Dizajn i analiza algoritama

Lekcija 2

leto 2019/2020

Prof. dr Branimir M. Trenkić

Rešenje domaćeg zadatka

- Problem ***najvećeg*** elementa niza
  - ***Naći indeks*** (prvog) najvećeg elementa niza
  - ***Naći vrednost*** najvećeg elementa niza

- Ako je dat niz ***a*** od ***n*** neuređenih brojeva $\boldsymbol{a_1}$, $\boldsymbol{a_2}$, ..., $\boldsymbol{a_n}$, treba naći ***indeks*** (prvog) najvećeg elementa tog niza

Rešenje domaćeg zadatka

- Problem najvećeg elementa niza

- Primer ***instance problema*** najvećeg elementa niza:

  a = [17, 24, 8, 24, 12, 10, 24];   (n = 7)

- ***Rešenje*** ove ***instance***: ***2***

Rešenje domaćeg zadatka

- Problem najvećeg elementa niza
- ***Algoritam*** u pseudo kodu:

```
// Ulaz: niz a, broj elemenata n niza a
// Izlaz: indeks najveceg elementa niza a
algorithm max(a, n)

 m = a[1];       // najveci element nadjen do sada
 j = 1;          // indeks najveceg elementa

 i = 2;
 while (i <= n) do
        if (m < a[i]) then      // nadjen veci element od
                                // privremeno najveceg
             m = a[i];               // zapamti veci broj
             j = i;                  // i njegov indeks
        i = i + 1;          // predji na sledeci element

 return j;       // vrati indeks najveceg elementa
```

Definicije

- ***Osnovne teme*** izučavanja sledeći ***postupci***:
  - Dizajn algoritama
  - Analiza algoritama
- ***Dizajn algoritama***: Formulisanje (pisanje) niza koraka koji čine ***korektan algoritam*** za dati problem i ***dokaz korektnosti*** datog algoritma
- ***Analiza algoritama***: Utvrđivanje koliko računarskih resursa troši taj algoritam (vreme, memorija, komunikacione resursa,...)
  - Obično se izražava u funkciji od veličine instance

Dizajn algoritama

- ***Postupak*** u kome se posvećuje posebna pažnja da algoritam ***zadovolji*** opšte ***kriterijume dobrih algoritama***
- ***Korektnost*** (tačnost, ispravnost) i ***efikasnost***
  - Količina računarskih resursa – procesorsko vreme, memorija, komunikacioni resursi,....
- ***Jednostavnost*** i ***jasnoća***
- Nažalost, u praksi vrlo često ***kontradiktorni zahtevi***

Dizajn algoritama

- ***Postupak*** dobijanja dobrih algoritama (za većinu problema) – iz ***tri koraka***:

1. ***Definisanje problema - Eliminacija*** nevažnih i remetilačkih ***detalja*** u problemu koji se rešava – ***primena apstrakcije***
   – Prikazuje se samo ***suština problema*** u vidu apstraktnih ***matematičkih pojmova***
2. Pošto je problem jasno matematički definisan – pristupa se ***pisanju algoritma*** i ***dokazivanju njegove ispravnosti***
3. Na kraju se sprovodi ***analiza algoritma***

Dizajn algoritama

- ***Dizajn*** dobrih algoritama za složene probleme ***zahteva od programera*** ***visoki stepen*** ***obrazovanja*** ***i*** ***iskustva***
- Dobro poznavanje
  - Domen problema
  - Računarstvo
  - Matematika
- Čak ni to ***ne garantuje uspeh*** – dizajn algoritama je ***kreativan posao*** za koji ***ne postoje šabloni***

Dizajn algoritama

- Za dobre algoritme ***ne postoji*** “čarobna formula”
- ***Algoritamske paradigme*** – ***standardne metode*** čija primena garantuje dobre algoritme
  - ***Iterativni/Rekurzivni pristup***
    - Rešenje problema – ***ponavljanjem određenog postupka***
  - “***Podeli pa vladaj” pristup***
    - ***Particija problema*** na ***nezavisne pod-probleme***
    - Zatim nastupa rekurzivno ***rešavanje podproblema***, kako bi se njihovim ***spajanjem*** dobilo ***rešenje polaznog problema***
    - *Merge* sort - sortiranje objedinjavanjem

