

# *Oblikovanje i analiza algoritama*

## *5. predavanje*

Saša Singer

`singer@math.hr`

`web.math.pmf.unizg.hr/~singer`

PMF – Matematički odsjek, Zagreb

# Sadržaj predavanja

- Primjer “sporog” algoritma — Hanojski tornjevi:
  - Jednostavni model složenosti — broj poteza.
  - Hanojski tornjevi — razne varijante (Pascal).
  - Hanojski tornjevi — razne varijante (C).
  - Prošireni model složenosti.

# Informacije — web stranica

Moja web stranica za Oblikovanje i analizu algoritama je

<https://web.math.pmf.unizg.hr/~singer/oaa/>

ili, skraćeno

<https://web.math.hr/~singer/oaa/>

Kopija je na adresi

<http://degiorgi.math.hr/~singer/oaa/>

Službena web stranica za Oblikovanje i analizu algoritama je

<https://web.math.pmf.unizg.hr/nastava/oaa/>

# Osnovni model složenosti

# Osnovni model = broj osnovnih poteza

Jednostavni model (vremenske) složenosti Hanojskih tornjeva dobivamo tako da gledamo samo

- broj osnovnih poteza = prebacivanja po jednog diska, tj.,
- broj poziva funkcije prebaci\_jednog.

Dakle, model za složenost, u ovisnosti o broju diskova  $n$ , je

$$t_n := \text{broj poteza za } n \text{ diskova} = h_n.$$

Pripadna rekurzija za  $t_n$  je ista kao i za  $h_n$  (v. Prog2)

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

## Oblik rekurzije za broj poteza

Za početak, ovo je **nehomogena** rekurzija, sa zadanim (**jednim**) **početnim** uvjetom  $t_0 = 0$ .

Pripadna **homogena** rekurzija ima oblik, kao u (1),

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

= **linearna** rekurzija **prvog** reda, s **konstantnim** koeficijentima.

**Nehomogeni** član u rekurziji ima “**standardni**” oblik, kao u (4),

$$g(n) = b^n p_d(n),$$

uz

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

Dakle, treba riješiti “**standardnu**” **nehomogenu** rekurziju **prvog** reda, s **konstantnim** koeficijentima.

## Rješenje rekurzije za broj poteza

Rješenje. Karakteristična jednažba homogenizirane rekurzije je

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

s korijenima  $r_1 = 1$  i  $r_2 = 2$ . Opće rješenje homogenizirane rekurzije je

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

Ovo rješenje uvrstimo u polaznu rekurziju, zato da dobijemo konstantu  $c_1$  uz onaj dio rješenja koji odgovara nehomogenom članu. Izlazi

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

$$c_1 = 2c_1 + 1$$

$$c_1 = -1.$$

## Rješenje rekurzije za broj poteza

Dakle, opće rješenje **polazne nehomogene** rekurzije je

$$t_n = c_2 \cdot 2^n - 1, \quad n \geq 0,$$

a konstanta  $c_2$  se računa iz **početnog uvjeta**  $t_0$ .

Početni uvjet je  $t_0 = 0$ , pa je

$$t_0 = c_2 \cdot 2^0 - 1 = 0 \quad \implies \quad c_2 = 1.$$

Konačno rješenje za **broj poteza** u Hanojskim tornjevima s  $n$  diskova je

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



## Model — trajanje jednog poteza

Broj poteza za  $n$  diskova:

$$F(n) = 2^n - 1.$$

Ako je izmjereno vrijeme  $T(n)$ , onda je trajanje jednog poteza

$$c(n) = \frac{T(n)}{F(n)} = \frac{T(n)}{2^n - 1}.$$

Uočiti:  $c(n)$  je obratno proporcionalan brzini poteza (= recipročna vrijednost brzine).

# Hanojski tornjevi u Pascalu

## *Hanoi — ispis na ekran*

```
procedure Hanoi ( n, i, j : integer ) ;  
  
begin  
  
    if n > 0 then  
        begin  
            Hanoi ( n - 1, i, 6 - i - j ) ;  
            writeln ( i, ' -> ', j ) ;  
            Hanoi ( n - 1, 6 - i - j, j ) ;  
        end ; { n > 0 }  
  
end ; { Hanoi }
```

## *Hanoi — ispis na **disk** (u datoteku Moves)*

```
procedure Hanoi ( n, i, j : integer ) ;  
  
begin  
  
    if n > 0 then  
        begin  
            Hanoi ( n - 1, i, 6 - i - j ) ;  
            writeln ( Moves, i, ' -> ', j ) ;  
            Hanoi ( n - 1, 6 - i - j, j ) ;  
        end ; { n > 0 }  
  
end ; { Hanoi }
```

# Hanoi — opis varijanti potprograma

Za “normalnije” mjerjenje vremena (i brzine) — bez ispisa, testiramo  $2 \times 2 = 4$  varijante potprograma Hanoi.

## Dubina rekurzije:

- varijanta 0 — ima pozive za  $n = 0$ , koji ništa ne rade,
- varijanta 1 — nema pozive za  $n = 0$ , s posebnim testom za  $n = 1$ .

## Broj argumenata:

- obična varijanta — ima samo dva štapa kao argumente (odakle, kamo) i računa pomoćni stap,
- varijanta a — ima sva tri štapa kao argumente (“vrti” ih, bez računa).

