

10 . Razvoj algoritama

Prevalili smo polovinu semestra i vreme je da napravimo malu rekapitulaciju pređenog i preostalog puta.

Podaci

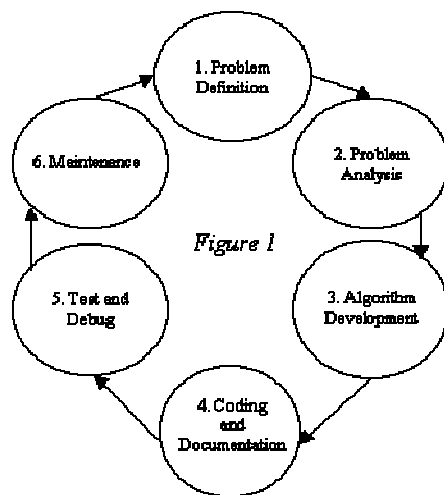
Iscrpno smo se bavili podacima, analizirajući različite vidove u kojima se oni javljaju (tekstualni, numerički, zvučni, grafički, video, itd.), analizirajući načine na koji se podaci memorišu u računarima (bitovi, bajtovi itd.), tipove podataka (int, char, float, pointer, itd...) i strukture podataka (nizovi, liste, stekovi, redovi, grafovi, heš tabele, datoteke), što sve zajedno čini osnovu za kompleksnu primene računara za rešavanje različitih problema u raznim domenima primene (tehnicima, biznisu, medicini, upravi, itd...)

Životni ciklus softvera - Software Development Life Cycle (SDLC)

Razvoj i proizvodnja softvera je jedna od danas najatraktivnijih poslovnih sfera. Setite se samo da je vlasnik Microsofta dugo godina prvi na listi najbogatijih ljudi na svetu, a njegova kompanija prodaje softverske proizvode (Windows operativne sisteme i drugo) u stotinama miliona primeraka širom sveta.

Razvoj i proizvodnja softvera je složen zadatak, koji zahteva visoko kvalifikovane ljude, vrhunsku organizaciju posla, vrhunski marketing i menadžment.

Mi ćemo se detaljnije baviti fazama kroz koje svaki softverski proizvod prolazi od ideje o proizvodu do njegove realizacije, upotrebe, zastarevanja i nestajanja. Sledeća slika ilustruje taj ciklus.



Slika 10.1 Životni ciklus softvera

Svaka od faza zahteva posebnu metodologiju, posebno obučene ljude za tu fazu i posebne tehnike rada.

Naše težište biće na Fazi 3 – Razvoju algoritama jer to i jeste naša osnovna tema. Međutim moraćemo se, bar delimično, baviti i svim drugim fazama.

Pojam algoritma (programa)

Podsetićemo se na neke osnovne pojmove koje smo definisali u poglavlju 9.

Dali smo jednu intuitivnu definiciju algoritma (Efektivna procedura kojok se rešava neki problem u konačnom broju koraka). Veza između programa, algoritma i podataka najbolje se odslikava formulom (Niklaus Wirth, autor jezika Pascal):

$$\text{Program} = \text{Algoritam} + \text{Podaci}$$

gde se pod programom podrazumeva algoritam izražen u nekom konkretnom programskom jeziku (C, C++, C#, Java, Cobol, Basic, itd.).

Teorema o programskoj strukturi

Analizirajući razne algoritme Bohm i Jacopini su uočili da se svaki algoritam može izraziti sa sledeće tri vrste koraka.

1. Sekvenca - proces
2. Odluka - selekcija
3. Ponavljanje - repeticija, iteracija, ciklus, petlja

O ovim osnovnim algoritamskim strukturama bilo je reči u poglavlju 9. Ovde se samo podsećamo na njih jer one predstavljaju korake koji se mogu naći u svakom algoritmu.

