# METODIKA NASTAVE RAČUNARSTVA A

Python – pretraga i sortiranje

Stefan Mišković

1. Koristeći linearnu pretragu, proveriti da li niz sadrži dati broj.

*Rešenje.* Kod linearne pretrage se prolazi kroz ceo niz i za svaki element se proverava da li je jednak traženom broju. Niz ne mora da bude sortiran, ali je vremenska složenost algoritma *O*(*n*), gde je *n* broj elemenata niza.

**def** l i n e a r n a Pr e tr a g a ( broj , n i z ) :

**for** element **in** n i z :

**i f** b r o j == element :

**return** True

**return** False

n i z = [ 5 , 1 , 2 , 7 , 10 , 3 , 9 , 13 , 1 2 ]

**for** b r o j **in range** ( 1 , 1 4 ) :

**i f** l i n e a r n a Pr e tr a g a ( broj , n i z ) :

**print** ( " Broj." + **str** ( b r o j ) + ". j e .u. nizu . " )

## else :

**print** ( " Broj." + **str** ( b r o j ) + ". n i j e .u. nizu . " )

1. Korsteći binarnu pretragu, proveriti da li sortirani niz sadrži dati broj. Binarnu pretragu implementirati rekurzivno.

*Rešenje.* Binarna pretraga se izvršava nad nizom koji je već sortiran i određuje da li u njemu postoji traženi element. U svakom koraku se vrednost traženog elementa upoređuje sa vrednošću koja je na sredini trenutnog intervala. Ukoliko su one jednake, nađen je element. Inače, ukoliko je tražena vrednost manja, posmatra se leva polovina intervala elemenata, a ukoliko je veća, desna polovina intervala. Funkcija se rekurzivno poziva sve dok nije pronađen element ili dok leva i desna strana intervala ne postanu jednake, što znači da element nije pronađen. Vremenska složenost algoritma *O*(log *n*), gde je *n* broj elemenata niza.

**def** binarna Pretraga R ( broj , niz , l e v i , d e s n i ) :

**i f** l e v i > d e s n i :

**return** False

s r e d n j i = ( l e v i + d e s n i ) // 2

**i f** b r o j == n i z [ s r e d n j i ] :

**return** True

**e l i f** b r o j < n i z [ s r e d n j i ] :

**return** binarna Pretraga R ( broj , niz , l e v i , s r e d n j i 1 )

:

*−*

## else

**return** binarna Pretraga R ( broj , niz , s r e d n j i + 1 , d e s n i )

n i z = [ 5 , 1 , 2 , 7 , 10 , 3 , 9 , 13 , 1 2 ]

**for** b r o j **in range** ( 1 , 1 4 ) :

**i f** binarna Pretraga R ( broj , **sorted** ( n i z ) , 0 , **len** ( n i z ) 1 ) :

*−*

**print** ( " Broj." + **str** ( b r o j ) + ". j e .u. nizu . " )

## else :

**print** ( " Broj." + **str** ( b r o j ) + ". n i j e .u. nizu . " )

1. Korsteći binarnu pretragu, proveriti da li sortirani niz sadrži dati broj. Binarnu pretragu implementirati iterativno.

*Rešenje.* Kod iterativne binarne pretrage, polovljenje intervala se, umesto rekurzivnog poziva, izvršava u ciklusu. U svakoj iteraciji, ako je traženi element manji od srednjeg elementa niza, krajnji desni element postaje prvi element koji se nalazi levo od srednjeg elementa. Slično, ako je traženi element veći od srednjeg elementa niza, krajnji levi element postaje prvi element koji se nalazi desno od srednjeg elementa.

**def** binarna Pretraga I ( broj , n i z ) : l e v i = 0

d e s n i = **len** ( n i z ) 1

*−*

**while** l e v i <= d e s n i :

s r e d n j i = ( l e v i + d e s n i ) // 2

**i f** b r o j == n i z [ s r e d n j i ] :

**return** True

**e l i f** b r o j < n i z [ s r e d n j i ] :

**else** d e s n i = s r e d n j i *−* 1 l e v i = s r e d n j i + 1

:

**return** False

n i z = [ 5 , 1 , 2 , 7 , 10 , 3 , 9 , 13 , 1 2 ]

**for** b r o j **in range** ( 1 , 1 4 ) :

**i f** binarna Pretraga I ( broj , **sorted** ( n i z ) ) :

**print** ( " Broj." + **str** ( b r o j ) + ". j e .u. nizu . " )

## else :

**print** ( " Broj." + **str** ( b r o j ) + ". n i j e .u. nizu . " )

1. Sortirati niz korišćenjem bubble sorta.

*Rešenje.* Bubble sort je jednostavan algoritam za sortiranje kvadratne složenosti, koji u svakom paru iteracija menja dva susedna elementa ukoliko oni nisu u odgovarajućem poretku. Posle prve iteracije kroz niz, najveći element se nalazi na najvećoj poziciji u nizu, tj. na desnoj strani niza. Nakon druge iteracije, drugi najveći element će se nalaziti na pretposlednjoj poziciji, itd.

**def** bubble Sort ( n i z ) : n = **len** ( n i z )

**for** i **in range** ( n 2 ) :

*−*

**for** j **in range** ( n 1 ) :

*−*

**i f** n i z [ j ] > n i z [ j + 1 ] :

n i z [ j ] , n i z [ j + 1 ] = n i z [ j + 1 ] , n i z [ j ]

n i z = [ 5 , 1 , 2 , 7 , 10 , 3 , 9 , 13 , 1 2 ]

**print** ( " Niz . pre . s o r t i r a n j a : " )

**print** ( n i z ) bubble Sort ( n i z )

**print** ( " Niz . nakon. s o r t i r a n j a : " )

**print** ( n i z )

1. Sortirati niz korišćenjem quick sorta.

*Rešenje.* Quick sort se u proseku izvršava u *O*(*n* log *n*). Iako je najgori slučaj izvršavanja ovog algoritma kvadratni, u praksi mu je vreme izvršavanja najčešće jednako prosečnom, zbog čega se i najčešće koristi. Zasniva se na biranju elementa koji je označen kao pivot, a zatim se vrši deljenje niza na tri dela: elemente koji su manji od pivota, elemente koji su jednaki pivotu i elemente koji su veći od njega. Zatim se algoritam

rekurzivno poziva za sortiranje levog dela niza (gde su u tom trenutku svi elementi manji od pivota) i za sortiranje desnog dela niza (gde su u tom trenutku svi elementi veći od pivota). Pivot može biti prvi, srednji, posledenji ili proizvoljni element trenutnog niza, ili izabran na neki drugi način.

**def** quick Sort ( n i z ) :

**i f len** ( n i z ) <= 1 :

**return** n i z

## else :

manji = [ ]

je d n a k i = [ ] v e c i = [ ]

p ivot = n i z [ 0 ]

**for** b r o j **in** n i z :

**i f** b r o j < p ivot :

manji . append ( b r o j )

**i f** b r o j == p ivot :

je d n a k i . append ( b r o j )

**i f** b r o j > p ivot :

v e c i . append ( b r o j )

**return** quick Sort ( manji ) + je d n a k i + quick Sort ( v e c i )

n i z = [ 5 , 1 , 2 , 7 , 10 , 3 , 9 , 13 , 1 2 ]

**print** ( " Niz . pre . s o r t i r a n j a : " )

**print** ( n i z )

n i z = quick Sort ( n i z )

**print** ( " Niz . nakon. s o r t i r a n j a : " )

**print** ( n i z )