ALGORITMI I STRUKTURE PODATAKA – GRUPA 9
DODACI UZ PRVI, DRUGI I TREĆI TJEDAN NASTAVE

Robert Manger, 14. ožujka 2007.

Osnovni pojmovi, ciljevi i svrha kolegija

Osnovni pojmovi:
- *Strukture podataka*: sastojak, ono sa čime radimo.
- *Algoritmi*: postupak, ono što radimo.

Tijekom nekoliko desetljeća otkad postoje računala ljudi su pronašli velik broj upotrebljivih algoritama i struktura podataka, proučili su njihova svojstva, te su im dali imena.

Cilj kolegija (*što?*)
- Proučiti standardne algoritme i strukture podataka.
- Usvojiti njihova imena.
- Analizirati njihova svojstva.
- Primijeniti ih u vlastitim programima.

Svrha kolegija (*zašto?*)
- Da bi pisali bolje, čitljivije i efikasnije programe.
- Da ne bi nepotrebno otkrivali «toplu vodu».
- Da bi svoje rješenje znali opisati drugima.
- Da bi znali usporediti i vrednovati različita rješenja istog problema.

Odnos prema prethodnom kolegiju.
- Kolegij Programiranje i programsko inženjerstvo bilo je učenje abecede programiranja.
- Kolegij Algorimi i strukture podataka je usvajanje naprednijih znanja o programiranju.

Argumenti komandne linije

Htjeli bismo napisati program koji se pokreće iz komandne linije s argumentima. Dakle, naredba za pokretanje programa izgleda ovako:

```
> ime_prog argument1 argument2 argument3 ...
```

Primjeri takvih programa su DOS naredbe:

```
> copy file1 file2
> ren staro_ime novo_ime
> del data1 data2 data3
```

U C-u postoji ugrađeni mehanizam koji programu omogućuje da za vrijeme svog rada dohvati argumente iz komandne linije kojom je bio pokrenut. Da bi taj mehanizam aktivirali, potrebno je funkciju main( ) deklarirati s parametrima:

```
int main (int argc, char *argv[ ]) { ... }
```

Ti se parametri tumače ovako:

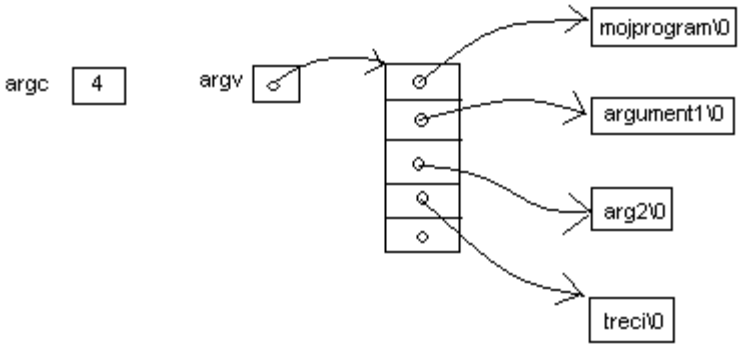
argc ... broj argumenata (riječi) u komandnoj liniji, uključujući i ime samog programa. Znači uvijek je argc <= 1, a ako je argc == 1 program je bio pozvan bez argumenata.

argv[ ] ... polje pokazivača na početke stringova u kojima se nalaze argumenti. To polje ima argc+1 elemenata, dakle indeksi su 0, 1, 2, ... argc.

Ako je program bio na primjer pozvan s

```
> mojprogram argument1 arg2 treci
```

tada su unutar main-a njegovi parametri već inicijalizirani i izgledaju ovako:



Dakle:

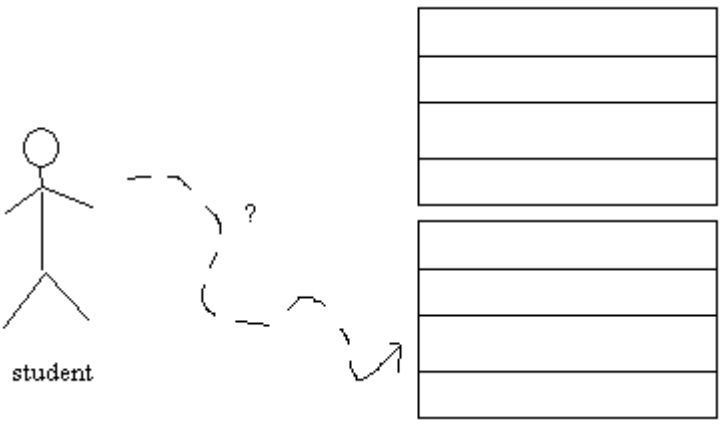
- agv[0] pokazuje na ime programa,
- agv[1] pokazuje na tekst prvog argumenta,
- ....
- argv[argc-1] pokazuje na tekst zadnjeg argumenta,
- agv[argc] je nul-pokazivač.

