PANEVROPSKI UNIVERZITET APEIRON, BANJA LUKA

FAKULTET INFORMACIONIH TEHNOLOGIJA

Redovne studije

Smjer : “Inženjering Informacionih Tehnologija“

Predmet:

ALGORITMI I STRUKTURE PODATAKA

ALGORITMI ZA SORTIRANJE I NJIHOVA PRIMJENA U PROGRAMSKOM JEZIKU PYTHON

(Seminarski rad)

Predmetni nastavnik

Prof. dr Zoran Avramović

Student: Pavlović Ivan

Br. Indeksa : 92-20/RITP-S

Banja Luka, jul 2021.

Sadržaj

UVOD

Nizovi predstavljaju veoma važne strukture pomoću kojih na jednom velikom mjestu, smještamo određeni broj informacija istog tipa. Poznavanje rada sa nizovima je krucijalan uslov koji će biti potreban za rad u većini programskih jezika.

Sortiranje podataka je kompleksan dio programiranja u kojoj je potreban što efikasniji i brži metod da bi to bilo moguće. Da bi sortiranje bilo efikasnije informatičari su izmislili sortiranje selekcijom, sortiranje umetanjem, sortiranje mjehurom itd.

1. STUKTURE PODATAKA

Struktura podataka je način predstavljanja podataka u računarskoj memoriji, kojim se omogućuje njihovo lako predstavljanje i obrada.

Najbitnije strukture su: nizovi, liste, stekovi, redovi, grafovi i binarna stabla.

1.1. Nizovi

Kao elementarne strukture mogli bi se navesti nizovi – mada, neko se možda nece složiti da su nizovi strukture.

Nizovi su strukture podataka koje se mogu koristiti za čuvanje velikog broja istorodnih podataka. U računarskoj memoriji se uglavnom realizuju kao kontinualni memorijski blokovi.

Direktan pristup je veoma efikasan, kao i sekvencijalan. Također, postoji veliki broj efikasnih algoritama za pretraživanje nizova i uređivanje nizova po nekom kriterijumu.

Na primjer: ako je adresa početka niza A, a traži se i-ti element niza, do njega se dolazi veoma jednostavno.

a[i] = vrijednost lokacije (A + i * velicina_pojedinacnog_el_niza)

Mane nizova su veoma zahtjevno umetanje elemenata između dva vec postojeća, njihovo brisanje (potrebno je pomjeriti sve elemente niza od mjesta gdje se umeću jedno mjesto prema kraju niza).

1.2. Liste

Lista je struktura podataka koja se odlikuje linearnim rasporedom pripadajućih elemenata. Po svojom prirodi, lista je najsrodnija nizu, ali se uglavnom implementira koristeci dinamičko alociranje memorije i pokazivače.

1.2.1. Osobine listi

Svaka lista mora da zadovoljava sljedeće osobine:

- Lista moze biti prazna

- Moguce je ubaciti novi element na bilo koju poziciju u listi
- Moguce je izbaciti bilo koji element iz liste
- Lista ima svoj broj elemenata
- Svakom elementu liste se moze pristupiti preko rednog broja, tj. indeksa

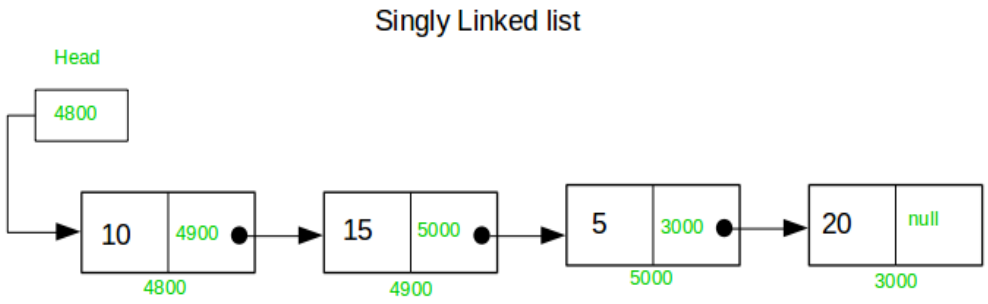
1.2.2. Vrste listi

Po vrsti povezanosti, liste se najčešće dijele na jednostruko povezane i dvostruko povezane. Po ovom obliku, dijele se na linearne i kruzne liste.

