Algoritmi in podatkovne strukture 1

Visokošolski strokovni študij Računalništvo in informatika

Zahtevnost algoritmov

Jurij Mihelič, UniLj, FRI

Analiza algoritmov

Kolikokrat je potrebno obrniti pogonsko ročico?

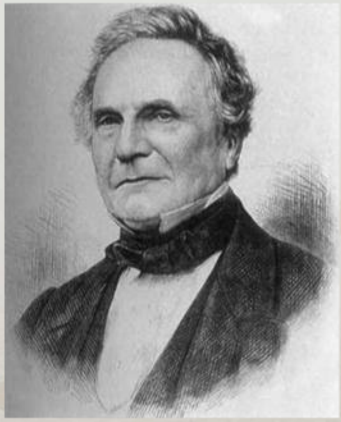
Analiza algoritmov

- Temeljno področje algoritmike
  - proučuje porabo virov algoritmov

*As soon as an Analytic Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will arise - By what **course of calculation** can these results be arrived at by the machine in the shortest time?*

- Charles Babbage (1864)

Charles Babbage



1791 – 1871

Zahtevnost algoritma

- Katere vire potrebuje algoritem za svoje izvajanje?

- Viri:
  - **čas**: realni čas, št. korakov, št. operacij, št. dostopov do pomnilnika
  - **prostor**: poraba pomnilnika, diska
  - energija: poraba električne energije
  - komunikacija: pasovna širina, št. paketov

Zahtevnost algoritma

- Koliko vira potrebuje algoritem za svoje izvajanje?
  - koliko časa, koliko operacij
  - koliko pomnilnika
  - koliko električne energije

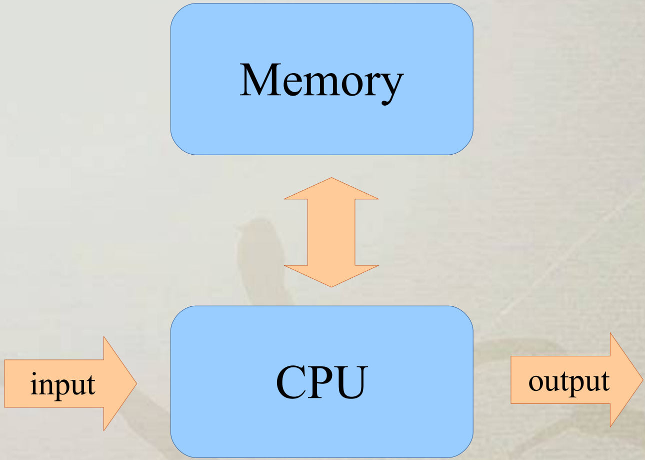
- Porabo virov navadno le ocenimo
- Zahtevnost ugotavljamo glede na nek bolj ali manj realen **model računanja**.

Model računanja

John Von Neumann

1903 – 1957

- Von Neumannov model
  - računalniška arhitektura
  - CPU
    - aritmetično logična enota, kontrolna enota
    - registri (ukazni register, programski števec)
  - pomnilnik
    - vsebuje **podatke** in **ukaze**
    - Von Neumannovo ozko grlo
      - branje ukazov in podatkov