Korištenjem parametara argc i argv[ ] program može saznati svoje ime i dohvatiti sve svoje argumente, te zatim djelovati na odgovarajući način.

Tehnike adresiranja

Imamo datoteku koja se sastoji od zapisa istog tipa. Zanima nas kako u toj datoteci pronaći i (direktno) pročitati određeni zapis?

Na primjer moglo bi se raditi o datoteci sa zapisima o studentima. Pitamo se kako pronaći podatke o nekom određenom studentu?



Da bi problem bio bolje definiran, uvodimo pojam ključa. *(Primarni) ključ* je podatak unutar zapisa čija vrijednost jednoznačno određuje cijeli zapis. Drugim riječima, u istoj datoteci ne mogu postojati dva različita zapisa s istom vrijednošću primarnog ključa.

Problem adresiranja tada preciznije definiramo ovako: za zadanu vrijednost primarnog ključa pronaći mjesto u datoteci (adresu na disku) gdje se nalazi zapis s tom vrijednošću primarnog ključa.

U primjeru sa studentima primarni ključ bi mogao biti JMBAG. Zanima nas kako na primjer odrediti mjesto u datoteci gdje pišu podaci o studentu s JMBAG-om 36418339?

Problem adresiranja zaista jest problem zato što ključ obično ne možemo poistovijetiti s adresom (rednim brojem zapisa). Na primjer, kad bi htjeli da JMBAG izravno odgovara rednom broju zapisa, tada bi morali napraviti datoteku s $10^8 = 100$ milijuna zapisa, makar stvarni broj studenata ne prelazi nekoliko tisuća.

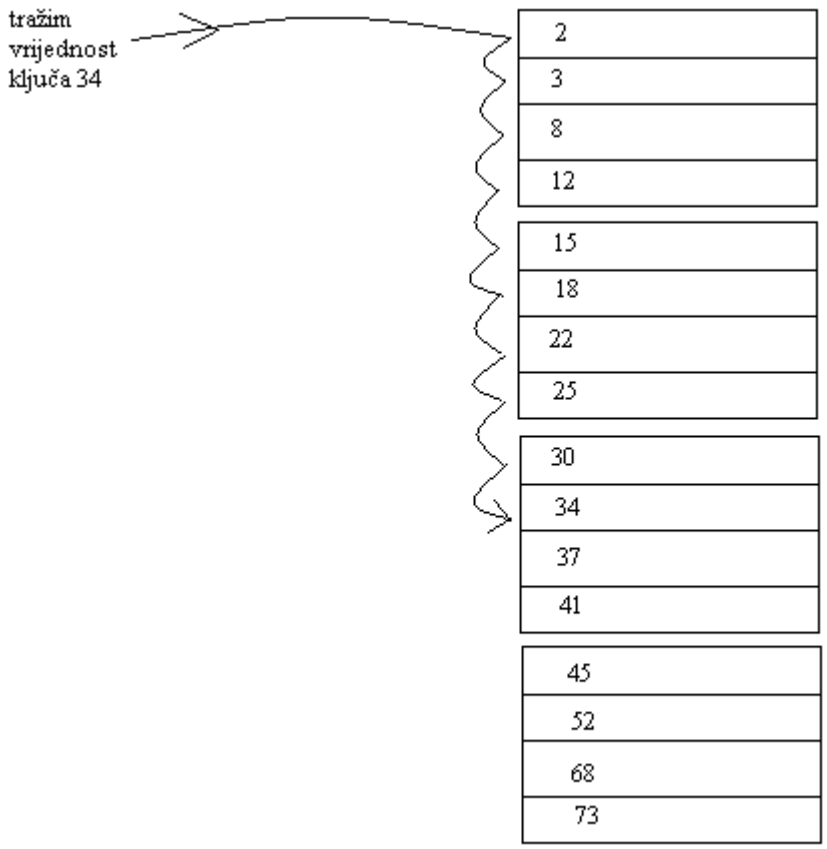
Bilo koje rješenje opisanog problema adresiranja zove se *tehnika adresiranja*. To rješenje obično u sebi sadrži dvije stvari:
- Pravilo kako ćemo organizirati datoteku, dakle pravilo kako će zapisi biti poredani, odnosno gdje će se nalaziti zapis s određenom vrijednošću ključa (*transformacija ključa u adresu*).
- Algoritam pronalaženja zapisa sa zadanom vrijednošću ključa u tako organiziranoj datoteci (*postupak pretraživanja*).