1.2.2.1. Jednostruko povezane liste

Jednostruko povezane liste podržavaju pretraživanje elemenata samo u jednom smjeru. Kod implementacija liste pomoću pokazivača i dinamičke memorije, svaki element sadrži po tačno jedan pokazivač koji pokazuje na sljedeći element u listi.

Najčešće se koriste u situacijama kada je jednosmjerno pretraživanje dovoljno za dati problem.

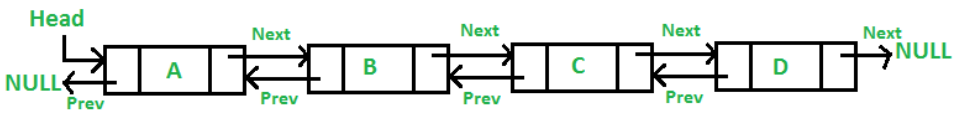


Slika 1 Jednostruko povezana lista

1.2.2.2. Dvostruko povezane liste

Dvostruko povezane liste su liste po kojima je moguće pretraživanje pripadajućih elemenata u dva smjera. Ovo se u programiranju najčešće implementira postojanjem dva pokazivača u svakom elementu liste, od kojih jedan pokazuje na prethodni, a drugi na sljedeci element u listi.

Ova vrsta liste je jako udobna za rad, i u većini slučajeva olakšava rješenje u odnosu na jednostruku povezanu listu.

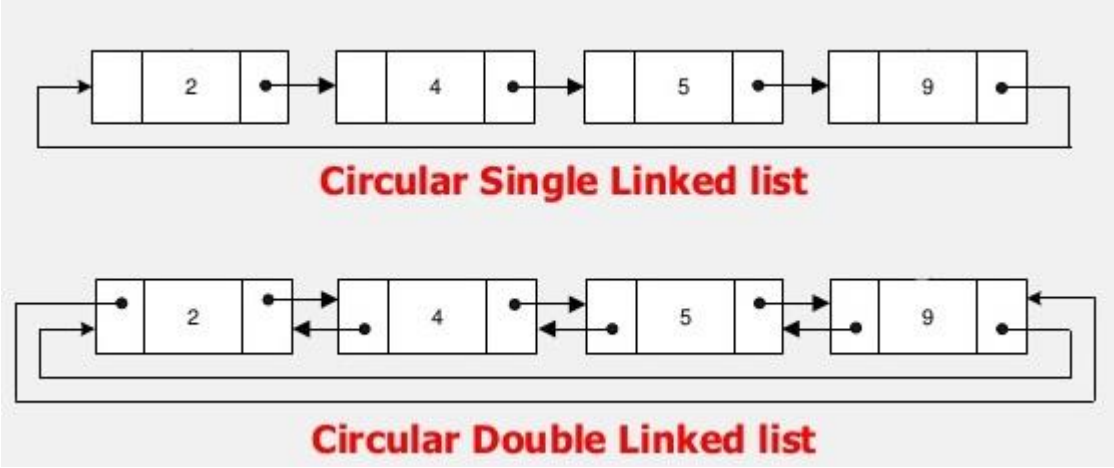


Slika 2 Dvostruko povezana lista

1.2.2.3. Kružne liste

Kružne liste se odlikuju odsustvom početka liste. Naime, za razliku od običnih listi koje imaju jasno izražen početni element i krajnji element, čiji pokazivač je jednak nuli, posljednje ubačeni element liste se uvijek postavlja da pokazuje na prvi, a prvi element moze da šeta, tj. u svakom trenutku bilo koji element moze da se proglasi za prvi bez remećenja strukture.

Na ovaj način se postiže da se iteracija po listi ne mora nikada završiti, jednostavno prelazeci svaki put na sljedeci element, sto čini određene algoritme jednostavnijim.



Slika 3 Kruzna lista

1.3. Stek

Stek je struktura podataka u memoriji slična nizu u kojoj podaci mogu biti spremljeni i iz nje uklonjeni sa lokacije koja se naziva „vrh“ (top) steka.

Podaci koji trebaju biti spremljeni se „guraju“ (push) u stek, a podaci koje treba dobiti se „vade“ (pop) iz steka.