## *Hanoi — varijanta 0*

```
procedure Hanoi ( n, i, j : integer ) ;  
  
begin  
  
    if n > 0 then  
        begin  
            Hanoi ( n - 1, i, 6 - i - j ) ;  
            Prebaci ( i, j ) ;  
            Hanoi ( n - 1, 6 - i - j, j ) ;  
        end ; { n > 0 }  
  
end ; { Hanoi }
```

## *Hanoi — varijanta 1*

```
procedure Hanoi ( n, i, j : integer ) ;  
  
begin  
  
    if n <= 1 then  
        Prebaci ( i, j )  
    else  
        begin  
            Hanoi ( n - 1, i, 6 - i - j ) ;  
            Prebaci ( i, j ) ;  
            Hanoi ( n - 1, 6 - i - j, j ) ;  
        end ; { n > 0 }  
  
end ; { Hanoi }
```

## *Hanoi — varijanta a0*

---

```
procedure Hanoi ( n, i, j, k : integer ) ;  
  
begin  
  
    if n > 0 then  
        begin  
            Hanoi ( n - 1, i, k, j ) ;  
            Prebaci ( i, j ) ;  
            Hanoi ( n - 1, k, j, i ) ;  
        end ; { n > 0 }  
  
end ; { Hanoi }
```

---



## *Hanoi — varijanta a1*

```
procedure Hanoi ( n, i, j, k : integer ) ;  
  
begin  
  
    if n <= 1 then  
        Prebaci ( i, j )  
    else  
        begin  
            Hanoi ( n - 1, i, k, j ) ;  
            Prebaci ( i, j ) ;  
            Hanoi ( n - 1, k, j, i ) ;  
        end ; { n > 0 }  
  
end ; { Hanoi }
```

## Potprogram Prebaci

Potprogram **Prebaci** samo **zbraja** poteze (globalni brojač):

---

```
procedure Prebaci ( i, j : integer ) ;
```

```
begin
```

```
    { Povecaj globalni brojac poteza.
```

```
    Trajanje je (skoro) konstantno:
```

```
        T(Prebaci) = c.}
```

```
    Broj_poteza := Broj_poteza + 1 ;
```

```
end ; { Prebaci }
```

---

# Sudionici (žrtve) eksperimenta

Test je napravljen 2003. godine, s 2 Pascal kompajlera:

- Turbo Pascal 7 — generira 16-bitni kôd za obični DOS.
- Free Pascal 1.0.4 — generira 32-bitni kôd za tzv. DOS-ekstender GO32v2.

Izvršavanje je kroz DOS prozor na Windowsima (2000, XP).

Računala u eksperimentu su:

- Pentium na 166 MHz, zvani P120\_166, rođen 1996., (ranije je imao Pentium na 120 MHz);
- Pentium 2 na 333 MHz, zvani Klamath, rođen 1998., (kasnije je imao Pentium 3 na 500 MHz);
- Pentium 4/660 na 3.6 GHz, zvan(a) BabyBlue, rođen(a) 2005., test iz 2006. godine (isti .exe kao prije);

## TP7 — Tablica izmjerenih vremena $T(n)$

Usporedba izmjerenih vremena  $T(n)$  (u s) za razna računala i razne varijante:

Varijanta	P120_166	Klamath	BabyBlue
ekran, $n = 15$	47.04	64.43	0.84
disk, $n = 15$	0.35	0.43	0.05
0, $n = 30$	694.82	747.91	305.82
1, $n = 30$	480.39	495.98	202.58
a0, $n = 30$	679.74	757.03	305.16
a1, $n = 30$	465.31	499.48	202.28

Uočiti: Zelena vremena za varijantu 1 (bez poziva za  $n = 0$ ) su oko  $2/3$  vremena za varijantu 0.

## TP7 — Tablica trajanja poteza za $n = 30$

Usporedba trajanja jednog poteza  $c(n)$  (u s) za razna računala i razne varijante:

Varijanta	P120_166	Klamath	BabyBlue
ekran, $n = 15$	$1.44 \cdot 10^{-3}$	$1.97 \cdot 10^{-3}$	$2.55 \cdot 10^{-5}$
disk, $n = 15$	$1.06 \cdot 10^{-5}$	$1.31 \cdot 10^{-5}$	$1.55 \cdot 10^{-6}$
0, $n = 30$	$6.47 \cdot 10^{-7}$	$6.97 \cdot 10^{-7}$	$2.85 \cdot 10^{-7}$
1, $n = 30$	$4.47 \cdot 10^{-7}$	$4.62 \cdot 10^{-7}$	$1.89 \cdot 10^{-7}$
a0, $n = 30$	$6.33 \cdot 10^{-7}$	$7.05 \cdot 10^{-7}$	$2.84 \cdot 10^{-7}$
a1, $n = 30$	$4.33 \cdot 10^{-7}$	$4.65 \cdot 10^{-7}$	$1.88 \cdot 10^{-7}$

## FPC 1.0.4 — Tablica izmjerenih vremena $T(n)$

Usporedba izmjerenih vremena  $T(n)$  (u s) za razna računala i razne varijante:

Varijanta	P120_166	Klamath	BabyBlue
ekran, $n = 15$	48.53	65.04	0.89
disk, $n = 15$	0.43	0.41	0.02
0, $n = 30$	453.34	193.76	19.12
1, $n = 30$	297.91	160.75	23.43
a0, $n = 30$	466.29	175.76	16.79
a1, $n = 30$	277.67	155.67	22.58

Grubo objašnjenje zelenih i crvenih rezultata za varijantu 1 (bez poziva za  $n = 0$ ) ide malo kasnije!

