Za kraj priče o cjelobrojnim algoritmima, evo odgovora na davno spomenuti problem:

Zadan je (recimo) prirodni broj n. Treba provjeriti da li je n potencija od 2.

Što znači "potencija od 2"?
To znači da je n oblika

$$n = 2^k$$

za neki $k \geq 0$, k cijeli broj.
Ovdje dozvoljavam i $k = 0$, tj. $n = 1 = 2^0$ je potencija od 2 (i potencija od bilo kojeg drugog broja - prirodnog).

Sad znamo što znači "potencija od 2", pa idemo na konstrukciju algoritma.

Za početak, zaboravite sve "izlete" u realnu aritmetiku, poput:

"$\log_2 n$ je cijeli broj",

zbog grešaka zaokruživanja.

- Pravi, "lijepi" algoritam radi samo s prirodnim brojevima.

Ideja je slična onoj kod brojanja znamenki, samo što ćemo ovdje "izbrojati" dvice koje su faktori od n.

Broj n sigurno mogu zapisati u obliku

$$n = 2^k \cdot \text{nešto}$$

s tim da "nešto" ne smije biti djeljiv s 2 (paran).
Tada je k = eksponent od dvice jednoznačno određen (sjetite se rastava broja na faktore, pa onda na proste faktore).

Dakle, strogo matematički: (dokaz = ?!)

- za svaki prirodni broj n postoje jednoznačno odreođeni (jedinstveni) brojevi
  $k \in \mathbb{N}_0 = \mathbb{N} \cup \{\emptyset\}$
  $\ell \in \mathbb{N}$, ℓ nije djeljiv s 2

takvi da je $n = 2^k \cdot \ell$.

I sad je lako:
n je potencija od 2, ako i samo ako je $\ell = 1$.

Algoritam se svodi na traženje $\underline{\ell}$, a to ide ovako:

sve dok je n paran (djeljiv s 2)
podijeli ga s 2;
rezultat je ℓ
i pogledamo da li je $\ell = 1$.

Ovaj algoritam "uništava" n, pa ako nam n kasnije treba (recimo za ispis), zgodno ga je spremiti u nešto privremeno.
Idealna oznaka (ime) za taj privremeni objekt je upravo ℓ.

{u C-u mod ⇔ %, div ⇔ /}

Algoritam: učitaj n
$\ell = n$

ovo "briše" faktore 2 iz n:
sve dok je $\ell \bmod 2 = 0$ ponavljaj
[$\ell = \ell \text{ div } 2$

odgovor = ($\ell = 1$) ... logička vrijednost {u C-u logičke su $\emptyset$, 1}

ili: ako je $\ell = 1$ onda
napiši "da, n je potencija od 2"
inače
napiši "ne, n nije potencija od 2".

Ako još želite, možemo naći i eksponent k iz gornjeg rastava!
Dodati: $k = \emptyset$ ispred petlje
i $k = k + 1$ u petlju (broji "dvice")

Par komentara i pitanja:

- Da li algoritam radi "uvijek" – tj. za svaki prikazivi (pozitivni) cijeli broj n?
  (DA, zašto?)
- Da li isti princip (algoritam) radi ako broj 2 zamijenimo bilo kojim drugim (prikazivim) brojem $m \in \mathbb{N}$, uz $m > 1$.
  (Što bi se dogodilo da uzmem $m = 1$?
  → BESKONAČNA PETLJA!
  Matematički razlog: $n = 1^k \cdot \ell$
  k nije jedinstven, već može biti bilo koji!)

OBRADA NIZOVA PODATAKA

Niz podataka = složeni tip podataka, sastavljen od konačnog broja ($n \geq 0$) podataka istog tipa.

Na pr: niz x s n podataka

$$x_1, x_2, \ldots, x_n.$$

Pojam – ili struktura "niz" – odgovara (matematički) uređenoj n-torci $(x_1, x_2, \ldots, x_n)$, s tim da svi "članovi" (elementi, komponente) $x_1, \ldots, x_n$ pripadaju istoj domeni, tj. imaju isti tip.
(tj. $x_i \in T$, $i = 1, \ldots, n$ i onda $X \in T^n$).

- Napomena: broj n članova niza je, u principu, zadan.
Obično smatramo da je n fiksan (ali može biti $n = \emptyset$, tako da imamo prazan niz)
- Najbliži pojam u matematici je vektor (oznake su vrlo slične) – s tim da ovi ne trebaju nikakve operacije za komponente (aksiomi vekt. prostora).
- Dakle, imam uređeni niz (n-torku) podataka ISTOG tipa.

$$x_1, x_2, \ldots, x_n.$$

Obzirom da su svi podaci istog tipa, pristup pojedinim podacima ide preko indeksa,

$$x_i = i\text{-ti član niza}.$$

- Standardno je $i \in \{1, 2, \ldots, n\}$ i nije "zgodno" izaći izvan ovog skupa (tj. tražiti članove kojih nema)
- U C-u se standardno uzima

$$i \in \{0, 1, \ldots, n-1\}.$$

Razlog = računanje adresa!