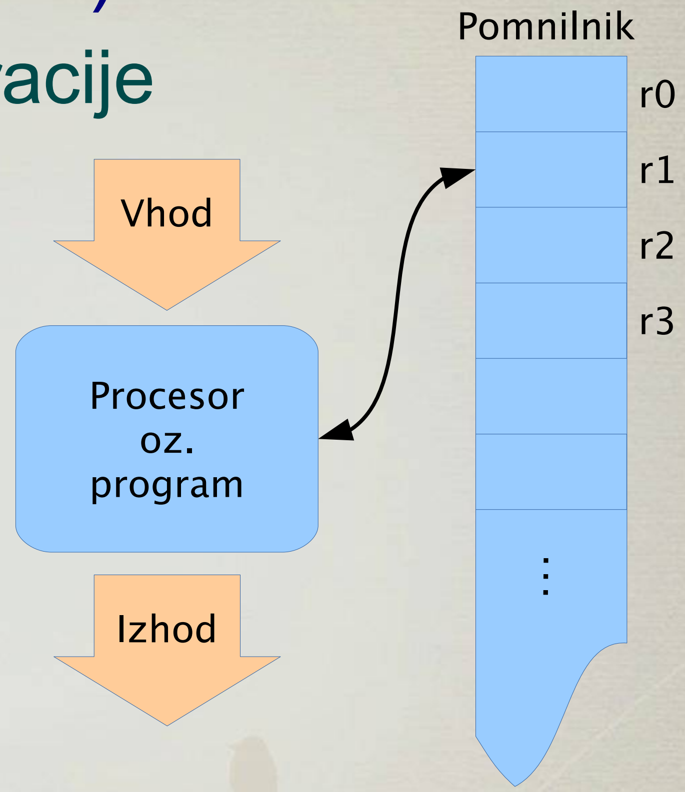
Model računanja

- Model računanja (*model of computation*)
  - množica dovoljenih **operacij**
    - realnost operacij
    - kompleksnost operacij
  - vsaka operacija ima neko **ceno**
    - cena ene izvedbe
    - cene so lahko različne
  - **enostavnost** in **realnost** modela
    - uporabnost

Model računanja ni enako kot računski model (computational model).

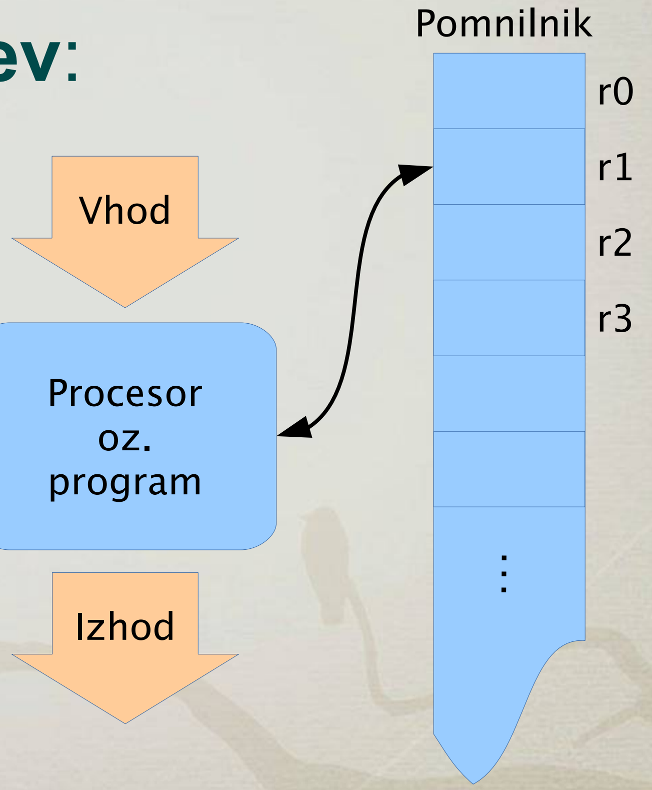
Model računanja

- RAM (*Random Access Machine*)
  - zaporedno izvaja običajne operacije
  - program je zapečen v procesor
  - ocena zahtevnosti
    - (solidna) ocena časa
    - (dobra) ocena prostora
  - RAM kot ciljni stroj
    - Algoritme pišemo
      v višjem programskem jeziku
      RAM pa si predstavljamo kot ciljni stroj.



Model računanja

- RAM (*Random Access Machine*)
  - dolžina besede in naslovni prostor
    - w bitov
  - predstavitev **števil** in **kazalcev**:
    - nepredznačeno
      od 0 do $2^w - 1$
    - predznačeno
      od -2^{w-1} do $2^{w-1} - 1$



Model računanja

- Veliko vrst modelov
  - avtomati, Turingovi stroji,
  - stroji s števcem, kazalcem,
  - RAM, PRAM, RASP,
  - programski jeziki, MMIX,
  - programi brez zank
  - bitni izračun (logična vezja)
  - odločitveno drevo
  - itd.

Zahtevnost algoritma

- Zahtevnost algoritma

Katere in koliko virov
potrebuje algoritem za svoje izvajanje
v nekem modelu računanja?

