PT&SCHE

Introduction of part-time and short cycle studies in Serbia

Funded by the Erasmus+ Programme of the European Union

This project has been funded with support from the European Commission. This publication [communication] reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

UNIVERZITET
METROPOLITAN
BEOGRAD

KI102 - OSNOVE PROGRAMIRANJA

Implementacija algoritama

Lekcija 04

PRIRUČNIK ZA STUDENTE

KI102 - OSNOVE PROGRAMIRANJA

Lekcija 04

IMPLEMENTACIJA ALGORITAMA

UVOD

U okviru ove lekcije naučićemo sledeće:

Jedan način razvrstavanja algoritama je po metodologiji projektovanja ili primenjenom obrascu, čime se bavi oblast koja se naziva algoritamske strategije.

Postoji više algoritamskih strategija koje se primenjuju kada je potrebno razviti algoritam za neki problem. Ipak, ne postoje garancije da će neka strategija dati tačno rešenje (a iako je rešenje tačno ne mora biti efikasno).

U nastavku ćemo se baviti osnovnim strategijama za rešavanje problema: gruba sila, podeli i osvoji, dinamičko programiranje i pohlepni algoritam.

Poglavlje 1

Dijagram toka (flowchart)

OSNOVNI ELEMENTI DIJAGRAMA TOKA

Osnovni elementi su operatori dodele, učitavanja i štampanja, uslovni operator i petlja

Dijagrami toka (flowchart) se koriste sa ciljem da se vizuelno prikaže tok ili kotrola nekog algoritma kako se on izvršava korak po korak.
Dijagrami toka se sastoje iz sledećih komponenti:

- Početna ovalna labela pod nazivom START
- Sekvenca pravougaonika koje sadrže operacije algoritma
- Strelice koje označavaju redosled u kojem se izvršavaju operacije algoritmu
- Završna ovalna labela pod nazivom STOP ili END

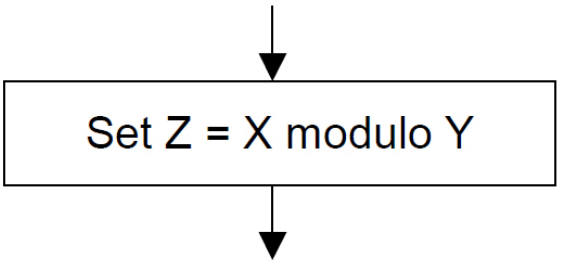
Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Algoritmi se uglavnom koriste za matematičke i računarske programe, dok se dijagrami toka uglavnom koriste za opisivanje različite vrste procesa: poslovnih, edukacionih, ličnih, ali naravno i algoritama.
Osnovni elementi dijagrama toka su:

- operacija dodele
- učitavanje podataka
- štampanje podataka
- uslovni operator
- operator ponavljanja (petlja)

Operacija dodele

Operacija dodele postavlja vrednost nekoj promenljivoj ili umesto stare promenljivoj dodeljiuje novu vrednost. Ova operacija je u ddijagramu toka(blok dijagramu) predstavljena korišćenjem pravougaonika u kome se navodi operator dodele (=). Primer:



Slika 1.1 Operator dodele vrednosti

UČITAVANJE I ŠTAMPANJE PODATAKA

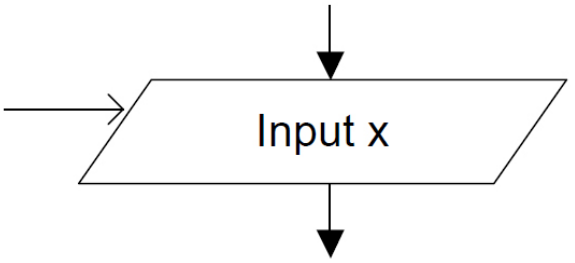
Operacija učitavanja postavlja promenljivu na vrednost koju je uneo korisnik. Izlazni operator prikazuje vrednost odgovarajuće promenljive ili štampa poruku namenjenu korisniku algoritma.

Ulaz (učitavanje podataka)

Operacija učitavanja postavlja promenljivu na vrednost koju je uneo korisnik.

- U dijagramu toka je ova operacija predstavljena korišćenjem paralelograma (romba) i strelice koja sa leva pokazuje ka rombu.
- Unutar romba je navedena promenljiva koja se inicijalize sa vrednošću koju unese korisnik.

Primer (Slika-2):



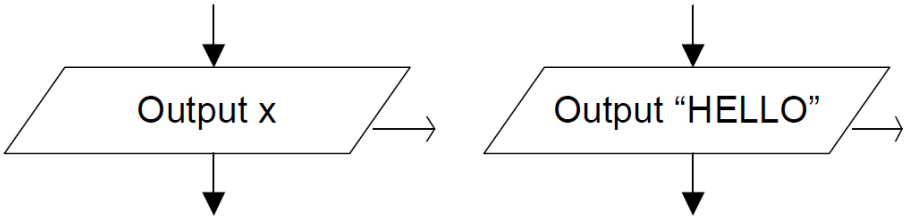
Slika 1.2 Učitavanje podataka

Izlaz (štampanje podataka)

Izlazni operator prikazuje vrednost odgovarajuće promenljive ili štampa poruku namenjenu korisniku algoritma.

- Predstavljen je korišćenjem paralelograma (romba) sa malom strelicom koja je od romba usmerena u desno
- Unutar romba je navedena promenljiva ili poruka koja se štampa na standardnom izlazu.

Primer (Slika-3):



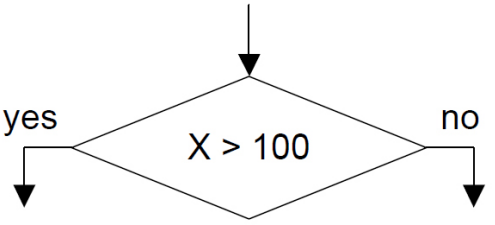
Slika 1.3 Štampanje vrednosti promenljive i štampanje poruke

Napomena. Alternativno, ulazni i izlazi operatori su predstavljeni korišćenjem trapeza, pri čemu je kod ulaznog operatora veća gornja osnovica, a kod izlaznog operatora je veća donja osnovica trapeza.

USLOVNI OPERATOR

Uslovni operator određuje smer u kome će teći vaš algoritam

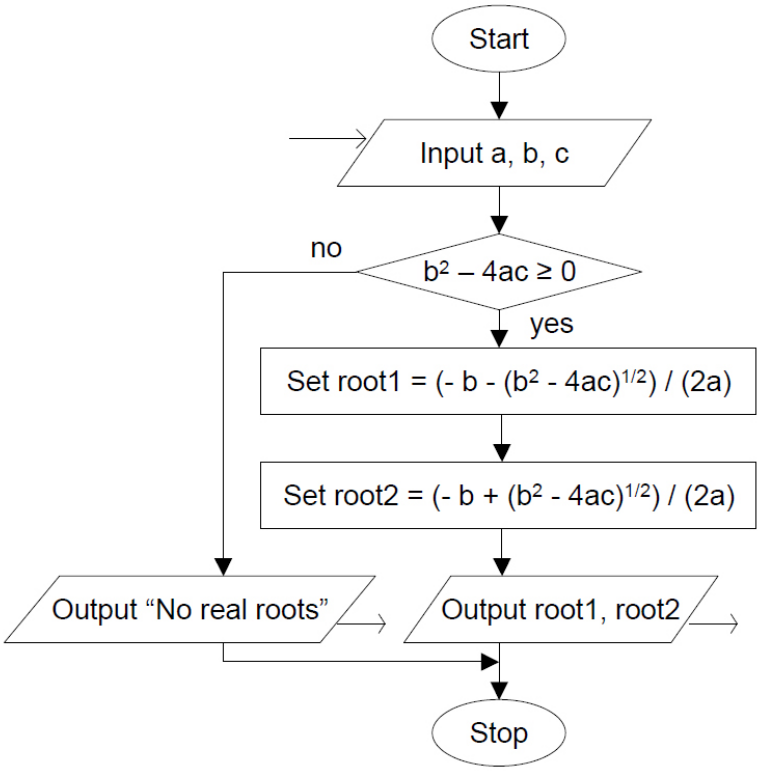
Uslovni operator određuje smer u kome će teći vaš algoritam. U dijagramu toka je predstavljen korišćenjem dijamanta u kome se nalazi uslov provere, Slika-4.