Kako se niz sprema u memoriji?

- Uređaj među podacima (članovima) je FIZIČKI, tj. članovi niza su spremljeni jedan za drugim (redom) u memoriji

x_1 (x_0 u C-u)
x_2
$\vdots$
x_n (x_{n-1} u C-u)

- ISTI TIP za sve x_i $\Rightarrow$ svaka "kućica" $\boxed{x_i}$ zauzima istu količinu prostora.
- Zbog toga se NE pamte adrese svih članova niza, već se pamti samo:
  - adresa prvog člana niza (tzv. početna adresa cijelog niza)

    $\longrightarrow \boxed{x_1}$ (x_0 u C-u)

  - adresiranje ostalih (ili svih!) članova niza ide preko: ove početne adrese i indeksa.

Dakle: adresa od x_i $\longrightarrow \boxed{x_i}$

je: početna adresa niza + $(i-1) *$ veličina svakog člana
(tj. ~~člana~~ prvog člana x_1)

i ovo je za $i = 1, \ldots, n$,

a u C-u indeksi idu $i = 0, \ldots, n-1$, pa je:

adresa od x_i = početna adresa niza (prvog člana x_0) + $i *$ veličina svakog člana

zbog ovog i brojanje članova u C-u ide od $\varnothing$ (lakše – brže računanje adresa)

- Jedina stvarna operacija s nizom (kao složenom strukturom podataka) je:
  - pristup pojedinom članu x_i

(meni će ⓘ šetati od 1 do n, po ugledu na vektore u matematici, a na "C-u" ću pisati od ∅ do n-1)

- Napomena: taj pristup se bazira na fizičkom uređaju podataka - članova u memoriji i ovom pravilu za adresiranje članova, tj.

član x_i = ono što se nalazi na odgovarajućem bloku adresa
(veličina = prostor za član - recimo, 4 byte-a za cijeli broj
startna adresa = po onom pravilu, i-1 članova iza prvog !!!)

[zato NE SMIJEMO s indeksom ⓘ izaći IZVAN dozvoljenog skupa → "BRLJAMO PO MEMORIJI"]

- Sve ostalo što s nizovima možemo raditi - ovisi o tipu članova (tj. "operacije po komponentama").

- Za nizove po sebi - jedina operacija = pristup članu, možemo "samo": pristupati članovima
mijenjati njihov poredak.

~~Ako je skup strani ... bazirane ... osnovne~~
Uz pristup članu - možemo testirati njegovu vrijednost (operacije =, ≠).

- Ako osnovni podaci imaju neku relaciju uređaja (na onom skupu T), onda možemo koristiti SVE operacije uspoređivanja (tj. i <, ≤, >, ≥).

- I baš to nam je dovoljno za dva osnovna algoritma (ili 2 klase algoritama) na nizovima:
  - pretraživanje nizova = provjera da li je neki objekt član niza ili ne,
  - uređivanje ili sortiranje nizova = dovođenje članova u "uređeni" poredak (rastući ili padajući)

— Uočiti: petlje "po indeksima" su idealan način za obradu nizova ("for")

— Operacije u petlji:
  — uspoređivanje članova ($=, \neq$ i ostalo, ako ima uređaj!)
  — ostale operacije "po članovima", kad postoje.

— Na primjer — par "trivijalnih" algoritama za nizove BROJEVA (brojevi — osnovni podaci u računalu i za njih imam aritmetičke operacije i usporedbe)

(1) Zbroj svih članova niza

Imam (učitan!!) niz $x_1, \dots, x_n$ od $n\ (\in \mathbb{N})$ brojeva (svejedno — cijelih ili realnih ili "nekih drugih").

Tražim: $S = \sum_{i=1}^{n} x_i$

Algoritam:
$S = \emptyset$
za $i=1$ do n ponavljaj
  [$S = S + x_i$

(Usporediti sa zbrajanjem znamenki broja od prošlog puta).

Napomena: koristim da se petlja ne izvršava ako je $n \leq 0$ (iako sam pretpostavio da je $n \in \mathbb{N}$ na početku).

