenog dijela u (5)}} = 0$$

U općem rješenju te rekurzije reda m

$$m = k + (d_1+1) + \dots + (d_e+1),$$

samo k konstanti treba odrediti iz početnih uvjeta, a sve ostale se mogu dobiti direktnim uvrštavanjem u (5).

<u>Primjer 18.</u> (Alg., Ex. 2.3.6., p. 71-72) Rijesi rekurziju

$$t_n = 2t_{n-1} + n + 2^n, \quad n \geq 1$$

uz početni uvjet $t_0 = 0$.

<u>Rješenje</u>: U standardnom obliku, rekurzija glasi:

$$t_n - 2t_{n-1} = n + 2^n.$$

Ona je oblika (5), uz

$$b_1 = 1, \quad P_1(n) = n$$

$$b_2 = 2, \quad P_0(n) = 1.$$

Karakteristična jednadžba za ovu (ili pripadnu homogenu) rekurziju je:

$$\underbrace{(x-2)}_{\text{homogeni dio}} \cdot \underbrace{(x-1)^2}_{\substack{\text{prvi} \\ \text{nehom.} \\ \text{član}}} \cdot \underbrace{(x-2)}_{\substack{\text{drugi} \\ \text{nehom.} \\ \text{član}}} = 0$$

imaju zajednički faktor.

Korijeni su $r_1 = 1$, $r_2 = 2$, oba multipliciteta 2.

Opće rješenje ima oblik:

$$t_n = c_1 \cdot 1^n + c_2 \cdot n \cdot 1^n + c_3 \cdot 2^n + c_4 \cdot n \cdot 2^n .$$

- Trebamo dodatna 3 početna uvjeta, pa iz polazne rekurzije računamo:

$$t_0 = 0, \quad t_1 = 3, \quad t_2 = 12, \quad t_3 = 35 .$$

Iz tih početnih uvjeta izlazi linearni sustav:

$$\begin{aligned}
n = 0: &\quad c_1 + c_3 = 0 \\
n = 1: &\quad c_1 + c_2 + 2c_3 + 2c_4 = 3 \\
n = 2: &\quad c_1 + 2c_2 + 4c_3 + 8c_4 = 12 \\
n = 3: &\quad c_1 + 3c_2 + 8c_3 + 24c_4 = 35
\end{aligned}$$

s rješenjem:

$$c_1 = -2, \quad c_2 = -1, \quad c_3 = 2, \quad c_4 = 1 .$$

Rješenje rekurzije je:

$$t_n = -2 - n + 2^{n+1} + n \cdot 2^n .$$

Odavde je asimptotski:

$$t_n = \Theta(n 2^n) \quad \text{ili} \quad t_n \sim n 2^n .$$

- Odmah, iz općeg rješenja, slijedi da je

$$t_n = \mathcal{O}(n 2^n).$$

Lako se dokazuje da je $t_n \geq 0$ (iz $t_0 \geq 0$) i da je tada $t_n \geq 2^n$ (iz nehomogenog člana). Na isti se odmah da je $c_4 \neq 0$, tj $t_n \geq n \cdot 2^n$ ili $t_n = \Omega(n 2^n)$.

- U ovom primjeru, homogeni dio i jedan nehomogeni član daju faktore koji nisu relativno prosti (zajednički faktor je $x-2$).

Interesantno je uvrstiti opće rješenje u rekurziju i izračunati konstante koje se daju izračunati.

Uvrštavanje dalje:

$$(c_1 + c_2 \cdot n + c_3 \cdot 2^n + c_4 n 2^n) - 2\left[c_1 + c_2(n-1) + c_3 2^{n-1} + c_4 (n-1) 2^{n-1}\right] =$$

$$= n + 2^n$$

$$c_1 + c_2 n + \cancel{c_3 \cdot 2^n} + \cancel{c_4 \cdot n 2^n} + (-2c_1 + 2c_2) - 2c_2 n + (\cancel{-2c_3} + 2c_4) 2^{n-1} \cancel{- 2c_4 n \cdot 2^{n-1}} =$$

$$= n + 2^n \qquad (c_4 - c_3) 2^n \qquad - c_4 n 2^n$$

Sredimo:

$$(2c_2 - c_1) - c_2 n + c_4 2^n = n + 2^n \quad , \quad n \geq 1$$

Zbog linearne nezavisnosti funkcija u općem rješenju, smijemo izjednačiti koeficijente uz iste funkcije:

uz 1: $2c_2 - c_1 = 0$ $\Rightarrow \underline{c_1 = -2}$

uz n: $-c_2 = 1 \Rightarrow \underline{c_2 = -1}$

uz 2^n: $\underline{c_4 = 1}$.

- Dakle, bez početnog uvjeta $t_0 = 0$, opće rješenje polazne rekurzije je:

$$\underline{t_n = -2 - n + c_3 2^n + n \cdot 2^n}.$$

Iz početnog uvjeta $t_0 = 0$, odmah izlazi $c_3 = 2$, tj. općenito:

$$\underline{c_3 = t_0 + 2}$$ ■

- U nekim slučajevima, kompliciranije rekurzije možemo rješiti zamjenom varijable - tj. odgovarajućom supstitucijom.

Potreba za tim se javlja u tzv. "podijeli pa vladaj" algoritmima, gdje se rješenje velikog problema svodi na nekoliko rješenja bitno manjih problema.
Vrlo često se veličina problema raspolavlja (bisekcija, binarno pretraživanje).

Zamjena varijable

U algoritmima primjenama, $T(n)$ označava članove polazne rekurzije, a t_k članove nove rekurzije dobivene zamjenom varijable.

Primjer 19. (Alg., Ex. 2.3.7., p. 72-73) Zadana je rekurzija oblika

$$T(n) = 4T(n/2) + n, \quad n > 1$$

gdje je n potencija od 2.
Treba naći red veličine za $T(n)$.

Rješenje: Supstitucija je samim prirodna, jer je n potencija od 2:

$$n = 2^k$$

ili $$k = \lg n.$$

Dobivamo rekurziju:

$$T(2^k) = 4T(2^{k-1}) + 2^k, \quad k > 0.$$

Uz oznaku:

$$t_k = T(2^k) = T(n)$$

dobivamo novu rekurziju:

$$t_k = 4t_{k-1} + 2^k, \quad k > 0.$$

Ova rekurzija znamo rješiti. Karakteristična jednadžba je

$$(x-4)(x-2) = 0$$

Opće rješenje rekurzije je

$$t_k = c_1 \cdot 4^k + c_2 \cdot 2^k, \quad k > 0.$$

Možemo naći i konstantu c_2 (uz "nehomogeni" dio rješenja). Uvrstimo u rekurziju:

$$c_1 4^k + c_2 2^k = 4(c_1 4^{k-1} + c_2 2^{k-1}) + 2^k$$
$$= c_1 4^k + c_2 \cdot 2^{k+1} + 2^k$$
$$2c_2 \cdot 2^k$$

$$\Rightarrow \quad c_2 \cdot 2^k = -2^k$$

ili $$\underline{c_2 = -1}.$$

Dakle:

$$t_k = c_1 \cdot 4^k - 2^k \quad , \quad k>0 .$$

Vratimo li n, umjesto k, izlazi:

$$\underline{T(n) = c_1 \cdot n^2 - n \quad , \quad n>1}$$

uz uvjet da je n potencija od 2.

- Za asimptotsko ponašanje vrijedi

$$\underline{T(n) = \Theta(n^2), \ n \text{ potencija od } 2}$$

To smo mogli zaključiti i bez nalaženja c_2 ■
(Osim što je i $T(n) = \Theta(n^2)$ tj. $T(n) \sim c_1 n^2$, jer $c_1 > 0$ za $T(n) \geq 0$)

Vidimo da $T(n)$ kvadratno ovisi o n. To se moglo i očekivati. Pri prijelazu s veličine $n/2$ problema na veličinu n, potrebno vrijeme je uvećano 4 puta, uz dodatak n (nehomogeni član). Taj nehomogeni član je dovoljno mali da <u>ne</u> narušava opće ponašanje. Ako ga povećamo do n^2 - dobit ćemo do promjene!

<u>Primjer 2.6.</u> (Alg., Ex. 2.3.8., p. 72) Nadi red veličine (asimpt. ponašanje) za $T(n)$, ako je n potencija od 2 i vrijedi

$$T(n) = 4T(n/2) + n^2 \quad , \quad n>1 .$$

Rješenje: Postupak je isti. Stavimo $n = 2^k$ i izlazi

$$T(2^k) = 4T(2^{k-1}) + 4^k$$

tj.

$$t_k = 4t_{k-1} + 4^k \quad , \quad k>0 .$$

Karakteristična jednadžba je

$$(x-4)^2 = 0$$

a opće rješenje je

$$t_k = c_1 4^k + c_2 k \cdot 4^k .$$

Vratimo li u termin od n, $k = \lg n$, izlazi:

$$\underline{T(n) = c_1 n^2 + c_2 n^2 \lg n} \quad , \quad n \text{ pot. od } 2$$

Odavde zaključujemo: $\underline{T(n) = \Theta(n^2 \log n)}$, n pot. od 2

Konstante iz rekurzije:

$$\cancel{c_1 4^k} + c_2 k \cdot 4^k = \cancel{c_1 4^k} + c_2 (k-1) 4^k + 4^k$$

$$c_2 k \cancel{4^k} = c_2 k \cancel{4^k} - c_2 4^k + 4^k$$

$$\Rightarrow \quad \underline{c_2 = 1}$$

Dakle:

$$\underline{T(n) = c_1 n^2 + n^2 \lg n}$$, n potencija od 2

i

$$\underline{T(n) \sim n^2 \lg n}$$, n pot. od 2

i to neovisno o početnom uvjetu!! ■

U prostoru primjera nehomogeni član - overhead je bitno povećao kompleksnost algoritma, koji - uz manji overhead ima kvadratnu kompleksnost.

Pogledajmo još 2 primjera "raspolavljanja", u kojima broj rješavanja prepolovljenih problema, nije više 4, nego 2 odn. 3.

<u>Primjer 21.</u> (Alg., Ex. 2.3.9., p. 73) Nađi red veličine (asimpt. ponašanje) za $T(n)$, ako je n potencija od 2 i vrijedi

$$T(n) = 2T(n/2) + n \lg n \,, \quad n > 1.$$

<u>Rješenje</u>:
Da nema nehomogenog člana, odmah bi $T(n)$ linearno ovisio o n (2× veći problem – 2× veća kompleksnost).
Nehomogeni član je reda veličine $n \lg n$ - dakle "gori" od linearnog → prevelik overhead.
Supstitucija i postupak su isti: $n = 2^k$, $k = \lg n$.

Dobivamo:

$$T(2^k) = 2T(2^{k-1}) + k \cdot 2^k$$

ili

$$t_k = 2t_{k-1} + k \cdot 2^k \,, \quad k > 0.$$

Karakteristična jednadžba je

$$(x-2)^3 = 0$$

(faktor $(x-2)$ od homogenog dijela i $(x-2)^2$ od nehomogenog dijela, jer $b = 2$, $p_1(k) = k$).

Opće rješenje ove rekurzije je:

$$t_k = c_1 \cdot 2^k + c_2 \cdot k \cdot 2^k + c_3 \cdot k^2 \cdot 2^k .$$

Vraćamo u termin od n:

$$\underline{T(n) = c_1 n + c_2 \cdot n \lg n + c_3 \cdot n \cdot \lg^2 n} \quad , \; n \text{ pot. od } 2$$

Zaključujemo da je

$$\underline{T(n) = \Theta(n \log^2 n)} \quad , \; n \text{ pot. od } 2 .$$

Konstante iz rekurzije: (bilo iz t_k, bilo iz $T(n)$):

$$c_1 \cdot 2^k + c_2 k 2^k + c_3 k^2 \cdot 2^k = 2[c_1 2^{k-1} + c_2 (k-1) 2^{k-1} + c_3 (k-1)^2 2^{k-1}] + k \cdot 2^k$$

$$c_2 k 2^k + c_3 k^2 2^k = c_2 k 2^k - \underline{c_2 2^k} + c_3 k^2 2^k - 2 c_3 k 2^k + \underline{c_3 2^k} + k \cdot 2^k$$

$$\Rightarrow \quad (c_3 - c_2) 2^k + (1 - 2c_3) k \cdot 2^k = 0$$

$$\Rightarrow \quad 2c_3 = 1 \quad \text{tj.} \quad c_3 = \frac{1}{2}$$

$$c_3 = c_2 \quad \text{tj.} \quad c_2 = \frac{1}{2}$$

Dakle:

$$\underline{T(n) = c_1 n + \frac{1}{2} n \lg n + \frac{1}{2} n \lg^2 n} \quad , \; n \text{ pot. od } 2$$

i asimptotski:

$$\underline{T(n) \sim \frac{1}{2} n \lg^2 n} \quad , \; n \text{ pot. od } 2 \; \blacksquare$$

<u>Primjer 22.</u> (Alg., Ex. 2.3.10; p. 74) Nađi red veličine (asimpt. ponašanje) za $T(n)$, ako je n potencija od 2 i mjedi

$$T(n) = 3T(n/2) + cn \quad , \; n > 1$$

gdje je $c \neq 0$, konstanta.

<u>Rješenje</u>: Potpuno istom metodom, $n = 2^k$, $k = \lg n$, dobivamo:

$$T(2^k) = 3T(2^{k-1}) + c \cdot 2^k$$

tj.:

$$t_k = 3t_{k-1} + c \cdot 2^k, \quad k > 0 .$$

U nehomogenom članu je $b = 2$, $p_0(k) = c$ - konstanta, pa je karakteristična jednadžba:

$$(x-3)(x-2) = 0 .$$

Opće rješenje je:

$$t_k = c_1 \cdot 3^k + c_2 \cdot 2^k$$

ili

$$T(n) = c_1 \cdot 3^{\lg n} + c_2 \cdot n .$$

Za logaritme vrijedi:

$$a^{\lg b} = b^{\lg a} = 2^{\lg b \cdot \lg a}$$

pa je:

$$\underline{T(n) = c_1 \cdot n^{\lg 3} + c_2 n \text{ , } n \text{ pot. od } 2} .$$

Dakle je

$$\underline{T(n) = O(n^{\lg 3})} \text{ , } n \text{ pot. od } 2$$

Konstante iz rekurzije:

$$c_1 3^k + c_2 2^k = 3(c_1 3^{k-1} + c_2 2^{k-1}) + c \cdot 2^k \text{ , } k > 0$$

$$c_2 2^k = \frac{3}{2} c_2 \cdot 2^k + c \cdot 2^k$$

$$\frac{1}{2} c_2 = -c \text{ , } \underline{c_2 = -2c}$$

Dakle:

$$\underline{T(n) = c_1 \cdot n^{\lg 3} - 2c \cdot n} \text{ , } n \text{ pot. od } 2$$

Uz zahtjev $T(n) \geq 0$ za n pot. od 2, onda vrijedi

$$\underline{c_1 > 0}$$

pa je:

$$\underline{T(n) \sim c_1 n^{\lg 3}} \text{ , } n \text{ pot od } 2$$

Ovdje je: $\lg 3 = 1.58496\ldots$ ■

Skica rekurzija, javlja se kod jedne ubrzane varijante algoritma za množenje n znamenkastih cijelih brojeva. Standardnoj algoritam zahtijeva reda veličine n^2 množenja pojedinačnih znamenki. Ubrzana varijanta - tzv. Karacubin algoritam, zahtijeva reda veličine $n^{\lg 3}$ množenja pojedinačnih znamenki.

Množenje velikih brojeva - rekurzivni algoritam.

Pretpostavimo da su U, V veliki brojevi, dani pozicionim zapisom u pogodno odabranoj bazi b, tako da aritmetičke operacije na znamenkama možemo efikasno izvesti (direktno - arhitekturom računala)

Radi jednostavnosti, pretpostavimo da oba broja imaju točno n znamenki u bazi b (u protivnom, dopunimo sprijeda nulama do n.). Neka je $n = 2k$ paran.

$$U = \sum_{i=0}^{n-1} u_i \cdot b^i = U_1 \cdot b^k + U_0 \quad , \quad U_1 \text{ i } U_0 \text{ po } n/2 = k \text{ znamenki}$$

$$U_1 = \sum_{i=0}^{k-1} u_{i+k} b^i \quad , \quad U_0 = \sum_{i=0}^{k-1} u_i b^i$$

i analogno za V:

$$V = \sum_{i=0}^{n-1} v_i b^i = V_1 b^k + V_0 \quad , \quad V_1 = \sum_{i=0}^{k-1} v_{i+k} b^i \quad , \quad V_0 = \sum_{i=0}^{k-1} v_i \cdot b^i .$$

Tj. U i V gledamo kao dvoznamenkaste brojeve u bazi $b^{n/2} = b^k$

- Produkt $W = U \cdot V$ u toj bazi ima oblik:

$$W = (U_1 \cdot V_1) \cdot b^{2k} + (U_1 \cdot V_0 + U_0 \cdot V_1) \cdot b^k + U_0 \cdot V_0$$

tj. trebamo 4 produkta $n/2$ znamenkastih brojeva.