Slika 1.4 Uslovni operator

U primeru sa Slike-5, izraz *X>100* je logički izraz i predstavlja uslov koga želimo da testiramo. Ako važi da je *x > 100* (Ako je X veće od 100) onda se nastavlja putem *yes*. Ukoliko ne važi da je *X > 100* (Ako X nije veće od 100 B) onda algoritam nastavlja svoj put granom označenom sa *no*.

U nastavku je primer koji određuje rešenja kvadratne jednačine $ax^2 + bx + c = 0$, Slika-5.



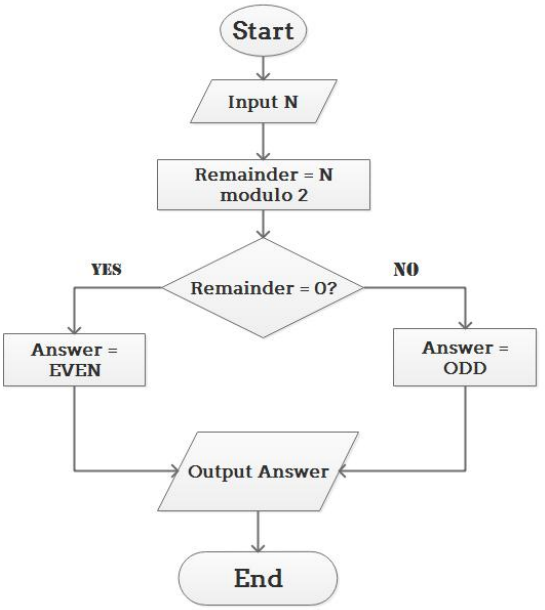
Slika 1.5 Primer određivanja korena broja

PRIMERI UPOTREBE USLOVNOG OPERATORA

Provežbati naredne primere radi utvrđivanja gradiva iz uslovnih operatora

Primer 1: Odrediti da li je broj paran ili nepara i odštampati odgovarajuću poruku. Algoritam i dijagram toka (Slika-6) su u nastavku:

```
Step 1: Read number N,
Step 2: Set remainder as N modulo 2,
Step 3: If remainder is equal to 0 then number N is even
        else number N is odd
Step 4: Print output.
```

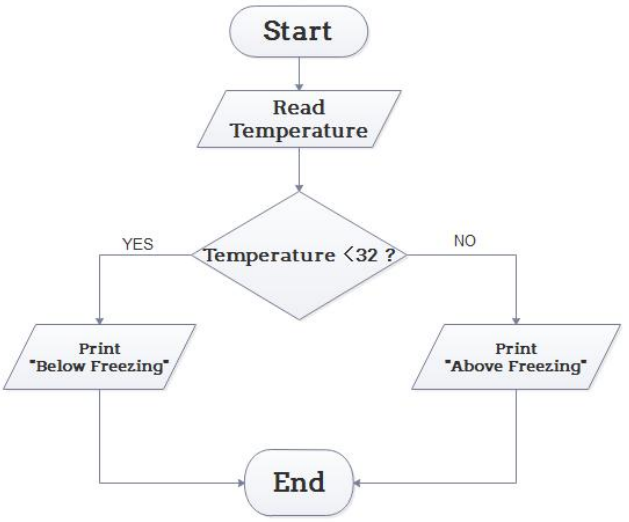


Slika 1.6 Odrediti da li je broj paran ili neparan

Primer 2: Proveriti da li temperatura iznad ili ispod tačke smrzavanja. Temperatura je uneta u Farenhajtima. Algoritam:

```
Step 1: Input temperature,
Step 2: If it is less than 32, then
            print "below freezing point",
        otherwise
            print "above freezing point"
```

Dijagram toka (Slika-7):



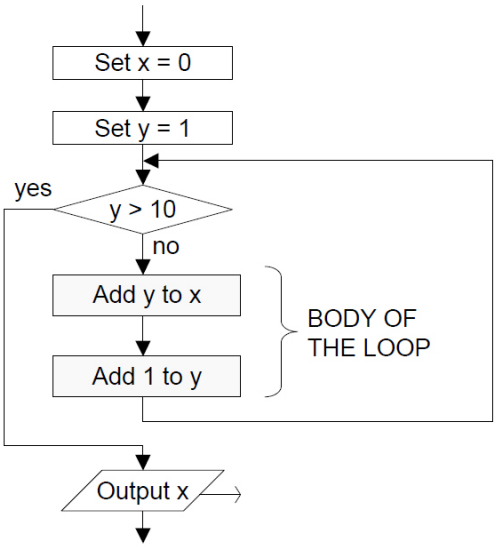
Slika 1.7 Provera da li temperatura iznad ili ispod tačke smrzavanja

OPERATOR PONAVLJANJA ILI PETLJE

Sekvenca operacija koje se ponavljaju iznova u skladu sa uslovom koji kontroliše broj ponavljanja

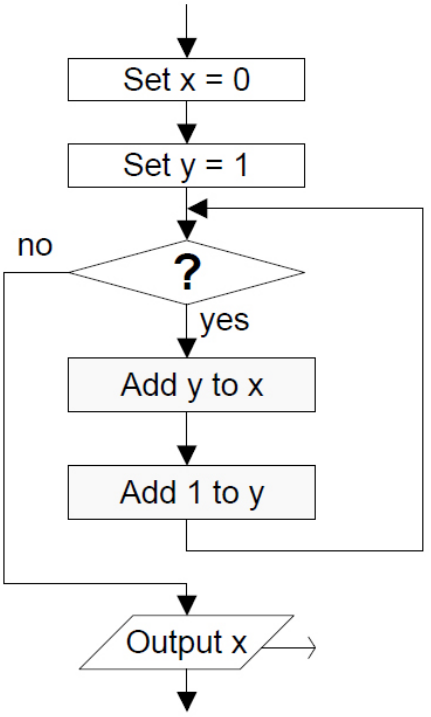
Operator ponavljanja ili *petlje* označava sekvencu operacija koje se ponavljaju iznova i iznova u skladu sa uslovom koji kontroliše broj ponavljanja

- U dijagramu toka je predstavljen korišćenjem dijamanta u kome je naveden uslov provere
- uključuje jedan ili više operatora dodele nakon kojih sledi stralica koja se vraća nazad na prethodno navedeni uslovni operator u dijagramu toka vašeg algoritma.



Slika 1.8 Primer koji određuje zbir prvih 10 brojeva

Zadatak za samostalni rad: Koji uslov treba postaviti umesto znaka ? da bi se algoritam odvijao na korektan način?