Voljeli bismo da naša tehnika adresiranja rezultira što bržim pronalaženjem traženog zapisa. Dakle, željeli bismo da postupak pretraživanja ima što manju vremensku

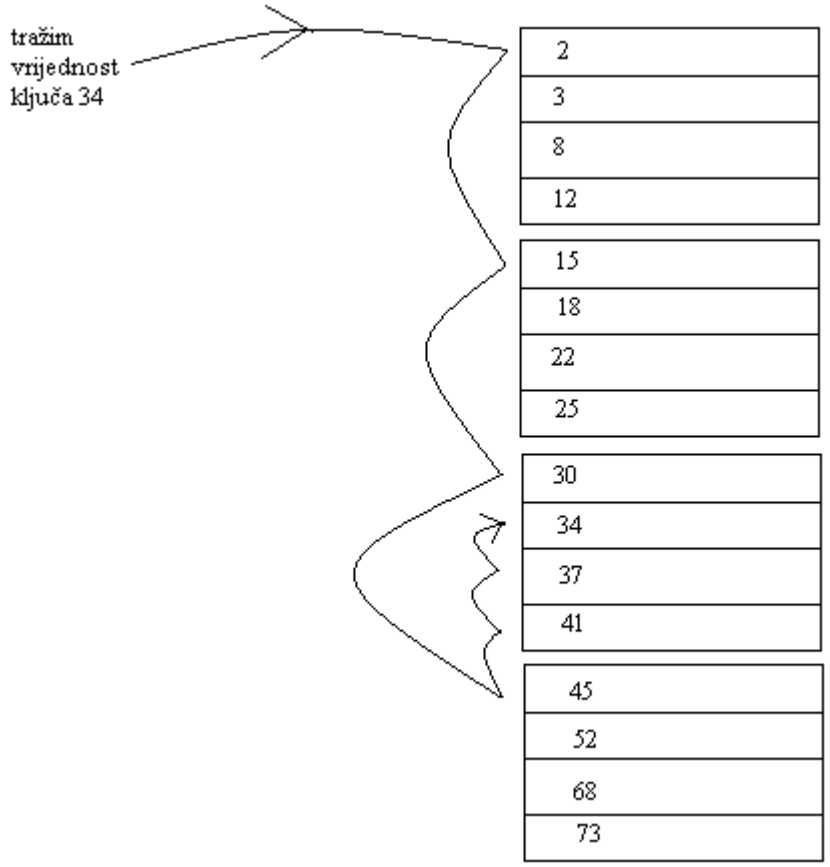
složenost. Ideal kojem težimo (no obično ga ne možemo doseći) je *direktni pristup*, dakle vremenska složenost koja je konstantna tj neovisna o veličini datoteke.

Slijedi pregled nekih jednostavnih postupaka pretraživanja. Kod većine od njih pretpostavlja se da je datoteka organizirana na određeni način.