- Srednji faktor možemo zapisati u obliku:

$$U_1 V_0 + U_0 V_1 = (U_1 + U_0)(V_1 + V_0) - \underbrace{U_1 V_1}_{\text{već imamo}} - \underbrace{U_0 V_0}_{\text{}}$$

tj. za njega je potrebno još samo 1 množenje, ali $k+1$ znam. brojeva. No, to ne umanjuje bitno kompleksnost.

- Dalje, W možemo naći s 3 produkta $\times\, n/2$ znam. brojeva:

$$\left.\begin{aligned} p &\leftarrow U_1 V_1 \\ q &\leftarrow U_0 V_0 \\ r &\leftarrow (U_1 + U_0)(V_1 + V_0) \end{aligned}\right\} \text{ova 3 produkta se računaju rekurzivno, istim ovim alg.}$$

$$W \leftarrow p \cdot b^{2k} + (r - p - q) b^k + q .$$

(Karacuba, 1962).

Za W, treba još obaviti množenja s b^{2k}, b^k što se svodi na pomake, i zbrajanja / oduzimanja te normalizaciju. Sve ove operacije imaju trajanje $c \cdot n$ (linearno ovisno o duljini broja W, a ta je $\approx 2n$)

U zadnja četiri primjera, rekurzija za $T(n)$ je mjerila samo uz uvjet da je n potencija od 2. Analogno, i zaključak o asimptotskom ponašanju mjeri samo uz taj uvjet.

Uvjet "n je pot. od 2" je bio potreban zato da $n/2$ uvijek bude cijeli broj.

Ti oblici rekurzija su, zapravo, pojednostavljeni oblici rekurzija koje se obično javljaju u analizi algoritama "podijeli pa vladaj".

Na pr. umjesto $n/2$, obično se javljaju

$$\lfloor n/2 \rfloor \quad \text{ili} \quad \lceil n/2 \rceil .$$

Primjer 23. (Alg., Ex. 2.1.2., p. 46) Zadana je rekurzija oblika:

$$T(n) = \begin{cases} a & , \text{ za } n = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + bn & , \text{ inače} \end{cases}$$

gdje su a, b konstante, $a \geq 0$, $b \geq 0$.

Ova rekurzija je korektno definirana za $\forall n \in \mathbb{N}$, ali se teško rješava, zbog $\lceil\ \rceil$, $\lfloor\ \rfloor$.

Promatramo li samo slučajeve u kojima je n potencija od 2, rekurzija postaje:

$$T(n) = \begin{cases} a & , \text{ za } n = 1 \\ 2T(n/2) + bn & , \text{ za } n > 1, \ n \text{ pot. od } 2. \end{cases}$$

Metodom iz Primjera 21 odmah izlazi da za asimptotsko ponašanje rješenja mjeri:

$$T(n) \sim c \cdot n \log n \quad , \ n \text{ pot. od } 2 .$$

Postavlja se pitanje:

Pod kojim uvjetima možemo ovaj rezultat generalizirati, tako da on mjeri za bilo koji n?

Tj. kada se uvjetna asimptotska relacija, sličnog tipa uvjeta, može prevesti u bezuvjetnu?

Da to bude moguće, treba postaviti odredene uvjete na funkciju T - odn. kompleksnost algoritma.

<u>Definicija 6</u>. Neka je $f: D \to \mathbb{R}_0^+$ nenegativna funkcija na odozgo neograničenom podskupu $D \subseteq \mathbb{R}_0^+$.
Funkcija f je <u>asimptotski rastuća</u> (nepadajuća, ili nužno i strogo rastuća) ako je f rastuća za dovoljno velike argumente y.

$\exists M \in D$ takav da za svaki $x, y \in D$ vrijedi:

$$x, y \geq M \text{ i } x < y \Rightarrow f(x) \leq f(y) \blacksquare$$

- Za $D = \mathbb{N}$ (ili $\mathbb{N}_0$), to znači da $\exists n_0 \in \mathbb{N}$ takav da za svaki $n \in D$ vrijedi:
$$n \geq n_0 \Rightarrow f(n) \leq f(n+1).$$
(Dovoljno je uzeti $y = x+1$, onda ide indukcijom).
- U Def. 6 - mogli smo staviti i da je f f-a kompleksnosti, ali nije potrebno.
- Treba nam još jedan pojam:

<u>Definicija 7</u>.
Neka je f asimptotski rastuća funkcija, $f: D \to \mathbb{R}_0^+$, i neka je $b > 1$. Funkcija f je <u>b-glatka</u> ako vrijedi
$$f(bx) = O(f(x))$$
(Naravno, smatramo da je $bx \in D$ za $x \in D$, barem za dovoljno velike x).

Može se pokazati da ako je f b-glatka za neki $b > 1$, onda je ona i c-glatka za bilo koji $c \geq b$, uz uvjet da je domena D kvalitetna.

Mi ćemo posebno koristiti $b \in \mathbb{N}$. Najčešće je $b = 2$, a tada je f i c-glatka za bilo koji $c \in \mathbb{N}$, $c \geq 2$.

Funkcija je <u>glatka</u>, ako je
$$f(bx) = O(f(x))$$
za svaki $b \in \mathbb{N}$, $b \geq 2$ $\blacksquare$

Može se pokazati da mjeri:

Propozicija 2.

Neka je $f: \mathbb{N}_0 \to \mathbb{R}_0^+$ glatka funkcija. Neka je $b \in \mathbb{N}$, $b \geq 2$ i $T: \mathbb{N}_0 \to \mathbb{R}_0^+$ asimptotski rastuća funkcija, takva da mjeri uvjetna asimptotska relacija:

$$T(n) = X(f(n)), \quad n \text{ je potencija od } b,$$

gdje je X jedan od simbola O, Ω, Θ.

Tada mjeri i bezuvjetna asimptotska relacija:

$$T(n) = X(f(n)).$$

Dodatno, ako je $X = \Theta$, tj. $T(n) = \Theta(f(n))$, onda je i funkcija T također glatka ■

Napomena: Primjerima se može pokazati da je srati od uvjeta
(a) T je asimptotski rastuća
(b) f je b-glatka odn. $f(bn) = O(f(n))$
uvjan za zaključak u propoziciji 2.

Primjer 23 (nastavak):

Želimo pokazati da asimptotska relacija

$$T(n) \sim c \cdot n \log n$$

mjeri bezuvjetno.

Za primjenu propozicije 2., treba dokazati da je:
(a) $T(n)$ asimptotski rastuća,
(b) $f(n) = n \log n$ glatka.

(a) Ovi dio rješenja dvostruko - mat. indukcijom. Tvrdimo zato i jače - da je T rastuća, tj. $n_0 = 1$.

Baza za $n = 1$:

$$T(1) = a$$
$$T(2) = 2T(1) + 2b = 2a + 2b$$

Uvjet na a i b je:

$$a \leq 2a + 2b$$

tj. $\underline{a + 2b \geq 0}$.

To vrijedi mjeri za $a, b \geq 0$.

Korak:

Neka je n strogo veći od 1, $n > 1$. Pretpostavimo da mijedi:

$$\forall m,\ m < n \Rightarrow T(m) \leq T(m+1).$$

Posebno je: (v. 59-1)

$$T(\lfloor n/2 \rfloor) \leq T(\lfloor \tfrac{n+1}{2} \rfloor)$$

$$T(\lceil \tfrac{n}{2} \rceil) \leq T(\lceil \tfrac{n+1}{2} \rceil)$$

$T(\lfloor \frac{n+2}{2} \rfloor)$ $\quad$ Ali $\lfloor \frac{n+1}{2} \rfloor = \lfloor \frac{n}{2} \rfloor$

Tada je:

$$T(n+1) = T(\lfloor \tfrac{n+1}{2} \rfloor) + T(\lceil \tfrac{n+1}{2} \rceil) + \underbrace{b(n+1)}_{\geq bn \text{ jer } b \geq 0}$$

$$\geq T(\lfloor \tfrac{n}{2} \rfloor) + T(\lceil \tfrac{n}{2} \rceil) + bn = T(n).$$

Dakle T je rastuća.

(Napomena: dokaz da je T rastuća mora ići direktno iz definicije funkcije T - tj. iz rekurzije. Ne možemo koristiti argument da je T asimptotski proporcionalna s $n \log n$, jer smo to pokazali tek za n je potencija od 2, a beskonačnu relaciju uporavo dokazujemo (tj. naporali bi dokaz tipa $A \Rightarrow A$!)).

(b) Dakle je $f(n) = n \log n$ asimptotski rastuća, pa je dovoljno pokazati da je na pr. 2-glatka:

$$f(2n) = 2n \log(2n) = 2n \cdot \log 2 + 2n \cdot \log n \leq$$

$$\leq (\log 2 \leq \log n, \text{ čim je } n \geq 2)$$

$$\leq 2n \log n + 2n \log n = 4n \log n = 4 \cdot f(n), \quad n \geq 2$$

$$\Rightarrow f(2n) = O(f(n)).$$

Dalle, $f(n) = n \log n$ je glatka.

Iz Propozicije 2, zaključujemo da je

$$\underline{T(n) \sim c \cdot n \log n}, \quad n \to \infty \quad \blacksquare$$

— Dodaj detaljnije min-max kao primjer!

$$T(\lfloor n/2 \rfloor) \leq T(\lfloor n/2 \rfloor + 1)$$
$$T(\lceil n/2 \rceil) \leq T(\lceil n/2 \rceil + 1)$$

- $\lfloor \frac{n+1}{2} \rfloor = \lfloor \frac{n}{2} \rfloor$, n par
  $= \lfloor \frac{n}{2} \rfloor + 1$, n impar
  $\Rightarrow \lfloor \frac{n+1}{2} \rfloor \in \{\lfloor \frac{n+1}{2} \rfloor, \lfloor \frac{n}{2} \rfloor\}$
  $i \in \{\lfloor \frac{n}{2} \rfloor, \lfloor \frac{n}{2} \rfloor + 1\}$

$$\Rightarrow T(\lfloor \tfrac{n}{2} \rfloor) \leq T(\lfloor \tfrac{n+1}{2} \rfloor)$$

analogo:

$\lceil \frac{n+1}{2} \rceil = \lceil \frac{n}{2} \rceil + 1$, n par
$= \lceil \frac{n}{2} \rceil$, n impar
$\Rightarrow \lceil \frac{n+1}{2} \rceil \in \{\lceil \frac{n}{2} \rceil, \lceil \frac{n+1}{2} \rceil\}$
$i \in \{\lceil \frac{n}{2} \rceil, \lceil \frac{n}{2} \rceil + 1\}$

$$\Rightarrow T(\lceil \tfrac{n}{2} \rceil) \leq T(\lceil \tfrac{n+1}{2} \rceil)$$

Na isti način kao u Primjeru 23., može se pokazati da asimptotske relacije u ovim primjerima – od Primjera 19 do Primjera 22, vrijede bezuvjetno.

Rezultate tih primjera možemo generalizirati u obliku:

Propozicija 3. (Alg., Prob. 2.3.6, p.74)

Neka je $T : \mathbb{N} \to \mathbb{R}^+$ asimptotski rastuća funkcija za koju vrijedi rekurzivna relacija

$$T(n) = a \cdot T(n/b) + c \cdot n^k, \quad \text{za } n > n_0, \text{ i } n/n_0 \text{ je potencija od } b$$

gdje su:

$n_0 \geq 1$, $b \geq 2$, $k \geq 0$ cijeli brojevi,

$a; c > 0$ su realni brojevi.

Za red relacije od T vrijedi:

$$T(n) \in \begin{cases} \Theta(n^k) & \text{ako je } a < b^k \\ \Theta(n^k \log n) & \text{ako je } a = b^k \\ \Theta(n^{\log_b a}) & \text{ako je } a > b^k. \end{cases}$$

Ova relacija vrijedi bezuvjetno (tj. ne samo za n takav da je n/n_0 potencija od b) ■

Dokaz je divertan – supstitucijom i proučavanjem karakteristične jednadžbe.

Ovaj rezultat je vrlo koristan za analizu algoritama tipa "podijeli pa vladaj", u kojem se "veliki" problem veličine n svodi na a rješavanja "manjih" problema – veličine n/b.

U našim primjerima je bilo $b = 2$.

Nehomogeni član opisuje posao koji je potreban da se rješenja manjih problema iskombiniraju u rješenje polaznog problema.

- U ovoj propoziciji je nehomogeni član najjednostavnijeg oblika: $c \cdot n^k$.

  Naši primjeri su koristili i općenitiji oblik $c \cdot n^k \cdot \lg^m n$.

- S tim oblikom se jednostavno izlazi na kraj ako je $m \geq 0$.

U nekim slučajevima analize algoritama tipa "podijeli pa vladaj" potrebno je još općenitiji rezultat. Neki dijelovi tog rezultata ne mogu se dobiti metodom karakteristične jednadžbe, već zahtijevaju finiju analizu.

Propozicija 4. (Alg., Prob. 2.3.13, p. 76-77)

Neka su $n_0 \geq 1$ i $b \geq 2$ cjelobrojne konstante, a $a, d \in \mathbb{R}^+$ realne pozitivne konstante.

Neka je X skup svih prirodnih brojeva oblika

$$n = n_0 b^i \quad , \quad i \in \mathbb{N}_0$$

tj.
$$X = \{ n \in \mathbb{N} \mid \log_b (n/n_0) \in \mathbb{N}_0 \}.$$

Neka je $f : X \to \mathbb{R}_0^+$ bilo koja funkcija i neka je $T : X \to \mathbb{R}_0^+$ funkcija definirana rekurzivnom relacijom

$$T(n) = \begin{cases} d & \text{za } n = n_0 \\ a \cdot T(n/b) + f(n) & \text{za } n \in X, \; i \; n > n_0 \end{cases}$$

[Rekurzija je istog oblika kao u Prop 3., osim što je nehomogeni član $f(n)$ općenit].

Neka je $p = \log_b a$.

Najjednostavniji način za opis asimptotskog ponašanja funkcije T, ovisi o odnosu između f i n^p.

(i) Ako stavimo $f(n_0) = d$ (što ne utječe na definiciju funkcije T - jer se tamo koristi $f(n)$ samo za $n > n_0$), onda je međusobno $T(n)$ za $n \in X$, dana sljedećom sumom:

$$T(n) = \sum_{i=0}^{\log_b (n/n_0)} a^i f(n/b^i).$$

[Odavde je teško zaključiti asimptotsko ponašanje funkcije T. Zbog toga - uspoređujemo f i n^p].