1

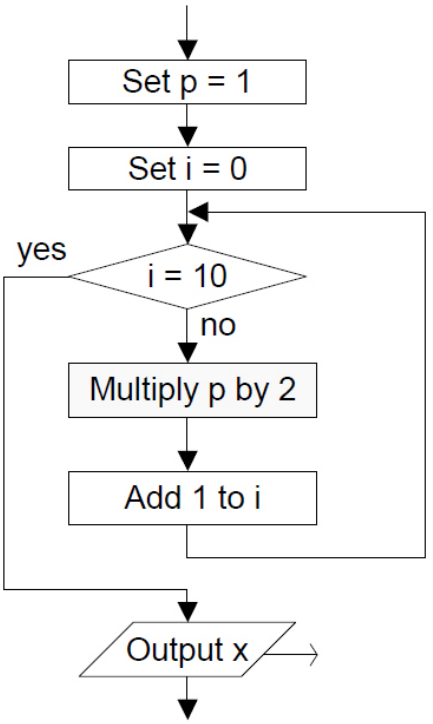
Slika 1.9 Primer za samostalni rad

PRIMERI KORIŠĆENJA OPERATORA PONAVLJANJA (PETLJI)

Provežbati naredne primere radi utvrđivanja gradiva iz operatora ponavljanja

Primer 3. Odrediti vrednost 2^{10}. Algoritam i dijagram toka (Slika-10):

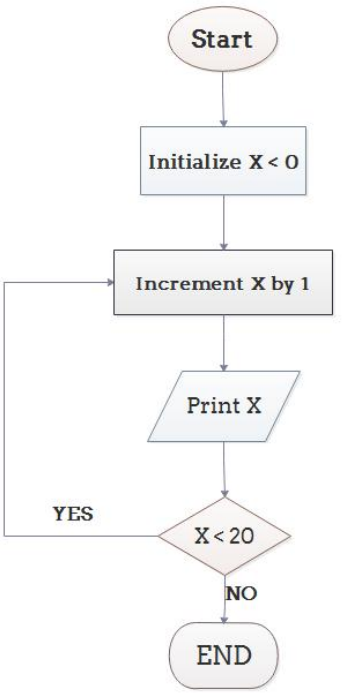
```
Set p = 1
Do the following 10 times:
     Multiply p by 2
Output p
```



Slika 1.10 Primer upotrebe petlji

Primer 4: Štampati brojeve od 1 to 20. Algoritam i dijagram toka (Slika-11):

```
Step 1: Initialize X as 0,
Step 2: Increment X by 1,
Step 3: Print X,
Step 4: If X is less than 20 then go back to step 2.
```



Slika 1.11 Dijagram toka algoritma za štampanje prvih 20 brojeva

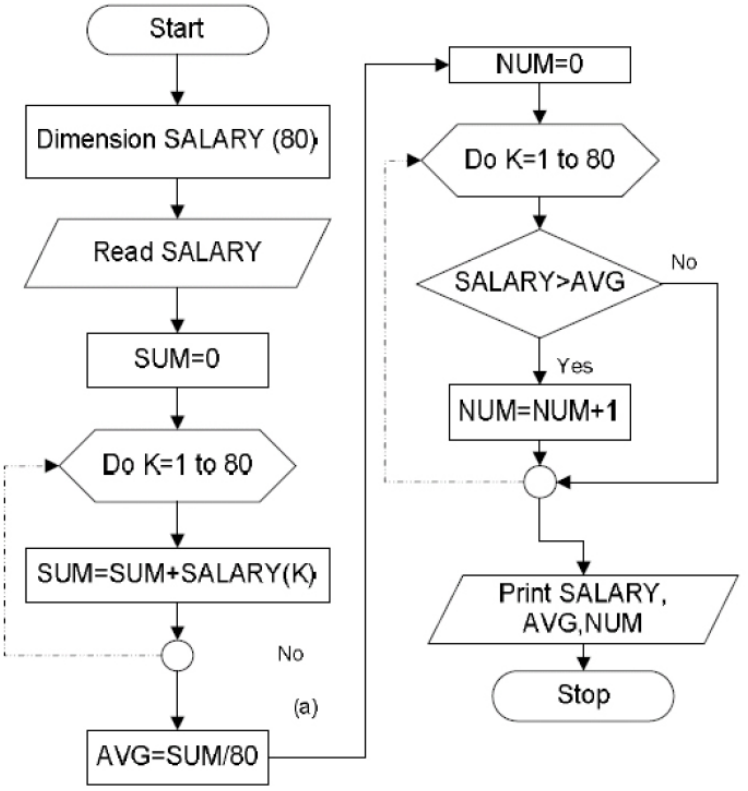
PRIMER: PROSEČNA PLATA RADNIKA

Primer određivanja prosečne plate i broja zaposlenih čija je plata iznad proseka

Kompanija ima 80 zaposlenih. Kreirati dijagram toka za algoritam koji određuje prosečnu platu radnika i broj radnika čija je plata veća od prosečne.

Rešenje je prikazano na Slici-12. Primetiti kako se plate učitavaju u nizu SALARY, a zatim se sračunava prosečna plata AVG. Zatim se za svakog radnika K poredi njegova plata, tj. SALARY(K) sa prosekkom AVG da bi se odredio broj NUM radnika koji imaju platu veću od AVG.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.



Slika 1.12 Dijagram toka za određivanje prosečne plate radnika, i broja radnika čija je plata veća od proseka

ZADACI ZA SAMOSTALNI RAD

Na osnovu svega obrađenog uraditi samostalno sledeće zadatke

Zadatak 1. Nacrtati dijagram toka za naredni pseudo kod:

```
1. Input n
2. Input a vector of values A[0], A[1], …, A[n-1]
3. Set i = 0
4. Set max = A[0]
5. Do the following n-1 times:
    a. Add 1 to i
    b. If A[i] > max, set max = A[i]
6. Output max
```

Zadatak 2. Nacrtati blok dijagram algoritma koji pronalazi površinu kružnice poluprečnika r.

Zadatak 3. Nacrtati blok dijagram algoritma koji konvertuje temperaturu iz Farenhajta u Celzijuse.

Zadatak 4. Nacrtati blok dijagram algoritma koji učitava dva broja i pronalazi njihovu razliku.

Zadatak 5. Nacrtati blok dijagram algoritma koji štampa sve parne brojeve između 9 i 100.

Zadatak 6. Nacrtati blok dijagram algoritma koji štampa neparne brojeve manje od zadatog broja N. Takodje treba da sračuna njihov zbir i njihov broj.

Zadatak 7. Nacrtati blok dijagram algoritma srednju vrednost ocena na ispitu medju 25 studenata.

Zadatak 8. Nacrtati blok dijagram algoritma koji određuje zbir prvih 100 prirodnih brojeva.

Zadatak 9. Nacrtati blok dijagram algoritma koji nalazi najveći među prirodnim brojevima x, y i z.

Zadatak10. Nacrtati blok dijagram algoritma koji određuje da je uneti broj prost? Broj je prost ako je deljiv samo sa 1 i sa samim sobom.

Zadatak 11. Nacrtati blok dijagram algoritma koji generiše listu od prvih 50 brojeva iz Fibonačijeve serije: 1, 1, 2, 3, 5, 8,...? Fib(1) = 1, Fib (2) = 1, Fib (3) = Fib (1) + Fib (2), ..., Fib (N) = Fib (N-2) + Fib (N-1)

Poglavlje 2

Osnovne algoritamske strategije

UVOD U ALGORITAMSKE STRATEGIJE

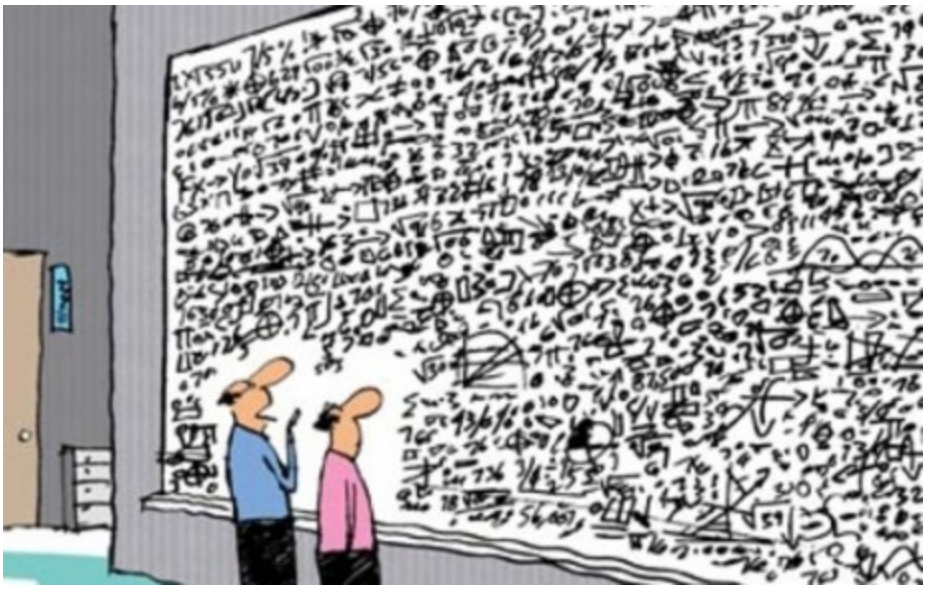
Algoritmi se obično razvrstavaju prema metodologiji projektovanja ili primenjenom obrascu. Ovom problematikom se bavi oblast koja se zove algoritamske strategije

Jedan način razvrstavanja algoritama je po metodologiji projektovanja ili primenjenom obrascu, čime se bavi oblast koja se naziva *algoritamske strategije*.