Dizajn algoritama

- Za dobre algoritme ***ne postoji*** "čarobna formula"
- ***Algoritamske paradigme*** – ***standardne metode*** čija primena garantuje dobre algoritme
  - ***"Pohlepni" algoritmi***
    - *Lokalno najbolje* (optimalno) *rešenje*
    - Problemi moraju imati ***optimalnu strukturu***
      - Optimalno rešenje problema se sastoji od optimalnih rešenja potproblema
    - Primer: problem prebrojavanja novca

Dizajn algoritama

- Primer: ***problem prebrojavanja novca***
- Pretpostavite da treba da izbrojite određenu sumu novca u dolarima ***korišćenjem najmanjeg mogućeg broja*** novčanica i novičića (apoena)
- U ovom slučaju se može primeniti ***pohlepni algoritam*** koji u svakom koraku uzima najveći mogući apoen takav da ukupna do tada izbrojana suma ne prelazi zadatu
- Primer: $6.39 ($5, $1, 25c, 10c,1c)
- Rešenje: 8 novčića ($5 - 1, $1 - 1, 25c -1 , 10c - 1,1c – 4)

Dizajn algoritama

- Primer: ***problem prebrojavanja novca***
- Primer: 15 perpera (10p, 7p, 1p)
- Rešenje: ***6 novčića*** (10p - 1, 1p – 5)
- Da li je rešenje optimalno?

Dizajn algoritama

- Za dobre algoritme ***ne postoji*** "čarobna formula"
- ***Algoritamske paradigme*** – ***standardne metode*** čija primena garantuje dobre algoritme
  - ***Dinamičko programiranje***
    - Particija glavnog problema na podprobleme (eng. *overlapping subproblems*), ali koji nisu nezavisni
    - Pod-problem i njegovo optimalno rešenje
    - Koristimo ga u nalaženju rešenja celokupnog problema
  - ***Randomizacija***

Dizajn algoritama

- ***Randomizacija***
- **Ulaz**: Niz od $n \geq 2$ elemenata, u kome je polovina ‘*a*’ elemenata, a drugu polovinu čine ‘*b*’ elementi.
- **Izlaz**: Pronađeno neko ‘*a*’ u nizu

```
nadjiA_LV(A, n)
{
   repeat
      Slučajan izbor jednog od n
      elementa.
   until 'a' je nadjeno
}
```

\+ Postiže uspeh sa verovatnoćom 1
\- Vreme izvršavanja varira

```
nadjiA_MC(A, n, k)
{
   i=0
   repeat
      Slučajan izbor jednog od n
      elementa.
      i = i + 1
   until i=k ili 'a' je nadjeno
}
```

\- Ne garantuje uspeh
\+ Vreme izvršavanja konstantno

Ispravnost iterativnih algoritama

- ***Iterativni algoritmi***?
- U dokazu ispravnosti iterativnog algoritma treba pokazati da ***petlja zadovoljava*** ***dve vrste uslova***:

1. ***Izlazni uslov*** – ***neophodan uslov*** za svaku petlju koji ***mora biti zadovoljen u nekom trenutku***, kako bi se ***petlja sigurno završila***
2. Uslov(i) čija se (istinitosna)***vrednost ne menja*** izvršavanjem petlje - ***invarijanta petlje***
   - Invarijanta petlje - ***formalni iskaz*** koji je ***tačan*** ***pre svake iteracije*** petlje

Ispravnost iterativnih algoritama

- Ispitivanje (***dokazivanje***) ***da li je neki iskaz - invarijanta petlje***
- ***Dokaz*** sličan ***matematičkoj indukciji***
- Dakle, ***treba dokazati***:

1. Invarijanta petlje je ***tačna*** ***pre prve iteracije*** petlje (***bazni slučaj***)
2. Ako je invarijanta petlje tačna ***pre neke iteracije*** (***induktivna hipoteza***) – ona je tačna i ***nakon te iteracije***, tj. pre izvršavanja sledeće iteracije

Primer dokaza ispravnosti

- Problem izračunavanja najvećeg zajedničkog delioca dva cela broja – ***NZD(x, y)***

```
// Ulaz:  pozitivni celi brojevi x i y
// Izlaz: nzd(x,y)
algorithm gcd(x, y)

   d = min{x,y};
   while ((x % d != 0) || (y % d != 0)) do
      d = d - 1;

   return d;
```

Prvi korak – izlazni uslov

- ***Petlja će se sigurno završiti***
- Jer će se uzastopnim smanjivanjem vrednosti ***d*** doći do slučaja ***d = 1***
- Izlazni uslov ***while*** petlje će ***postati netačan***
  - 1 deli bilo koji ceo broj bez ostatka ***x % 1 = 0***

```
// Ulaz:  pozitivni celi brojevi x i y
// Izlaz: nzd(x,y)
algorithm gcd(x, y)

   d = min{x,y};
   while ((x % d != 0) || (y % d != 0)) do
      d = d - 1;

   return d;
```

Drugi korak – invarijanta petlje

- Prepoznata ***invarijanta*** ***while petlje*** za ovaj problem je uslov:

$$d \geq nzd(x, y)$$

Drugi korak – invarijanta petlje

- Uslov je zadovoljen ***pre prve iteracije*** jer je u tom slučaju:

  $d = min(x, y)$ a uvek je ***$min(x, y) \geq nzd(x, y)$***

- Prema tome ***uslov je tačan pre prve iteracije***

```
// Ulaz:  pozitivni celi brojevi x i y
// Izlaz: nzd(x,y)
algorithm gcd(x, y)

   d = min{x,y};
   while ((x % d != 0) || (y % d != 0)) do
      d = d - 1;

   return d;
```

Drugi korak – invarijanta petlje

- Ako uslov invarijantnosti važi pre neke iteracije – važi i posle nje
- Predpostavimo da uslov ***d* ≥ *nzd(x, y)*** ***važi pre iteracije***
- Predpostavimo da je ta ***while iteracija izvršena***
- Vrednost promenljive ***d*** je smanjena za ***jedan***
- ***Stara vrednost d*** nije bila delilac – pa kako je nakon iteracije smanjena za 1 zaključujemo da je pre iteracije bila ***striktno veća*** ***od nzd(x, y)***

Drugi korak – invarijanta petlje

- ***Zaključak***:
- ***Nova vrednost d*** koja je u toku tekuće iteracije smanjena za 1 – je ***opet veća ili jednaka nzd(x, y)*** pre sledeće iteracije
- Ovim je dokazano da je uslov ***d ≥ nzd(x, y)*** stvarno invarijanta za datu petlju

Drugi korak – invarijanta petlje

- ***Kada se while petlja završi***, ***d*** je delilac oba broja (x i y)
- Kako je ***po definiciji nzd(x, y) najveći takav*** delilac ***sledi***

$$d \leq nzd(x, y)$$

- Pošto posle završetka petlje ***važi i invarijanta petlje,*** tj.

$$d \geq nzd(x, y)$$

- Možemo zaključiti da važi:

$$d = nzd(x, y)$$

Analiza algoritama

- ***Ocena efikasnosti algoritma*** na osnovu kriterijuma iskorišćenosti računarskih resursa
- Dragoceni ***računarski resursi***
  - ***Vreme!***
    - Skoro ekskluzivno proučavamo **vremensku složenost (vreme izvršavanja algoritma)**
  - Memorija
  - Mrežni resursi
  - …
- Vreme izvršavanja : (I) ***empirijski***; (II) ***analitički***

Vremenska složenost

- ***Empirijsko*** određivanje vremena izvršavanja algoritma

1. ***Napisati*** računarski ***program prema algoritmu***
2. ***Izmeriti njegovo vreme*** koliko radi na računaru