(ii) Neka je $q>0$ realna konstanta. Za asimptotsko ponašanje funkcije T mjerodavne sljedeće asimptotske relacije - umjetno - za $n \in X$

$$T(u) \overset{\in}{=} \begin{cases} \Theta(n^p) & \text{ako } f(u) = O(n^p/(\log n)^{1+q}) \\ \Theta(f(n)\log n \log\log n) & \text{ako } f(u) = \Theta(n^p/\log n) \\ & \text{(ovo odgovara } q=0 \text{ u gornjoj varijanti – kritična granica)} \\ \Theta(f(u)\log n) & \text{ako } f(u) = \Theta(n^p \cdot (\log n)^{q-1}) \\ \Theta(f(u)) & \text{ako } f(u) = \Theta(n^{p+q}) \end{cases}$$

Treća alternativa uključuje i $f(u) = \Theta(n^p)$, uz izbor $q=1$.
Također, specijalni slučaj prve alternative glasi:

$$T(u) = \Theta(n^p) \quad \text{za} \quad f(u) = O(n^r) \text{ uz } r<p.$$

[Ovaj rezultat usporedbe nehomogenih članova $f(u)$ s funkcijama oblika

$$n^k \cdot (\log n)^m$$

primjenu je dozvoljeno i $m<0$.
Očito je kritična granica za red veličine od f

$$n^p \cdot (\log n)^{-1} . \ ].$$

(iii) Zadnja alternativa se može profinitи - jer usporedbe f samo s funkcijama oblika

$$n^k \quad \text{za} \quad k>p.$$

Ako je na pr.

$$f(u) = \Theta(n^{p+q} \log n)$$

ili

$$f(u) = \Theta(n^{p+q}/\log n)$$

također vrijedi

$$T(u) = \Theta(f(u))$$

ali uz uvjet – ako postoji funkcija $g: X \to \mathbb{R}_0^+$ takva da je $f(u) = \Theta(g(u))$ i postoji $\alpha \in \mathbb{R}$, $\alpha > a$ takav da je

$$g(bn) \geq \alpha g(u), \quad \forall n \in X.$$

(uže dovoljna samo b. glatkoća, još mora i $\alpha > a$) ■

6. Kompleksnost algoritma i kompleksnost problema

(Case study – Problem računanja Fibonaccijevog broja)

[v. Alg., §1.7.5, p.16]

Niz Fibonaccijevih brojeva F_n, $n \in \mathbb{N}_0$, definiran je slijedećom dvočlanom homogenom rekurzijom (red jednadžbe = 2):

$$F_0 = 0, \quad F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2}, \quad n \geq 2.$$

Prvih par članova niza su

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, \ldots$$

[Alg., Ex. 2.3.2., p.66-67]

U standardnom obliku, rekurzija glasi:

$$F_n - F_{n-1} - F_{n-2} = 0, \quad n \geq 2$$

s karakterističnom jednadžbom:

$$x^2 - x - 1 = 0$$

Korijeni su

$$r_{1,2} = \frac{1 \pm \sqrt{5}}{2} \qquad \begin{matrix} r_1 \to + \\ r_2 \to - \end{matrix}$$

Opće rješenje je:

$$F_n = c_1 r_1^n + c_2 r_2^n.$$

Iz početnih uvjeta izlazi:

$$n = 0: \quad c_1 + c_2 = 0 \quad \Rightarrow \quad c_2 = -c_1$$

$$n = 1: \quad c_1 r_1 + c_2 r_2 = 1$$

$$c_1(r_1 - r_2) = 1 \quad \text{i} \quad r_1 - r_2 = \frac{1+\sqrt{5}}{2} - \frac{1-\sqrt{5}}{2} = \sqrt{5}$$

$$\Rightarrow \quad \sqrt{5} c_1 = 1$$

To daje:

$$c_1 = \frac{1}{\sqrt{5}}, \quad c_2 = -\frac{1}{\sqrt{5}}.$$

Rješenje je:

$$F_n = \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right].$$

Broj $r_1 = (1+\sqrt{5})/2$ se zove <u>zlatni omjer</u> ili <u>omjer zlatnog reza</u> i obično se označava s ϕ.

Za r_2 vrijedi $r_1 \cdot r_2 = -1$ (koef. uz x^0 u kar. jedn.) što daje

$$r_2 = -\frac{1}{\phi}.$$

U terminima zlatnog omjera, relacija za F_n glasi:

$$F_n = \frac{1}{\sqrt{5}}\left[\phi^n - (-\phi)^{-n}\right], \quad n \in \mathbb{N}_0.$$

---

i to je tzv. De Moivreova formula.

- Fibonaccijev niz ima mnogo primjena u matematici - teoriji brojeva, teoriji igara i računarstvu.

  Na pr. Euklidov algoritam za određenje najveće zajedničke mjere dva broja x i y, gledan na brojevima x i y sličnog reda veličine, traje najdulje upravo u slučaju da su x i y dva susjedna člana Fibonaccijevog niza. (v. kasnije - SC).

- Promatramo sljedeći problem:

  <u>Problem FIB</u>: Za zadani broj $n \in \mathbb{N}_0$, treba naći (izračunati) n-ti Fibonaccijev broj F_n.

  Zanimaju nas efikasni algoritmi za rješavanje ovog problema i njihova kompleksnost. Na kraju, zanima nas da li je ovaj <u>problem</u> težak ili ne, i kolika je njegova kompleksnost.

  Treba uočiti da tražimo <u>samo</u> n-ti član niza, a ne <u>sve</u> članove niza do F_n (tj. <u>ne</u> prvih n ili $n+1$ članova).

- Upravo ta razlika će biti fundamentalna za konačni zaključak.

Prvi očiti pristup rješenju ovog problema je korištenje De Moivreove formule.

Međutim brojevi ϕ, ϕ^{-1} i $\sqrt{5}$ su iracionalni (jer je broj $\sqrt{5}$ takav - kao korijen iz prostog broja).

Dokaz te činjenice je isti kao i klasični dokaz da je $\sqrt{2}$ iracionalan.

Približne vrijednosti za $\sqrt{5}$ i ϕ (u bazi 10, na 20 decimala) [v. Knuth, Fundam. Alg's, p. 613, za 40 decimala) su:

$$\sqrt{5} = 2.23606\ 79774\ 99789\ 69641 -$$

$$\phi = 1.61803\ 39887\ 49894\ 84820 +$$

Kako je $\phi^{-1} < 1$, a $\frac{1}{\sqrt{5}}\phi^{-n} < \frac{1}{2}$ za $\forall n \in \mathbb{N}_0$, slijedi da drugi član $(-\phi)^{-n}$ u DeMoivreovoj možemo ignorirati, znajući da je rezultat cijeli broj:

$$F_n = \left[\frac{1}{\sqrt{5}} \phi^n \right], \quad \forall n \in \mathbb{N}_0 .$$

Ova formula, očito, zahtijeva realnu aritmetiku. Osim toga, kako n raste, potrebne su sve preciznije vrijednosti za $\sqrt{5}$ i ϕ. Mašinska aritmetika ima ograničenu točnost, pa nakon nekog n, dobivamo pogrešan rezultat (čak i bez pretvaranja u cijeli broj).

Kompleksnost ovakvog algoritma je najviše $O(\log n)$ ako koristimo brzo potenciranje - preko binarnog prikaza broja n (kvadriranje i množenje).

(Ako mašinski potenciramo - preko funkcija iz biblioteke, kompleksnost je konstanta).

- To, naravno, vrijedi samo ako množenje i dijeljenje smatramo elementarnim operacijama - s konstantnim trajanjem.

Brojevi F_n, međutim, vrlo brzo rastu i potrebna točnost računanja vrlo brzo prelazi mogućnosti arhitekture. (Na pr. F_{100} ima 21 dekadsku znamenku, tj. $F_{100} > 10^{20}$).

Ako i dalje želimo koristiti ovakav algoritam, trebamo vlastite rutine za množenje i dijeljenje realnih brojeva velike točnosti, a te nemaju konstantnu kompleksnost. (Usput - trebamo i vrlo točni $\sqrt{5}$).

(Problem: sami analizirajte kompleksnost takvog algoritma - za razne varijante algoritama za množenje i dijeljenje u velikoj točnosti).

Kasnije ćemo pokazati da se složena brzina (čistog veola velične) može postići i cjelobrojnom aritmetikom.

Zbog toga, nećemo dalje analizirati algoritam na bazi De Moivreove formule u realnoj aritmetici.

Primjer 24. Turbo Pascal ima realni tip extended za 80x87 matematičke koprocesore. Točnost tog tipa je 19-20 dekadskih znamenki - mantisa ima 64 bita.

Najveći Fibonaccijev broj koji se može egzaktno prikazati i ispisati u tipu extended je:

$$F_{87} = 679\ 89163\ 76386\ 12258$$

(egzaktno prikazivi su još neki, ali se ne mogu točno ispisati, zbog poteškoća u dekadskom sustavu)

Algoritam na bazi De Moivre-ove formule za F_n je:
(v. FIB-MOIV.PAS)

```
function Fib ( n:integer ) : real;    { real = extended = operacijom $N+ }

var Sqrt5, Fi : real;

begin
  Sqrt5 := sqrt(5.0);
  Fi := ( 1 + Sqrt5 ) / 2;
  Fib := exp ( n * ln(Fi)) / Sqrt5 ;   { Fi ** n / Sqrt5 }
end; {Fib}
```

Rezultat je realan, jer tipovi integer i longint imaju mnogo manji raspon. Najbliži cijeli broj se lako dobiva 12 ispisa, ili se dobiva direktno s

```
writeln ( n:5, Fib(n):25:0 );
```

ispisom s 0 decimalnih mjesta (TP tada sam zaokružuje).

- Sve mjerodavne cifre dobivene i F_{85} su točne

$$F_{85} = 259\ 69549\ 69111\ 22585.$$

Vrijednosti za F_{86} i F_{87} su pogrešne (za 1 prevelike) - zbog grešaka zaokruživanja pri realnim operacijama i funkcijama (i, eventualno, ispisu).

- Trajanje izvođenja funkcije Fib je konstantno (do na točnost mjerenja) i iznosi

$$T(n) = 0.0001 \pm 0.000033 \text{ sekundi}, \quad 0 \le n \le 87$$

(1 milisekunda ± 33 mikrosekunde)

na AT 8 MHz s 80287 koprocesorom (6 MHz stranci takt), u Turbo Pascalu 6.0 ■

Promotrimo sad algoritme na bazi gjelobrojne aritmetike. Problem brzog rasta Fibonaccijevih brojeva ostaje prisutan, i vlo brzo F_n prelaze prikazivi raspon. U prikazivoj domeni, osnovne aritmetičke operacije možemo smatrati elementarnim - tj. konstantnog trajanja. Van prikazivog raspona, aritmetičke operacije moramo realizirati sami, a njihova kompleksnost postaje ovisna o veličini brojeva. Ovu razliku ćemo analizirati na kraju, kad formuliramo sve algoritme - u terminima standardnih aritmetičkih operacija.

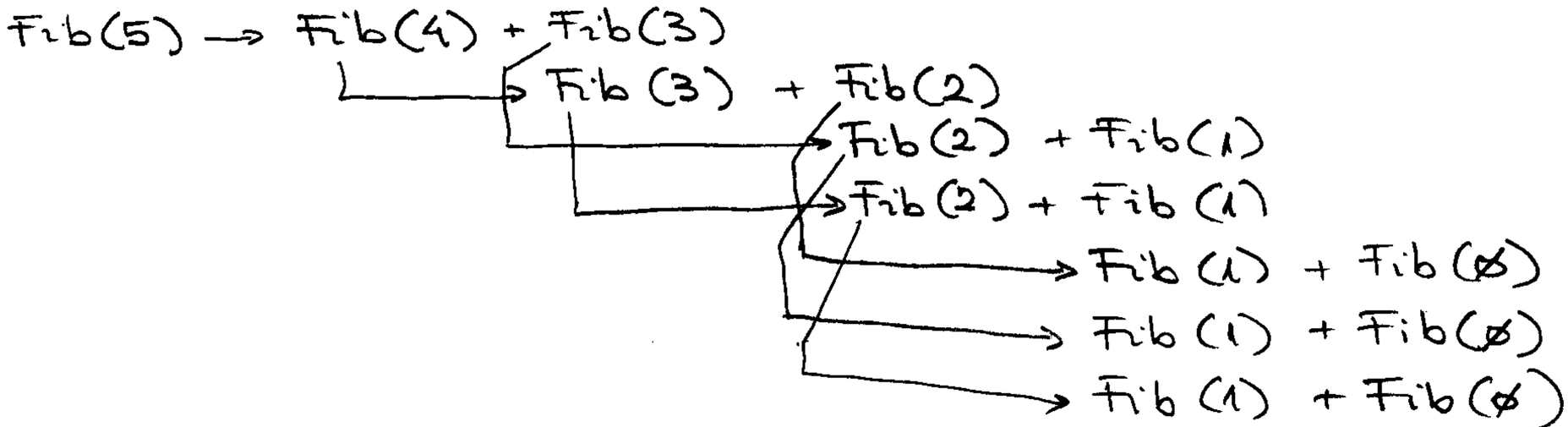
Prvi algoritam izlazi direktno iz definicije niza Fibonaccijevih brojeva - rekurzijom.

Algoritam 1. F_n - rekurzivno

```
function Fib (n : integer) : integer;
begin
  if n <= 1 then
    Fib := n
  else
    Fib := Fib(n-1) + Fib(n-2)
end; {Fib}
```

Ovaj algoritam je izrazito neefikasan, jer mnogo puta računa iste međurezultate.

Na pr. da izračunamo Fib(5), trebamo međurezultate za Fib(4) i Fib(3). No, Fib(4) također poziva izračunavanje Fib(3). Dakle, Fib(3) se računa 2 puta. Nastavimo li dalje, do Fib(1) i Fib(0), dobivamo ovakvo stablo poziva:



1 1 2 3 5 3 puta.

Ako je n fiksan, spuštanjem po rekurziji, broj poziva Fib(k) vrlo brzo raste, kako k pada. Označimo sa c_k broj poziva za Fib(k), $k = n, \rightarrow, 0$.

Očito je: $c_n = 1$ (vanjski poziv)

$c_{n-1} = 1$

$\vdots$

$c_k = c_{k+1} + c_{k+2}$, $1 \leq k \leq n-2$

jer i Fib(k+1) i Fib(k+2) zovu Fib(k) - pa se Fib(k) zove onoliko puta koliko i oba prethodna.

Na kraju, Fib(1) i Fib(ø) ne zovu nikog ispod sebe, pa je:

$$c_0 = c_2$$

jer samo Fib(2) zove Fib(ø).

- Ovo je rekurzija istog tipa kao i rekurzija za Fibonaccijeve brojeve. Očito je:

$c_n = F_1$

$c_{n-1} = F_2$

$\vdots$

$c_k = F_{n+1-k}$, $1 \leq k \leq n-2$ ili n.

$\vdots$

$c_0 = c_2 = F_{n-1}$.

Dakle, u računanju Fib(n), vrijednost za Fib(k) se računa točno F_{n+1-k} puta, za $k \geq 1$.

Na dnu, to znači da se:

Fib(2) i Fib(ø) zovu F_{n-1} puta
Fib(1) zove F_n puta.

- Ostaje još odrediti trajanje svakog od poziva – tj. trajanje operacija u jednom izvršenju tijela - naredbe funkcije (od begin do end).
- Prije nego što to detaljnije analiziramo, očito je da je to trajanje odozdo ograničeno nekom konstantom, ovisno o jedinicama u kojima se mjeri.
  Ta konstanta uključuje na pr. vrijeme za pripremu poziva i čišćenje nakon poziva – manipulaciju parametrima na stacku.
- Označimo s $T(n)$ vrijeme potrebno za izvršavanje neke implementacije ovog algoritma (na pr. u TP 6.0).
  Dakle zahtijevat ćemo:
  $$T(n) \geq c > 0 \quad , \forall n \in \mathbb{N}_0$$
  gdje je $c > 0$ pozitivna konstanta.
- Pošto već znamo broj poziva za Fib(k) u toku računanja Fib(n), izlazi relacija:

$$T(n) \geq c_{n-1} \cdot T(n-1) + c_{n-2} \cdot T(n-2) + \ldots + c_1 \cdot T(1) + c_0 \cdot T(0).$$

Odavde direktno slijedi:

$$T(n) \geq c \cdot (c_{n-1} + c_{n-2} + \ldots + c_1 + c_0) =$$

$$= c \cdot (F_2 + F_3 + \ldots + F_{n-1} + F_n + F_{n-1}) \geq$$

$$\geq \text{(jer je } n \geq 1\text{, onda je } F_{n-1} \geq F_0 = F_1\text{)}$$

$$\geq c \cdot \sum_{i=1}^{n} F_i$$

tj. još jednostavnije:

$$\underline{\underline{T(n) \geq c \cdot F_n \quad , \forall n \in \mathbb{N}_0}}$$

što pokazuje $\underline{\underline{T(n) = \Omega(F_n) = \Omega(\phi^n)}}$.

- Zaključujemo dakle da je Algoritam 1 barem eksponencijalan i to u n.

Iolemo odredotiti točan red veličine za $T(n)$.

- Za $T(n)$ - direktno iz algoritma, vijedi:

$$T(n) = T(n-1) + T(n-2) + D(n), \text{ za } n \geq 2$$

gdje je $D(n) > 0$ vrijeme potrebno za sve <u>dodatne</u> operacije pored poziva $Fib(n-1)$ i $Fib(n-2)$.
$D(n)$ uključuje test $n \leq 1$, skokove, zbrajanje F_{n-1} i F_{n-2} i dodjeljivanje vrijednosti za F_n - u tom nivou poziva!
Za početne uvjete možemo staviti

$$T(1) = D(1)$$
$$T(0) = D(0).$$

- Očito je $D(n) \geq c > 0$, gdje je c neka pozitivna konstanta, koja uključuje barem stack operacije za poziv $Fib(n)$ i nekoliko skokova (recimo 2 jump instr.).

- Preciznije određivanje $D(n)$ ovisi o tome što sve smatramo elementarnim operacijama. Preostaju još test veličine za n, dodjeljivanja za F_n i, <u>kao ključno</u>, <u>zbrajanje</u> F_{n-1} i F_{n-2}.