Postoji više algoritamskih strategija koje se primenjuju kada je potrebno razviti algoritam za neki problem. Ipak, ne postoje garancije da će neka strategija dati tačno rešenje (a iako je rešenje tačno ne mora biti efikasno). Razlog za primenu je taj što su se ove strategije pokazale uspešnim za rešavanje mnogih problema pa je verovatno da će neka od njih dati dovoljno dobro rešenje. Najpoznatije algoritamske strategije su:

- *Gruba sila* (brute-force) ili iscrpna pretraga
- Algoritmi zasnovani na razlaganju ili *"Podeli pa vladaj"* algoritmi (divide and conquer)
- *Dinamičko programiranje*: problem ranca, najkraći putevi iz zadatog čvora, izračunavanje n-tog Fibonačijevog člana niza.
- *Pohlepni metod* (greedy method).
- *Bektreking* - iscrpna pretraga rešenja uz ispitivanje svih puteva primenom obrnute pretrage.

U nastakvu ćemo opisati samo neke od njih



Slika 2.1 Izbor prave strategije za rešavanje postavljenog problema je nekada težak posao

ALGORITAM ISCPNE PRETRAGE

Iscrpna pretraga je opšta tehnika rešavanja problema koja se sastoji od sistematičnog nabrajanja svih kandidata kao mogućih rešenja i provere da li svaki zadovoljava rešenje

Brute-force **pretraga (gruba sila)** ili *iscrpna pretraga*, je opšta tehnika rešavanja problema koja se sastoji od sistematičnog nabrajanja svih mogućih kandidata kao potencijalnih rešenja i provere da li svaki kandidat zadovoljava rešenje problema.

Primer 1: Da bi algoritam iscrpne pretrage pronašao delioce prirodnog broja *n*, on nabraja sve cele brojeve od *1* do *n/2*, i proverava da li svaki od njih deli *n* bez ostatka.

Ova metoda se uglavnom koristi kada je jednostavnost implementacije važnija od brzine. Iscrpna pretraga je takođe korisna i kao "*osnovni*" metod kod testiranja drugih algoritama.

Određivanje NZD dva broja

Kao što je već poznato, najveći zajednički delilac (NZD) brojeva 4 i 2 je 2. Najveći zajednički delilac brojeva 16 i 24 je 8. Ono što se postavlja kao pitanje je kako izgleda kreiranje algoritma koji će odrediti najveći zajednički delilac dva broja?

Osnovni algoritam se može napisati na sledeći način:

- Neka su uneta dva cela broja *n1 i n2*.
- Znamo da je broj *1* sigurno zajednički dellilac dva uneta broja, ali ne mora biti najveći. Stoga je potrebno ispitati da li je *k* (gde je*k* = *2*, *3*, *4*, itd) najveći zajednički delilac brojeva *n1* i *n2*, sve dok *k* ne postane veći od *n1* ili *n2*.
- Koristićemo pomoćnu promenljivu *gcd* da bi u njoj čuvali NZD. Inicijalno, *gcd* je *1*.
- Kad god pronađemo zajednički delilac, u petlji gde *k* ide od *2*, do manjeg broja *n1* i *n2*, on postaje novi najveći zajednički delilac.

- Kada ispitamo sve moguće zajedničke delioce između brojeva *2* i *n1* ili *n2*, vrednost koja je sačuvana u *gcd* će ustvari biti najveći zajednički delilac.

Ova ideja može biti pretočena u sledeći kod:

```
gcd = 1 // Initial gcd is 1
k = 2 // Possible gcd
while (k <= n1 AND k <= n2)
   if (n1 modulus k == 0 AND n2 modulus k == 0)
      gcd = k // Update gcd
   k++
end while
```

„PODELI I OSVOJI“ ALGORITAM

Podeli i osvoji algoritam smanjuje stepen složenosti problema podelom na dva ili više manjih problema iste vrste dok od problema ne ostane toliko mali deo da se može jednostavno rešiti

Podeli i osvoji (lat. divide et impera*. ili „zavadi pa vladaj“*) predstavlja algoritam u kojem se veći problem raščlanjuje na više manjih, koji su jednostavniji za sagledavanje i pojedinačno rešavanje. Podela na manje se vrši dok potproblemi ne postanu dovoljno jednostavni da se mogu direktno rešiti. Ovi potproblemi se mogu rešavati paralelno (istovremeno rešavanje dva ili više potproblema) ili jedan za drugim. Rešenja potproblema se kombinuju da bi se dobilo rešenje početnog problema.

U cilju razumevanja osnovnih koncepata "podeli i osvoji" algoritma možete pogledati sledeći video.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

POHLEPNI ALGORITAM

Pohlepni algoritam u svakom koraku bira u tom trenutku naizgled najbolje rešenje, bez obzira da li će to uticati na tačnost konačnog rešenja problema

Pohlepni algoritam u svakom koraku bira u tom trenutku naizgled najbolje rešenje. Prednost ovih algoritama je što se lako konstruišu i što se njihova složenost obično lako izračunava. Mana je da je dokaz korektnosti ovih algoritama obično komplikovan, i lako dolazi do grešaka pri njihovoj konstrukciji, jer često dobra rešenja ne odgovaraju intuiciji.

Pohlepni algoritmi vrlo rano donose određene odluke, koje ih mogu sprečiti da kasnije nađu najbolje rešenje. Ipak oni su korisni, jer su u stanju da brzo daju dobru aproksimaciju optimalnog rešenja.

Za mnoge probleme, pohlepni algoritmi mogu biti pogrešan izbor strategije. Recimo, u partiji šaha, pohlepni algoritam će uvek odigrati onaj potez koji izgleda najbolje u datom trenutku, ne obazirući se na njegove posledice. Na primer, ukoliko najveću vrednost za igrača u datoj situaciji ima potez u kome on uzima protivničkog lovca, on će to i učiniti, sve i ako taj potez otvara protivniku mogućnost da matira igrača.

Problem kusura

Trgovac treba da vrati mušteriji kusur od 41 cent, na raspolaganju su mu novčići od 25, 10, 5 i 1 centi. Instinktivno će većina ljudi vratiti 1x25, 1x10 i 5x1, i 1x1 cent. Takav algoritam vraća tačan iznos uz najkraću moguću listu novčanica (Slika-2).

Slika 2.2 Primena pohlepnog algoritma kod problema kusura

Algoritam: izabere se najveća novčanica koja ne prelazi ukupnu sumu (25 centi), stavlja se na listu za vraćanje, oduzme se vraćena vrednost od ukupne kako bi se izračunao ostatak za vraćanje (16 centi), zatim se bira sledeća novčanica koja ne prelazi ostatak koji treba vratiti (10 centi), dodaje se na listu za vraćanje i računa novi ostatak za vraćanje (6 centi).

Ovo se vrši sve dok se ne vrati tačan iznos. Dakle, strategija pohlepnog pristupa je u konkretnom primeru sa 41 centa dovela do najfikasnijeg rešenja problema vraćanja novca. Međutim, za većinu problema, pohlepni algoritmi uglavnom (ali ne i uvek) neće uspeti da nađu globalno optimalno rešenje, jer oni obično ne obrađuju temeljno sve mogućnosti.