## FPC 1.0.4 — Tablica trajanja poteza za $n = 30$

Usporedba trajanja jednog poteza  $c(n)$  (u s) za razna računala i razne varijante:

Varijanta	P120_166	Klamath	BabyBlue
ekran, $n = 15$	$1.48 \cdot 10^{-3}$	$1.98 \cdot 10^{-3}$	$2.70 \cdot 10^{-5}$
disk, $n = 15$	$1.30 \cdot 10^{-5}$	$1.25 \cdot 10^{-5}$	$6.11 \cdot 10^{-7}$
0, $n = 30$	$4.22 \cdot 10^{-7}$	$1.80 \cdot 10^{-7}$	$1.78 \cdot 10^{-8}$
1, $n = 30$	$2.77 \cdot 10^{-7}$	$1.50 \cdot 10^{-7}$	$2.18 \cdot 10^{-8}$
a0, $n = 30$	$4.34 \cdot 10^{-7}$	$1.64 \cdot 10^{-7}$	$1.56 \cdot 10^{-8}$
a1, $n = 30$	$2.59 \cdot 10^{-7}$	$1.45 \cdot 10^{-7}$	$2.10 \cdot 10^{-8}$

# Objašnjenje rezultata za varijante 0 i 1

Free Pascal 1.0.4 generira loš 32-bitni kôd. Izvršavanje tog kôda (i to kroz DOS-ekstender)

- je sve sporije na modernim 32-bitnim procesorima.

Do Pentium 4—Northwood procesora ponašanje je zeleno,

- isplati se izbaciti pozive za  $n = 0$ .

Od Pentium 4—Prescott procesora ponašanje je crveno,

- ne isplati se izbaciti pozive za  $n = 0$ .

Napomena. Rezultate ne treba koristiti za usporedbu računala. Posebno to vrijedi za TP7, jer

- izvršavanje 16-bitnog kôda postaje sve sporije.



# Hanojski tornjevi u C-u

# Hanoi — početak i štoperica

```
#include <stdio.h>
#include <time.h>

/* Hanojski tornjevi s n diskova.
   Broji poteze i mjeri vrijeme. */

int nmax = 30;
long int broj_poteza;

double dsecnd (void) {
    return (double)( clock( ) ) / CLOCKS_PER_SEC;
}
```

## *Hanoi* — funkcija `prebaci_jednog`

Funkcija `prebaci_jednog` samo **zbraja** poteze:

---

```
void prebaci_jednog(int odakle, int kamo)
{
    /* Umjesto pisanja poteza:
       printf("  %d -> %d\n", odakle, kamo);
       samo povecava globalni brojac poteza. */

    ++broj_poteza;

    return;
}
```

---

# Hanoi — opis varijanti potprograma

Za razumno **mjerenje vremena** (i brzine) — **bez ispisa**, testiramo  $2 \times 2 = 4$  varijante funkcije **Hanoi**.

## Dubina rekurzije:

- varijanta **0** — **ima** pozive za  $n = 0$ , koji **ništa** ne rade,
- varijanta **1** — **nema** pozive za  $n = 0$ , s posebnim **testom** za  $n = 1$ .

## Broj argumenata:

- **obična** varijanta — ima samo **dva** štapa kao argumente (odakle, kamo) i **računa** pomoćni štap,
- varijanta **a** — ima sva **tri** štapa kao argumente (“vrti” ih, **bez** računa).

## Hanoi — varijanta 0

```
void Hanoi(int n, int odakle, int kamo)
{
    if (n > 0) {
        Hanoi(n - 1, odakle, 6 - odakle - kamo);
        prebaci_jednog(odakle, kamo);
        Hanoi(n - 1, 6 - odakle - kamo, kamo);
    }

    return;
}
```

# Hanoi — varijanta 1

```
void Hanoi(int n, int odakle, int kamo)
{
    if (n <= 1)
        prebaci_jednog(odakle, kamo);
    else {
        Hanoi(n - 1, odakle, 6 - odakle - kamo);
        prebaci_jednog(odakle, kamo);
        Hanoi(n - 1, 6 - odakle - kamo, kamo);
    }

    return;
}
```

## Hanoi — varijanta a0

```
void Hanoi(int n, int odakle, int kamo, int pomocni)
{
    if (n > 0) {
        Hanoi(n - 1, odakle, pomocni, kamo);
        prebaci_jednog(odakle, kamo);
        Hanoi(n - 1, pomocni, kamo, odakle);
    }

    return;
}
```

## Hanoi — varijanta *a1*

```
void Hanoi(int n, int odakle, int kamo, int pomocni)
{
    if (n <= 1)
        prebaci_jednog(odakle, kamo);
    else {
        Hanoi(n - 1, odakle, pomocni, kamo);
        prebaci_jednog(odakle, kamo);
        Hanoi(n - 1, pomocni, kamo, odakle);
    }

    return;
}
```



## Hanoi — funkcija main za 0, 1

```
int main(void) {
    int n;
    double t1, t2, time, c;

    /* Zaglavlje tablice. */

    printf("      n      Broj_Poteza(n)      Vrijeme      "
           "C = T(n) / (2^n - 1)\n");
    printf("      n      2^n - 1      T(n)\n");
    printf("=====\n");
    printf("=====\n");
}
```

## *Hanoi* — funkcija main za 0, 1 (nastavak)

```
for (n = 1; n <= nmax; ++n) {  
  
    /* Inicijalizacija brojaca za ovaj n. */  
    broj_poteza = 0;  
  
    t1 = dsecnd( );  
    Hanojski_tornjevi(n, 1, 3);  
    t2 = dsecnd( );  
    time = t2 - t1;  
  
    if (broj_poteza > 0)  
        c = time / broj_poteza;  
    else  
        c = 0.0;
```