Vremenska složenost

- ***Empirijsko*** određivanje vremena izvršavanja algoritma
- ***Nedostaci*** ovog pristupa:
  - Ne može se oceniti ***efikasnost pojedinih delova***
  - Efikasnost zavisi od ***konkretnih ulaznih podataka*** (***benčmarking***)
  - Efikasnost zavisi od ***konkretnog računara*** na kome se meri

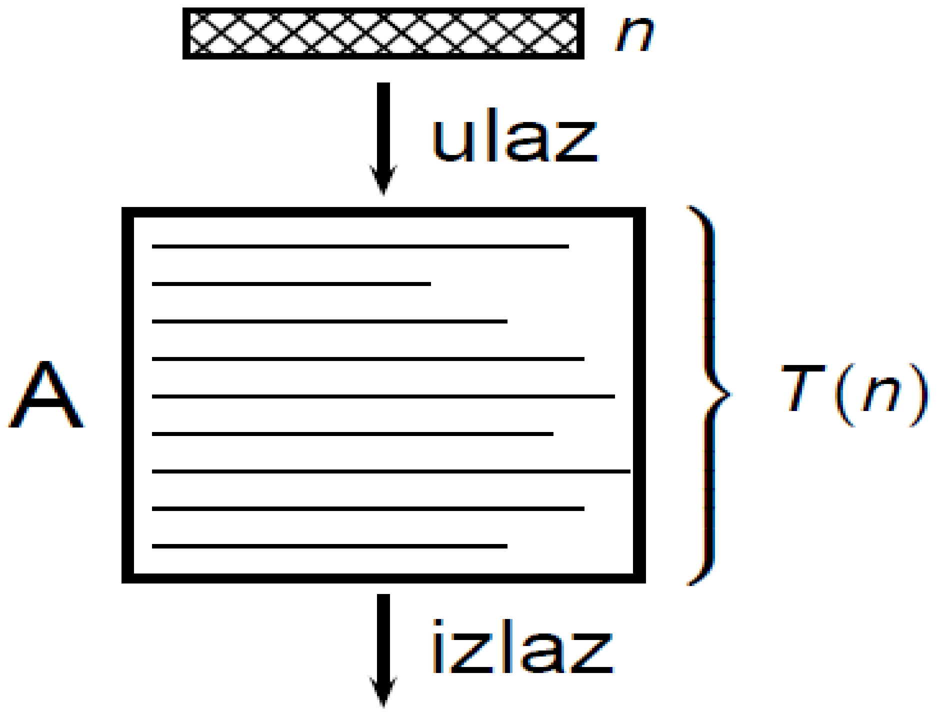
Vremenska složenost

- ***Analitičko*** određivanje vremena izvršavanja algoritma
- ***Prebrojati ukupan broj*** osnovnih ***koraka*** (***jediničnih instrukcija***) koji se izvršavaju u algoritmu
- ***Problemi analize***:
  - Šta su ***jedinične instrukcije***?
  - Njihov broj ***zavisi od broja*** ulaznih podataka
  - Njihov broj ***zavisi od prirode*** ulaznih podataka

Jedinične instrukcije

- Jedinične (osnovne, primitivne) instrukcije = ***naredbe*** čije je ***vreme*** izvršavanja ***konstantno***
  - *dodela vrednosti* promenljivoj
  - *poređenje vrednosti* dve promenljive
  - *aritmetičke operacije*
  - *logičke operacije*
  - *ulazno/izlazne operacije*
  - ...
- ***Pojednostavljenje***: ***jedinične instrukcije*** se izvršavaju za ***jednu jedinicu vremena***