OSNOVI DINAMIČKOG PROGRAMIRANJA

Kada problem pokazuje optimalnu podstrukturu, možemo ga rešiti brzo koristeći dinamičko programiranje - pristup koji izbegava ponovno izračunavanje rešenja koja su već izračunata

U situacijama kada se optimalno rešenje problema može konstruisati iz optimalnog rešenja potproblema, i preklapanjem potproblema, glavni problem možemo rešiti brzo korišćenjem **dinamičkog programiranja**, pristupa koji izbegava ponovno izračunavanje rešenja koja su već izračunata. Sama reč programiranje u terminu dinamičko programiranje se odnosi na popunjavanje tabele pri rešavanju problema, a ne na upotrebu programskog jezika ili računara.

Pregled dinamičkog programiranja

Dakle, problem se može rešiti u sledećim koracima:

- Razbiti problem na manje potprobleme.
- Rešiti date potprobleme koristeći ova tri koraka rekurzivno.
- Iskoristiti pronađena optimalna rešenja potproblema kako bi se našlo optimalno rešenje glavnog problema.
- Potprobleme je potrebno deliti na pod-potprobleme, sve dok se ne dođe do jednostavnih slučaja koje je jednostavno rešiti.

Pomoću algoritama dinamičkog programiranja, potproblemi jednog problema postaju međusobno zavisni, pa je dovoljno da ih samo jednom rešite, da bi rad bio lakši i brži.

Ideja je sačuvati rešenja onih problema koji su već rešeni, kako bi se mogli kasnije iskoristiti. Ovo se zove *memoizacija*. Ukoliko je očigledno da rešeni potproblemi nisu više potrebni, mogu se odbaciti kako bi se sačuvao memorijski prostor.

Uobicijeni postupak je da se svi potproblemi redom rešavaju i koriste za nalaženje većih.

```
function fib(n)
    var previousFib := 1, currentFib := 1
    repeat n − 1 times
        var newFib := previousFib + currentFib
        previousFib := currentFib
        currentFib  := newFib
    return currentFib
```

ZADACI ZA SAMOSTALNI RAD

Na osnovu svega obrađenog uraditi samostalno sledeće zadatke:

Ove zadatke treba da rešite ručno i da smanjite broj koraka za generalizovane verzije. Ne treba da pišete programe, već da otkrijete koji su koraci potrebni za rešavanje i da ih zapišete kao algoritme.

- Novčići dolaze u apoenima od 1 cent, 5 centi, 10 centi, 20 centi i 1 dolar. Ako se podrazumeva da ima neograničen broj novčića, kako biste smislili algoritam za vraćanje kusura koji računa minimalan broj novčića potrebnih da se dobije određenja suma novca. Na primer, za 46 centi, treba 4 novčića: dva od 20 centi, jedan od 5 centi i jedan od jednog centa.
- Kako biste sortirali 10 brojeva u rastućem redosledu? Da li znate neke od osnovnih tehnika sortiranja?
- U igri *Mastermind*, prvi igrač skriva tajni kod koji se sastoji od 4 cifre, a drugi igrač treba da pogađa sve dok ne provali taj kod. Prvi igrač pomaže drugom na sledeći način, ukazuje mu na neki od sledeća dva slučaja:
  Pogađanje prave cifre i tačne pozicije,
  Pogađanje prave cifre, ali pogrešne pozicije.
  Napišite algoritam koji odgovara na pogađanje drugog igrača. Prvo, radite na specijalnom

slučaju, gde nema ponavljanja cifara. Posle toga, pređite na slučaj gde se cifre mogu ponavljati u kodu.

Poglavlje 3

Impementacija raznih algoritama

PRIKAZ PROSTIH BROJEVA

Predstavljamo program koji na ekranu prikazuje prvih 50 prostih brojeva, i to u 5 linija, pri čemu u svakoj liniji ima po 10 prostih brojeva

Kao što je poznato, pozitivan ceo broj veći od *1* je ***prost*** ako je deljiv sa *1* i sa samim sobom. Tako su na primer brojevi *2*, *3*, *5*, i *7* su prosti brojevi, dok *4*, *6*, *8*, i *9* nisu. Cilj ovog zadatka je da se na ekranu prikaže prvih 50 prostih brojeva, i to u 5 linija, gde svaka linija sadrži 10 prostih brojeva. Problem se može podeliti u nekoliko sledećih koraka, odnosno podzadataka:

- Napisati kod koji određuje da li je proizvoljni broj prost.
- Zatim za brojeve *number* = *2*, *3*, *4*, *5*, *6*, ..., ispitati da li su prosti.
- Vršiti brojanje prostih brojeva.
- Prikazati svaki prost broj, ali tako da ih ima 10 u jednoj prikazanoj liniji na ekranu.

Na osnovu prethodnih koraka, očigledano je da morate napisati petlju koja će iznova ispitivati da li je novi broj *number* prost. Ukoliko je *number* prost uvećati brojač *count* za *1*. Naravno brojač *count* je inicijalno jednak *0*. Kada brojač dostigne *50*, petlja se prekida.

Naravno, svaki put kada je brojač *count* deljiv sa 10, treba prebaciti kursor u sledeći red. Stoga, algoritam za rešenje ovog problema ima oblik:

```
Set the number of prime numbers to be printed as
a constant NUMBER_OF_PRIMES;
Use count to track the number of prime numbers and
set an initial count to 0;
Set an initial number to 2;
while (count < NUMBER_OF_PRIMES) {
 Test whether number is prime;
 if number is prime {
  Display the prime number and increase the count;
 }
 Increment number by 1;
}
```

Da bi smo ispitali da li je broj prost, ispitaćemo da li je deljiv sa nekim brojem u intervalu od *2* do *number / 2*. Ukoliko je pronađen delilac onda broj nije prost. Algoritam može biti napisan na sledeći način:

```
Use a boolean variable isPrime to denote whether
the number is prime; Set isPrime to true initially;
```

```
for (int divisor = 2; divisor <= number / 2; divisor++) {
 if (number % divisor == 0) {
  Set isPrime to false
  Exit the loop;
 }
}
```

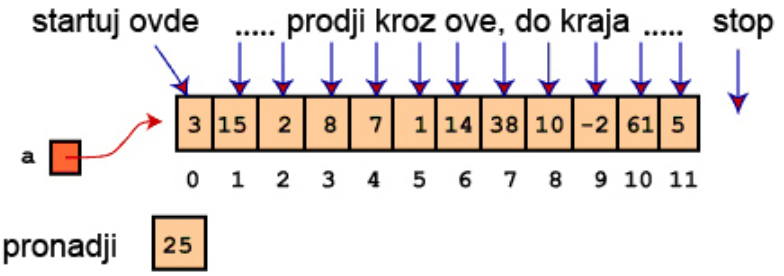
LINEARNO PRETRAŽIVANJE NIZOVA

Pretraživanje je proces traženja odgovarajućeg elementa u nizu.
Osnovni tipovi pretraživanja jesu linearno i binarno pretraživanje.

Pretraživanje je proces traženja odgovarajućeg elementa u nizu, na primer, pronalaženje da li se neki rezultat nalazi u tabeli rezultata. Mnogi algoritmi i strukture podataka su posvećeni pretraživanju, i to će biti detaljnije opisano u lekcijama kao što su heširanje ili stabla pretrage.

Linearno pretraživanje

Pristup linearnog pretraživanja upoređuje ključni element *key* sekvencionalno (redom) sa svim elementima niza. Pretraga se izvršava sve dok se ključ pretrage ne poklopi sa nekim elementom niza, ili dok se ne potroše svi elementi niza, a ne dođe do poklapanja. Ukoliko se desi poklapanje sa ključem, linearno pretraživanje vraća indeks elementa niza koji odgovara ključu pretrage (Slika-1). Ukoliko se ne pronađe ključ u nizu, algoritam vraća vrednost *-1*. Metod linearnog pretraživanja je prikazan u sledećem listingu.