## *Hanoi* — funkcija main za 0, 1 (nastavak)

```
        printf("    %2d    %10ld    %11.6f    "  
              "%15e\n",  
              n, broj_poteza, time, c);  
    }  
  
    return 0;  
}
```

---

## Intel C (2007. g.) — Izmjerena vremena $T(n)$

Usporedba izmjerenih vremena  $T(n)$  (u s) za razna računala i razne varijante (Intel C 9.1, /03):

Varijanta	Klamath5	Mali 3.0	Veliki_P	BabyBlue
0, $n = 30$	57.12	7.74	8.55	7.28
1, $n = 30$	28.30	4.17	4.52	3.83
a0, $n = 30$	59.69	8.90	8.62	7.45
a1, $n = 30$	30.66	4.67	4.55	3.83

Na “normalnom” prevoditelju dobivamo očekivano zeleno ponašanje rezultata za varijantu 1 (bez poziva za  $n = 0$ ):

🟡 ušteda u vremenu je skoro 50%!

Tih 50% je i gornja granica za uštedu — v. malo dalje.

## Intel C (2007. g.) — Trajanje poteza za $n = 30$

Usporedba trajanja jednog poteza  $c(n)$  (u s) za razna računala i razne varijante (Intel C 9.1, /03):

Varijanta	Klamath5	Mali 3.0	Veliki_P	BabyBlue
0, $n = 30$	$5.32 \cdot 10^{-8}$	$7.20 \cdot 10^{-9}$	$7.97 \cdot 10^{-9}$	$6.78 \cdot 10^{-9}$
1, $n = 30$	$2.64 \cdot 10^{-8}$	$3.89 \cdot 10^{-9}$	$4.20 \cdot 10^{-9}$	$3.57 \cdot 10^{-9}$
a0, $n = 30$	$5.56 \cdot 10^{-8}$	$8.29 \cdot 10^{-9}$	$8.03 \cdot 10^{-9}$	$6.94 \cdot 10^{-9}$
a1, $n = 30$	$2.86 \cdot 10^{-8}$	$4.35 \cdot 10^{-9}$	$4.23 \cdot 10^{-9}$	$3.57 \cdot 10^{-9}$

Razlika Mali, Veliki\_P na varijantama 0, 1 je “misterij”.

Vjerojatni krivac:

- razlika u arhitekturi procesora (Northwood C, Prescott) i optimizacija te verzije kompilera!

# Intel C (2018. g.) — Izmjerena vremena $T(n)$

Usporedba izmjerenih vremena  $T(n)$  (u s) za jedno računalo

📍 Sasa-PC = Intel Core i7 4770K (Haswell) na 3.5 GHz,  
i razne varijante (Intel C 18.0.3, ia32 i x64, razne opcije):

Varijanta	ia32			x64		
	/0d	/03	/fast	/0d	/03	/fast
0, $n = 30$	9.148	3.478	2.417	9.418	2.312	2.267
1, $n = 30$	6.389	1.935	1.541	6.078	1.532	1.537
a0, $n = 30$	9.320	4.379	2.347	9.431	2.566	2.351
a1, $n = 30$	6.768	2.339	1.556	6.224	1.581	1.672

Očekivano, varijanta 1 (bez poziva za  $n = 0$ ) je bitno brža:

📍 ušteda u vremenu je 30–40%!

# Intel C (2018. g.) — Trajanje poteza za $n = 30$

Usporedba trajanja jednog poteza  $c(n)$  (u  $10^{-9}$  s) za jedno računalo (Sasa-PC) i razne varijante (Intel C 18.0.3, ia32 i x64, razne opcije):

Varijanta	ia32			x64		
	/Od	/O3	/fast	/Od	/O3	/fast
0, $n = 30$	8.52	3.24	2.25	8.77	2.15	2.11
1, $n = 30$	5.95	1.80	1.44	5.66	1.43	1.43
a0, $n = 30$	8.68	4.08	2.19	8.78	2.39	2.19
a1, $n = 30$	6.30	2.18	1.45	5.80	1.47	1.56

# Prošireni model složenosti



# Hanojski tornjevi — modeli složenosti

Osnovni model složenosti mjeri samo broj poteza

$t_n :=$  broj poteza za  $n$  diskova.

Pripadna rekurzija za  $t_n$  je bila

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

Rješenje za broj poteza je  $t_n = 2^n - 1$ , za svaki  $n \geq 0$ .

U ovom modelu

• nema razlike između varijanti 0 i 1.

Model je prejednostavan — brojimo samo poteze, bez poziva.

## Mane osnovnog modela složenosti

Taj model **nije dobar** za procjenu **vremenske** složenosti.

Problem se vidi već u **početnom uvjetu**  $t_0 = 0$ . Po tom modelu, poziv s  $n = 0$  uvijek “traje” **nula** vremena. No,

🔴 znamo da to **nije istina** u varijanti **0**, iako **nema poteza**.

Treba nam model u kojem je  $t_0 > 0$ . Dovoljno je uzeti da je

$$t_0 = \text{const} > 0.$$

**Zadatak**. Probajte sami napraviti takav model, sa **samo jednim** parametrom — konstantom **const**.

🔴 Što je značenje tog parametra **const**?

🔴 Testirajte ovaj model, tj. probajte **eksperimentalno** odrediti parametar **const** i pogledajte kolike su **greške**.

## Prošireni model složenosti

Još **bolji** model **vremenske** složenosti dobivamo tako da uvedemo dva “**parametra**” — **dvije konstante** u model.