Zahtevnost algoritma

- Zahtevnost je odvisna od naloge (vhoda)
  - ogromno različnih nalog
  - različne naloge algoritem lahko rešuje različno časa
  - odvisnost zahtevnosti od
    - **velikosti** naloge
    - **podatkov** v nalogi

Zahtevnost algoritma

- Odvisnost od **velikosti** naloge
  - Množenje: 2*3 vs 1234*5678
  - Urejanje: 2 1 3 vs 3 1 4 2 5 9 6 0 7 8

- Zanima nas zahtevnost
  ob spremembi velikosti naloge
  - Časovna zahtevnost
    - $T(n) = \dots$
  - Prostorska zahtevnost
    - $S(n) = \dots$

Zahtevnost algoritma

- Odvisnost od **podatkov** v nalogi
  - Množenje: 1234*1000 vs 1234*5678
  - Urejanje: 0 1 2 3 4 5 6 7 8 9 vs 3 1 4 2 5 9 6 0 7 8

- Glede na vse možne naloge velikosti n govorimo o zahtevnosti:
  - v najboljšem primeru (*best case*)
  - **v najslabšem primeru (*worst case*)**
  - v povprečju (*average*)

Zahtevnost algoritma

- Zakaj najpogosteje uporabljamo zahtevnost v najslabšem primeru?
  - podaja največjo možno porabo vira za izvedbo algoritma na katerikoli nalogi
  - za veliko algoritmov je najslabši primer zelo pogost
    - npr. iskanje elementa, ko elementa ni v seznamu
  - zahtevnost v povprečju je pogosto (asimptotično) enaka zahtevnosti v najslabšem primeru.
  - zahtevnost v povprečju je pogosto težko analizirati

Primeri

Zaporedno iskanje

- Ideja algoritma
  - zaporedoma poglej vse elemente

Zaporedno iskanje

```
for i = 0 to n-1 do
    if a[i] == key then return i
return -1
```

Odločitveni ali iskalni problem
Naloga:
- tabela elementov
- iskani element

Rešitev:
- odgovor da/ne
- indeks iskanega elementa

- Zahtevnost algoritma
  - čas in prostor
    - kaj dejansko merimo?
  - odvisnost
    - od podatkov? od velikosti naloge?

Zaporedno iskanje

- Čas: **št. primerjav** elementov
  - tabela velikosti n
  - best: 1
  - worst: n
  - avg: $(n + 1) / 2$

Zaporedno iskanje

```
for i = 0 to n-1 do
    if a[i] == key then return i
return -1
```

Kakšna je naloga za najboljši in najslabši primer?

Zaporedno iskanje

- Čas: **št. primerjav** elementov
  - Kako izračunamo povprečno zahtevnost?
  - predpostavke
    - vsi možni vhodi enako verjetni
    - permutacije števil 1 … n
    - vedno iščemo isti element 1 (ostalo je simetrično)
    - iskani element vedno najdemo
  - Koliko permutacij ima 1 na 1., 2., 3., … n-tem mestu?

$$C_{avg}(n) = \sum_{i=1}^{n} \frac{(n-1)!}{n!} i = \frac{1}{n} \sum_{i=1}^{n} i = \frac{n+1}{2}$$

Zaporedno iskanje

- Čas: **realni čas**
  - ocena trajanja posameznih operacij
  - predpostavimo model računanja RAM

Zaporedno iskanje

```
for i = 0 to n-1 do
    if a[i] == key then return i
return -1
```

c_1 … pogoj v zanki
c_2 … primerjava elementov
c_3 … stavek **`return`**

- Zahtevnost:
  - best: $c_1 + c_2 + c_3$
  - worst: $c_1 \cdot (n+1) + c_2 \cdot n + c_3$ (elementa ne najdemo)