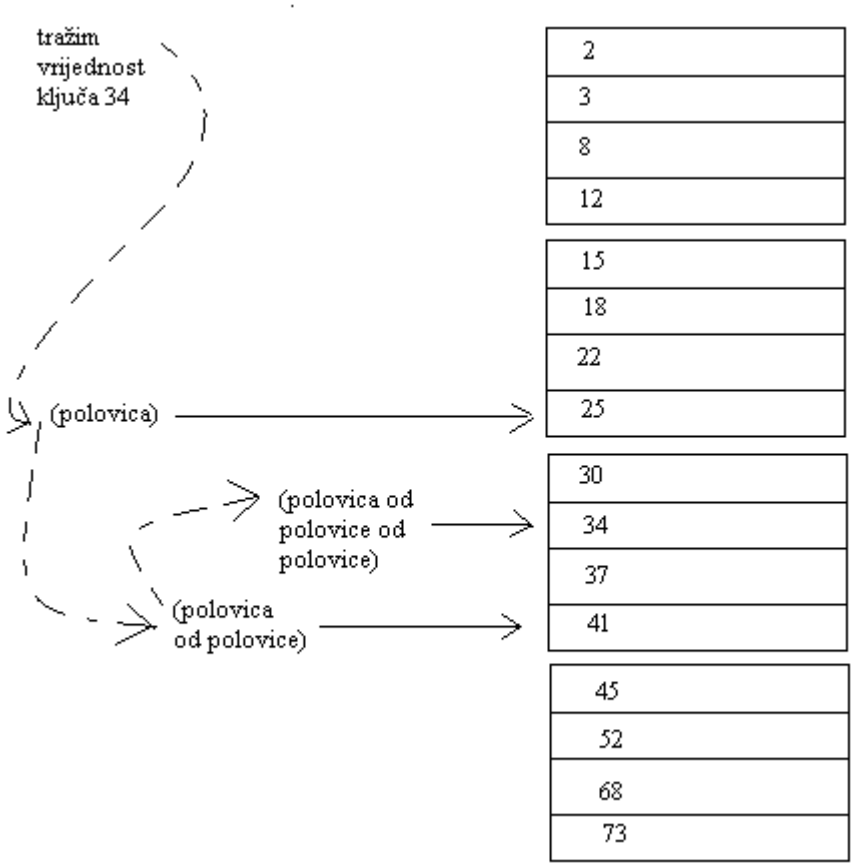
- *Slijedno (sekvencijalno) pretraživanje*. Tu nema nikakve pretpostavke o organizaciji datoteke, dakle datoteka može biti posve «neuređena». Čitamo redom zapise iz datoteke od prvog prema zadnjem, sve dok ne naiđemo na traženi zapis ili ne pročitamo cijelu datoteku.



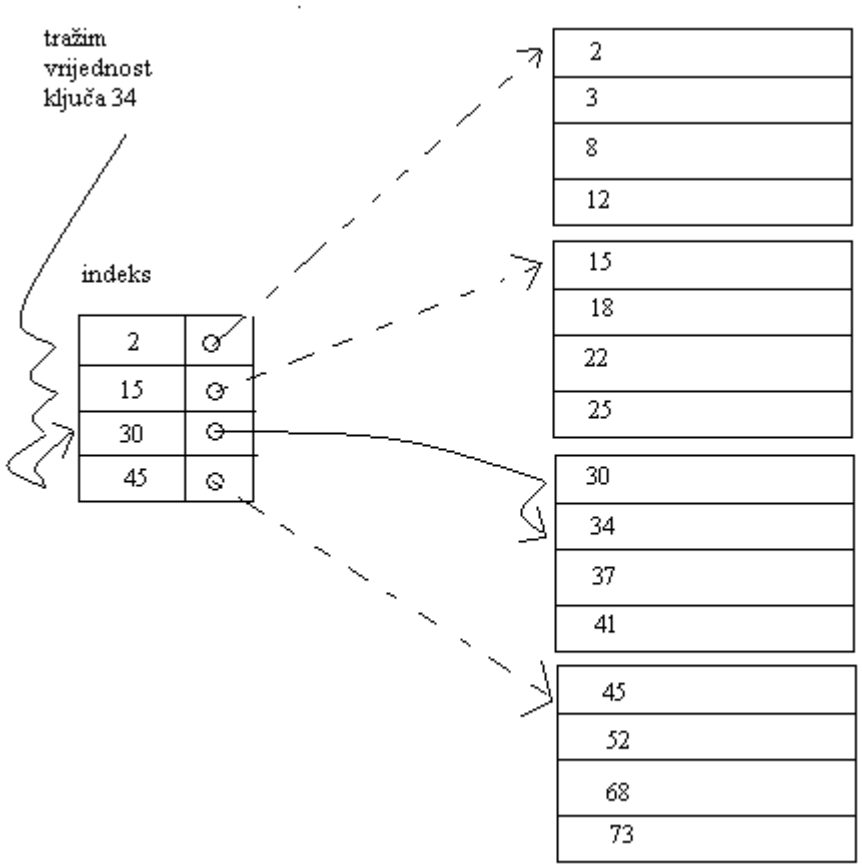
- *Pretraživanje po blokovima*. Pretpostavka je da je datoteka sortirana po ključu i podijeljena u «blokove» jednakih veličina. Čitamo po prvi zapis iz svakog bloka, sve dok ne «prijeđemo» traženi zapis tj dok ne naiđemo na zapis koji bi u sortiranom poretku trebao biti nakon traženog. Zatim detaljno pregledamo prethodni blok.