(A) <u>NE</u> vodimo računa o veličini brojeva F_n, tj. smatramo da zbrajanje (i dodjeljivanje) <u>ne</u> ovise o veličini brojeva.
To odgovara pretpostavci da su sve operacije direktno realizirane aritmetikom računala, tj. da zbrajanje traje (približno) konstantno, neovisno o veličini brojeva koji se zbrajaju.
Ostatak tijela funkcije, također, ima najviše konstantno trajanje.
Dakle, za $D(n)$ vijedi

$$D(n) = \Theta(1)$$

ili preciznije:

$$d_1 \leq D(n) \leq d_2 \quad , \forall n \in \mathbb{N}_0$$

(<u>ne</u> samo za dovoljno velike, nego baš za <u>sve</u>)

gdje su $d_1, d_2 > 0$ pozitivne konstante.

Realna pretpostavka je

$$D(n) = \begin{cases} d_1 & za \quad n = 0, 1 \\ d_2 & za \quad n \geq 2 \end{cases}$$

jer su tako odvojene i grane u algoritmu. Dato je i $d_2 > d_1$, jer druga grana sadrži i zbrajanje (pored testa - koji je zajednički i dodjeljivanja).

- Ostajemo ipak kod $d_1 \leq D(n) \leq d_2$, dozvoljavajući da operacije imaju poredstvo varijabilno trajanje - ali omeđeno konstantama.
  (To ima smisla i za arhitekture računala, ako se detaljno analiziraju operacije!)

- Intuitivno se dokazuje da je niz $T(n)$ monotono rastući (i to strogo, zbog $d_1 > 0$).

  Pošto nehomogeni član sadrži nejednakosti, ne možemo direktno analizirati $T(n)$.

  Logično je, zbog toga, promatrati 2 niza $T_1(n)$ i $T_2(n)$. Prvi koristi stalno donju granicu za $D(n)$, a drugi gornju:

$$\begin{cases} T_1(n) = T_1(n-1) + T_1(n-2) + d_1 \;, & n \geq 2 \\ T_1(0) = T_1(1) = d_1 \end{cases}$$

$$\begin{cases} T_2(n) = T_2(n-1) + T_2(n-2) + d_2 \;, & n \geq 2. \\ T_2(0) = T_2(1) = d_2 \end{cases}$$

Lako se dokazuje da je

$$T_1(n) \leq T(n) \leq T_2(n) \;, \quad \forall n \in \mathbb{N}_0.$$

- Rekurzije za T_1 i T_2 se lako rješavaju - čak su istog oblika - s različitom konstantom u nehomogenom članu.

Rekurzija je oblika:

$$t_n = t_{n-1} + t_{n-2} + d \;, \quad n \geq 2$$
$$t_0 = t_1 = d.$$

Rješenje (nehomogeni član $= 1^n \cdot p_0(n)$ s $p_0(n) = d$):

Karakteristična jednadžba je:

$$\underbrace{(x^2-x-1)}_{\text{homogeni dio}}\underbrace{(x-1)}_{\text{nehom. dio}} = 0 .$$

Korijeni su:

$$r_1 = \frac{1+\sqrt{5}}{2} \;,\quad r_2 = \frac{1-\sqrt{5}}{2} \;,\quad r_3 = 1$$

a opće rješenje je:

$$f_n = c_1 r_1^n + c_2 r_2^n + c_3 r_3^n =$$

$$= c_1 \cdot \left(\frac{1+\sqrt{5}}{2}\right)^n + c_2 \left(\frac{1-\sqrt{5}}{2}\right)^n + c_3 .$$

Konstante (ove 3) iz 3 prva clana rekurzije:

$$\begin{aligned} n=0: &\quad c_1 + c_2 + c_3 = d \\ n=1 &\quad c_1\left(\tfrac{1+\sqrt{5}}{2}\right) + c_2\left(\tfrac{1-\sqrt{5}}{2}\right) + c_3 = d \\ n=2 &\quad c_1\left(\tfrac{1+\sqrt{5}}{2}\right)^2 + c_2\left(\tfrac{1-\sqrt{5}}{2}\right)^2 + c_3 = 3d \end{aligned}$$

Posto su r_1, r_2 korijeni jednadžbe $x^2-x-1=0$, onda od zadnje jedn. oduzmemo prvu i dvaju:

$$c_1\underbrace{(r_1^2-r_1-1)}_{=0} + c_2\underbrace{(r_2^2-r_2-1)}_{=0} - c_3 = d$$

$$\Rightarrow \qquad \underline{\underline{c_3 = -d}} \qquad \text{(ovo se dobije i umstavanjem općeg rješenja u rekurziju!)}$$

Prve dvije jednadžbe glase:

$$\begin{aligned} c_1 + c_2 &= 2d \qquad \text{(može i determinantama)} \\ c_1\left(\tfrac{1+\sqrt{5}}{2}\right) + c_2\left(\tfrac{1-\sqrt{5}}{2}\right) &= 2d \end{aligned}$$

Prva – druga daje:

$$c_1\left(\frac{1-\sqrt{5}}{2}\right) + c_2\left(\frac{1+\sqrt{5}}{2}\right) = 0$$

tj.

$$c_1 = -\frac{1+\sqrt{5}}{1-\sqrt{5}}\,c_2 = \frac{-(1+\sqrt{5})^2}{(1-\sqrt{5})(1+\sqrt{5})}\,c_2 = \frac{-(1+2\sqrt{5}+5)}{1-5}\,c_2$$

$$\underline{c_1 = \frac{3+\sqrt{5}}{2}\,c_2}$$

Iz prve jednadžbe:

$$\frac{3+\sqrt{5}}{2}\,c_2 + c_2 = 2d$$

$$\Rightarrow \qquad \frac{5+\sqrt{5}}{2}\,c_2 = 2d$$

ili

$$c_2 = \frac{4}{5+\sqrt{5}}\, d$$

$$= \frac{4(5-\sqrt{5})}{2\phi}\, d = \frac{5-\sqrt{5}}{5}\, d$$

Za c_1:

$$c_1 = \frac{(3+\sqrt{5})(5-\sqrt{5})}{1\phi} = \frac{15+2\sqrt{5}-5}{1\phi} = \frac{5+\sqrt{5}}{5}\, d$$

Rješenja su: $\frac{1+\sqrt{5}}{\sqrt{5}}$ $\frac{\sqrt{5}-1}{\sqrt{5}}$

$$\boxed{c_1 = \left(1+\frac{1}{\sqrt{5}}\right)d,\ c_2 = \left(1-\frac{1}{\sqrt{5}}\right)d,\ c_3 = -d}$$

Na kraju, rješenje rekurzije je:

$$t_n = d\cdot\left[\left(1+\frac{1}{\sqrt{5}}\right)\cdot\left(\frac{1+\sqrt{5}}{2}\right)^n + \left(1-\frac{1}{\sqrt{5}}\right)\cdot\left(\frac{1-\sqrt{5}}{2}\right)^n - 1\right],\ n\in\mathbb{N}_0$$

- Donju ogradu T_1 dolivamo za $d = d_1$, a gornju T_2 za $d = d_2$.
- U terminima broja ϕ, t_n glasi:

$$t_n = d\left[\left(1+\frac{1}{\sqrt{5}}\right)\phi^n + \left(1-\frac{1}{\sqrt{5}}\right)(-\phi)^{-n} - 1\right],\ n\in\mathbb{N}_0$$

Dalje je:

$$0 < 1-\frac{1}{\sqrt{5}} < 1$$

$$0 < |(-\phi)^{-n}| \leq 1,\ \forall n\in\mathbb{N}_0$$

$$\Rightarrow \left|\left(1-\frac{1}{\sqrt{5}}\right)(-\phi)^{-n}\right| < 1,\ \forall n\in\mathbb{N}_0$$

- Za gornju ogradu T_2, dolivamo:

$$T_2(n) < d_2\cdot\left(1+\frac{1}{\sqrt{5}}\right)\phi^n,\ n\in\mathbb{N}_0$$

jer je suma zadnja dva člana u t_n uvijek negativna!

- Zaključujemo da je

$$T_2(n) = \mathcal{O}(\phi^n)$$

a zbog $T(n) \leq T_2(n)$, vrijedi i

$$T(n) = \mathcal{O}(\phi^n)$$

Već ranije smo pokazali da je $T(n) = \Omega(\phi^n)$, pa je

$$\underline{\underline{T(n) = \Theta(\phi^n)}}.$$

- Ako želimo $T(n)$ usporediti baš sa F_n, a ne sa ϕ^n, očekujemo, naravno, isti rezultat.

- Za precizan dokaz, treba vrijednost t_n izraziti preko F_n. Slijedi:

$$t_n = d\cdot\left[\frac{1}{\sqrt{5}}[\phi^n - (-\phi)^{-n}] + \phi^n + (-\phi)^{-n} - 1\right]$$

$$= d\left[\underbrace{\frac{1}{\sqrt{5}}[\phi^n - (-\phi)^{-n}]}_{F_n} + \underbrace{[\phi^n - (-\phi)^{-n}]}_{\sqrt{5}\cdot F_n} + 2(-\phi)^{-n} - 1\right]$$

$$\underline{t_n = d\cdot\left[(\sqrt{5}+1)F_n + 2(-\phi)^{-n} - 1\right], \quad n\in\mathbb{N}_0}.$$

- Sličnim razmatranjima dolazimo željeni oblik ograda za $T(n)$:

Odozgo:

(~~a~~) Slijedi da je $|2(-\phi)^{-n}| \leq 2$, $\forall n\in\mathbb{N}_0$

$\longrightarrow$ odn. $|2(-\phi)^{-n}| < 1$, za sve dovoljno velike n ($\forall n \geq n_0$, uz neki mali n_0) – dovoljno je $n_0 = 2$

($\phi^{-1} = 0.61803\ldots \Rightarrow \phi^{-2} < \frac{1}{2}$)

Slijedi:

(baci)
$$T_2(n) \leq d_2\cdot[(\sqrt{5}+1)F_n + 1] < (F_n > 1 \text{ za } n \geq 3)$$
$$< \underline{d_2\cdot(\sqrt{5}+2)F_n}, \text{ za } \forall n \geq 3.$$

$\longrightarrow$ ili direktno, za $n \geq n_0$:

$$T_2(n) < \underline{d_2(\sqrt{5}+1)F_n}, \quad \forall n \geq n_0 (=2)$$

Zaključak: $\underline{T_2 = \mathcal{O}(F_n)}$ i $\underline{T(n) = \mathcal{O}(F_n)}$.

Odozdo: poći od nekog n_0, očito je:

$$F_n + \underbrace{2(-\phi)^{-n} - 1}_{>-3} > 0, \quad \forall n \geq n_0$$

ti. čim je $F_n \geq 3$, ovo vrijedi. (Dakle, $n_0 = 4$). Slijedi

$$T_1(n) > d_1\cdot\sqrt{5}F_n, \quad n \geq 4.$$

Zaključak: $\underline{T_1 = \Omega(F_n)}$ i $\underline{T(n) = \Omega(F_n)}$.

Dakle:

$$\underline{d_1\sqrt{5}\cdot F_n < T(n) < d_2\cdot(\sqrt{5}+1)F_n}\ , \quad \forall n \geq 4$$

što dokazuje $\underline{T(n) = \Theta(F_n)}$.

Ovdje zak zvamo i gornju i donju ogradu!

— • —

<u>Napomena</u> Do ovog rezultata $T(n) = \Theta(F_n)$ može se doći i direktno iz rekurzije, bez njenog rješavanja. Koristimo tehniku tzv. "<u>konstruktivne indukcije</u>".

- Rekurzija glasi:

$$T(n) = T(n-1) + T(n-2) + D(n),\ n \geq 2$$

uz početne uvjete

$$T(0) = D(0),\ T(1) = D(1)$$

Znamo da za nehomogeni dio vrijedi

$$D(n) = \Theta(1).$$

Vrijedi i jače - $D(n) > 0$, pa možemo staviti

$$0 < d_1 \leq D(n) \leq d_2\ , \quad \forall n \in \mathbb{N}_0.$$

Kao i obično, kod indukcije, treba znati što želimo dokazati.

Da dokažemo da je $T(n) = \Theta(F_n)$, treba pokazati da postoje konstante a, b takve da je

$$aF_n \leq T(n) \leq b\cdot F_n\ , \quad \forall n \geq n_0,$$

za neki n_0.

- Ideja konstruktivne indukcije je u tome da se, osim dokaza egzistencije, efektivno nađu tražene konstante.

- Dokazimo prvi dio: $aF_n \leq T(n)$.

Pretpostavka indukcije je da ta relacija vrijedi za prethodna 2 člana:

$$aF_{n-2} \leq T(n-2)$$
$$aF_{n-1} \leq T(n-1)$$

gdje je $n \geq 2$.

Vratit da nam fali baza indukcije. Pošto imamo slobodu izbora za a, pokušavamo a <u>odrediti</u> tako da pokrije i korak i baza indukcije.

Od koraka krećemo zato da vidimo koja ograničenja odnje izlaze na a.

Za očekivati je da će baza indukcije praviti manje problema - a se valjda lako naštima za bazu.

Vratimo pretpostavku u rekurziju:

$$T(n) = T(n-1) + T(n-2) + D(n) \geq \quad (D(n) \geq d_1)$$

$$\geq \underbrace{a \cdot F_{n-1} + a \cdot F_{n-2}}_{= a \cdot F_n} + d_1 =$$

$$= a \cdot F_n + d_1 .$$

Kako je $d_1 > 0$, slijedi direktno da je $T(n) \geq a \cdot F_n$.

- Dakle, korak indukcije ne postavlja nikakva ograničenja na a (tj. mjesto za $\forall a$).

Ostaje podesiti a tako da mjesto i baza indukcije, pa će i čitava indukcija biti korektna.

Za prva 2 slučaja mjesto:

$n=0$: $T(0) = D(\emptyset) \geq d_1$ i želimo $T(0) \geq a \cdot \underbrace{F_0}_{0} = 0$.

No, $d_1 > 0$ po pretp., pa je odmah

$$T(0) \geq d_1 > \emptyset = a \cdot F_0 . \qquad \forall a .$$

$n=1$: $T(1) = D(1) \geq d_1$ i želimo $T(1) \geq a \cdot \underbrace{F_1}_{=1} = a$.

Stavimo li $d_1 \geq a$ izlazi

$$T(1) \geq d_1 \geq a = a \cdot F_1 \quad \text{✓}.$$

Dakle, baza indukcije mjesto uz uvjet $a \leq d_1$. Biramo, naravno, najbolju mogućnost - najveći a (za što bolju donju ogradu), tj.

$$\underline{a = d_1 .}$$

Time smo dokazali:

$$\underline{T(n) \geq d_1 \cdot F_n \ , \ \forall n \in \mathbb{N}_0 .}$$

Napomena: ovo je, očito, slabija donja ograda od ranije - za faktor $\sqrt{5}$, ali je izvedena direktno, bez rješavanja rekurzije, a zadovoljava sve naše potrebe.

- Preostaje pokazati drugi dio : $T(n) \leq b \cdot F_n$.

Rekurzija sadrži 2 prethodna člana, pa je pretpostavka indukcije da vrijedi

$$T(n-1) \leq b \cdot F_{n-1}$$
$$T(n-2) \leq b \cdot F_{n-2}$$

za neki $n \geq 2$. Uvrstimo u rekurziju i iskoristimo $D(n) \leq d_2$

$$T(n) = T(n-1) + T(n-2) + D(n) \leq \quad (D(n) \leq d_2)$$
$$\leq b \cdot F_{n-1} + b \cdot F_{n-2} + d_2 = b \cdot F_n + d_2.$$

Zaključak je $T(n) \leq b \cdot F_n + d_2$.

Međutim, kako je $d_2 > 0$, <u>nismo</u> uspjeli dokazati $T(n) \leq bF_n$.

Odavde, izgleda kao da tabva tvrdnja uopće ne vrijedi.

- Srećom, nije tako. Problem je u obliku pretpostavke indukcije, koji ne omogućava provođenje koraka. Ono što nam smeta je dodavanje pozitivne konstante d_2 u rekurziji.
U našoj pretpostavci nema slobode - nema člana koji bi kompenzirao taj dodatak d_2.

- Zvuči apsurdno, ali pokušajmo dokazati ovo:

$$T(n) \leq b \cdot F_n - c \quad , \text{ za } n \geq n_0.$$

gdje je c neka konstanta. Ako dokažemo da možemo uzeti $c > 0$, onda smo dokazali jaču tvrdnju, iz koje odmah izlazi i $T(n) \leq b \cdot F_n$.

Ideja zapravo nije tako apsurdna. Ako povećamo b, onda možda ima prostora i za $-c$, jer F_n rastu.