- **Jedna** od njih mjeri **samo** trajanje **poteza**, a
- **druga** mjeri trajanje “**preostatka**” svakog **poziva** rekurzivne funkcije **Hanoi**, tj. sve ostalo, **bez** poteza.

Takav model precizno razlikuje

- **poteze** od **poziva**,

i mora pokazati **razliku** između varijanti **0** i **1**.

# Dvoparametarski model složenosti — parametri

Preciznije, neka je:

- $c_m$  = trajanje **jednog poteza** (“move”), tj.
  - trajanje **jednog poziva** funkcije **prebaci\_jednog**.

Što god smatrali potezom, “skriveno” je u pozivu ove funkcije!

- $c_0$  = trajanje **jednog poziva** funkcije **Hanoi**, **bez poteza** i **bez rekurzivnih poziva** te funkcije. Dakle,  $c_0$  uključuje
  - **poziv** funkcije **izvana**,
  - **ulaz** u funkciju (priprema argumenata — stack/reg),
  - **testiranje** vrijednosti od  $n$  u **if** naredbi,
  - **izlaz** iz funkcije (iza **if**), što uključuje povratak i “čišćenje” argumenata sa stacka ili iz registara.

## Vremenska složenost za obje varijante

Varijanta **0** ima pozive funkcije **Hanoi** s argumentom  $n = 0$ .

Rekurzija za pripadnu vremensku složenost  $t_n^{(0)}$  je

$$t_n^{(0)} = \begin{cases} c_0, & \text{za } n = 0, \\ c_0 + 2t_{n-1}^{(0)} + c_m, & \text{za } n > 0. \end{cases}$$

Varijanta **1** nema pozive funkcije **Hanoi** s argumentom  $n = 0$ .

Rekurzija za pripadnu vremensku složenost  $t_n^{(1)}$  je

$$t_n^{(1)} = \begin{cases} c_0 + c_m, & \text{za } n = 1, \\ c_0 + 2t_{n-1}^{(1)} + c_m, & \text{za } n > 1. \end{cases}$$

Uočimo da su rekurzije **iste** u **obje** varijante!

## Poravnanje početka na $n = 0$ u varijanti 1

Za lakše rješavanje rekurzija, isplati se “poravnati” početak, tako da obje rekurzije počinju s  $n = 0$ .

Stvarni početak varijante 1 je za  $n = 1$ , i glasi

$$t_1^{(1)} = c_0 + c_m.$$

Kad umjetno “vratimo” drugu rekurziju za jedan član unatrag

$$t_1^{(1)} = c_0 + c_m = c_0 + 2t_0^{(1)} + c_m,$$

dobivamo da mora biti

$$t_0^{(1)} = 0.$$

To savršeno odgovara pravom stanju stvari, jer poziva s  $n = 0$  zaista nema, tj. trajanje im je nula! Točno to smo htjeli.

# Zajednička rekurzija za vremensku složenost

Rekurzija za vremensku složenost  $t_n$  u obje varijante je ista

$$t_n = 2t_{n-1} + c_0 + c_m, \quad \text{za } n > 0,$$

a početni uvjeti su različiti

$$t_0 = \begin{cases} c_0, & \text{za varijantu 0,} \\ 0, & \text{za varijantu 1.} \end{cases}$$

Rješenje. Nehomogeni član u rekurziji ima standardni oblik

$$g(n) = b^n p_d(n),$$

uz

$$b = 1, \quad p_0(n) = c_0 + c_m.$$

# Rješenje rekurzije za vremensku složenost

Karakteristična jednačba **homogenizirane** rekurzije je

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

s korijenima  $r_1 = 1$  i  $r_2 = 2$ . Opće rješenje **homogenizirane** rekurzije je

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

Ovo rješenje **uvrstimo** u **polaznu** rekurziju, zato da dobijemo konstantu  $c_1$  uz onaj **dio rješenja** koji odgovara **nehomogenom** članu. Izlazi

$$c_1 + c_2 \cdot 2^n = 2(c_1 + c_2 \cdot 2^{n-1}) + c_0 + c_m$$

$$c_1 = 2c_1 + c_0 + c_m$$

$$c_1 = -(c_0 + c_m).$$



## Rješenja za vremensku složenost

Dakle, opće rješenje **polazne** nehomogene rekurzije je

$$t_n = c_2 \cdot 2^n - (c_0 + c_m), \quad n \geq 0,$$

a konstanta  $c_2$  se računa iz **početnog uvjeta**  $t_0$ .

🔴 Za varijantu **0**, početni uvjet je  $t_0 = c_0$ , pa je

$$t_0 = c_2 \cdot 2^0 - (c_0 + c_m) = c_0 \quad \implies \quad c_2 = 2c_0 + c_m.$$

🔴 Za varijantu **1**, početni uvjet je  $t_0 = 0$ , pa je

$$t_0 = c_2 \cdot 2^0 - (c_0 + c_m) = 0 \quad \implies \quad c_2 = c_0 + c_m.$$

# Rješenja za vremensku složenost (nastavak)

Konačna rješenja za vremensku složenost obje varijante algoritma, uz polazne oznake, možemo napisati ovako:

## ● Varijanta 0

$$\begin{aligned}t_n^{(0)} &= (2c_0 + c_m) \cdot 2^n - (c_0 + c_m) \\ &= (c_0 + c_m) \cdot (2^n - 1) + c_0 \cdot 2^n, \quad n \geq 0.\end{aligned}$$

## ● Varijanta 1

$$\begin{aligned}t_n^{(1)} &= (c_0 + c_m) \cdot 2^n - (c_0 + c_m) \\ &= (c_0 + c_m) \cdot (2^n - 1), \quad n \geq 0.\end{aligned}$$

Drugo rješenje  $t_n^{(1)}$  je “skalirani” broj poteza. U pripadnoj rekurziji piše  $c_0 + c_m$ , umjesto 1, a početni uvjeti su isti.