Stek je LIFO (Last in First Out) struktura podataka, tj. podaci koji su spremljeni prvi se dobijaju posljednji.

1.3.1. Python funkcije koje su povezane sa stekom

Funkcije povezane sa stekom su:

- empty() – Prikazuje da li je stek prazan
- size() – Prikazuje veličinu steak
- top() – Vraca reference na najviši element steka
- push(g) – Dodaje element “g” na vrh steka
- pop() – Briše gornji element steka

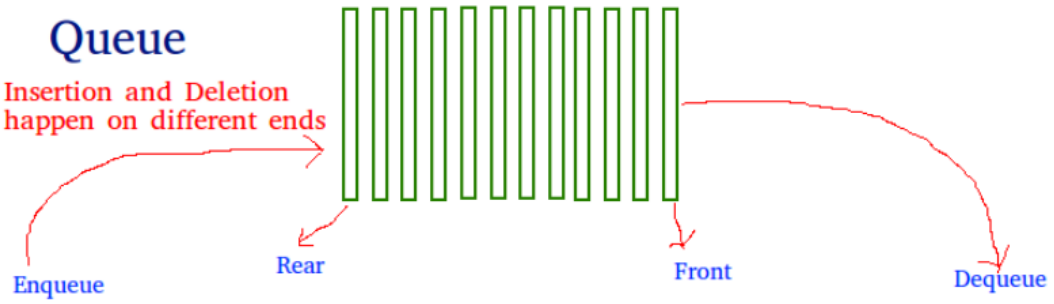
```
# Python program to
# demonstrate stack implementation
# using list


stack = []

# append() function to push
# element in the stack
stack.append('a')
stack.append('b')
stack.append('c')

print('Initial stack')
print(stack)

# pop() fucntion to pop
# element from stack in
# LIFO order
print('\nElements poped from stack:')
print(stack.pop())
print(stack.pop())
print(stack.pop())

print('\nStack after elements are poped:')
print(stack)

# uncommenting print(stack.pop())
# will cause an IndexError
# as the stack is now empty
```

Slika 4 Primjer stack-a u Python-u

1.4. Red

Kao i stek, red (queue) je linearna struktura podataka koja skladišti stavke na nacin FIFO (First in First Out).

Pomocu reda prvo se uklanja najmanje dodata stavka. Dobar primjer reda je bilo koji red potrošaca za resurs gdje se prvo usluži potrošač koji je došao prvi.



Ilustracija 1Slikoviti prikaz Reda

1.4.1. Implementacija Reda pomoću liste

Lista je ugrađena Python-ova struktura podataka koja se može koristiti kao red. Umjesto enqueue() i dequeue(), koristi se append() i pop() funkcija.

Međutim liste su poprilično spore jer umetanje ili brisanje elemenata na početku zahtijeva pomjeranje svi ostalih elemenata.

```
# Python program to
# demonstrate queue implementation
# using list

# Initializing a queue
queue = []

# Adding elements to the queue
queue.append('a')
queue.append('b')
queue.append('c')

print("Initial queue")
print(queue)

# Removing elements from the queue
print("\nElements dequeued from queue")
print(queue.pop(0))
print(queue.pop(0))
print(queue.pop(0))

print("\nQueue after removing elements")
print(queue)

# Uncommenting print(queue.pop(0))
# will raise and IndexError
# as the queue is now empty
```

Slika 5 Implementacija reda koristeci listu u Python-u

2. Algoritmi za sortiranje

2.1. Uvod u algoritme za sortiranje

Sortiranje je, bez sumnje, najosnovniji algoritamski problem sa kojim se suočavamo u ranim danima na računaru.

U stvari, većina istraživanja računarskih nauka bila je fokusirana na pronalaženje najboljeg načina za sortiranje skupa podataka. Tokom godina, informatičari su stvorili mnogo algoritama za sortiranje podataka.

Neki od njih su:

- Bubble sort
- Selection sort
- Insertion sort
- Merge sort

2.2. Bubble sort

Babl sort ili Mehurićasto sortiranje je jedan od jednostavnijih algoritama za sortiranje koji neprekidno prolazi kroz listu, uspoređuje susjedne elemente i zamjenjuje ih ako su u pogrešnom redosljedu. Prolazak kroz listu se ponavlja dok se lista ne sortira.