Zaporedno iskanje

- Čas: **realni čas**
  - predpostavke (kot prej)
    - element je na indeksu p-1 (izvede se p iteracij)

$$T(n,p)=c_1\cdot p+c_2\cdot p+c_3$$

  - povprečna zahtevnost

$$T_{avg}(n)=\sum_{p=1}^{n}\frac{1}{n}T(n,p)= \overset{\text{preveri}}{\ldots} =\frac{(c_1+c_2)}{2}n+\frac{(c_1+c_2)}{2}+c_3$$

$$=a\cdot n+b$$

$$a=\frac{c_1+c2}{2},\qquad b=a+c_3$$

Zaporedno iskanje

- Čas: **realni čas** – praktični preizkus
  - povprečna časovna zahtevnost
    - *T*(*n*) = *a* ì *n* + *b*
  - Kako določimo a in b?
    - naredimo praktični preizkus
      pri različnih n_1 in n_2

$$T(n_1) = a \cdot n_1 + b \qquad T(n_2) = a \cdot n_2 + b$$

    - odštejemo enačbi in izrazimo a in b

$$a = \frac{T(n_2) - T(n_1)}{n_2 - n_1}$$

$$b = T(n_2) - a \cdot n_2$$

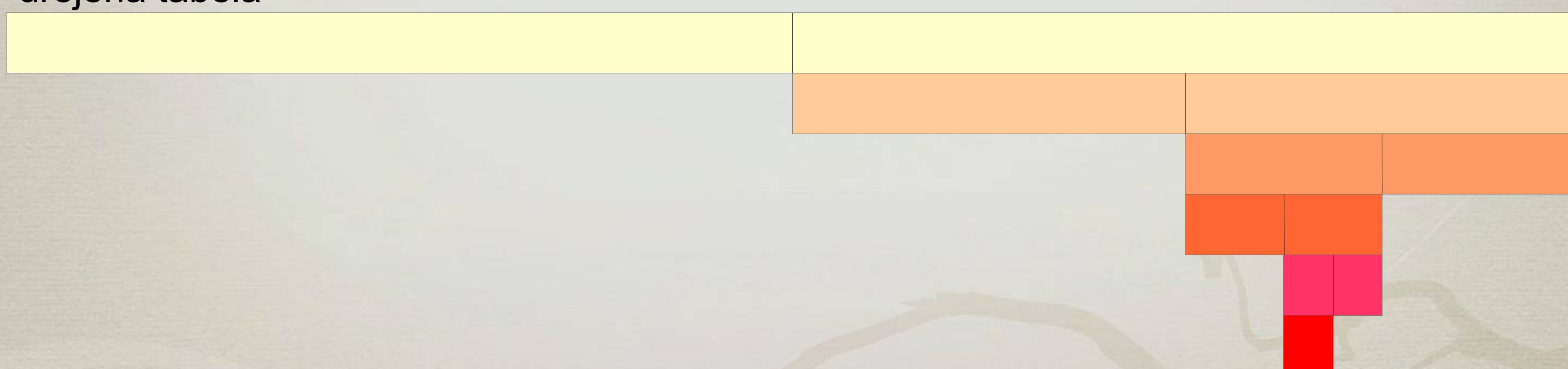
Dvojiško iskanje

- Iskanje elementa v **urejeni** tabeli
- Ideja algoritma
  - tabelo delimo na **dve** polovici
  - rekurzivno iščemo le v **eni** polovici

Odločitveni ali iskalni problem
Naloga:
- **urejena** tabela elementov
- iskani element