- Pretpostavka indukcije je sad

$$T(n-1) \leq b \cdot F_{n-1} - c$$
$$T(n-2) \leq b \cdot F_{n-2} - c$$

za neki $n \geq 2$. Uvrštavanjem u rekurziju, dobivamo:

$$T(n) = T(n-1) + T(n-2) + D(n) \leq (\text{pretp. ind. i } D(n) \leq d_2)$$
$$\leq b \cdot F_{n-1} - c + b \cdot F_{n-2} - c + d_2 =$$
$$= b \cdot F_n + d_2 - 2c.$$

Mi želimo $T(n) \leq b \cdot F_n - c$. To ćemo sigurno dobiti, ako vrijedi

$$b \cdot F_n + d_2 - 2c \leq b \cdot F_n - c \quad , \forall n, n \geq n_0$$
(n_0 treba naći)

Članovi sa F_n se kratite - tj. taj dio ima dobar oblik!

Odavde slijedi: $\underline{d_2 \leq c}$.

Ako tako odaberemo c, onda iz pretpostavke indukcije slijedi

$$T(n) \leq b \cdot F_n - c$$

i to <u>bez</u> dodatnih uvjeta na n. Dakle, ostaje samo podesiti bazu indukcije.

Vozimo još da slijedi i $c > 0$ (jer $d_2 > 0$), što nam upravo odgovara.

- Za podešavanje b, c tako da uvjedi baza indukcije, možemo odmah uzeti najbolji c, $c = d_2$.
Pogodnije je, međutim, raditi u općem obliku, a na kraju odabrati b i c. (Može se desiti da c treba još povećati!).

- Za bazu indukcije, krećemo od $n = 0$:

$n = 0$: $T(\emptyset) = D(\emptyset) \leq d_2$ i želimo $T(0) \leq b \cdot F_0 - c = -c$

Zahtjev na c bi bio $d_2 \leq -c$ ili $\underline{c \leq -d_2}$.

To je u kontradikciji s ranijim zahtjevom, jer je $d_2 > 0$.

Dakle, baza indukcije <u>ne</u> uvjedi za $n = 0$. No, to nas ne smeta, jer je dovoljno dokazati da je $T(n) \leq b \cdot F_n - c$ za sve n, počev od nekog n_0.
Tj. ne mora nužno biti $n_0 = 0$.

- Pokušajmo krenuti sa $n = 1$:

$n = 1$: $T(1) = D(1) \leq d_2$ i želimo $T(1) \leq b \cdot \underbrace{F_1}_{1} - c = b - c$.

Ako uvjedi $\underline{d_2 \leq b - c}$, onda je sigurno i $T(1) \leq bF_1 - c$.

(Nap. - ne možemo odavde odrediti b, c - jer moramo bazu osigurati za 2 člana - uje dovoljno za jedan!).

$n = 2$: $T(2) = T(1) + T(\emptyset) + D(2)$.

Iz $T(0), T(1) \leq d_2$ i $D(2) \leq d_2$, slijedi $T(2) \leq 3d_2$.

Želimo: $T(2) \leq b \cdot \underbrace{F_2}_{1} - c = b - c$.

Dakle, ako vrijedi $\underline{3d_2 \leq b - c}$, onda je sigurno i $T(2) \leq bF_2 - c$.

Imamo sistem od 3 uvjeta - nejednadžbe na b i c, koji, ako vrijede, osiguravaju dokaz $T(n) \leq bF_n - c$, za $\forall n \geq 1$.

$$\left.\begin{array}{l} d_2 \leq c \\ d_2 \leq b - c \\ 3d_2 \leq b - c \end{array}\right\} \Rightarrow \begin{array}{l} b \geq d_2 + c \\ b \geq 3d_2 + c \end{array}$$

ti najmanji c $\Rightarrow$ najmanji b.

Vrijedno bi u prvom uvjetu optimalni c, $c = d_2$, iz preostala dva uvjeta slijedi

$$\begin{array}{l} 2d_2 \leq b \\ 4d_2 \leq b \end{array},$$

ti optimalni b je $b = 4d_2$.

- Vidimo da nas sistem uvjeta zna bar gdje vrijedi

$$b = 4d_2, \quad c = d_2.$$

Tj. dokazali smo da vrijedi:

$$\underline{T(n) \leq d_2(4F_n - 1), \quad \forall n \geq 1}.$$

tj. naravno, $\underline{T(n) < 4d_2F_n, \quad \forall n \geq 1}$.

Napomena: I ova ocjena je losija od ranije (tamo je faktor bio $d_2 \cdot (\sqrt{5}+1)$)

$$\sqrt{5}+1 < 4.$$

No, ona ocjena je vrijedila za $n \geq 2$, a ova vrijedi za $n \geq 1$.

- Gornji sistem uvjeta se može formulirati i za drugačije početne vrijednosti - na pr. da baza indukcije kreće od 2 ili od 3 i sl.

Taj sistem uvjeta se onda može rješavati kao problem linearnog programiranja - ako zadamo kriterij optimalnosti koeficijentata:

na pr: $b \to min$
ili $c \to min$
ili $b+c \to min$

i slično.

- Za to nema potrebe - jer nas zanima asimptotsko ponašanje za $T(n)$, a to smo odredili.

Iz ove analize slijedi ovaj zaključak:

" Kompleksnost algoritma 1 za računanje F_n je <u>eksponencijalna u n</u>, čak i kad zbrajanje smatramo elementarnom operacijom ".

$$Compl\,(Alg\,1) = \Theta(\phi^n).$$

<u>Primjer 25.</u> Za implementaciju algoritma 1, opet koristimo realni tip extended za F_n - da vidimo dovoljno veliki raspon egzaktno prikazivih Fibonaccijevih brojeva.

Odgovarajući potprogram za TP 6.0 je (FIB-REC.PAS)

```
function Fib (n: integer): real;  { real = extended s opcijom $N+ }
var F1, F2: real;
begin
  if n = 0 then                    { može i <= }
    Fib := 0.0                     } može i ovo: if n <= 1 then
  else                             }               Fib := n
    if n=1 then
      Fib := 1.0
    else
      begin
      F1 := Fib(n-1);
      F2 := Fib(n-2);
      Fib := F1 + F2;
      end;
end; {Fib}
```

Ako koristimo 80x87, <u>ne smijemo</u> koristiti zapis

$$Fib := Fib(n-1) + Fib(n-2)$$

jer on proizvodi 8087 stack overflow za $n \geq 8$.
Zbog toga, međurezultate u rekurziji spremamo u privremene varijable F1 i F2 (pomoćne), na 80x86 stacku.

- Račun je proveden do $n=30$. Karakteristična vremena su

$F_{10} = 55$, $T(10) = 0.001871$ s
$F_{20} = 6765$, $T(20) = 2.319690$ s
$F_{30} = 832040$, $T(30) = 285.304618$ s

- U pojednostavljenom modelu $T(n) = c \cdot F_n$, konstanta c teži prema vrijednosti

$$\underline{c = 3.428977 \cdot 10^{-4}\ \text{s}}$$ ■

[O kompleksnosti u slučaju da zbrajanje nije elementarna operacija - ovisi o veličini brojeva, vrsto kasnije].

Zadatak Smislite algoritam za zbrajanje "velikih" nenegativnih cijelih brojeva, koristeći pokaz u nekoj odabranoj bazi $b \in \mathbb{N}$. Pokažite da tada, za zbrajanje F_{n-1} i F_{n-2}, potrebno vrijeme zadovoljava asimptotsku relaciju

$$T(n) = \Theta(\log F_n)$$

odnosno: $$T(n) = \Theta(n).$$

Odredite potrebne konstante što precizirate.
Na kraju, pokažite da za kompleksnost algoritma 1 uz takve dodatne operacije i dalje vrijedi

$$T(n) = \Theta(F_n) \blacksquare$$

- Bolji algoritam se, uzavrno, dolazi tako da naprosto zbrajamo prethodna dva člana niza i srodi se na pamćenje ta dva prethodna člana.

Direktni algoritam je:

Algoritam 2. F_n - zbrajanjem

```
function Fib (n: integer ) : integer;

var  F, F1, F2, k : integer; { k - indeks petlje, F pamti F_k,
                               F1 pamti F_{k-1}, a F2 pamti F_{k-2} }

begin
  if n <= 1 then
    Fib := n
  else
    begin
    F1 := 0;
    F  := 1;
    for k := 2 to n do
      begin
      F2 := F1;   { pomak }
      F1 := F;
      F := F1 + F2
      end;
    Fib := F;
    end;
end;  {Fib}
```

Zbog jednostavnog oblika rekurzije, možemo uštedjeti jednu varijablu i ne pamtiti F_{n-2}, već ga računati iz novih F_n, F_{n-1} – oduzimanjem.

To je ušteda memorije, ali donosi jednu aritmetičku operaciju više.

(Općenito – ta "ušteda" za rekurziju reda k izgleda ovako:

$$\epsilon_n = \overset{a_{k-1}}{a_1}\epsilon_{n-1} + \dots + \overset{a_0}{a_k}\epsilon_{n-k}$$

oduzimanjem dobiti: $\epsilon_{n-k} = (\epsilon_n - a_1\epsilon_{n-1} - \dots - a_{k-1}\epsilon_{n-k+1})/a_k$.

Pri tome ne smijemo koristiti dodatnu varijablu za $a_{k-1}\epsilon_{n-k+1}$, jer onda ništa ne štedimo.

Dakle – broj operacija se približno udvostručuje, za "ušteđu" jedne varijable).

– Za Fibonaccijeve brojeve, taj algoritam ima oblik:

Algoritam 2a. F_n – "štedljivo zbrajanje"

```
function Fib (n: integer) : integer;
var ∈, F, k : integer; { k - indeks petlje, F pamti F_k,
                         ∈ pamti F_{k-1} }
begin
  ∈ := 1 ;   { odgovara F_{-1} = 1, tako da u rekurziji dolazimo
               F_1 = F_0 + F_{-1} = 1 }
  F := 0 ;   { F_0 }
  for k := 1 to n do
    begin
      F := ∈ + F ;   { Fib(k) ← F_{k-2} + F_{k-1} }
      ∈ := F - ∈ ;   { F_{k-1} ← F_k - F_{k-2} }
    end;
  Fib := F;
end;
```

– Ovo je "vrlo mudro", ali je neelegantno – jer računa ∈ na onim petlje – za sljedeći korak, kojeg možda nema (za zadnji prolaz).

Osim toga, to je sporije od obične varijante (zbrajanje i oduzimanje ipak traje dulje od dodjeljivanja).

– Zbog toga, analiziramo običnu varijantu.

Algoritam 2 nije rekurzivan i analiza njegove kompleksnosti je jednostavna.

Dodjeljivanje vrijednosti možemo smatrati operacijom konstantnog trajanja, neovisno o veličini rezultata. Dovoljno je raditi s pokazivima na objekt za pohranu integera.

Većinu vremena odnosi izvršavanje petlje za $k=2,\dots,n$. Označimo s $D(k)$ trajanje izvršavanja petlje za vrijednost indeksa k.

Tada je

$$T(n) = \begin{cases} c + \sum_{k=2}^{n} D(k) & , \text{za } n \geq 2 \\ c_0 & , \text{za } n = 0, 1 \end{cases}$$

- Konstanta $c_0 > 0$ uključuje trajanje poziva i povratka iz funkcije i dodjeljivanje `Fib := n`.
- Konstanta $c > 0$ uključuje poziv i povratak, 3 dodjeljivanja u else bloku, ali izvan petlje i, eventualno, osnovni korak pripreme ili kraja petlje.
- Vrijeme $D(k) > 0$ sadrži 3 dodjeljivanja, kontrolu petlje i zbrajanje F1 i F2.

Samo $D(k)$ može ovisiti o veličini Fibonaccijevih brojeva.

- Ako zbrajanje smatramo elementarnom operacijom, onda trajanje ne ovisi o veličini brojeva, onda (kao i ranije) mjerimo

$$d_1 \leq D(k) \leq d_2 \quad , \forall k \geq 2$$

gdje su $d_1, d_2 > 0$ pozitivne konstante.

- Onda je trivijalno:

$$(n-1)\,d_1 \leq \sum_{k=2}^{n} D(k) \leq (n-1)\,d_2 \quad , \forall n \geq 2$$

odnosno:

$$\underbrace{c + (n-1)\,d_1}_{d_1 n + c - d_1} \leq T(n) \leq \underbrace{c + (n-1)\,d_2}_{d_2 n + c - d_2}$$

tj, uz oznaku $c_i = c - d_i$, $i = 1, 2$ (c_i ne mora biti pozitivan), dobivamo:

$$\underline{d_1 n + c_1 \leq T(n) \leq d_2 n + c_2 \quad , \forall n \geq 2}$$

Time je dokazano $\underline{T(n) = \Theta(n)}$. Compl(Alg2) $= \Theta(n)$.

Ovo je bitno poboljšanje u odnosu na algoritam 1. Trajanje algoritma <u>linearno</u> ovisi o n.

<u>Napomena</u>: Duljina zadaće $x = \{n\}$ problema FIB je

$$|x| = \lfloor \log_b n \rfloor + 1$$

gdje je $b \geq 2$ baza za pohranu brojeva (b=10 na ulazu programa odnosno b=2 u matrici).

Dakle:

$$n = b^{\log_b n} = b^{\lfloor \log_b n \rfloor + c}$$

gdje je $0 \leq c < 1$ takav broj da je

$$\log_b n = \lfloor \log_b n \rfloor + c$$

(tj. $c = \log_b n - \lfloor \log_b n \rfloor$).

U terminu duljine zadaće $|x|$, izlazi:

$$n = b^{|x| + \underbrace{c-1}_{-1 \leq c-1 < 0}}$$

što daje:

$$\frac{1}{b} \cdot b^{|x|} \leq n < b^{|x|}$$

ili

$$n = \Theta(b^{|x|}).$$

Uvrstimo li to u kompleksnost algoritma 2, dolazimo:

$$\underline{T(n) = \Theta(b^{|x|})}.$$

Dakle, algoritam 2 je još uvijek <u>eksponencijalni</u> algoritam u ovisnosti o duljini zadaće.

Tj. ne znamo još da li je problem FIB polinomni ili ne ■

<u>Primjer 26.</u> U implementaciji koristimo realni tip extended za prikaz Fibonaccijevih brojeva.

```
Promjene su:  function Fib (n: integer) : real;  { = extended }
2.            var F, F1, F2 : real;
                  k : integer;
```

Ostatak algoritma je isti.

Račun je proveden do $n = 87$ (jer F_{87} je najveći broj koji će moće egzaktno prikazati i ispisati u extended tipu).

Karakteristična vremena su:

$$F_{30}\ , \quad T(30) = 0.004\,092$$
$$F_{60}\ , \quad T(60) = 0.008\,159$$
$$F_{87}\ , \quad T(87) = 0.011\,825\,.$$

U pojednostavljenom modelu za kompleksnost

$$T(n) = c \cdot n$$

(koji odgovara asimptotskoj kompleksnosti, uz $\underbrace{d_1 = d_2}_{\text{ovo je korektna pretp. za zbrajanje - konst. trajanja}} = c$)

konstanta c tada prema mjerodavnosti

$$c = 1.36 \cdot 10^{-4}\ s\,.$$

(ostatak znamenki iza druge decimale je neupotrebljiv, zbog točnosti mjerenja, koja je približno 10^{-6} s.) ■

Algoritam 2 ima tu manu, da za računanje F_n, mora redom izračunati sve Fibonaccijeve brojeve do F_n. Tj. za n-tog, on računa prvih n.

Ovo ponašanje možemo usporediti s algoritmima za računanje n-te potencije broja a (rekurzija za taj problem je: $t_0 = 1$, $t_n = a \cdot t_{n-1}$, $n \geq 1$). Algoritam 2 odgovara sukcesivnom množenju:

Algoritam POT1:
```
t := 1;                    ← početni uvjeti
for k := 1 to n do
    t := a * t;            ← ovo je zapravo rekurzija
                             (usporedi s Alg. 2).
```

Ovdje, isto tako, redom računamo sve potencije od a, do n-te.