Ovaj jednostavan algoritam ima loše rezultate u stvarnom svijetu i koristi se prvenstveno kao obrazovni alat.

Efikasnije algoritme kao sto su „quick sort“, „time sort“ ili „merge sort“ koriste biblioteke za sortiranje ugrađene u popularne programske jezike kao sto su Python i Java.

Slika 6 Slikoviti prikaz Bubble sort-a

2.2.1. Performanse Bubble sort-a

Babl sort ima najgoru složeno O (n^2), gdje je *n* broj elemenata koji se sortiraju. Postoji mnogo algoritama za sortiranje koji imaju znatno bolju složenost O (*n* log *n*). Čak i drugi algoritmi složenosti O (n^2), kao što sortiranje umetanjem, tzv. Insertion sort, imaju tendenciju da imaju bolje performanse od bablsorta. Dakle, babl sort nije praktičan algoritam za sortiranje kada je *n* veliko.

Jedina značajna prednost bablsorta za razliku od drugih implementacija, čak i „Quick sort-a“, ali ne „Insertion sort-om“, je sposobnost da otkrije da je sortiran niz efikasno ugrađen u algoritam.

Složenost babl sort-a nad već sortiranim nizovima (u najboljem slučaju) je O (*n*). Nasuprot tome, većina drugih algoritama, čak i oni sa boljom prosečnom složenošću, obavljaju ceo proces sortiranja na setu i na taj način su složeniji.

Međutim, ne samo da insertion sort ima ovaj mehanizam, već i bolje radi na nizu koji je znatno sortiran.

2.2.2. Zečevi i kornjače

Pozicije elemenata u bablsortu igraju veliku ulogu u određivanju složenosti. Veliki elementi na početku niza ne predstavljaju problem jer se brzo zamjene.

Mali elementi pri kraju niza se kreću na početak veoma sporo. Zbog toga se ove vrste elemenata respektivno nazivaju „zečevi“ i „kornjače“.

Učinjeni su razni napori da se eliminišu kornjače kako bi se poboljšala brzina bablsorta.

Koktel sort je dvosmjerni bablsort koji ide od početka do kraja, a onda se poništava i ide od kraja do početka. On moze da pomjera kornjače prilično brzo, ali zadržava složenost O (n^2).

Kombsort poredi elemente razdvojene velikim prazninama, a moze da pomjera kornjače izuzetno brzo prije nego sto pređe na manje praznine. Njegova prosječna brzina se može brzim algoritmima kao što su „Quick sort“.

2.2.3. Korak po korak primjer

Niz brojeva „5 1 4 2 8“ potrebno je sortirati od najmanjeg do najvećeg broja (u rastućem poretku) pomoću bablsorta. U svakom koraku **boldirani elementi** se uspoređuju.

Biće potrebna tri prolaska kroz niz:

Prvi prolaz:

(**5** 1 4 2 8) → (1 **5** 4 2 8), Algoritam upoređuje prva dva elementa i zamjenjuje 1 i 5, 5 > 1;

(1 **5 4** 2 8) → (1 **4 5** 2 8), zamjena pošto je 5 > 4;

(1 4 **5 2** 8) → (1 4 **2 5** 8), zamjena pošto je 5 > 2;

(1 4 2 **5 8)** → (1 4 2 **5 8**), sada, pošto su ovi elementi već sortirani (8 > 5), algoritam ih nece zamjeniti.

Drugi prolaz:

(1 4 2 **5 8**) → (**1 5** 4 2 8)

(1 **4 2** 5 8) → (1 **2 4** 5 8), zamjena pošto je 4 > 2;

(1 2 **4 5** 8) → (1 2 **4 5** 8)

(1 2 4 **5 8**) → (1 2 4 **5 8**)

Sada, niz je već soriran, ali naš algoritam još uvijek ne zna da je sortiranje završeno. Algoritam je završen tek kada prođe kroz čitav niz bez i jedne zamjene.