Izložićemo sada jednu opštu metodologiju (strategiju) za razvoj algoritama. Naravno, ovo nije jedina metodologija, ali se pomoću nje jasno ilustruje proces razvoja algoritama.

Strategija razvoja algoritama

Korak 1: Istraživački korak

- Identifikovati procese
- Identifikovati glavne odluke
- Identifikovati ponavljanja
- Identifikovati varijable

Korak 2: Izrada preliminarnog (grubog) algoritma

- Izrada algoritma "višeg nivoa"
- Proći kroz algoritam misaonom simulacijom. Ako simulacija otkrije probleme ispraviti algoritam.

Korak 3: Izrada finalnog (detaljnog) algoritma

- Do detalja razraditi "grubi" algoritam napravljen u koraku 2.
- Grupišite procese koji se mogu grupisati
- Grupišite varijable koje se mogu grupisati
- Testirajte algoritam simulacijom korak po korak.

Metode prikazivanja algoritama

Sada ćemo se baviti sa nekoliko najrasprostranjenijih tehnika (načina) za prikazivanje algoritama. Jedan od načina smo već koristili u primeru »Stabilni parovi«. Reč je o opisnom prikazu algoritma korišćenjem prirodnog (srpskog, engleskog ili nekog drugog) jezika.

Sledi lista tehnika koje ćemo proučiti:

- Prirodni jezik (koračna forma)
- Dijagrami toka (Flowcharts)
- Pseudo kod (Pseudo cod)
- Nasi-Šnajderman dijagrami (Nassi-Sneiderman – N/S diagrams)
- Džeksonovi Strukturalni dijagrami (Jackson Structured Diagrams – JSD)

Za svaku od navedenih tehnika, pored opisa, daćemo i kratak komentar o uzajamnim prednostima i nedostacima.

Prirodni jezik

Ovom tehnikom se algoritam prikazuje kao niz brojem označenih koraka. Svaki korak se sadrži jednu ili više rečenica prirodnog jezika (srpskog, na primer) kojim se opisuje proces (operacija) koju u tom koraku treba izvršiti.

Evo jednog jednostavnog primera:

Problem: Prikazati (na monitoru računara, na primer) dvostruku vrednost broja koji je predhodno unet u računar (pomoću tastature, na primer).

Algoritam u prirodnom jeziku:

1. Tražiti od korisnika da uz pomoć tastature unese broj.
2. Učitati broj koji korisnik ukuca na tastaturi.
3. Pomnožiti učitani broj sa brojem 2.
4. Prikazati rezultat operacije iz koraka 3 na monitoru računara.

Sekvenca se prikazuje jednostavno nizanjem koraka jedan za drugim, pri čemu svaki korak dobija redni broj u redosledu kojim treba da se izvršavaju.

Odluka se prikazuje opisom uslova i uputstvom na koji korak se ide za slučaj da je uslov ispunjen, a na koji korak kada uslov nije ispunjen. Na sličan način možemo izvršiti i selekciju iz više mogućih rezultata nekog uslova.

Repeticija – iteracija se postiže tako što se izvršenje nastavlja nekim korakom koji ima manji redni broj od onog u kojem se postavlja uslov repeticije.

Prednosti prirodnog jezika:

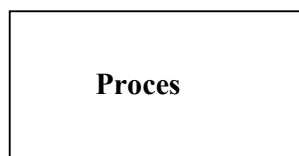
- Jednostavan za učenje, jer se ionako služimo prirodnim jezikom.

Nedostaci:

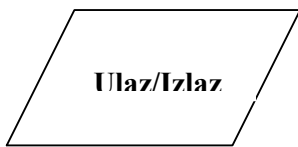
- Koraci su predugački jer se mora koristiti puno reči za njihov opis.
- Prevođenje iz prirodnog jezika u kompjuterski jezik može biti teško jer za razliku od prirodnih jezika, kompjuterski (programski) jezici imaju vrlo precizno definisanu sintaksu (gramatiku) i semantiku (značenje).