Funkcija vremena izvršavanja



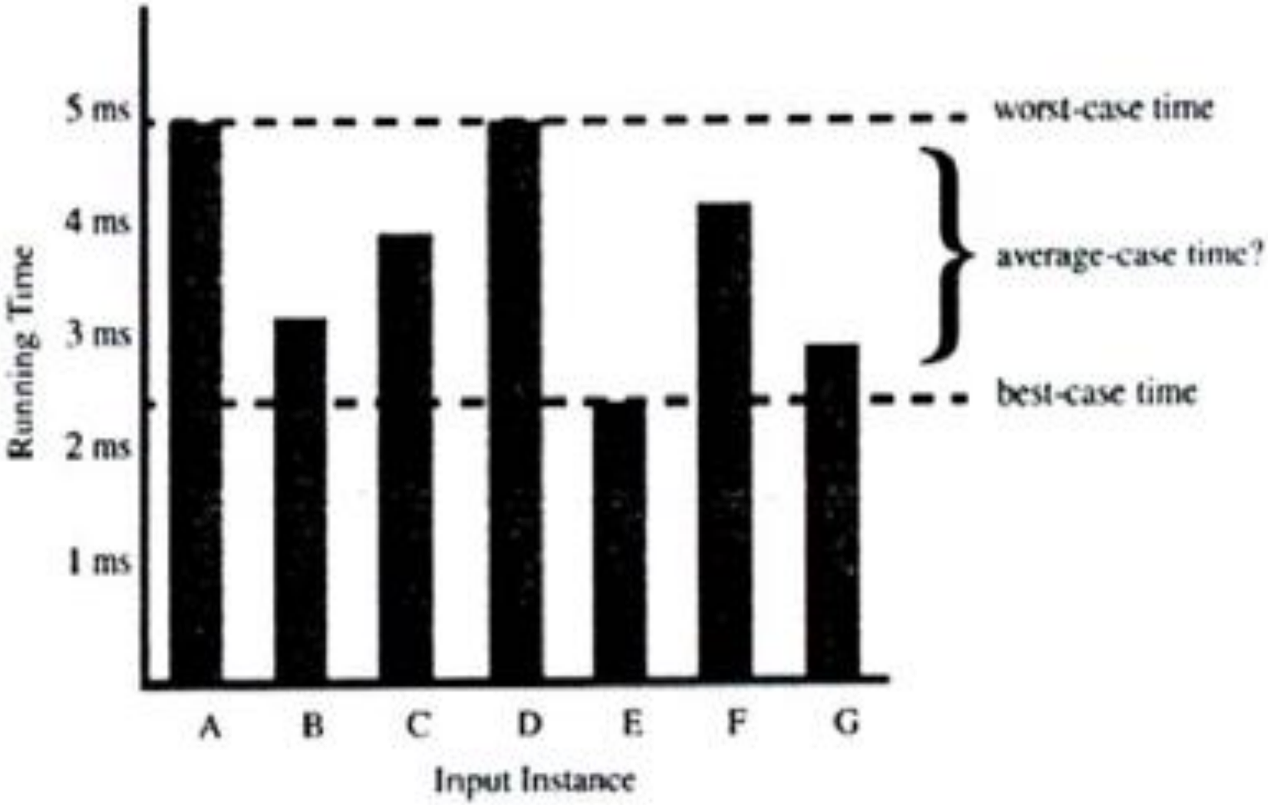
- ***T(n)*** - funkcija vremena izvršavanja
- ***n*** - broj ulaznih podataka (veličina ulaza)

Funkcija vremena izvršavanja

- Više pristupa:
- ***Pesimistički pristup***
  - Uzimamo u obzir ***najgori mogući slučaj*** što se tiče vremena izvršavanja
  - Posmatra se ***najgori slučaj ulaznih podataka***
- ***Probabilistički pristup***
  - Analiza ***srednjeg vremena*** izvršavanja nekog algoritma

Funkcija vremena izvršavanja

- Više pristupa:



Funkcija vremena izvršavanja

- ***Pesimistički pristup***: uzima se da se izvršava najveći mogući broj instrukcija
- ***Najgori slučaj izvršavanja***
- Funkcija ***T(n)*** – vreme izvršavanja u najgorem slučaju
- Primer:

```
n = 5;
repeat
   read(m);
   n = n - 1;
until ((m == 0) || (n == 0));
```

```
read(n);
repeat
   read(m);
   n = n - 1;
until ((m == 0) || (n == 0));
```