# Vremenska složenost — usporedba

Uz pretpostavku da su konstante  $c_0$  i  $c_m$

- jednake za obje varijante algoritma,

- što je skoro “očito” — usporedbom tih algoritama, smijemo usporediti dobivena rješenja. Dobivamo da je

$$t_n^{(0)} = (c_0 + c_m) \cdot (2^n - 1) + c_0 \cdot 2^n = t_n^{(1)} + c_0 \cdot 2^n.$$

Zaključak:

- trajanje varijante 0 je veće za  $c_0 \cdot 2^n$ ,

što točno odgovara trajanju

- $2^n$  bespotrebnih rekurzivnih poziva s  $n = 0$  diskova.

Dakle, ovaj model, bar u teoriji, radi!

# Korektnost modela za vremensku složenost

Naravno, ovaj dvoparametarski model je korektan u praksi,

- ako i samo ako vrijede osnovne pretpostavke tog modela.

Te pretpostavke treba provjeriti — verificirati u praksi.

Osnovna pretpostavka našeg modela je

- da su parametri  $c_m$  i  $c_0$  zaista konstante,
- tj. da ovi parametri ne ovise o broju diskova  $n$ .

Sada, kad imamo izraze za vremensku složenost  $t_n$  u obje varijante algoritma, to se lako može

- eksperimentalno provjeriti,
- tako da iskoristimo izmjerena vremena trajanja  $T(n)$ .

# Verifikacija modela za vremensku složenost

No, i tu treba biti oprezan — da ne tražimo **previše**.

Za **verifikaciju** pretpostavki i modela — “striktno” govoreći, trebalo bi eksperimentalno **provjeriti dvije** stvari. **Prvo**,

- jesu li  $c_m$  i  $c_0$  “**skoro**” **konstante**, gledano u funkciji od  $n$ ,
- i to za **svaku** varijantu algoritma — **posebno**.

**Zatim** treba vidjeti dobivamo li u **obje** varijante

- “**približno**” **iste** vrijednosti za **odgovarajuće** konstante.

To bi značilo da,

- na početku, **svaka** varijanta ima **svoje** parametre  $c_m$  i  $c_0$ ,
- a onda provjeravamo jesi li oni **isti** u **raznim** varijantama.

Takav pristup **ne valja** — dobivamo **previše** parametara.

## Previše parametara za pouzdanu verifikaciju

Za naše **dvije** varijante algoritma, imamo čak **4** parametra. To je **previše**, u smislu da ih je

- **nemoguće pouzdano** odrediti iz mjerenja.

U čemu je problem?

Kako bismo, za svaku **pojedinu** varijantu, provjerili jesu li pripadni  $c_m$  i  $c_0$  “**konstantni**”? Recimo, ovako:

- izmjerimo vremena  $T(n)$  za neke vrijednosti od  $n$ ;
- postavimo odgovarajući “**model**”  $T(n) = t_n$ , s **dva nepoznata konstantna** parametra, i parametre odredimo diskretnom metodom najmanjih kvadrata;
- ako su dobivene **greške** dovoljno **male**, možemo uzeti da su nađeni parametri zaista “**konstantni**”.

## Tablica vremena za varijantu a0

Na primjer, za varijantu a0 (Intel C 9.1, BabyBlue računalo) dobivamo sljedeći izlaz:

$n$	$2^n - 1$	$T(n)$	$T(n)/(2^n - 1)$
1	1	0.000	0.000000e+000
⋮	⋮	⋮	⋮
19	524287	0.000	0.000000e+000
20	1048575	0.015	1.430513e-008
21	2097151	0.016	7.629398e-009
22	4194303	0.015	3.576280e-009
23	8388607	0.063	7.510186e-009
24	16777215	0.109	6.496907e-009
25	33554431	0.235	7.003546e-009
26	67108863	0.453	6.750226e-009
27	134217727	0.906	6.750226e-009
28	268435455	1.828	6.809831e-009
29	536870911	3.688	6.869435e-009
30	1073741823	7.453	6.941147e-009

## Model složenosti za varijantu a0

Pripadni model vremenske složenosti za varijantu 0 je

$$\begin{aligned}t_n^{(0)} &= (2c_0 + c_m) \cdot 2^n - (c_0 + c_m) \\ &= (c_0 + c_m) \cdot (2^n - 1) + c_0 \cdot 2^n, \quad n \geq 0.\end{aligned}$$

Iz ova dva zapisa odmah dobivamo dva različita modela za trajanje  $T_0(n)$  — razlika je samo u izboru funkcija baze.

● Model (a):

$$T_0(n) = a_1 \cdot 2^n + a_2 \cdot 1,$$

s koeficijentima  $a_1 = 2c_0 + c_m$  i  $a_2 = c_0 + c_m$ .

● Model (b):

$$T_0(n) = b_1 \cdot (2^n - 1) + b_2 \cdot 2^n,$$

s koeficijentima  $b_1 = c_0 + c_m$  i  $b_2 = c_0$ .



## Nestabilnost računanja dva parametra

Oba modela su **linearna** i imaju **dva** nepoznata koeficijenta. Kad ih izračunamo, lako nađemo **parametre**  $c_0$  i  $c_m$  za varijantu **a0** (linearni sustav reda 2).