Dijagrami toka (Flowcharts)

Ova tehnika koristi niz grafičkih simbola povezanih usmerenim linijama (strelicama) kojima se pokazuje sekvenca (niz) u kojoj će koraci opisani grafičkim simbolima biti izvršavani. Unutar grafičkog simbola upisuje se prirodnim jezikom (ili pseudo kodom) proces (operacija) koju treba izvršiti. Grafički simboli koji se najčešće koriste za prikaza algoritama dijagramom toka su sledeći:



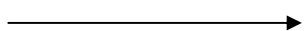
Pravougaonik – sadrži opis procesa (naredbi) koje se izvršavaju jedna za drugom u redosledu kako su napisane. Nakon što se naredbe (jedna ili više) izvrše nastavlja se sa sledećim grafičkim simbolom koji sledi. Dakle, pravougaonik odgovara programskoj sekvenci.



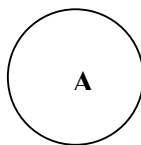
Romb – predstavlja ili ulaznu ili izlaznu naredbu kojim se podaci unose u kompjutersku memoriju (ulaz) ili iz kompjuterske memorije prikazuju na nekom spoljnjem uređaju (izlaz). Nakon što se naredba ulaza (izlaza) izvrši, algoritam se nastavlja naredbom iz sledećeg simbola.



Dijamant – predstavlja proces donošenja odluke. Odluka sadrži pitanje koje obično ima dva odgovora, DA ili NE, pa se nakon utvrđivanja koji od tih odgovora je tačan algoritam nastavlja jednom od dve moguće putanje koje slede (izlaze) iz ovog simbola.



Strelica – služi za povezivanje grafičkih simbola u smeru njihovog logički sukcesivnog izvršavanja.

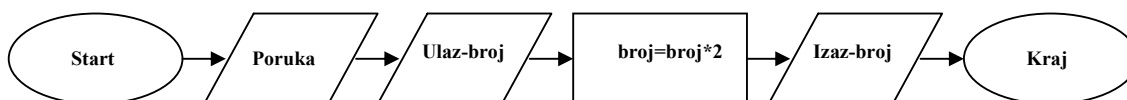


Krug – koristi se za povezivanje delova dijagrama toka. Kada dijagram toka ne može da bude prikazan na jednoj stranici (što je čest slučaj) ovaj znak se koristi za povezivanje delova dijagrama koji se nalaze na različitim stranicama.



Oval (elipsa) – se koristi za označavanje početka i kraja algoritma. Oval za start pokazuje gde algoritam započinje i obično sadrži reč “Start”, a oval za kraj pokazuje gde se algoritam završava i obično sadrži reč “Kraj”.

Evo kako bi algoritam iz predhodnog primera (prikaz dvostruke vrednosti učitano broj) bio prikazan ovom tehnikom:



Kako se prikazuje repeticija dijagramima toka?

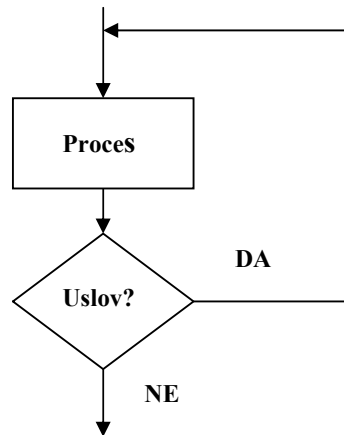
Kako smo već videli dijagrami toka imaju simbole za sekvencu (pravougaonik) i odluku (dijamant). Ali, kao prikazati repeticiju (do...until i while tipa na primer).

To se postiže kombinacijom simbola za sekvencu i odluku.