- Međutim, znamo da postoji i bitno bolji algoritam za računanje a^n - preko binarnog prikaza eksponenta n.

$$n = n_k \cdot 2^k + n_{k-1} \cdot 2^{k-1} + \ldots + n_1 \cdot 2 + n_0$$

gdje je $n_i \in \{0, 1\}$ i $n_k = 1$, za $n > 0$.

$$a^n = a^{\sum_{i=0}^{k} n_i 2^i} = \prod_{i=0}^{k} a^{n_i \cdot 2^i} = \left\{ n_i \in \{0,1\}, \text{ pa je } a^{n_i 2^i} = 1 \text{ za } n_i = 0 \right\}$$

$$= \prod_{\substack{i=0 \\ n_i = 1}}^{k} a^{2^i} .$$

Uz oznaku $b_i = a^{2^i}$, imamo:

$$b_0 = a^1 = a$$

$$b_i = a^{2^i} = a^{2^{i-1} \cdot 2} = \left(a^{2^{i-1}}\right)^2 = b_{i-1}^2 , \quad i = 1, \ldots, k$$

Članovi niza b_i dobivaju se kvadriranjem prethodnih, dalje jedinim množenjem.

Formula za a^n glasi:

$$a^n = \prod_{\substack{i=0 \\ n_i = 1}}^{k} b_i .$$

Broj faktora u tom produktu jednak je broju jedinica u binarnom zapisu broja n. Taj broj je najviše

$$\lfloor \lg n \rfloor + 1 . \quad (= k+1 = \text{broj binarnih znamenki broja } n).$$

Algoritam za brzo potenciranje srodi se na računanje članova b_i i akumulaciji produkta za a^n - prema binarnom rastavu za n:

Algoritam POT 2: a^n - binarnim potenciranjem, $n \in \mathbb{N}_0$

```
pot := 1;  { inicijalizacija produkta za a^n }
b   := a;  { inicijalizacija b niza, b_0 = a }
while n > 0 do
  begin
  while not odd(n) do  { n_i = 0, jer n paran }
    begin
    b := sqr(b);  { b_i = b_{i-1}^2 }
    n := n div 2;  { brišemo znamenku n_i }
    end;
  { ovdje je n_i = 1, jer n neparan => akumuliraj b_i
    u produkt za a^n }
  pot := pot * b;
  n := n - 1          { obriši - oduzmi obrađenu jedinicu! }
  end;                  Ako smo ušli u while -> n>0, najniža
                        znamenka je sigurno 1 }
{ovdje je pot = a^n}.
```

Tipovi ovise o tome kojeg je tipa a (cijeli ili real). Varijable pot i b imaju isti tip kao a.

Broj prolaza kroz unutarnji while jednak je broju binarnih znamenki broja n (svaki prolaz briše točno jednu bin. znam.), tj:

$$k+1 = \lfloor \lg n \rfloor + 1$$

a broj prolaza kroz dio vanjskog while-a (tj. kroz vanjski while) jednak je broju jedinica u binarnom prikazu za n. Onda je:

$$Compl(POT2) = \Theta(\lg n)$$

uz pretpostavku da kvadriranje ($b \leftarrow sqr(b)$) i množenje ($pot \leftarrow pot * b$) imaju najviše konstantno trajanje - neovisno o veličini brojeva b i pot.

- Vidimo da je ovo pravi polinomni, čak linearni algoritam, jer je duljina zadaće dana s

$$|x| = \lfloor \lg n \rfloor + 1 + c$$

(c je duljina za a, a, po pretp, kompleksnost ne ovisi o duljini za a, pot, b).

Tj. $$Compl(POT2) = \Theta(|x|)$$ ■

Ovaj algoritam se, naravno, može generalizirati i na baze koje nisu binarne (v. Knuth, Seminumerical Alg's).
Osnovni zaključak ostaje isti i za $b>2$ – algoritam je linearan!
[U većim bazama se može još malo smanjiti broj množenja].

– Osnovna ideja je da pokušamo nekako skćno umnožiti za računanje F_n.
Prvo treba problem nalaženja F_n formulirati u obliku potenciranja.

Vidimo da je rekurzija za potenciranje ima red jednak 1

$$t_0 = 1$$
$$t_n = a \cdot t_{n-1}\ ,\ n \geq 1\ .$$

– Rekurzije višeg reda možemo prikazati kao sistem rekurzivnih jednadžbi prvog reda (isto kao za diferencijalne jednadžbe).

Linearna, homogena rekurzivna relacija reda k, s konstantnim koeficijentima ima oblik (1):

$$t_n = a_1 t_{n-1} + \dots + a_k t_{n-k}\ ,\ za\ n \geq k \qquad (6)$$
$$a_{k-1} \qquad a_0$$

(ovaj oblik izlazi iz (1) dijeljenjem s $a_0^k \neq 0$),
s tim da se zadaje k početnih uvjeta $t_0, -, t_{k-1}$.

– Uvedimo vektorski niz $x^{(n)}$ dimenzije k, tj. $x^{(n)} \in \mathbb{C}^k$ (na pr.) formulom:

$$x^{(n)} = \begin{bmatrix} t_n \\ t_{n+1} \\ \vdots \\ t_{n+k-1} \end{bmatrix}\ ,\ n \in \mathbb{N}_0\ .$$

Vektor $x^{(0)}$ je upravo vektor početnih inicijalnosti rekurzije.
Rekurzivnu relaciju (6) možemo zapisati u matričnom obliku:

$$x^{(n)} = A \cdot x^{(n-1)}\ ,\ za\ n \geq 1$$

gdje je A matrica reda k ($A \in \mathbb{C}^{k\times k}$) oblika:

$$A = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & & & & & \\ 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 \\ a_k & a_{k-1} & a_{k-2} & \dots & a_2 & a_1 \end{bmatrix}$$
$$a_0 \quad a_1 \quad a_2 \quad \dots \quad a_{k-2} \quad a_{k-1}$$

} oraj dio predstavlja pomak komponenti za jedno mjesto unatrag
$x^{(n)}_i = x^{(n-1)}_{i+1}\ ,\ i = 1, -, k-1$

} nova inicijalnost po (6).
za $t_{n-k+1} = \dots$

- Matrica A ne ovisi o n, jer rekurzija (6) ima konstantne koeficijente.

Vidimo da smo upravo dobili oblik rekurzije za potenciranje, samo u vektorsko - matričnoj formi:

$$x^{(0)} \quad \text{zadan}$$
$$x^{(n)} = A \cdot x^{(n-1)}, \quad n \geq 1$$

gdje je A matrica reda k. Tj. broj a prelazi u matricu A.

- Ovdje se $x^{(n)}$ trivijalno izražava preko početnog vektora $x^{(0)}$

$$x^{(n)} = A^n \cdot x^{(0)}, \quad n \geq 0$$

(jer formula vrijedi i za $n=0$, pošto je $A^0 = I$).

- Dakle, naša rekurzija (6) je svedena na potenciranje.
- Naš problem je: izračunati n-ti član t_n niza rješenja zadane rekurzije.

Taj član se pri puta javlja u vektoru $x^{(n-k+1)}$, kao zadnja (k-ta) komponenta. Tada je

$$t_n = x_k^{(n-k+1)}, \quad n \geq k$$

i $$\underline{x^{(n-k+1)} = A^{n-k+1} x^{(0)}}.$$

čak ne treba cijela matrica!

Treba naći $n-k+1$-u potenciju početne matrice A.
Uz oznaku $m = n-k+1$, problem se svodi na potenciranje - naći vektor $A^m x^{(0)}$.

- Analogan algoritma POT1, valjalo bi redom vektore $x^{(i)}$, umjesto matricu na prosti vektor.
- Analogan algoritma POT2, računa niz matrica

$$B_0 = A$$
$$B_i = B_{i-1}^2, \quad i = 1, \ldots, \lfloor \lg m \rfloor$$

i akumulira produkt

$$A^m x^{(0)} = \left(\prod_{\substack{i=0 \\ m_i = 1}}^{\lfloor \lg m \rfloor} B_i \right) x^{(0)}$$

gdje su m_i, $i = 0, \ldots, \lfloor \lg m \rfloor$ binarne znamenke broja m.

Akumulacija se prorooli u vektorskoj formi, uz oznaku.

pot = produkt B-ova · x_0 (vektor dim. k)

Nova mjeolmost za pot se doliva vektorski

pot := B_i · pot (i takav da je $m_i = 1$)

Tj <u>ne</u> treba akumulirati matrični produkt B-ova, pa na kraju pomnožiti na $x^{(0)}$, već stalno računati vektor.

- Ova varijanta oblika POT2 izgleda mnogo gora po utrošku memorije - jer mora spremati matricu B i vektor pot. To, međutim, nije točno, jer matrice B imaju poseban oblik, pošto je i A posebnog, jednostavnog oblika.

— Ilustrirajmo to na primjeru Fibonaccijevog niza.

Rekurzija je reda $k=2$, tj. matrica A je reda 2.

$$F_0 = 0,\ F_1 = 1$$
$$F_n = F_{n-1} + F_{n-2},\ n \geq 2.$$

Vektorska forma je, uz oznaku $f^{(n)}$ (umjesto $x^{(n)}$) za vektore i oznaku F (umjesto A) za matricu:

$$f^{(n)} = \begin{bmatrix} F_n \\ F_{n+1} \end{bmatrix},\ n \geq 0$$

početni uvjet je: $f^{(0)} = \begin{bmatrix} F_0 \\ F_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = e_2$ (drugi vektor kanonske baze u $\mathbb{R}^2$)

a za matricu $F = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$. (tj. matrična rekurzija je: $f^{(n)} = F \cdot f^{(n-1)},\ n \geq 1$)

Znamo da je

$$f^{(n-1)} = \begin{bmatrix} F_{n-1} \\ F_n \end{bmatrix} = F^{n-1} \cdot f^{(0)} = F^{n-1} e_2,\ n \geq 1.$$

- Vidimo da je $F^{n-1}e_2$ upravo drugi stupac matrice F^{n-1}

$$F^{n-1} = [\, F^{n-1}e_1,\ F^{n-1}e_2 \,] \text{ po stupcima.}$$

- Lako se nalazi i prvi stupac u toj matrici. Iz matrice F se čita da je

$$Fe_1 = (\text{prvi stupac u } F) = e_2$$

Odavde direktno slijedi, umnoženjem s F^{n-2}, da je

$$F^{n-1}e_1 = F^{n-2}e_2 = f^{(n-2)} = \begin{bmatrix} F_{n-2} \\ F_{n-1} \end{bmatrix}.$$

Imamo direktni oblik za F^{n-1}.

$$F^{n-1} = \begin{bmatrix} F_{n-2} & F_{n-1} \\ F_{n-1} & F_n \end{bmatrix}, \quad n \geq 2.$$

Ovo slučajno vrijedi i za $n=1$, jer možemo definirati F_{-1} tako da primjenom rekurzije na F_{-1}, F_0 dobijemo F_1. Izlazi da treba definirati $F_{-1} = F_1 = 1$.

- Napomena: u općem slučaju, retke matrice A^{n-1} treba pomaknuti za jedno mjesto prema gore, da dobijemo prvih $n-1$ redaka u A^n (To ide direktno iz $A^n = A \cdot A^{n-1}$).
  Ovdje, zbog simetrije matrice F, to isto vrijedi i za stupce. Dakle je i F^{n-1} simetrična ■

~ Napomena: Ovdje matrice F^{n-1} sadrže upravo Fibonaccijeve brojeve, tj. elemente niza koji zadovoljava rekurziju.
  Općenito, to ne vrijedi. Svaki član niza t_n, za $n \geq k$, možemo izraziti kao linearnu kombinaciju prethodnih vrijednosti $t_0, \dots, t_{k-1}$. Reci matrica A^n (A^{n-k+1}) sadrže upravo te koeficijente ■

- Da pojednostavnimo algoritam, računat ćemo vektor $f^{(n-1)}$

$$f^{(n-1)} = \begin{bmatrix} F_{n-1} \\ F_n \end{bmatrix} = F^{n-1}e_2 = F^n e_1$$

relacijom

$$\begin{bmatrix} F_{n-1} \\ F_n \end{bmatrix} = F^n e_1, \quad n \geq 0$$

jer ta relacija vrijedi i za $n=0$, uz dogovor $F_{-1} = 1$.

- U algoritmu, označavamo

$$pot = \begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} F_{e-1} \\ F_e \end{bmatrix} = F^e e_1 = f^{(e-1)}$$

tj. u tom vektoru akumuliramo vektor $f^{(n-1)}$.
Pritom, e ide po nekom skupu vrijednosti kojeg nećemo posebno opisivati, a određen je binarnim prikazom broja n.

Analogno, matrica B sadrži matrice F^ℓ za neke nepoznanice ℓ (opet ih nećemo posebno specificirati - osim da je $\ell = 2^i$).

$$B = F^\ell = \begin{bmatrix} e & f \\ f & e+f \end{bmatrix}. \qquad \begin{aligned} e &= F_{\ell-1} \\ f &= F_\ell \\ e+f &= F_{\ell+1}. \end{aligned}$$

Upravo ovo je pogodna forma, jer tražimo $F^n e_1$, tj. bitan nam je prvi stupac u B. Kod akumulacije pot, lako dolazimo do drugog stupca, onda kad ga trebamo. (Tj. ne računamo jedan Fibonaccijev broj previše!).

- Ako u algoritmu POT2 pišemo vektor pot i matricu B, uz odgovarajuću inicijalizaciju, posao je gotov.

Jedini problem su operacije

$$B := sqr(B) \quad \{= B^2\}$$

i

$$pot := B * pot \quad \{\text{mora tim redom}\}$$

koje su matrično-vektorske i zahtijevale bi paralelno izvršavanje po elementima.

Zbog toga ćemo ih zapisati kao operacije po elementima, tako da rade korektno i pri sekvencijalnom izvođenju.

(a) operacija $B := B^2$ ima zapis po elementima:

$$\begin{bmatrix} e & f \\ f & e+f \end{bmatrix} := \begin{bmatrix} e & f \\ f & e+f \end{bmatrix} \cdot \begin{bmatrix} e & f \\ f & e+f \end{bmatrix}.$$

Dorađuo je; naravno, nađi nove mjesdnosti za e, f.

$$\begin{bmatrix} e & f \\ f & \cdot \end{bmatrix} := \begin{bmatrix} e^2+f^2 & 2ef+f^2 \\ \cdot & \cdot \end{bmatrix}.$$

Sekvencijalni zapis je:

```
t := sqr(f);  {pomoćna var}
f := 2*e*f + t;
e := sqr(e) + t;
```

Moramo naći prvo f, jer on koristi stari e (iuate bismo uzeli noviq - pogrešan). Novi e ne treba stari f jer imamo pomoćni $t = f^2$.

(b) analogno, operacija $pot := B \cdot pot$ ima zapis po elementima:

$$\begin{bmatrix} i \\ j \end{bmatrix} := \begin{bmatrix} e & f \\ f & e+f \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} ei + fj \\ fi + ej + fj \end{bmatrix}$$

Zajednički konad fj spremamo u pomoćnu varijablu, i opet okrećemo redoslijed, jer novi j treba oba stara (i, j), a novi i treba samo stari i:

```
t := f * j;
j := f * i + e * j + t;
i := e * i + t;
```

- Inicijalizacije su: $pot := e_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, tj. $i := 1;\ j := 0;$
  i $B := F = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$ tj. $e := 0;\ f := 1;$.

Supstitucijom u algoritam POT2 dobivamo:

Algoritam 3: F_n - brzim potenciranjem

```
function Fib (n: integer): integer;
var i, j, e, f, t: integer;

begin
  i := 1; j := 0;   { pot := 1 , = e1 }
  e := 0; f := 1;   { B := F }
  while n > 0 do
    begin
    while not odd(n) do
      begin  { B := sqr(B) }
      t := sqr(f);
      f := 2*e*f + t;
      e := sqr(e) + t;
      n := n div 2;
      end;
        { pot := B * pot }
    t := f * j;
    j := f * i + e * j + t;
    i := e * i + t;
    n := n - 1;
    end;
  Fib := j;
end; { Fib }
```

<u>Zadatak</u>: Sastavi algoritam koji računa F_n na osnovi relacije

$$f^{(n-1)} = \begin{bmatrix} F_{n-1} \\ F_n \end{bmatrix} = F^{n-1} e_2$$

brzim potenciranjem za $F^{n-1}e_2$, preko binarnog rastava broja $m = n-1$ ■

— Analiziramo kompleksnost algoritma 3.

- Inicijalizacija algoritma ocito traži konstantno vrijeme, označimo ga s c_0. Inicijalno postavljamo:

$$pot \leftarrow F^0 e_1 = \begin{bmatrix} F_{-1} \\ F_0 \end{bmatrix} \quad (= e_1)$$

$$B \leftarrow F^1 = F$$

Inicijalizacijsko vrijeme: $t_{inic} = c_0$.

- Zapišimo n u binarnom sustavu:

$$n = n_k \cdot 2^k + n_{k-1} 2^{k-1} + \dots + n_1 \cdot 2 + n_0.$$

(v. ranije).