Slika 3.1 Primer linearnog pretraživanja niza

```
function linearSearch(list of elements, key, listSize)
    for i = 0, (listSize-1)
        if (key == list[i])
            return i
    return -1;
end function
```

Metod linearnog pretraživanja upoređuje ključ sa svim elementima niza. Elementi niza mogu biti u bilo kakvom poretku (uređenom tj. sortiranom ili neuređenom). U proseku, algoritam mora da ispita najmanje polovinu elemenata dok ne pronađe ključ, ukoliko on postoji.

PROVERA DA LI JE REČ PALINDROM

U ovoj sekciji je predstavljen program koji ispituje da li je string palindrom. Neki string je palindrom ako se čita isto sa obe strane

Neki string je palindrom ako se čita isto sa obe strane. Reči kao što su “mom,” “dad,” “anavolimilovana” su palindromi. Na primer, "123321" i "RADAR" su takođe palindromi.

Napišite algoritam koji proverava da li je reč palindrom.

Cilj je da se napiše algoritam koji od korisnika traži da unese string a zatim da izvesti o tome da li je palindrom. Jedno rešenje bi bilo da se ispita da li je prvi karakter u stringu jednak poslednjem. Ukoliko jeste, onda ispitati da li je drugi karakter jednak predposlednjem karakteru u stringu. Ovaj proces se nastavlja sve dok se desi neslaganje ili dok se ne provere svi karakteri u stringu, osim središnjeg karakter u stringu koji ima neparan broj karaktera.

Zadatak za samostalnu vežbu:

Prost palindromski broj je prost broj koji je istovremeno i palindrom. Na primer, 131 je prost ali istovremeno i palindrom, kao i brojevi 313 i 757.

Napisati algoritam koji prikazuje prvih 10 prostih brojeva koji su i palindromi.

Primer prvih 10:

2 3 5 7 11 101 131 151 181 191

313 353 373 383 727 757 787 797 919 929

PRIMER: ZBIR PRVE I POSLEDNJE CIFRE BROJA

Za proizvoljan prirodni broj veći od 0 vrši se izdvajanje cifara kombinovanjem operatora moduo i deljenje, i na kraju se vrši sabiranje prve i poslednje cifre

Napisati funkciju koji za prosleđeni ceo broj vraća sumu njegove prve i poslednje cifre. Ukoliko je prosleđeni broj jednocifren, metod treba da vrati -1. Primer:

- X=9, output: -1
- X=123, output: 4
- X=1235, output: 6

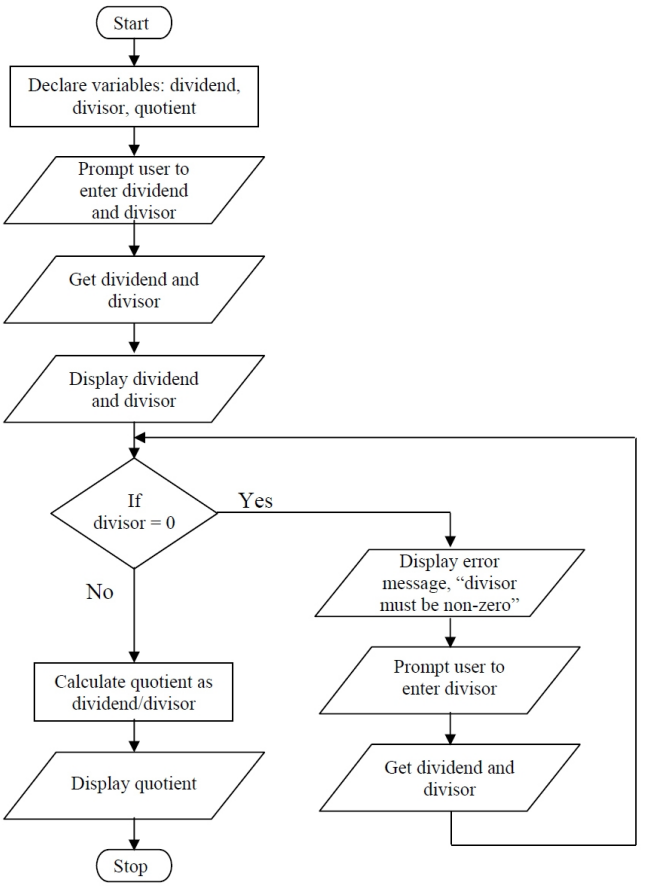
```
function prva_poslednja(int n)

    if(n>=0 AND n<10)
        return -1

    poslednja_cifra = n MODUO 10
```

```
    while(n>=10)
        n=n/10

    prva_cifra = n

    return (prva_cifra+poslednja_cifra)
end function
```

PRIMER: KOLIČNIK BROJEVA

Blok dijagram za određivanje količnika brojeva. Vrši se provera da li je delilac jednak nuli

Nacrtati blok dijagram algoritma koji od korisnika zahteva da učita dva broja (deljenik i delilac), zatim provera da li je delilac različit od nule (pošto je deljenje nulom nedozvoljena operacija) i ukoliko jeste ona količnik brojeva štampa na standardni izlaz.



Slika 3.2 Dijagram toka algoritma za određivanje količnika dva uneta broja

PRIMER. STEPEN ODGOVARAJUĆEG BROJA

Primer pseudo koda kao i korišćenje table u cilju praćenja rada algoritma

Napisati pseudokod i nacrtati dijagram toka algoritam koji određuje 2^4 korišćenjem petlji. Verifikovati rezultat korišćenjem tabele za praćenje rada programa. Algoritam:

```
Step 1: Base <- 2
Step 2: Power <- 4
Step 3: Product <- Base
Step 4: Counter <- 1
Step 5: While Counter < Power  Repeat Step 5 through step 7
Step 6:     Product <- Product * Base
Step 7:     Counter <- Counter +1
Step 8: PrintProduct
```

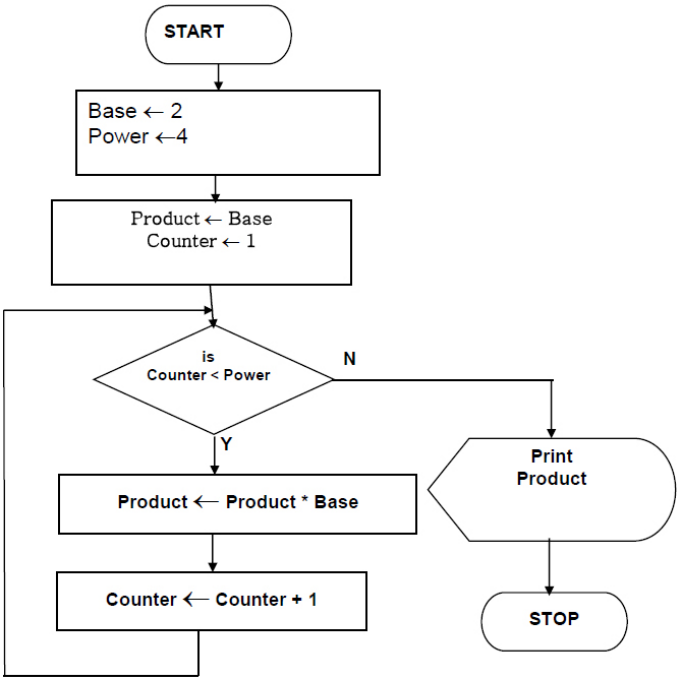
Tabela za praćenje rada algoritma (Slika-3)

	BASE	POWER	PRODUCT	COUNTER	COUNTER < POWER
STEP 1:	2	?	?	?	?
STEP 2:	2	4	?	?	?
STEP 3:	2	4	2	?	?
STEP 4:	2	4	2	1	T
STEP 5:	**2**	**4**	**2**	**1**	**T**
STEP 6:	2	4	2x2=4	1	T
STEP 7:	2	4	4	1+1=2	T
STEP 5:	**2**	**4**	**4**	**2**	**T**
STEP 6:	2	4	4x2=8	2	T
STEP 7:	2	4	8	2+1=3	T
STEP 5:	**2**	**4**	**8**	**3**	**T**
STEP 6:	2	4	8x2=16	3	T
STEP 7:	2	4	16	3+1=4	F
STEP 5:	**2**	**4**	**16**	**4**	**F**
STEP 8:	print **16**.				