Tako se repeticija tipa:

```
do {  
  proces;  
}while(uslov)
```

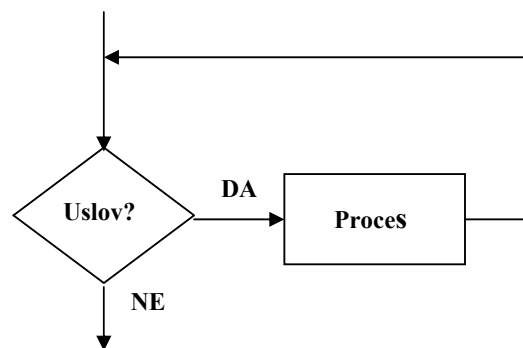
postiže sledećim dijagramom:



Slično tome repeticija:

```
while (uslov) {  
  proces;  
}
```

može da se izrazi dijagramom toka na sledeći način:



Prednosti dijagrama toka:

- Grafička prezentacija algoritma olakšava pronalaženje logičkih grešaka u algoritmu (“slika vredi hiljadu reči” – stara kineska poslovice).
- Postojanje znaka za povezivanje (krug) omogućava veoma lako dodavanje novig delova algoritma.

- Algoritam prikazan grafički lakše se prevodi u programski kod nego što je to slučaj sa prirodnim jezikom ili pseudokodom.

Nedostaci:

- Treba upamtiti značenje grafičkih simbola.
- Kada postoji mnogo koraka odluke i ponavljanja, dijagram toka može da se pretvori u vrlo zamršenu mrežu iz koje je teško sačiniti valjan programski kod.

Pseudo kod

Ova tehnika je veoma slična prirodnom jeziku s tom razlikom što se umesto prirodnog jezika koristi neki drugi jezik (sličan prirodnom) koji ima precizniju sintaksu, koristi manji broj unapred zadatih reči, pa je time lakše definisati i semantiku (značenje) rečenica koje se formiraju u tom jeziku. Takav jezik često podseća i na programske jezike, to jest on predstavlja kompromis između prirodnog i programskog jezika. Kompromis u smislu da je dovoljno jednostavan i razumljiv za čoveka, a istovremeno pogodan za dalje transformisanje algoritma u program. Rcimo takav jezik može sadržati reči kao što su: `display` (za prikaz poruke na monitoru), `read` (za učitavanje podatka), kao i simbole `+`, `-`, `*`, `/`, `=` za korišćenje u matematičkim formulama itd.

Tako naš predhodni primer algoritma može pseudo kodom biti prikazan kako sledi:

1. **display** poruka
2. **read** broj
3. rezultat = broj*2
4. **display** rezultat

Prednosti:

- Jednostavan za učenje skoro kao i kod prirodnog jezika.
- Lakši za prevođenje u programski jezik, jer kao što je rečeno jako podseća na stvarne programske jezike.

Nedostaci:

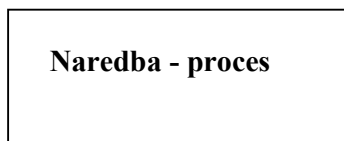
- Ova tehnika se oslanja na poznavanje takozvanih imperativnih (proceduralnih) jezika, pa za one koji se prvi put sreću sa ovom vrstom jezika može biti malo zbunjujuće – meša se prirodni i simbolički jezik.

Nasi-Šnajdermanovi dijagrami (N/S dijagrami)

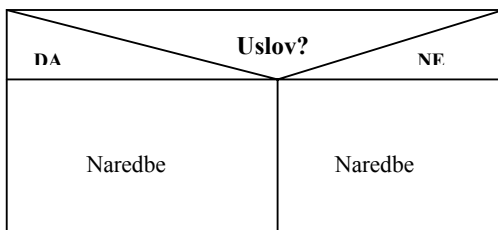
I kod Nasi-Šnajdermanovih algoritama se koriste grafički simboli kao i kod dijagrama toka, ali se ovde čitav algoritam stavlja u jedan jedini pragougaonik (boks). Simboli (koraci algoritma) se izvršavaju počev od prvog simbola na vrhu boksa i nastavljaju redom do poslednjeg na prikazanog dnu boksa. Svaki simbol sadrži ili prirodnim jezikom ili pseudokodom prikazane naredbe (processe).