- Svaki prolaz kroz unutarnji while kvadrira matricu B i briše zadnji znamenku broja n. Ta petlja redom računa

$$B = F^2 \leftarrow (F)^2 \quad - \text{briše } n_0 \quad (F^2 = F^{2^1})$$
$$B = F^4 \leftarrow (F^2)^2 \quad - \text{briše } n_1 \quad (F^4 = F^{2^2})$$
$$\vdots$$
$$F^{2^k} \leftarrow (F^{2^{k-1}})^2 \quad - \text{briše } n_{k-1}.$$

Vodeću znamenku n_k ne brišemo na taj način, već to kontroliramo vanjski while.

Dakle, unutarnji while računa redom matrice

$$F^{2^i}, \quad i = 1, \dots, k$$

kvadriranjem prethodne.

- Označimo s $D(i)$ vrijeme potrebno za nalaženje F^{2^i}, tj. za operaciju

$$F^{2^i} \leftarrow (F^{2^{i-1}})^2, \quad i = 1, \dots, k.$$

Dakle, vrijeme potrošeno na unutarnji while u cijelom poslu je:

$$t_{unut} = \sum_{i=1}^{k} D(i), \quad k = \lfloor \lg n \rfloor.$$

(konstante za upravljanje petljom možemo uključiti u $D(i)$).

- Donji dio vanjske while petlje se obavlja samo ako je trenutni n neparan, tj ako je zadnja znamenka $n_i = 1$ u trenutnom n.

Taj dio akumulira potencije. Ako je $n_i = 1$, za neki $i \in \{0, \dots, k\}$, taj dio programa računa

$$pot \leftarrow F^{2^i} \cdot pot.$$

Naime, n_i je zadnja znamenka trenutnog n, pa je zadnji izračunati B upravo jednak F^{2^i} (zadnja obrisana znamenka je n_{i-1}).

- Lako se pokaže da je nakon te operacije

$$pot = F^{n \bmod 2^{i+1}} \cdot e_1$$

gdje je e_1 početna vrijednost za pot.

(Na kraju je sigurno $n_k = 1$ i tada je $n \bmod 2^{k+1} = n$, jer je $n < 2^{k+1}$. To dokazuje i korektnost algoritma, tj. da je na kraju

$$pot = F^n e_1.\ )$$

- Označimo s $\in(i)$ vrijeme potrebno za računanje produkta

$$pot \leftarrow F^{2^i} \cdot pot$$

za $i \in \{0, \dots, k\}$, uz uvjet da je $n_i = 1$.

- Vrijeme potrebno za donji dio vanjske while petlje je

$$t_{van} = \sum_{\substack{i=0 \\ n_i = 1}}^{k} \in(i).$$

(Upravljanje petljom uključeno u $\in(i)$).

— Za ukupno potrebno vrijeme $T(n)$, za zadani n, izlazi

$$\boxed{T(n) = c_0 + \sum_{i=1}^{k} D(i) + \sum_{\substack{i=\emptyset \\ n_i = 1}}^{k} \in(i)}, \quad n \in \mathbb{N}_0$$

- Oba vremena $D(i), \in(i)$ uključuju dodjeljivanja, testove, skokove i slične operacije konstantnog trajanja. No, najveći dio posla otpada na nekoliko zbrajanja i množenja.

- Kako i zbrajanje i množenje smatramo operacijama čije trajanje <u>ne</u> ovisi o veličini brojeva, onda mijenjamo: Postoje konstante $d_1, d_2, e_1, e_2 > 0$, takve da je

$$d_1 \leq D(i) \leq d_2, \quad \forall i = 1, \ldots, k$$
$$e_1 \leq \in(i) \leq e_2, \quad \forall i = 0, \ldots, k$$

i to neovisne o n (jer k, i ovise o n).

- Označimo s $m(n)$ broj jedinica u binarnom zapisu broja n. Tada je, za $n > 0$

$$1 \leq m(n) \leq k+1 = \lfloor \lg n \rfloor + 1.$$

i dogovor $m(0) = 0$.

- Za kompleksnost izlazi:

$$c_0 + d_1 \cdot \underbrace{\lfloor \lg n \rfloor}_{k} + e_1 \cdot m(n) \leq T(n) \leq c_0 + d_2 \lfloor \lg n \rfloor + e_2 \cdot m(n)$$

- Uzevši u obzir ocjenu za $m(n)$, dobivamo:

$$\underbrace{c_1}_{c_0+e_1} + d_1 \lfloor \lg n \rfloor \leq T(n) \leq \underbrace{c_2}_{c_0+e_2} + (d_2 + e_2) \cdot \lfloor \lg n \rfloor$$

što dokazuje

$$T(n) = \Theta(\lfloor \lg n \rfloor).$$

Asimptotski je onda i

$$T(n) = \Theta(\log n).$$

- Trajanje algoritma 3 <u>logaritamski</u> ovisi o n, što je bitno poboljšanje obzirom na algoritam 2.
- Lako se dokazuje i $T(n) = \Theta(\lfloor \lg n \rfloor + 1)$, tj. algoritam 3 je <u>linearan</u> u duljini zadane problema FIB.
- Ovime je dokazano da je problem FIB <u>polinomno</u> rješiv, naravno, uz pretpostavku da su zbrajanje i množenje elementarne operacije.

Primjer 27. U implementaciji, kao i ranije, koristimo realni tip extended za prikaz Fibonaccijevih brojeva.

Potrebne promjene su:

```
function Fib (n: integer): real; {= extended}
```

i

```
var  I, J, E, F, T : real;
```

Ostatak algoritma ostaje isti. Na ispisu uzorku, do $n=87$, kao i u algoritmu 2, karakteristična vremena su:

F_{30} , $T(30) = 0.002\,754$
F_{60} , $T(60) = 0.003\,045$
F_{87} , $T(87) = 0.003\,661$.

Na 80286 procesoru, možemo zaključiti da trajanje pojedinih operacija ne ovisi o veličini brojeva, tj. ima konstantno trajanje. To znači da je $d_1 = d_2$ i $e_1 = e_2$, tj. vrijedi

$$D(i) = d$$
$$E(i) = e$$

gdje su $d, e > 0$.

Za kompleksnost izlazi

$$T(n) = c_0 + d \cdot \lfloor \lg n \rfloor + e \cdot m(n), \quad n \geq 0.$$

Primijetimo da $m(n)$ - broj jedinica u binarnom prikazu broja n, nije monotona funkcija. Zbog toga, ni $T(n)$ nije monotona funkcija. Iz ocjene za $m(n)$ izlazi ($1 \leq m(n) \leq \lfloor \lg n \rfloor + 1$):

$$c_0 + e + d\lfloor \lg n \rfloor \leq T(n) \leq c_0 + e + (d+e)\lfloor \lg n \rfloor .$$

- Primijenimo li pojednostavljeni model

$$T(n) = c \cdot \lg n$$

za $n \geq 2$, dobivamo da c nije konstanta, odn. ne teži prema konstanti kad $n \to \infty$. Za ilustraciju:

F_{63} , $T(63) = 0.003\,720$, $c = 6.2238 \cdot 10^{-4}$
F_{64} , $T(64) = 0.002\,269$, $c = 3.7809 \cdot 10^{-4}$ ■

U algoritmu 3, kompleksnost $T(n)$ stalno oscilira između dviju monotonih funkcija od n (ali $\lfloor \lg n \rfloor$).

- Za $m(n)$ mjerodi

$$1 \leq m(n) < \lfloor \lg n \rfloor + 1 \quad , \quad n \geq 1$$

pri čemu se donja ograda 1 dostiže za sve brojeve oblika 2^k, a gornja ograda $\lfloor \lg n \rfloor + 1$ se dostiže za brojeve oblika $2^{k+1} - 1$.

Tj. za brojeve n iste duljine $l = k+1$, kompleksnost varira - od najbolje - za 2^k, do najgore - za $2^{k+1} - 1$.

- Označimo te dvije funkcije sa:

$T_B(n)$ - najbolja kompleksnost
$T_W(n)$ - najgora kompleksnost.

Ove su, zaporavo, funkcije od $|x| = l$ - duljine zadaće, ali se lako proširuju na cijelu domenu N_0, supstitucijom. $l = \lfloor \lg n \rfloor + 1$. Dobivamo po dijelovima konstantne, monotono rastuće funkcije.

$$T_B(n) = c_1 + d_1 \cdot \lfloor \lg n \rfloor$$
$$T_W(n) = c_2 + (d_2 + e_2) \cdot \lfloor \lg n \rfloor .$$

- Uz pretpostavke $d_1 = d_2 = d$, $e_1 = e_2 = e$, izlazi

$$T_B(n) = c_0 + e + d \cdot \lfloor \lg n \rfloor$$
$$T_W(n) = c_0 + e + (d+e) \cdot \lfloor \lg n \rfloor .$$

- Asimptotski je očito (u pojednostavljenom modelu):

$$T_B(n) \sim d \cdot \lfloor \lg n \rfloor$$
$$T_W(n) \sim (d+e) \cdot \lfloor \lg n \rfloor$$

To omogućava eksperimentalno određivanje konstanti d i e.

<u>Primjer 28</u>. Trip extended ima dovoljan raspon za pokaz F_n za n do 30000. Dakle možemo raditi s brojevima

$$n = 2^k \quad \text{do} \quad k = 14 \quad , \quad n = 16384$$
$$\text{i} \quad n = 2^{k+1} - 1 \quad \text{do} \quad k = 13 \quad , \quad n = 16383$$

Dobivene vrijednosti za F_n imaju realnu točnost oko 10^{-18}, i, naravno, nisu egzaktno jednake. No, to i ne tražimo. Bitno je da koristimo iste osnovne operacije - s istim trajanjem, kao u primjeru 27.

- Za najbolju kompleksnost T_B dobivamo:

$$k = 14, \quad F_{16384} \approx 5.\emptyset5839\ 63273\ 257 \cdot 10^{3423}$$

$$T_B(16384) = \emptyset.\emptyset\emptyset4\ 8\emptyset\emptyset \text{ s}$$

i konstanta d eksperimentalno tražimo prema:

$$d = 3.4 \cdot 10^{-4} \text{ s} \quad (\text{i pada} \searrow)$$

- Za najgoru kompleksnost T_W dobivamo:

$$k = 13, \quad F_{16383} \approx 3.12626\ \emptyset8588\ 549 \cdot 10^{3423}$$

$$T_W(16383) = \emptyset.\emptyset\emptyset9\ 367 \text{ s}$$

a konstanta $d+e$ eksperimentalno tražimo prema:

$$d + e = 6.7 \cdot 10^{-4} \text{ s} \quad (\text{i vrijeme} \nearrow)$$

- Odavde možemo izračunati konstantu e

$$e = 3.3 \cdot 10^{-4} \text{ s}$$

- Time smo približno odredili sve bitne parametre kompleksnosti. Čak smo dobili i onaj trajanja unutarnje vanjske petlje i dovoljeg dijela vanjske petlje, tj. onog trajanja operacija

$$B \leftarrow \text{sgr}(B)$$

i

$$\text{pot} \leftarrow B \cdot \text{pot}$$

u vektorsko-matričnom zapisu ■

U dosadašnjoj analizi kompleksnosti algoritama 2 i 3, pretpostavljali smo da su aritmetičke operacije elementarne, tj. da njihovo trajanje ne ovisi o veličini brojeva-operanada.

- Uz tu pretpostavku, dobili smo da je algoritam 3 logaritamski u n, odnosno linearan (polinoman) u veličini zadaće - duljini zapisa broja n.
- Međutim, ta pretpostavka nije realistična. Brojevi F_n tako brzo rastu, da vrlo brzo prelaze bilo koji unaprijed raspon egzaktno prikazivih brojeva.

  Izvan tog raspona, aritmetiku moramo programski realizirati.

  Najjednostavniji pristup je prikaz brojeva poljem ili listom znamenki u nekoj odabranoj bazi. Polje je pogodno ako znamo maksimalnu veličinu brojeva u problemu.

  Da izbjegnemo kopiranje polja pri dodjeljivanju, najlakše je koristiti polje na heapu - tj. dodjeljivanje vršiti promjenom adrese - kad god to možemo.

  - U tom slučaju, jedini problem je realizacija i trajanje aritmetičkih operacija.

    Algoritam 2 zahtijeva samo jedno zbrajanje, dok algoritam 3 zahtijeva još i množenja (uključivo i kvadriranje) i množenje malom konstantom (2).

- Svi brojevi u oba algoritma su nenegativni i stalno rastu (samo + i *), što olakšava realizaciju, jer ne treba pamtiti predznak. Tj. koristimo aritmetiku velikih brojeva na $\mathbb{N}_0$.

  Aritmetiku realiziramo preko operacija na znamenkama brojeva. Zbog toga je, uz znamenke, potrebno pamtiti i duljinu broja, odn. broj znamenki - jer se on stalno javlja u graničnim petlji.

- Shematska struktura za takav nenegativni broj velike ~~točnosti~~ (ili duljine) - multiprecision (MP) je:

  MP broj = duljina - $\in \mathbb{N}_0$ (najniža potencija baze!)
  polje ili lista znamenki.

Ako je b odabrana (pokazna) baza, veliki broj n prikazujemo u normaliziranom obliku:

$$n = \sum_{i=0}^{\ell} n_i b^i \quad , \quad \ell = \lfloor \log_b n \rfloor$$

a pamtimo ℓ i niz znamenki $n_0, \dots, n_\ell$. Nulu je najlakše prikazati praznim nizom znamenki, tj $\ell = -1$ (na pr.).

- Realizacija aritmetičkih operacija je otežana - oporaba algoritme koje koristimo za računanje "na ruke" u $b = 10$
- Kompleksnost zbrajanja i množenja ovdje ovisi o veličini brojeva.

  Zbrajanje linearno ovisi o duljini brojeva, a standardni algoritam za množenje ovisi kvadratno.

Precizirajmo (dokaz sami):

Za $x \in \mathbb{N}_0$, označimo "duljinu" zapisa - najvišu potenciju baze za pokaz broja x, sa

$$\ell(x) = \lfloor \log_b x \rfloor . \qquad (\text{za 1 manje od prave duljine})$$

Neka su $x, y \in \mathbb{N}_0$ dva broja zadana opisanim normaliziranim pokazom u bazi b (≥ 2).

— Označimo sa ADD(x,y) algoritam za zbrajanje tih brojeva, a sa MUL(x,y) algoritam za množenje tih brojeva.

Tada je

$$\text{Compl}(\text{ADD}) = \Theta(\ell(x+y)).$$

Ovo je očito, jer zbrajamo znamenku po znamenku, a treba postaviti sve znamenke rezultata $x+y$.
(Ovo dozvoljava da su x i y bitno različih duljina, pa se većina vremena troši na propagiranje prijenosa. Zbog toga i Θ, a ne $\mathcal{O}$, jer konstanta varira).

- Za standardni algoritam množenja svake znamenke sa svakom, mijenjali

$$\text{Compl}(\text{MUL}) = \Theta(\ell(x) \cdot \ell(y)).$$

- Ovo asimptotski mijenja i za prave duljine brojeva (koje su za 1 veće!).
- Ako su x i y iste duljine ℓ, onda je

$$\text{Compl}(\text{ADD}) = \Theta(\ell) \quad \text{i} \quad \text{Compl}(\text{MUL}) = \Theta(\ell^2).$$

- Osim je kompleksnost zbrajanja optimalna, ti standardni algoritam je optimalan (do na konstantu).
- Za množenje to ne mjedi - postoje bolji alg.

- Analizirajmo algoritme 2 i 3 uz uvjet realizacije aritmetičkih operacija.

- Algoritam 2: Za kompleksnost smo dobili:

$$T(n) = \begin{cases} c + \sum_{k=2} D(k), & za\ n \geq 2 \\ c_0, & za\ n = 0, 1 \end{cases}$$

Vrijeme $D(k)$ uključuje konstantno vrijeme za 3 dodjeljivanja i kontrolu petlje te vrijeme potrebno za realizaciju operacije

$$F_k = F_{k-1} + F_{k-2}.$$

Za kompleksnost zbrajanja mjedi:

$$Compl\,(ADD(F_{k-1}, F_{k-2})) = \Theta(\ell(F_k)). \quad (k \to \infty)$$

Ograničimo se na gornju ogradu za $T(n)$. Ova relacija je asimptotska. Radi jednostavnosti, pretpostavimo da je $d > 0$ konstanta za koju mjedi:

$$Compl\,(ADD(F_{k-1}, F_{k-2})) \leq d \cdot \ell(F_k), \quad \forall k \geq 2$$

a ne samo asimptotski.
(Možemo čak staviti i d_1, d_2, $d_2 > 0$, tako da je

$$Compl\,(ADD(F_{k-1}, F_{k-2})) \leq d_1 + d_2$$

ali nema potrebe. U ovom obliku se potpuno analogno može provesti dokaz - aditivne konstante neće bitno utjecati na kompleksnost).