Step 1: Base ← 2
Step 2: Power ← 4
Step 3: Product ← Base
Step 4: Counter ← 1
Step 5: While Counter < Power
Repeat Step 5 through step 7
Step 6: Product ← Product * Base
Step 7: Counter ← Counter +1
Step 8: *Print* Product

Slika 3.3 Praćenje rada programa

Dijagram toka (Slika-4):



Slika 3.4 Blok dijagram algoritma za određivanje stepena broja 2 na 4

PRIMER: MAKSIMUM SKUPA BROJEVA

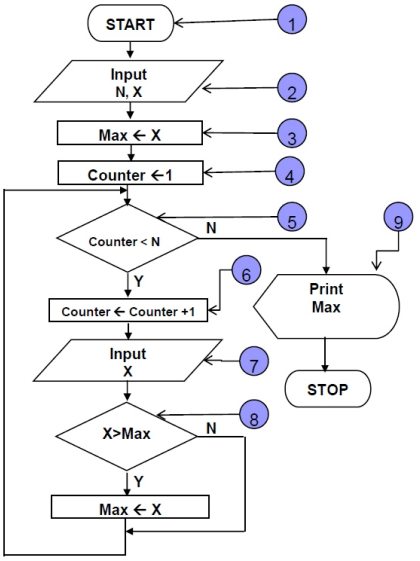
Učitava se N brojeva sa ulaza i pronalazi najveći među njima

Napisati pseudo kod i nacrtati dijagram toka algoritma koji pronalazi i štampa najveći od N (N može biti bilo koji broj) učitanih brojeva. Učitati brojeve jedan za drugim. Verifikovati rezultat korišćenjem tabele za prećenje promene vrednosti (trace table). Pritom pretpostaviti da je N jednako 5, a koristititi sledeći test uzorak brojeva {1 4 2 6 8 }.

Algoritam:

```
Step 1: InputN
Step 2: InputX
Step 3: Max  <- Current
Step 4: Counter  <- 1
Step 5: While (Counter < N)
            Repeat steps 5 through 8
Step 6: Counter  <- Counter + 1
Step 7: InputX
Step 8: If(X > Max) then
            Max <- X
        endif
Step 9: PrintMax
```

Dijagram toka i tabela za praćenje rada programa (Slika-5)



Tracing

	N	X	Max	Counter	Counter < N	Next > Max
Step 1	5	1				
Step 2	5	1				
Step 3	5	1	1			
Step 4	5	1	1	1	T	
Step 5	5	1	1	1	T	
Step 6	5	1	1	2	T	
Step 7	5	4	1	2	T	
Step 8	5	4	4	2	T	T
Step 5	5	4	4	2	T	F
Step 6	5	4	4	3	T	F
Step 7	5	2	4	3	T	F
Step 8	5	2	4	3	T	F
Step 5	5	2	4	3	T	F
Step 6	5	2	4	4	T	F
Step 7	5	6	4	4	T	T
Step 8	5	6	6	4	T	T
Step 5	5	6	6	4	T	F
Step 6	5	6	6	5	F	F
Step 7	5	8	6	5	F	T
Step 8	5	8	8	5	F	T
Step 5	5	8	8	5	F	F
Step 9			**8** output			

Slika 3.5 Blok dijagram i tabela za praćenje rada programa koji pronalazi maksimalni broj u skupu brojeva

Pitanje: Koliko puta će koraci 4, 6, i 7 biti izvršeni?

PRIMER: SREDNJA VREDNOST BROJEVA

Učitava se N brojeva sa ulaza i pronalazi njihova srednja vrednost

Za ulazni skup brojeva odrediti njihov zbir i srednju vrednost. Formula:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

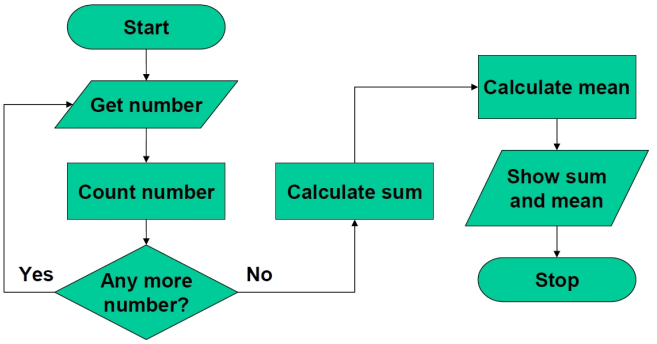
Slika 3.6 Formula za određivanje srednje vrednosti brojeva

pri čemu je n broj vrednosti u skupu. Algoritam može biti napisan kao:

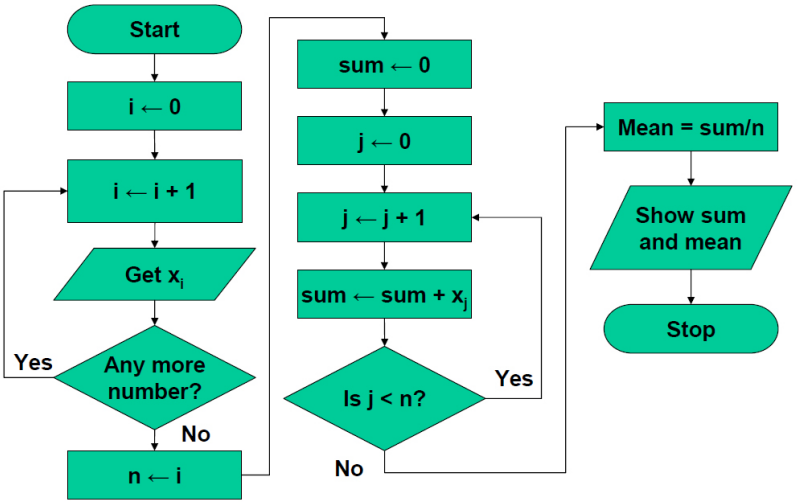
```
1. Start
2. Get one number in the set
3.     Count the numbers as it is obtained
4.     If there are still numbers to be obtained,
            go back to step 2.
5. Sum the numbers in the set
6. Divide the sum by the number of numbers
        in the set to get the average
7. Show the sum and the average
8. Stop
```

Dijagram toka i detaljni dijagram toka su prikazani na slikama Slika-7 i Slika 8.

Zadatak za samostalni rad: Na osnovu dijagrama sa Slike-8 napisati pseudo kod algoritma



Slika 3.7 Dijagram toka algoritma za određivanja srednje vrednosti skupa brojeva



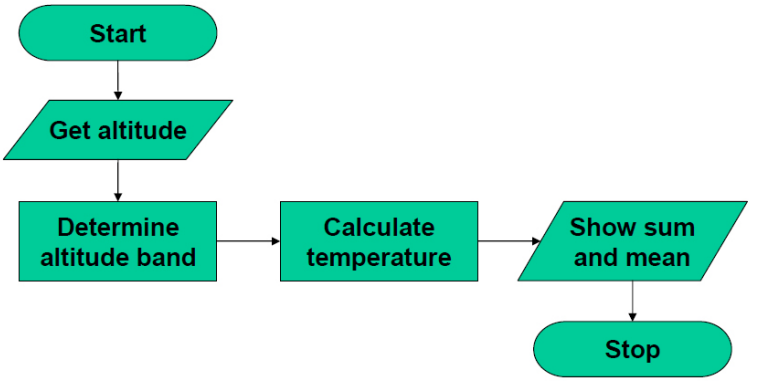
Slika 3.8 Detaljni dijagram toka

PRIMER: ODREĐIVANJE TEMPERATURE NA RAZLIČITIM NIVOIMA

Na osnovu unete visine, prema jednačinama koje važe, računa se temperatura u toj tački