- **Koeficijente** u modelu računamo **diskretnom metodom najmanjih kvadrata**, iz tablice izmjerenih vremena.

I tu je **problem**. Tablica je **vrlo netočna** za **male** vrijednosti  $n$ . Izmjerena vremena  $T(n)$  imaju neku točnost tek za  $n \geq 23$ .

- U modelu (a), član  $a_2 \cdot 1$  ima utjecaja samo za **male**  $n$ , tj. koeficijent  $a_2$  se **ne može** dobro odrediti iz mjerenja.
- U modelu (b), funkcije baze  $2^n - 1$  i  $2^n$  se “iole razlikuju” samo za **male**  $n$ . Inače su “skoro” linearno zavisne, pa  $b_1$  i  $b_2$  **ne možemo** nezavisno dobro odrediti.


Dakle, **dva** koeficijenta su **previše**. **Jedan** bi “prošao” (v. iza).

## Model složenosti za varijantu **a1**

Isti problem imamo i u varijanti **1**. Tu se još **bolje** vidi!

Model **vremenske** složenosti za varijantu **1** je

$$\begin{aligned}t_n^{(1)} &= (c_0 + c_m) \cdot 2^n - (c_0 + c_m) \\ &= (c_0 + c_m) \cdot (2^n - 1), \quad n \geq 0.\end{aligned}$$

Složenost ovisi samo o **zbroju** parametara  $c_0$  i  $c_m$ , tj. stvarno,  imamo samo **jedan** parametar.

Jedini razumni **model** za trajanje  $T_1(n)$  je

$$T_1(n) = C_1 \cdot (2^n - 1),$$

s **jednim** koeficijentom  $C_1 = c_0 + c_m$ .

# Stvarna ideja dvoparametarskog modela

Naš dvoparametarski model vremenske složenosti, s parametrima  $c_m$  i  $c_0$ , smišljen je zato da

- pokaže razliku između dviju varijanti 0 i 1,

a ne zato da dobijemo “precizniji” model složenosti za svaku pojedinu varijantu.

Dakle, na samom početku, u “izvodu” modela,

- koji je napravljen na osnovu obje varijante algoritma, pretpostavljamo da za obje varijante

- vrijede iste vrijednosti parametara  $c_m$  i  $c_0$ .

(Očito je da  $c_m$  mora biti isti — zovemo istu funkciju.)

Zato, u verifikaciji, treba iskoristiti izmjerena vremena za obje varijante, da bismo odredili/provjerili te parametre  $c_m$  i  $c_0$ .

# Verifikacija modela za vremensku složenost

Za **dvije** varijante algoritma sad imamo “**samo**” **2** parametra.

**Bitno**: Njih je **lako pouzdano odrediti** i verificirati!

**Najlakši** način za to je:

- za **obje** varijante algoritma uzmemo **jednoparametarski** model **trajanja** — s **jednim nepoznatim** koeficijentom,
- koji **dobro** opisuje trajanje za **velike  $n$**  — tamo gdje su izmjerene vrijednosti dovoljno **točne**.

Taj model **ne mora** biti **egzaktan**,

- možemo koristiti i **asimptotsko** ponašanje **vremenske** složenosti za **velike** vrijednosti  **$n$** .

Kad odredimo ta **dva** koeficijenta, **provjerimo** njihovu “**konstantost**” (grešku modela) i **nađemo** parametre  **$c_m$**  i  **$c_0$** .

# Jednoparametarski modeli trajanja

Za trajanje  $T_1(n)$  varijante **1** već imamo takav model

$$T_1(n) = C_1 \cdot (2^n - 1),$$

s **jednim** koeficijentom  $C_1 = c_0 + c_m$ . Ovaj model je **egzaktan**.

Jednoparametarski model za trajanje  $T_0(n)$  varijante **0**, izlazi **aproksimacijom** iz modela **vremenske složenosti**

$$\begin{aligned} t_n^{(0)} &= (2c_0 + c_m) \cdot 2^n - (c_0 + c_m) \\ &= (2c_0 + c_m) \cdot (2^n - 1) + c_0, \quad n \geq 0. \end{aligned}$$

Za iole **veći**  $n$ , zadnji član  $c_0$  postaje zanemariv, pa je

$$T_0(n) \approx C_0 \cdot (2^n - 1),$$

s **jednim** koeficijentom  $C_0 = 2c_0 + c_m$ .

# Određivanje parametara i verifikacija modela

Koeficijente  $C_0$  i  $C_1$  možemo odrediti na dva načina.

- Dijeljenjem izmjerenog vremena i broja poteza,

$$C_i(n) = \frac{T_i(n)}{2^n - 1},$$

za svaki  $n$  u tablici za pojedinu varijantu  $i = 0, 1$ .

Tako dobivamo “profil” koeficijenta  $C_i(n)$  u ovisnosti o  $n$ , odakle se lako vizuelno provjerava “konstantnost”.

Usput, točno to je zadnji stupac izlaza u programima.

- Diskretnom metodom najmanjih kvadrata s jednim parametrom — traženim koeficijentom, uz provjeru grešaka takve aproksimacije izmjerenih podataka.

Probajte sami!