Treci prolaz:

(**1 2** 4 5 8) → (**1 2** 4 5 8)

(1 **2 4** 5 8) → (1 **2 4** 5 8)

(1 2 **4 5** 8) → (1 2 **4 5** 8)

(1 2 4 **5 8**) → (1 2 4 **5 8**)

2.3. Selection sort

U računarskim naukama, eng. „Selection sort“ je algoritam sortiranja koji posjeduje vremensku složenost O (n^2), što ga čini nefikasnim na velikim listama, i uglavnom imaju lošije rezultate od slične vrste unosa.

Sortiranje izbora je zapaženo zbog jednostavnosti i ima prednost u performansama u odnosu na složenije algoritme u određenim situacijama, posebno tamo gdje je pomoćna memorija ograničena.

Algoritam dijeli ulaznu listu na dva dijela: sortiranu podlistu stavki koja se gradi slijeva nadesno, na prednjoj strani (lijeva) liste i podlistu preostalih nerazvrstanih stavki koje zauzimaju ostatak liste. U početku je sortirana podlista prazna, a nesortirana je čitava lista unosa.

Algoritam se nastavlja pronalazenjem najmanjeg (ili najveceg, u zavisnosti od redosljeda sortiranja), elemenata u nesortiranoj podlisti, zamjenjujuci ga sa krajnje lijevim nesortiranim elementom i pomjerajući granice podsistema za jedan element udesno.

Vremenska efikasnost selekcionog sortiranja je kvadratna, pa postoji niz tehnika sortiranja koje imaju bolju vremensku složenost od selekcionog sortiranja. Jedna stvar koja razlikuje sortiranje odabira od ostalih algoritama za sortiranje je ta što vrši najmanji mogući broj zamjena, *n – 1* u najgorem slučaju.

Sorted sublist	Unsorted sublist	Least element in unsorted list
()	(11, 25, 12, 22, 64)	11
(11)	(25, 12, 22, 64)	12
(11, 12)	(25, 22, 64)	22
(11, 12, 22)	(25, 64)	25
(11, 12, 22, 25)	(64)	64
(11, 12, 22, 25, 64)	()	

(Izgleda da se na ova posljednja dva reda ništa nije promijenilo jer su posljednja dva broja već bila u redu).

Sortiranje izbora također se može koristiti na strukturama liste koje čine dodavanje i uklanjanje efikasnim, kao što je povezana lista. U ovom slučaju je uobičajeno ukloniti minimalni element od ostatka liste, a zatim ga umetnuti na kraj, dosada sortiranih vrijednosti.

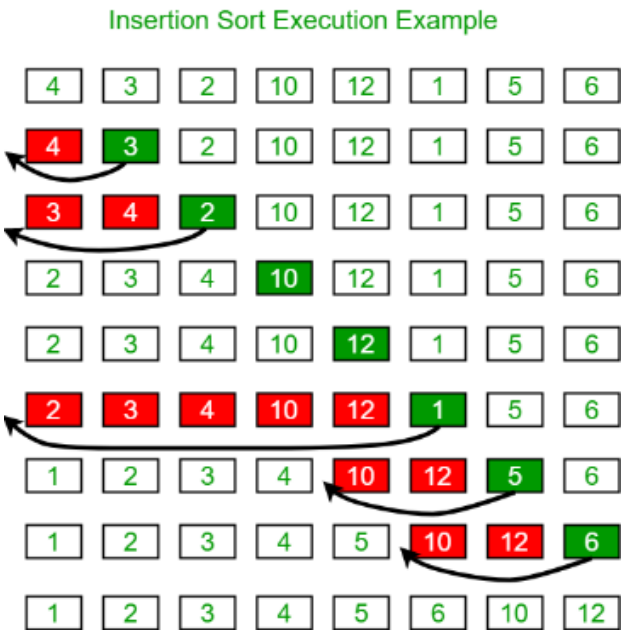
Na primjer:

```
arr[] = 64 25 12 22 11

// Find the minimum element in arr[0...4]
// and place it at beginning
11 25 12 22 64

// Find the minimum element in arr[1...4]
// and place it at beginning of arr[1...4]
11 12 25 22 64

// Find the minimum element in arr[2...4]
// and place it at beginning of arr[2...4]
11 12 22 25 64

// Find the minimum element in arr[3...4]
// and place it at beginning of arr[3...4]
11 12 22 25 64
```

2.4. Insertion sort

Insertion sort, srp. Sortiranje umetanjem je jednostavan algoritam za sortiranje koji radi slično načinu na koji sortirate karte u rukama. Niz je praktično podijeljen na sortirani i nesortirani dio.