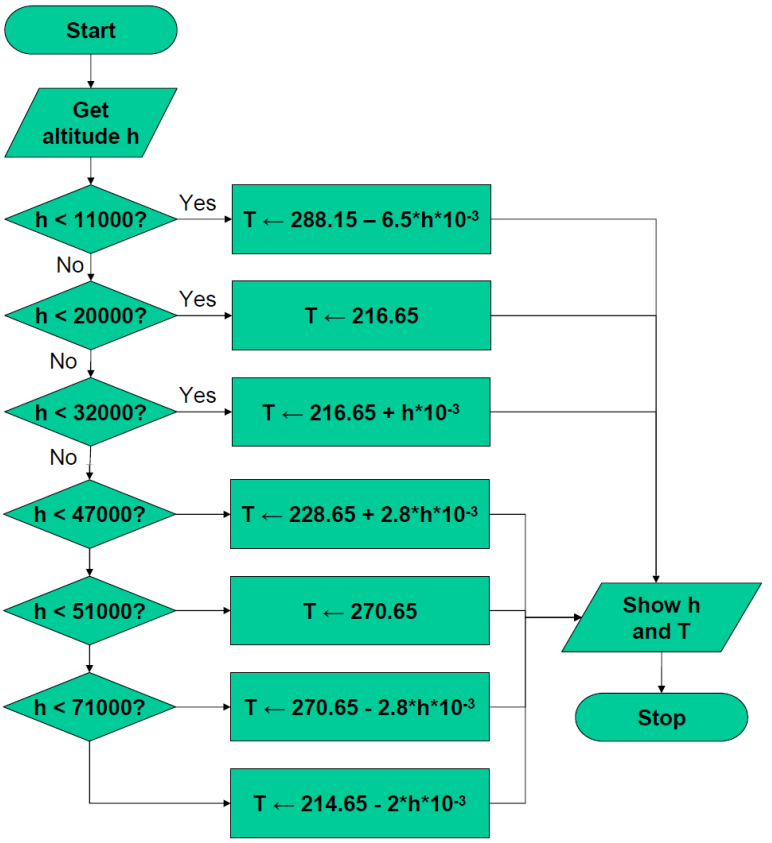
Trofosfera je sloj iznad nivoa mora na visini do 11000 m. Stratosfera je sloj izmedju 11000 i 51000m visine. Mezosfera je sloj na visini između 51000 to 71000m visine. Za odgovarajuću ulaznu vrednost visine sračunati temperaturu atmosfere:

```
1. Start
2. Get altitude
3. Determine which altitude band it is in
4. Calculate the temperature using the
        equation associated with that band
5. Show the altitude and the temperature
6. Stop
```



Slika 3.9 Dijagram toka za odredjivanje temperature

Detaljni dijagram toka (Slika-10):



Slika 3.10 Detaljni dijagram toka za odredjivanje temperature

PRIMERI

Provežbati sledeće zadatke u cilju utvrđivanja znanja iz algoritama

Primer 1. Malo istaživanje treba biti sprovedeno da bi se dobila informacija o tome koji je najomiljeniji sport medju studentima. Rezultat ankete treba biti prikazan kao rezultat. Napisati program koji će ovo da postigne.

```
REPEAT
    DISPLAY “Type in the letter chosen or Q to finish”
    DISPLAY “A: Athletics”
    DISPLAY “S: Swimming”
    DISPLAY “F: Football”
    DISPLAY “B: Badminton”
    DISPLAY “Enter data”
    ACCEPT letter
    If letter = ‘A’ then
        Athletics = athletics + 1
    If letter = ‘S’ then
        Swimming = Swimming + 1
    If letter = ‘F’ then
        Football = Football + 1
    If letter = ‘B’ then
        Badminton = Badminton + 1
UNTIL letter = ‘Q’
DISLAY “Athletics scored”, athletics, “votes”
DISLAY “Swimming scored”, swimming, “votes”
DISLAY “Football scored”, football, “votes”
DISLAY “Badminton scored”, Badminton, “votes”
```

Primer 2. Projektovati algoritam koji generiše parne brojeve izmedju 1000 i 2000, zatim ih štampa na izlaz. Program takođe treba da štampa ukupan zbir brojeva:

```
Step 1. Start
Step 2. I ← 1000 and S ← 0
Step 3. Write I
Step 4. S ← S + I
Step 5. I ← I + 2
Step 6. If (I <= 2000) then go to line 3
    else go to line 7
Step 7. Write S
Step 8. End
```

Primer 3. Projektovati algoritam koji učitava prirodan broj n, sračunava sledeću formulu i rezultat štampa na standardni izlaz:

S = ½ + ¼ + ... +1/n

```
Step 1. Start
Step 2. Read n
Step 3. I ← 2 and S ← 0
Step 4. S= S + 1/I
Step 5. I ← I + 2
Step 6. If (I <= n) then go to line 4
            else write S in standard output
Step 7. End
```

ZADACI ZA SAMOSTALNI RAD

Na osnovu svega obrađenog uraditi samostalno sledeće zadatke

Zadatak 1. Napisati pseudokod i nacrtati dijagram toka za algoritam koji štampa kvadrate (a na kvadrat) svih brojeva izmedju 1 i 10.

Zadatak 2. Napisati pseudokod i nacrtati dijagram toka za algoritam koji štampa zbir SUM svih brojeva od LOW do HIGH. Testirati rad algoritma kada je LOW=3 a HIGH=9.

Zadatak 3. Napisati pseudokod i nacrtati dijagram toka za algoritam koji štampa sve brojeve izmedju LOW i HIGH koji su deljivi brojem NUMBER.

Zadatak 4. Napisati pseudokod i nacrtati dijagram toka za algoritam koji učitava 10 brojevasa ulaza i štampa njihov zbir i njihov proizvod.

Zadatak 5. Napisati pseudokod i nacrtati dijagram toka za algoritam koji prebrojava i štampa sve brojeve od LOW do HIGH, pri čemu se štampa svaki na rastojanju STEP. Testirati programa za slučaj LOW=0 a HIGH=100, pri čemu je korak STEP=5.

Zadatak 6. Napisati pseudokod i nacrtati dijagram toka za program koji štampa tablicu množenja broja 6, tj:

----1 6 = 6
----2 6 = 12
…
----12 6 = 72

Zadatak 7. Napisati pseudokod i nacrtati dijagram toka za algoritam koji treba da štampa proizvod tri uneta broja.

Zadatak 8. Napisati pseudokod i nacrtati dijagram toka za algoritam koji određuje i štampa faktorija broja NUMBER. Testirati rad programa za slučaj kada je NUMBER=5.

ZAKLJUČAK

Na osnovu svega obrađenog možemo zaključiti sledeće:

Algoritmi se obično razvrstavaju prema metodologiji projektovanja ili primenjenom obrascu. Ovom problematikom se bavi oblast koja se zove algoritamske strategije.

Iscrpna pretraga je opšta tehnika rešavanja problema koja se sastoji od sistematičnog nabrajanja svih kandidata kao mogućih rešenja i provere da li svaki zadovoljava rešenje.

Podeli i osvoji algoritam smanjuje stepen složenosti problema podelom na dva ili više manjih problema iste vrste dok od problema ne ostane toliko mali deo da se može jednostavno rešiti.

Pohlepni algoritam u svakom koraku bira u tom trenutku naizgled najbolje rešenje, bez obzira da li će to uticati na tačnost konačnog rešenja problema. Za većinu problema, pohlepni algoritmi uglavnom (ali ne i uvek) neće uspeti da nađu globalno optimalno rešenje, jer oni obično ne obrađuju temeljno sve mogućnosti

Kada problem pokazuje optimalnu podstrukturu, možemo ga rešiti brzo koristeći dinamičko programiranje - pristup koji izbegava ponovno izračunavanje rešenja koja su već izračunata. Sama reč programiranje u terminu dinamičko programiranje se odnosi na popunjavanje tabele pri rešavanju problema. Na taj način sprečavamo ponovno rešavanje već rešenih potproblema