- *Binarno pretraživanje*. Pretpostavka je da je datoteka sortirana po ključu. Čitamo zapis na sredini datoteke. Ukoliko nismo pogodili traženi zapis, tada ovisno o zatečenoj vrijednosti ključa dalje promatramo jednu od dviju «polovica» datoteke u kojoj bi se morao nalaziti traženi zapis. Postupak rekurzivno ponavljamo u odabranoj «polovici».



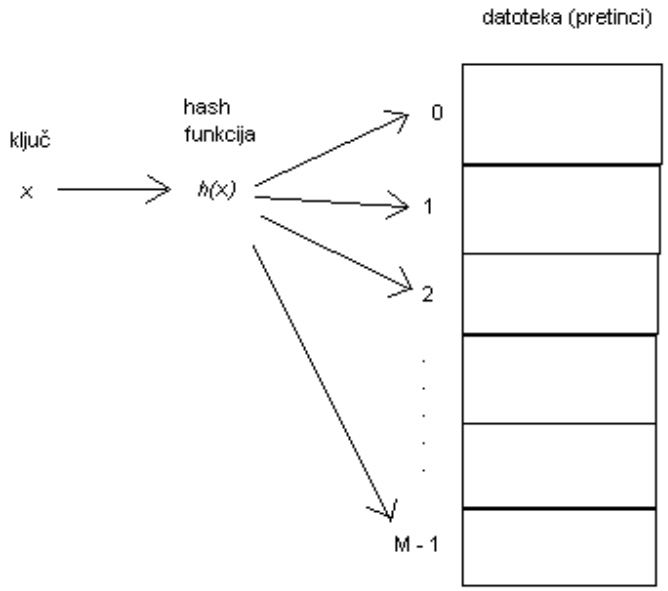
- *Pretraživanje pomoću indeksa*. Pretpostavljamo da je datoteka sortirana, te da osim nje postoji i pomoćna tablica (indeks) koja pobliže locira traženi zapis. Na primjer, indeks bi mogao sadržavati najmanju vrijednost ključa za svaki blok datoteke, te adresu svakog bloka. Najprije pretražimo indeks, te na osnovu informacija iz indeksa odredimo u kojem bloku datoteke bi se morao nalaziti traženi zapis. Zatim pretražimo samo taj jedan blok iz datoteke.



Raspršeno adresiranje (hashing)

Riječ je o nešto kompliciranijoj tehnici adresiranja, koja se najviše približava idealu direktnog pristupa. Kao i kod ostalih (jednostavnijih) tehnika adresiranja, i ovdje postoji pravilo kako je organizirana datoteka, kako se ključ transformira u adresu, te kako se obavlja postupak pretraživanja.

- Datoteka je podijeljena u dijelove (obično jednake veličine) koji se zovu *pretinci*.
- Transformacija ključa u adresu obavlja se pomoću «pseudoslučajne» funkcije *h( )*, koja zadanu vrijednost ključa *x* pretvara u redni broj pretinca *h(x)*.
- Postupak pretraživanja svodi se na ponovnu primjenu funkcije *h( )*. Dakle, zapis s vrijednošću ključa *x* tražimo tamo gdje smo ga spremali, dakle u pretincu *h(x)*.



Da bi ideja raspršenog adresiranja dobro funkcionirala, važne su tri stvari:

- *Dobro dimenzionirana datoteka.* Broj pretinaca *M* te kapacitet pretinca *C* moraju biti pogodno odabrani u odnosu na ukupan broj zapisa *N*. Ukupni kapacitet $M \times C$ treba biti nešto veći od *N*.
- *Dobro odabrana funkcija h( ).* Ta funkcija mora biti definirana za sve vrijednosti ključa *x*, mora pogađati sve redne brojeve pretinaca *0, 1, 2, ... M-1,* te mora *ravnomjerno* raspoređivati zapise u pretince.
- *Dobro riješen problem prelјeva.* Ako se neki pretinac ipak prepuni, tada moramo imati definiran postupak što ćemo dalje raditi sa zapisima koji bi trebali ići u taj prepunjeni pretinac. Postupak s preljevom ne smije odviše zakomplicirati postupak pretraživanja.

Glavna *prednost* raspršenog adsresiranja je (skoro) direktan pristup. Naime, da bi pristupili zapisu s vrijednošću ključa *x*, dovoljno je izračunati *h(x)* i pretražiti odgovarajući pretinac.

Glavna *mana* raspršenog adresiranja je što ono ne čuva sortirani redoslijed zapisa po vrijednostima ključa. Također, moguće su degradacije performansi zbog prepunjenja ili preljeva, te su potrebne povremene reorganizacije datoteke.