Funkcija vremena izvršavanja

- Funkcija ***T(n)*** – vreme izvršavanja u najgorem slučaju
- Primer:

```
n = 5;
repeat
   read(m);
   n = n - 1;
until ((m == 0) || (n == 0));
```

```
read(n);
repeat
   read(m);
   n = n - 1;
until ((m == 0) || (n == 0));
```

T = (iteracija)·5 + 1 ***T(n) = (iteracija)·n + 1***

Funkcija vremena izvršavanja

Konstrukcija	Vreme izvršavanja
Naredba serije S: P; Q;	$T_S = T_P + T_Q$
Naredba grananja S: **if** C **then** P; **else** Q;	$T_S = T_C + \max\{T_P, T_Q\}$
Naredba petlje S: (1) **while** C **do** P; (2) **repeat** P; **until** C; (3) **for** $i = j$ **to** k **do** P;	$T_S = n \cdot T_P$ n – najveći broj iteracija petlje

Benčmark podaci

- Postoji ***još jedan pristup*** u ocenjivanju efikasnosti algoritma
- Koristi se ***u slučajevima upoređivanja*** efikasnosti algoritama koji rešavaju ***isti problem***
- Formira se mala količina ***tipičnih ulaznih podataka*** – ***benčmark podaci***
- Benčmark podaci – smatraju se ***reprezentativnim*** za sve moguće ulazne podatke

Benčmark podaci

- Algoritami koji imaju dobre performanse ***za benčmark podatke*** podrazumeva se da će imati dobre performanse i ***za sve druge ulazne podatke***
- Benčmark analiza je u suštini ***empirijski postupak***

Primer 1

- ***Zamena vrednosti dve promenljive***

- Ako su ***date dve promenljive***, ***zameniti vrednosti tih promenljivih*** tako da nova vrednost jedne promenljive bude stara vrednost druge promenljive

Primer 1

- ***Zamena vrednosti dve promenljive***

```
// Ulaz:  promenljive x i y sa svojim vrednostima
// Izlaz: promenljive x i y sa zamenjenim vrednostima
algorithm swap(x, y)

   z = x;
   x = y;
   y = z;

   return x, y;
```

Primer 1

- ***Zamena vrednosti dve promenljive***

```
// Ulaz:  promenljive x i y sa svojim vrednostima
// Izlaz: promenljive x i y sa zamenjenim vrednostima
algorithm swap(x, y)

   z = x;
   x = y;
   y = z;

   return x, y;
```

$$T(n) = 1 + 1 + 1 + 1 = 4$$

Primer 2

- ***Najveći elemen niza***

- Ako je dat niz ***a*** od ***n*** neuređenih brojeva $\mathbf{a_1, a_2, . . . , a_n}$, treba naći indeks (prvog) najvećeg elementa tog niza

Primer 2

- ***Najveći elemen niza***

```
// Ulaz:  niz a, broj elemenata n niza a
// Izlaz: indeks najvećeg elementa niza a
algorithm max(a, n)

   m = a[1];  // najveći element nađen do sada
   j = 1;     // indeks najvećeg elementa

   i = 2;     // proveriti ostale elemente niza
   while (i <= n) do
      if (m < a[i]) then  // nađen veći element
         m = a[i];        // zapamtiti veći element
         j = i;           // ... i njegov indeks
      i = i + 1;  // preći na sledeći element niza

   return j;  // vratiti indeks najvećeg elementa
```

Primer 2

- ***Najveći elemen niza***

Vreme izvršavanja *max*

$$\begin{aligned} T(n) &= 1 + 1 + 1 + (n-1)(3+1) + 1 \\ &= 4 + 4(n-1) \\ &= 4n \end{aligned}$$

Domaći zadatak

Konstruisati algoritam koji konvertuje prirodan broj u njegov oktalni zapis. Dokazati korektnost tog algoritma.

Upitstvo:

Ulazni parametri: prirodan ***broj n***

Izlazni parametri: ***niz*** oktalnih cifara ***b*** broja n

Napomena: ***b[1]*** prva oktalna cifra s desna