Postoje tri vrste simbola: za sekvencu, za odluku i za repeticiju, u skladu sa teoremom o programskoj strukturi.

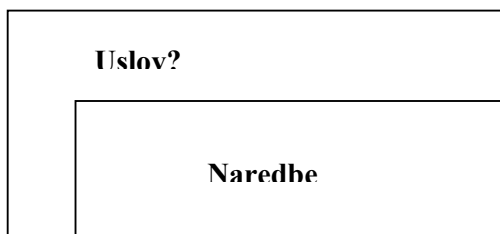
Simboli su:



Pravougaonik – koristi se za naredbe koje se izvršavaju jedna za drugom (sekvenca). Kada se sve naredbe iz pravouganika izvrše algoritam se nastavlja sledećim N/S simbolom.

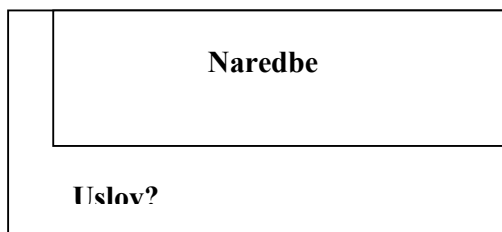


Simbol za odluku se sastoji od dela u kojem se nalazi uslov i dva pravougaonika koji sadrže alternativne naredbe ako je uslov ispunjen (DA) i ako nije (NE). Kada se izvrši odgovarajuća naredba algoritam se nastavlja sledećim N/S simbolom.



Repeticija se sastoji od uslova i naredbe (ili niza naredbi) koje se izvršavaju sve dok je uslov ispunjen (while ciklus). Kada uslov nije ispunjen algoritam nastavlja rad sledećim N/S simbolom.

Repeticija za slučaj do...while ciklusa izgleda ovako:



U ovom slučaju, za razliku od predhodnog, Naredbe će biti izvršene pre testiranja da li je uslov ispunjen. Ako jeste ponavlja se izvršavanje Naredbi, a ako nije algoritam nastavlja rad sledećim N/S simbolom.

Evo kako bi izgledao naš standardni primer korišćenjem N/S dijagrama:

display poruka
read broj
rezultat=broj*2
display rezultat

Prednosti N/S dijagrama:

- Grafička prezentacija algoritma olakšava pronalaženje logičkih grešaka u algoritmu (kao kod dijagrama toka).
- Lakše se prevodi u programski kod nego dijagram toka. Tri programske strukture (sekvenca, odluka i repeticija) zastupljene su u svim programskim jezicima.
- Pošto nema sterlica kao kod dijagrama toka ne mogu se kreirati zamršene strukture, već algoritam “glatko” sledi logiku rešenja.

Nedostaci:

- Moraju se pamtiti grafički simboli koji predstavljaju sekvencu, odluku i repeticije.
- Otežano je umetanje novih koraka u već sačinjeni algoritam, a što je bilo lako kod dijagrama toka (korišćenjem simbola za konekciju – kruga). S druge strane nastavljanje algoritma na sledećoj strani je jednostavno – nacrtate novi boks a stranu obeležite brojem 2, itd.

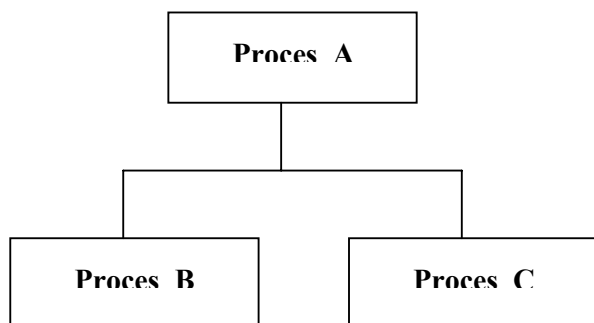
Džeksonovi strukturalni dijagrami (Jackson Structured Diagrams) – JSD dijagrami

Džeksonovi dijagrami slede ideju o podeli problema na niz podproblema manje složenosti (strategija poznata kao “divide and conquer” – podeli pa vladaj).