## Primjer određivanja parametara i verifikacije

Primjer. Pogledajmo rezultate programa za varijante **a0** i **a1** (Intel C 9.1, **BabyBlue**), za  $n \geq 25$  (vrlo slično je za **0** i **1**):

$n$	$2^n - 1$	$T_0(n)$	$T_0(n)/(2^n - 1)$	$T_1(n)$	$T_1(n)/(2^n - 1)$
25	33554431	0.235	7.003546e-009	0.109	3.248453e-009
26	67108863	0.453	6.750226e-009	0.250	3.725290e-009
27	134217727	0.906	6.750226e-009	0.484	3.606081e-009
28	268435455	1.828	6.809831e-009	0.953	3.550202e-009
29	536870911	3.688	6.869435e-009	1.907	3.552064e-009
30	1073741823	7.453	6.941147e-009	3.828	3.565103e-009

Vidimo da su koeficijenti  $C_0(n)$  i  $C_1(n)$  skoro **konstantni** i **stabiliziraju** se, kako  $n$  raste, približno na sljedeće vrijednosti

$$C_0 = 2c_0 + c_m \approx 6.9 \cdot 10^{-9},$$

$$C_1 = c_0 + c_m \approx 3.6 \cdot 10^{-9}.$$

## Primjer određivanja parametara i verifikacije

Oдавде dobivamo približne vrijednosti za **parametre** modela

$$c_0 \approx 3.3 \cdot 10^{-9} \text{ s}, \quad c_m \approx 0.3 \cdot 10^{-9} \text{ s}.$$

Jedinice su **sekunde** — parametri su **trajanje poziva** i **poteza**.

### Komentar.

- Naši “**potezi**” su jako **brzi** — **jedno** povećanje globalnog brojača u funkciji **prebaci\_jednog**.

Brzina približno odgovara frekvenciji računala (**3.6 GHz**).

- Jedan **poziv** traje **11 puta** dulje. Zato je **ušteta** u varijanti **1** skoro **50%** ( $c_0$  prema  $2c_0 + c_m$ , uz **mali**  $c_m$ ).

Uz veću **točnost štoperice** (ili na **sporijem** računalu), dobili bismo **veću** točnost parametara.



# Asimptotski jednoparametarski modeli trajanja

**Napomena.** Mogli smo koristiti i jednoparametarske modele na bazi **asimptotskog** ponašanja **vremenske** složenosti

$$T_0(n) \sim C_0 \cdot 2^n,$$

$$T_1(n) \sim C_1 \cdot 2^n,$$

s istim značenjima koeficijenata  $C_0$  i  $C_1$ .

Dobivamo **minijaturno veću** grešku nego prije, zbog zamjene funkcije baze  $2^n - 1 \mapsto 2^n$ .

Prema prethodnom **primjeru** iz **2007.** godine, stvari izgledaju **vrlo lijepo** i **razumno**,

🕒 u smislu **korektnosti** interpretacije i odnosa parametara.

Nažalost, u modernija vremena to više **nije tako!**

## Novi parametri (2018. g.) — velike razlike!?

Računalo = Sasa-PC, Intel C 18.0.3, ia32 i x64, razne opcije:

- usporedba parametara  $c_0$  i  $c_m$  (u  $10^{-9}$  s) za razne parove.

opcija	par	ia32				x64			
		$C_0$	$C_1$	$c_0$	$c_m$	$C_0$	$C_1$	$c_0$	$c_m$
/0d	0, 1	8.52	5.95	2.57	3.38	8.77	5.66	3.11	2.55
/0d	a0, a1	8.68	6.30	2.38	3.92	8.78	5.80	2.98	2.82
/03	0, 1	3.24	1.80	1.44	0.36	2.15	1.43	0.72	0.71
/03	a0, a1	4.08	2.18	1.90	0.28	2.39	1.47	0.92	0.55
/fast	0, 1	2.35	1.44	0.81	0.63	2.11	1.43	0.68	0.75
/fast	a0, a1	2.19	1.45	0.74	0.71	2.19	1.56	0.63	0.93

Za iste opcije, dobivamo velike razlike u trajanjima jednog poziva ( $c_0$ ), odnosno, jednog poteza ( $c_m$ ). Verifikacija??

# Moguća objašnjenja za velike razlike

- Procesor ima x64 arhitekturu, ali ju 32-bitni i 64-bitni kôd različito koriste.
- Optimizatori u pripadnim verzijama kompajlera nisu isti.
- Prijenos manjeg broja argumenata u funkciju više ne ide preko stacka, već se koriste registri.
  - Prvi i drugi rekurzivni poziv nemaju iste assembler (strojne) instrukcije.

Usput, tu je razlika između para (0, 1) i para (a0, a1).

Varijante 0 i 1 u istom paru ipak nemaju identični kôd, tj. pripadni  $c_0$  ne mora biti isti (problem modela). Dodatno:

- Moderni procesori imaju tzv. “speculative execution”, u kojeg spada i tzv. “branch prediction” — na osnovu statistike prethodnog “skakanja” u if.

# Završni komentar o modelu složenosti i praksi

Završni komentar o modelu složenosti i parametrima.

- Za parametar  $c_m$  zaista očekujemo da je isti u obje varijante, jer zovemo istu funkciju.
- Trajanje poziva — parametar  $c_0$ , bi mogao, eventualno, biti različit, jer algoritmi ipak nisu sasvim isti.

Tako bismo za dvije varijante dobili model s tri parametra,

- ali njih, opet, nije lako pouzdano odrediti!

Praksa: Na temelju vremena za jedan program, možemo

- točno odrediti samo jednu konstantu.

Iz dva programa — možemo naći dvije konstante. Onda ih (možda) možemo “prevesti” u neke druge dvije konstante, s “boljom” interpretacijom u modelu — poput  $c_0$  i  $c_m$ .