- Znamo da je $F_k = [\phi^k]$, tj.

$$\log_b F_k \approx k \cdot \underbrace{\log_b \phi}_{\text{konstanta}}.$$

- Zbog toga možemo naći konstantu $e > 0$, takvu da je

$$\ell(F_k) = \lfloor \log_b F_k \rfloor \leq e \cdot k$$

↳ približno $\log_b \phi$

Iz srega oraga izlazi da je

$$D(k) \leq c' + d \cdot e \cdot k \quad , \quad \forall k \geq 2 .$$

- Ako je $n \geq 2$, za $T(n)$ mijedi

$$T(n) \leq c + \sum_{k=2}^{n} (c' + de \cdot k) = c + c'(n-1) + d \cdot e \cdot \sum_{k=2}^{n} k$$

dodamo de. 1 i oduzmemo

$$= c - d \cdot e + c' \cdot (n-1) + d \cdot e \, \frac{n(n+1)}{2} .$$

- Time smo dokazali da je

$$T(n) = O(n^2) .$$

- Donja ograda se izvodi analogno (uz dodatak aditivnih konstanti - na pr. $\ell(F_k) \geq e_1 + e_2 \cdot k$, tako da mijedi za $\forall k \geq 2$).

- Dakle, dobivamo da je algoritam 2 kvadratni u n

$$\underline{T(n) = \Theta(n^2)} .$$

Primjer 29.

MLS (2 param.)

$$f(x) = c_1 f_1(x) + c_2 f_2(x)$$

$(x_i, y_i), i = 1, \dots, N$

$$S(c_1, c_2) = \sum [y_i - f(x_i)]^2 \to \min$$

$$(\| y - f(x) \|_2 \to \min, \quad y = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}, f(x) = \begin{bmatrix} f(x_1) \\ \vdots \\ f(x_N) \end{bmatrix}$$

$$\frac{\partial S}{\partial c_1} = 0 = -2 \cdot \sum_{i=1}^{N} (y_i - c_1 f(x_i) - c_2 f_2(x_i)) \cdot f(x_i)$$

$$\frac{\partial S}{\partial c_2} = 0 = -2 \sum_{i=1}^{N} (\quad) f_2(x_i)$$

$$s_{11} = \sum f_1^2(x_i), \quad s_{12} = \sum f_1(x_i) f_2(x_i)$$

$$s_{22} = \sum f_2^2(x_i)$$

$$t_1 = \sum y_i f_1(x_i), \quad t_2 = \sum y_i f_2(x_i)$$

$$\begin{bmatrix} s_{11} & s_{12} \\ s_{12} & s_{22} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$$

$f(x) =$

- Za kompleksnost Algoritma 3 (Fibonacci - brzim potenciranjem) dobili smo relaciju:

$$T(n) = c_0 + \underbrace{\sum_{i=1}^{k} D(i)}_{t_D} + \underbrace{\sum_{\substack{i=0 \\ n_i=1}}^{k} \in (i)}_{t_\in}, \quad \forall n \in \mathbb{N}_0.$$

Vrijeme $D(i)$ najvećim dijelom (do na aditivnu konstantu) sadrži vrijeme potrebno za operaciju

$$B \leftarrow sqr(B)$$

tj.

$$F^{2^i} \leftarrow (F^{2^{i-1}})^2.$$

Posto radimo aritmetičke operacije na elementima matrice B, treba naći duljine njenih elemenata e, f.

Za $B = F^e$ vrijedi:

$$B = \begin{bmatrix} e & f \\ f & e+f \end{bmatrix} = F^e = \begin{bmatrix} F_{e-1} & F_e \\ F_e & F_{e+1} \end{bmatrix}$$

tj. $e = F_{e-1}$, $f = F_e$.

- Na ulazu u blok umnožavanje vrijednosti petlje je $B = F^{2^{i-1}}$, tj. $\ell = 2^{i-1}$, pa je:

$$e = F_{2^{i-1}-1}, \quad f = F_{2^{i-1}}.$$

- Kao i u analizi algoritma 2, možemo naći konstantu c takvu da je

$$\ell(F_s) \leq c \cdot s, \quad \forall s \in \mathbb{N}_0.$$

Tada je, na ulazu:

$$\ell(e) \leq c \cdot (2^{i-1} - 1), \quad \ell(f) \leq c \cdot (2^{i-1}).$$

- Naš blok izgleda ovako:

```
t := sqr(f)
f := 2*e*f + t;
e := sqr(e) + t;
```

- Imamo 3 množenja velikih brojeva ($e*e$, $e*f$, $f*f$).
Za standardno množenje vrijedi:

$$\text{Compl}(MUL(x,y)) = \Theta(\ell(x) \cdot \ell(y)).$$

Radi jednostavnosti (jer je ovo asimptotska relacija), neka je $d_1 > 0$ konstanta za koju vrijedi

$$\text{Compl}(\text{MUL}(x,y)) \leq d_1 \cdot l(x) \cdot l(y)$$

za $\forall x, y \in \mathbb{N}_0$.

- Za nata 3 množenja vrijedi:

$$t_{mul_D} \leq d_1 [\, l(e) \cdot l(e) + l(e) \cdot l(f) + l(f) \cdot l(f)\,].$$

- No, e i f su bliski brojevi – po duljini, i očito je $e \leq f$ pa je i $l(e) \leq l(f)$.
- Zbog toga je lakša ocjena:

$$t_{mul_D} \leq 3 d_1 [l(f)]^2 \leq 3 d_1 [\, c\ 2^{i-1}\,]^2 =$$

$$= \underline{3 d_1 c^2 \cdot 2^{2(i-1)}}.$$

- Operaciji $2*(e*f)$ možemo realizirati kao zbrajanje (čak i ako množimo s 2 - znamenku po znamenku, kompleksnost se ne mijenja bitno)
Dakle, imamo ukupno 3 zbrajanja. Najveći rezultat zbrajanja koji se pojavljuje je izlazni f

$$f_{ize} = F_{2^i}$$

- Neka je $d_2 > 0$ konstanta za koju vrijedi

$$\text{Compl}(\text{ADD}(x,y)) \leq d_2 \cdot l(x+y)$$

za $\forall x, y \in \mathbb{N}_0$.

- Tada, za zbrajanja u umnožavanju vrijedi

$$t_{add_D} \leq 3 \cdot d_2 \cdot l(f_{ize}) \leq \underline{3 \cdot d_2 \cdot c \cdot 2^i}.$$

- Posto je $D(i) = c_D + t_{add_D} + t_{mul_D}$, (c_D ukljucuje i $n \leftarrow n \text{ div } 2$)

dobivamo ocjenu:

$$\underline{D(i) \leq c_D + 3 d_1 c^2 \cdot 2^{2(i-1)} + 3 d_2 c \cdot 2^i}$$

za $i = 1, \ldots, k$.

- Tada je:

$$t_D = \sum_{i=1}^{k} D(i) \leq c_D k + 3d_1 c_1^2 \cdot \sum_{i=1}^{k} 2^{2(i-1)} + 3d_2 c_1 \cdot \sum_{i=1}^{k} 2^i .$$

Obje sume su sume konačnih geometrijskih redova:

$$\sum_{i=1}^{k} 2^{2(i-1)} = \sum_{i=1}^{k} 4^{i-1} = \sum_{i=0}^{k-1} 4^i = \frac{4^k - 1}{4-1} = \frac{1}{3}(2^{2k} - 1)$$

$$\sum_{i=1}^{k} 2^i = 2 \cdot \sum_{i=0}^{k-1} 2^i = 2 \cdot \frac{2^k - 1}{2-1} = 2 \cdot (2^k - 1)$$

- Dakle:

$$\sum_{i=1}^{k} D(i) \leq c_D k + d_1 c_1^2 \cdot (2^{2k} - 1) + 6 d_2 c_1 (2^k - 1)$$

- Prevedimo to u termine od n:

$$k = \lfloor \lg n \rfloor$$ najveća potencija baze 2.

pa je

$$2^k \leq n < 2^{k+1}$$

što daje:

$$\sum_{i=1}^{k} D(i) \leq c_D \lfloor \lg n \rfloor + d_1 c^2 (n^2 - 1) + 6 d_2 c \cdot (n-1) .$$

To možemo napisati u obliku

$$t_D = \sum_{i=1}^{n} D(i) \leq c_D \lfloor \lg n \rfloor + a_0 + a_1 n + a_2 n^2 .$$

gdje su a_0, a_1, a_2 neke konstante.

- Vidimo da najviši član - u ovom slučaju kvadratni, direktno odražava kompleksnost množenja, jer dolazi od

$$t_{mul_1} \leq c' \cdot [l(f)]^2 = c'' \cdot 2^{2(i-1)} , \quad i = 1, \ldots, k$$

i nastaje sumacijom takvih članova.

- Linearni član odražava kompleksnost zbrajanja - jer nastaje iz t_{add_1}.

Dakle, kad bismo gledali samo zbrajanja, ovo bi bio linearan algoritam u n - što je bitno bolje od algoritma 2, koji sadrži samo zbrajanja, ali je kvadratan u n.

- Bitni dio kompleksnosti algoritma 3 dolazi od kompleksnosti množenja!!

- Potpuno analogno može se izvesti i donja ograda oblika:

$$\sum_{i=1}^{k} D(i) \geq c_D' \lfloor \lg n \rfloor + a_0' + a_1' n + a_2' n^2 .$$

- Preostaje ocijeniti drugi dio kompleksnosti

$$\sum_{\substack{i=0 \\ n_i=1}}^{k} \in (i) .$$

- Vrijeme $\in(i)$ odgovara jednoj akumulaciji potencije

$$pot \leftarrow B \cdot pot .$$

Pri tome je $B = F^{2^i}$, tj

$$e = F_{2^i - 1}$$
$$f = F_{2^i} .$$

- Na ulazu je $pot = F^{n \bmod 2^i} \cdot e_1 = \begin{bmatrix} F_{n \bmod 2^i - 1} \\ F_{n \bmod 2^i} \end{bmatrix} = \begin{bmatrix} i \\ j \end{bmatrix}$

tj.

varijabla "i" = $F_{n \bmod 2^i - 1}$

"j" = $F_{n \bmod 2^i}$.

- Na izlazu je (zbog $n_i = 1$), $pot = F^{n \bmod 2^{i+1}} e_1$, tj

"i" = $F_{n \bmod 2^{i+1} - 1}$

"j" = $F_{n \bmod 2^{i+1}}$.

Dosji dio umutanje vlule petlje ima oblik:

```
t := f * j;
j := f * i + e * j + t;
i := e * i + t
```

i sadrzi 4 umozenja i 3 zbrajanja. (Broj umozenja se moze smanjiti na 3!)

- Za umozenja, najveci operand je $f = F_{2i}$
(Odatle je $e < f$, $i < j$, a iz $n \bmod 2^i < 2^i \Rightarrow j < f$).
Vrijeme potrebno za umozenja mozemo ocijeniti sa:

$$t_{mul_\in} \leq 4 \cdot d_1 \cdot [l(f)]^2 \qquad l(f) = l(F_{2i}) \leq c \cdot 2^i$$
$$\leq \underline{4 \cdot d_1 c^2 \cdot 2^{2i}} .$$

- Za zbrajanja, najveci rezultat je izlazni j. Zbog toga, vrijeme za zbrajanja mozemo ocijeniti sa

$$t_{add_\in} \leq 3 \cdot d_2 \cdot l(j_{ize})$$
$$\leq 3 \cdot d_2 \cdot c \cdot \underbrace{n \bmod 2^{i+1}}_{< 2^{i+1}}$$
$$< \underline{3 \cdot d_2 c \, 2^{i+1}} .$$

- Vrijeme $\in(i)$ mozemo prikazati u obliku:

$$\in(i) = c_\in + t_{add_\in} + t_{mul_\in}$$

($c_\in$ ukljucuje dodjeljivanja, $n := n-1$ i operacije za petlju)

- Dobivamo gornju ogradu za $\in(i)$:

$$\in(i) < c_\in + 4 d_1 c^2 \cdot 2^{2i} + 3 d_2 c \, 2^{i+1} .$$

- Za ukupno vrijeme potrebno za akumulaciju potencija izlazi:

$$t_\in = \sum_{\substack{i=0 \\ n_i = 1}}^{k} \in(i) < c_\in \cdot m(n) + 4 d_1 c^2 \cdot \sum_{\substack{i=0 \\ n_i = 1}}^{k} 2^{2i} + 3 d_2 c \cdot \sum_{\substack{i=0 \\ n_i = 1}}^{k} 2^{i+1}$$

U najgorem slučaju, kada su sve znamenke $n_i = 1$, $i = 0, \dots, k$, tj. $n = 2^{k+1} - 1$, imamo ($m(u) = k+1$)

$$(t_\epsilon)_w \leq c_\epsilon \cdot (k+1) + 4 d_1 c^2 \cdot \frac{1}{3}\underbrace{(2^{2(k+1)} - 1)}_{(n+1)^2} + 6 d_2 c \underbrace{(2^{k+1} - 1)}_{n}$$

tj, u terminima od n, to možemo napisati kao

$$\boxed{t_\epsilon \leq (t_\epsilon)_w \leq c_\epsilon \lfloor \lg n \rfloor + b_0 + b_1 n + b_2 n^2}$$

Što je funkcija istog oblika kao i obje ocjene za t_D, tj t_ϵ neće nista bitno pokvariti.

- Donji ograde za t_ϵ, zapravo nema smisla tražiti, jer je već donja ograda za t_D istog oroj oblika, tj. najbolji t_ϵ ne može nista bitno popraviti.

- No, ipak, pogledajmo i najbolji slučaj. Tada je samo najmanja znamenka $n_k = 1$, tj. $n = 2^k$ i u sumi za t_ϵ ostaje samo zadnji član $\in(k)$

$$(t_\epsilon)_B = \in(k).$$

Za $\in(k)$ imamo donju ogradu. No, analogno se izvodi i donja ograda istog oblika, jer se obje gornje granice dubina dostižu. Moramo naći $f * f$ i izlazni j. Dakle:

$$\in(k) \geq c_\epsilon + c'_\epsilon \cdot \underbrace{2^{2k}}_{\text{iz } f \text{ i množ.}} + c''_\epsilon \cdot \underbrace{2^k}_{\text{iz } j \text{ izl. i zbraj.}}$$

a $j = F_n = F_{2^k}$

tj, u terminu od n, slijedi je:

$$t_\epsilon \geq (t_\epsilon)_B \geq c_\epsilon + b'_0 + b'_1 n + b'_2 n^2 .$$

- Jedina razlika je što donje ograde otpada logaritamski član. No, on se javlja u donjoj ogradi za t_D.

Zbrajanjem ocjena za t_D, t_E dolivamo ovaj zaključak za kompleksnost algoritma 3:

Postoje konstante c_e, c_0, c_1, c_2 i c_e', c_0', c_1', c_2', uz $c_2, c_2' > 0$, takve da je:

$$c_0' + c_e' \lg n + c_1' n + c_2' n^2 \leq T(n) \leq c_0 + c_e \lg n + c_1 n + c_2 n^2$$

$$\text{za } \forall n \in \mathbb{N}_0$$

Dakle: $\boxed{T(n) = \Theta(n^2)}$

Pri tome, kvadratni član potiče direktno od kompleksnosti množenja.

Zbrajanje doprinosi najviše linearni član.

- Dakle, možemo zaključiti da se isplati tražiti bolji algoritam za množenje velikih brojeva.

Pretpostavimo da imamo algoritam MUL za množenje velikih brojeva, za čiju kompleksnost vrijedi (bar za brojeve istih duljina)

$$\text{Compl}(\text{MUL}(x,x)) = \Theta([l(x)]^\alpha), \text{ za } 1 \leq \alpha \leq 2.$$

Iz analize kompleksnosti algoritma 3, direktno izlazi da je tada i vodeći član oblika n^α, tj.

$$\boxed{T(n) = \Theta(n^\alpha)}.$$

- Može se dokazati da je $\alpha \geq 1$, što je očito, jer moramo naći barem $\sim 2l(x)$ znamenki rezultata, koje su nezavisne.

Tj. kompleksnost množenja brojem dominira u kompleksnosti alg. 3.

- Najbolji poznati alg. za množenje velikih brojeva ima kompleksnost

$$\text{Compl}(\text{MUL}(x,x)) = \Theta(n \log n \log\log n).$$

gdje je $n = l(x)$.

(Strassen - Schönhage).