Ako baš moram "biti picajzla", pa neću odgovor $S = \emptyset$ kad je $n = \emptyset$ (tj. S mi je uvijek definiran!) onda ovako:

ako je $n > \emptyset$ onda
  [$S = \emptyset$
    za $i=1$ do n ponavljaj
      [$S = S + x_i$
    napiši S
inače
  napiši poruku "prazan niz"

{ $S = x_1$
  za $i=2$ do n ...
  $S = S + x_i$
može i ovo

- Produkt članova niza ide, naravno, po istom principu ($P=1$, ... – sjetite se priče o inicijalizacijama!)

(2) Najveći, najmanji član niza

Tu NEMA "vrdanja" s inicijalizacijom –
– min, max NISU definirani na PRAZNOM skupu.

Dakle, ovako:

Algoritam: (recimo za $\min(x_1, \ldots, x_n)$)

ako je $n \geq 1$ onda (ili $n > \emptyset$ – što vam draže)
  [min = x_1; poz = 1;
  za $i = 2$ do n ponavljaj
    [ako je $x_i <$ min onda
      [min = x_i; poz = i;
  napiši "najmanji član je", min, "na mjestu", poz
inače
  napiši "najmanji nije definiran".

- Uočiti – ovdje tražimo i vrijednost najmanjeg

$$\underset{\sim\sim}{\min} = \min_{1 \leq i \leq n} \{x_i\}$$

i njegovu poziciju = indeks – na kojem se (prvi puta) nalazi

$$\underset{\sim\sim}{poz} = \min \{ i \mid x_i = \underset{\sim\sim}{\min} \}$$

$i \in \{1, \ldots, n\}$

- Ovaj algoritam ćemo iskoristiti kod sortiranja!
- Trajanje? (što brojim? – elementarne operacije su usporedbe, indeksiranje, pristup članu)

ukratko, MORAM proći cijeli niz (i to JEDNOM, ne VIŠE puta)

tj. trajanje je "proporcionalno" s duljinom (n) niza (linearno u n). Zapis: $T(n) = O(n)$

trajanje   veliko O.

Ovdje nije bilo većih problema s inicijalizacijom za min, čim je niz neprazan — bilo koji član je "dobar" za inicijalizaciju.

Ali, nije uvijek tako.

(3) "Ekstrem uz uvjet" (tzv. uvjetni ekstrem)

Zadan je niz od n cijelih brojeva (može s predznakom). Naći najmanji PARNI član niza, ako takav postoji.

Naravno, ne valja — naći najmanji, pa pogledaj da li je paran.

Treba obratno — ići samo po parnima ("podniz") i tražiti najmanji.

Za početak, zabranjujem kopiranje podniza parnih u novi niz, recimo, y

$$y = \{ x_i \mid x_i \text{ paran}, i = 1, \ldots, n \}$$

pa onda nalaženje minimuma u y.

Probajte — i pazite na duljinu — broj članova u y.

Razlog zabrane — to predugo traje:

- proći cijeli x (n članova)
- proći cijeli y (m članova, $0 \leq m \leq n$)

i još se razbacuje memorijom.

— Dakle, hoću rješenje u JEDNOM PROLAZU kroz x.

Gruba skica algoritma:

proći kroz cijeli x niz — redom po svim članovima ili indeksima — nekim redom i radi:

[nađi prvi parni (bilo koji), ako ga ima;
ako ga nema, onda
  [napiši — nema parnih, pa ni najmanjeg parnog
inače
  [inicijaliziraj najmanji parni na prvog;
  prođi ostatak niza, gledaj parne i "spuštaj" minimum;
  napiši najmanjeg

Većinu ovih operacija nije teško napraviti, ali ključna stvar za korektnost algoritma je onaj prvi korak:

[naći neki (prvi, bilo koji) parni, ako ga ima.

Bez ovog nema korektne inicijalizacije najmanjeg.

– I tako stigosmo do prvog ključnog algoritma na nizovima, a to je

TRAŽENJE ili PRETRAŽIVANJE.

– U općem obliku, problem pretraživanja glasi:

Zadan je niz $x_1, \ldots, x_n$ bilo kakvih objekata (istog tipa) i još jedan objekt a (tog istog tipa)

Tražim odgovor na pitanje da li je

$$a \in \{x_1, \ldots, x_n\}$$

tj. da li se a (kao vrijednost) pojavljuje među članovima niza (poredak članova, trenutno, nije bitan), ili: da li POSTOJI član x_i niza koji je jednak a.

Matematički rečeno, pitam da li postoji INDEKS

$$i \in \{1, \ldots, n\}$$

takav da za pripadni član vrijedi ~~[illegible]~~

$$x_i = a.$$

Dodatno, može me zanimati i vrijednost tog indeksa i tj. na kojem mjestu se nalazi član jednak a ~~[illegible]~~ (bar jedno, ako ga ima, ili sva mjesta).