Ali oni takođe zadovoljavaju i teorem o programskoj strukturi, to jest JSD dijagrami imaju grafički simbol za sve tri osnovne komponente algoritma – sekvencu, odluku i repeticiju.

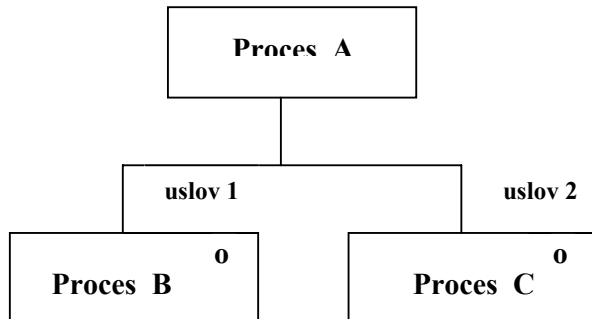
Sledi opis načina na koji se prikazuju ove komponente.

a) Sekvenca



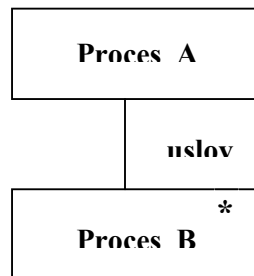
Ovde je proces A složen proces koji se sastoji od prostih procesa B i C. Da bi se obavio proces A potrebno je najpre obaviti proces B a onda proces C. Podproces se, dakle, izvršavaju sleva udesno jedan za drugim. Naravno proces A je mogao biti jedan jednostavan proces koji ne zahteva dalju podelu.

b) Odluka (selekcija)



Odluka se označava tako što se proces A deli na procese B i C, ali uslovno, tako da se proces A ispunjava bilo izvršavanjem procesa B (kada je uslov 1 ispunjen) bilo izvršavanjem procesa C (kada je uslov 2 ispunjen). Uočite mali kružić u gornjem desnom uglu pravougaonika procesa B i C. Ti se kružići koriste da se pravougaonik odluke razlikuje od pravougaonika sekvence.

c) Repeticija



Repeticija – iteracija je složena komponenta koja se izvršava ponavljanjem nekog procesa nula ili više puta u zavisnosti da li je uslov ispunjen ili ne.

Evo kako bi izgledao naš primer prikazan JSD dijagramom:



Prednosti:

- Grafička prezentacija algoritma olakšava pronalaženje logičkih grešaka u algoritmu (kao kod dijagrama toka).
- Jednostavniji grafički simboli nego kod dijagrama toka i N/S dijagrama.
- Sledi logiku rešenja podelom na podprobleme.

Nedostaci:

1. Ne uočavaju se lako odluke i repeticije.
2. Prevođenje u programski kod je nešto složenije – mora se voditi računa o redosledu izvršavanja procesa.

Pitanja

1. Nabrojite faze u životnom ciklusu softvera.
2. Definišite algoritam.
3. Objasnite zašto se može smatrati da je Program=Algoritam+Podaci.
4. Kako glasi teorema o programskoj strukturi?
5. Navedite korake u nekoj od strategija za razvoj algoritama.
6. Navedite nekoliko tehnika za prikaz algoritama.
7. Koje su osnovne komponente (grafički simboli) dijagrama toka?
8. Koje su osnovne komponente (grafički simboli) N/S dijagrama?
9. Koje su osnovne komponente (grafički simboli) JSD dijagrama?
10. Navedite komparativne prednosti i nedostatke algoritama izraženih: prirodnim jezikom, dijagramima toka, pseudokodom, N/S dijagramima, JSD dijagramima.