Rešitev:
- odgovor da/ne
- indeks iskanega elementa

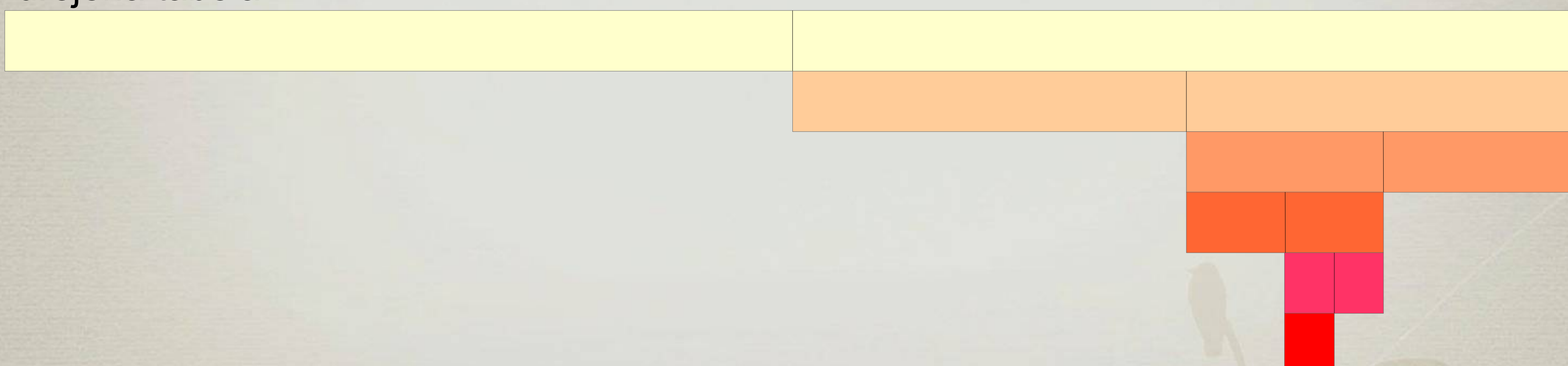
urejena tabela



Dvojiško iskanje

- Globina rekurzije:
  - best: 1
  - worst: $\lfloor \lg n \rfloor + 1$

urejena tabela



Dvojiško iskanje

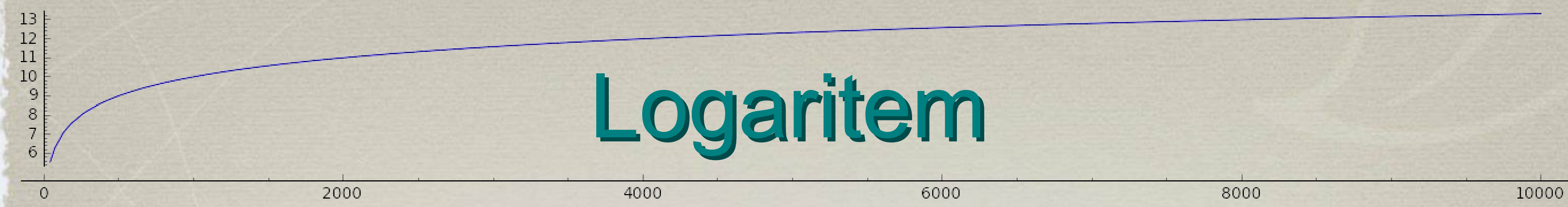
- Psevdokoda

Dvojiško iskanje (rekurzivno)

```
fun binarySearch(a, left, right, key) is
    if right > left then return -1
    mid = left + (right - left) / 2
    if (key < a[mid]) then
        return binarySearch(a, left, mid - 1)
    if (k > a[mid]) then
        return binarySearch(a, mid + 1, right)
    return mid
```

Dvojiško iskanje (iterativno)

```
while left <= right do
    mid = left + (right - left) / 2
    if key < a[mid] then right = mid - 1
    elif key > a[mid] then left = mid + 1
    else return mid
endwhile
return -1
```



Logaritem

- Dvojiški logaritem
  - a) Kolikokrat je potrebno razpoloviti n, da dobimo ≤ 1?
  - b) Koliko bitov potrebujemo za binarno predstavitev števil $\leq n$?
  - c) Koliko je globina celovitega (*complete*) dvojiška drevesa z n vozlišči?

V algoritmiki ima logaritem osnovno 2, če le ni drugače rečeno.

Načeloma velja

$\lg n = \log_2 n$

$\ln n = \log_e n$

$\log n = \log_{10} n$

a) $\lceil \lg n \rceil$
b) $\lceil \lg(n+1) \rceil$
c) $\lfloor \lg n \rfloor$

Povzetek

- Viri
  - čas in prostor
- Model računanja
  - RAM
- Odvisnost zahtevnosti
  - od velikosti naloge in od podatkov v nalogi
- Vrste zahtevnosti
  - najboljši primer, **najslabši** primer, povprečje
- Primeri
  - linearno iskanje, dvojiško iskanje, dvojne zanke