Vrijednost iz nesortiranog dijela se biraju i postavljaju na tacan položaj u sortiranom dijelu.



Slika 7 Ilustracija "Insertion sort-a"

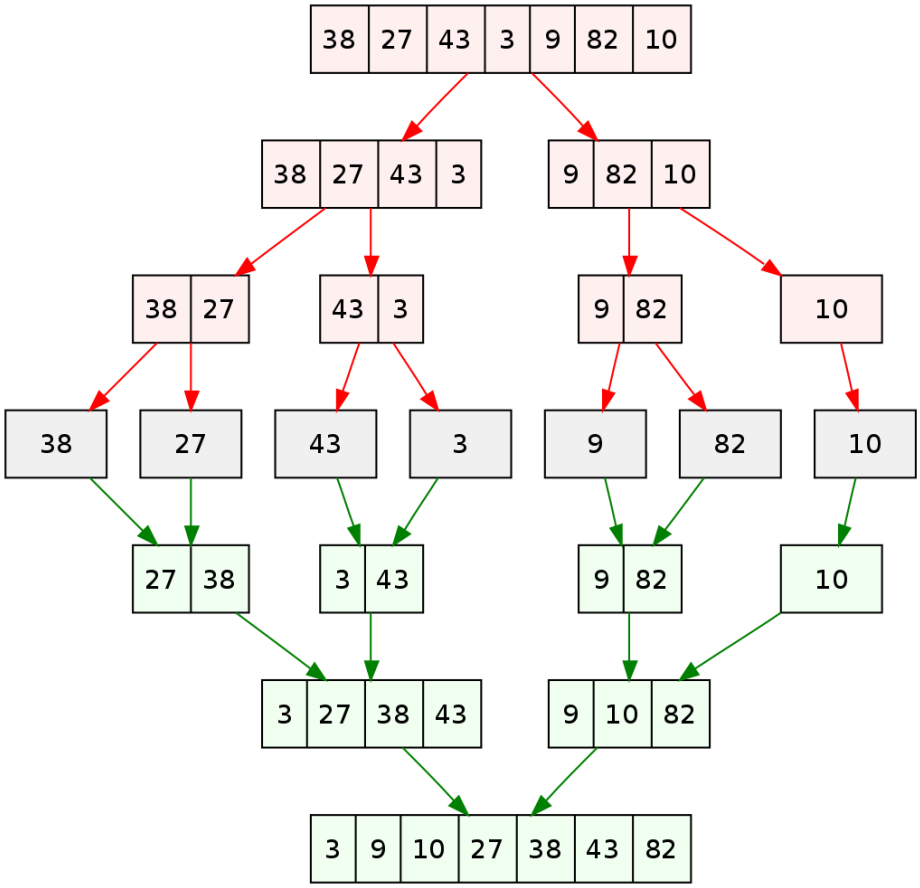
2.5. Merge sort

Merge sort, srp. Sortiranje umetanjem je efikasan algoritam sortiranja zasnovan na opštoj namjeni i zasnovan na poređenju.

Većina implementacija proizvodi stabilno sortiranje, što znači da je redoslijed jednakih elemenata jednak na ulazu i na izlazu.

U konceptu Merge sort, radi na sljedeci način:

1. Podjelimo nesortiranu listu na *n* podlista, od kojih svaka sadrži jedan element (lista jednog elementa smatra se sortiranim).
2. Neprekidno spajamo podliste da bismo stvorili nove sortirane podliste dok ne ostane samo jedna podlista. Ovo ce biti sortirana lista.



Slika 8 Ilustracija "Merge sort-a"

2.5.1. Natural Merge sort

Prirodna sorta spajanja, slična je sortiranju odozdo, prema gore, osim što se koriste svi prirodni pokreti u ulazu.

Mogu se koristiti i monotoni i bitonski (naizmjenično gore/dole) pokreti, pri čemu su liste (ili ekvivalentne trake ili datoteke) prikladne strukturi podataka (koriste se kao FIFO redovi ili LIFO stekovi).