Za početak, hoću samo odgovor DA/NE, IMA/NEMA

JESI ČLAN je ISTINA (1)

ili LAŽ (Ø).

Dakle, tražim "logičku" vrijednost kao rezultat pretrage.

Ključna riječ za razvoj algoritma pretrage je

<u>POSTOJI</u> (i).

Dakle, odgovor je DA (ISTINA) ako (i samo ako) je

$x_i = a$ za barem jedan indeks $i \in \{1, \dots, n\}$,

ili, jednostavnije rečeno, ako je

$$\underset{\substack{\uparrow\\ \text{na prvom}\\ \text{mjestu}}}{(x_1 = a)} \text{ ili } \underset{\substack{\uparrow\\ \text{na drugom}\\ \text{mjestu}}}{(x_2 = a)} \text{ ili } \dots \text{ ili } \underset{\substack{\uparrow\\ \text{na n-tom (zadnjem)}\\ \text{mjestu.}}}{(x_n = a)}$$

Dugački zapis, ali sad imamo logički izraz kojeg naprosto treba IZRAČUNATI – po istom principu kao i, recimo, ZBROJ svih članova niza.

- Uvedem logičku varijablu CLAN koja će (na kraju) sadržavati traženi odgovor

$$\text{CLAN} = (x_1 = a) \text{ ili } (x_2 = a) \text{ ili} \dots \text{ ili } (x_n = a)$$

- I sad, po ugledu na algoritam za zbrajanje

```
S = x1
za i = 2 do n ponavljaj
[ S = S + xi
```

imam algoritam za pretraživanje:

```
CLAN = (x1 = a)
za i = 2 do n ponavljaj
[ CLAN = CLAN or (xi = a)
```

već sam te našao (u prvih i−1) <u>ili</u> sam te našao na i-tom mjestu.

Radi!

A sad idemo to malo dotjerati!

- Po ugledu na klasičnu inicijalizaciju zbroja na neutral

$S = \emptyset$
za $i = 1$ do n ponavljaj
$S = S + x_i$

gdje $S = \emptyset$ "glumi" zbroj praznog niza, pitanje je:
Kako treba inicijalizirati CLAN, pa da analogni algoritam radi dobro?

Odgovor: na odgovor za praznan niz,
a to znači: a NIJE član praznog niza,
dakle: CLAN = laž (ili $\emptyset$ u C-u)

<u>Algoritam:</u> CLAN = laž;
za $i = 1$ do n ponavljaj
[CLAN = CLAN <u>or</u> ($x_i = a$);

Što se događa ako stavim CLAN = istina (1) na početku?
($\in$, pa <u>ne</u> možete promašiti!)

- Ovaj algoritam ima jednu manu, ako me zanima samo odgovor CLAN je istina ili laž:
  - prolazi čitav niz, uvijek – skroz do kraja

iako je, možda, već na početku ~~je~~ našao član (recimo, na trećem mjestu).

- Drugim riječima:
  - čim CLAN postane istina (našao sam ga!) NE MORAM dalje tražiti $\Rightarrow$ mogu VAN IZ PETLJE.
  - jedino ako ga NEMA (ili je na zadnjem mjestu) onda moram kroz cijeli niz.
- Dakle, želim ~~*~~ UBRZATI ovaj algoritam tako da prestane tražiti čim nađe član!

Tako dolazim do klasičnog algoritma pretrage "NA UVJET":

traži sve dok ne nađeš!

- Ali oprez – u principu NIJE jedan uvjet
  "pohlepa" = traži dok ne nađeš,

osim ako unaprijed nismo SIGURNI da ćemo ga naći!

- Ako se može dogoditi da ga ne nađemo, onda imamo 2 uvjeta

traži dok: ne nađeš
ⓘ ima smisla tražiti
(tj. ima gdje tražiti
tj. još NISMO pogledali sve članove)

- Realizacija – indekse "šetamo" sami:

$i = 1$ (prvi za pretragu)
CLAN = laž;
sve dok $(\underbrace{\text{not}}_{\text{nije}}\ \text{CLAN})\ \underbrace{\text{and}}_{i}\ (i \leq n)$ ponavljaj

$\text{CLAN} = \underbrace{\text{CLAN or}}_{\text{mogu izbaciti}}\ (x_i = a);$
$i = i + 1;$ ← makni se na sljedeći član

ili

$i = \emptyset;$ { ispred prvog / sljedećeg = zadnji pretraženi! }
CLAN = laž,
sve dok (not CLAN) and $(i < n)$ ponavljaj
$i = i + 1;$ ← makni se na sljedećeg
$\text{CLAN} = (x_i = a);$

- Prosječan broj prolaza kroz petlju je $\frac{n}{2}$ (za "slučajne podatke")