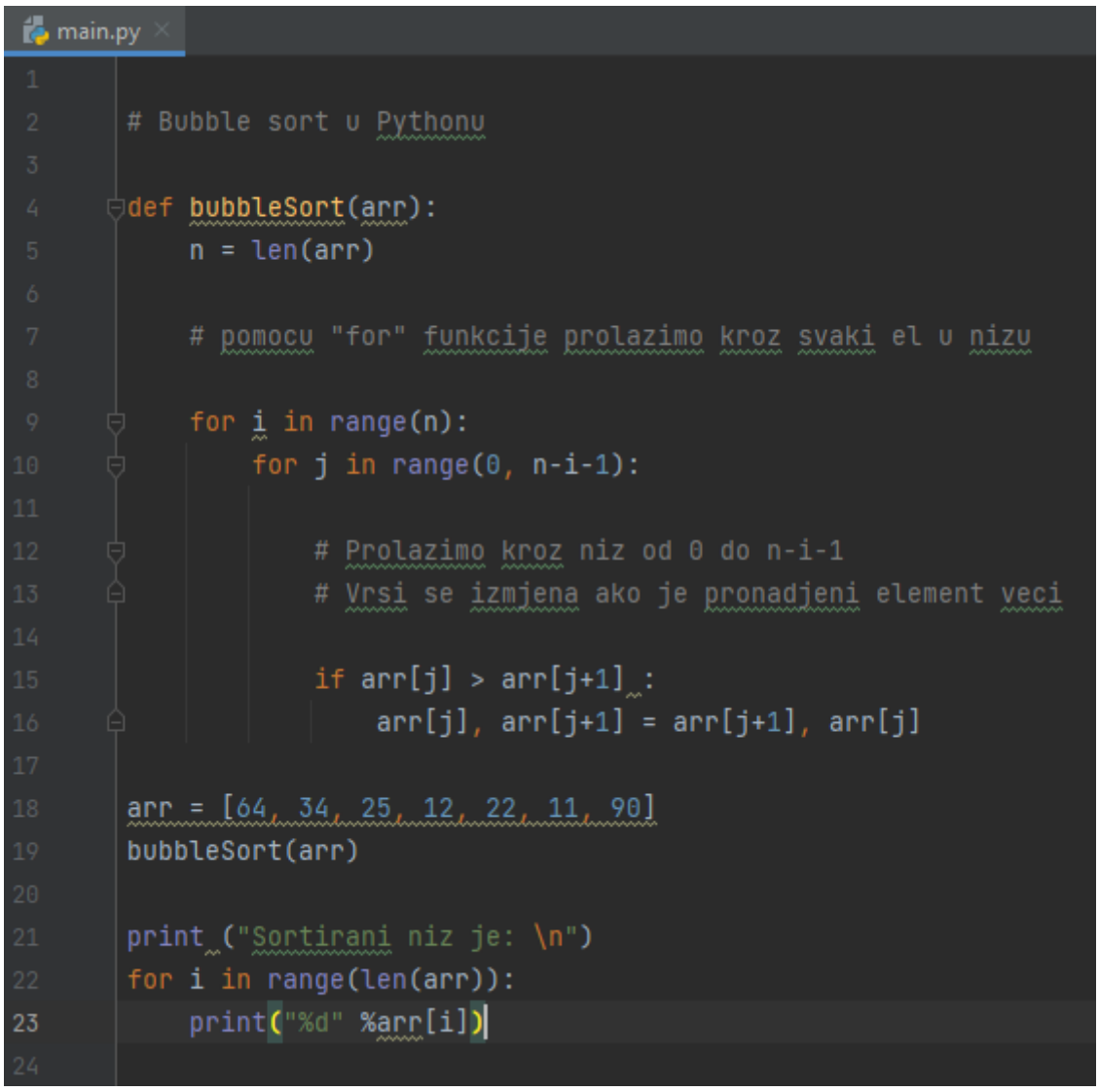
U sortiranju spajanja odozdo prema gore, početna tačka predpodstavlja da je svako pokretanje dugačko po jednu stavku. U praksi ce slučajni ulazni podaci imati mnogo kratkih prolaza koji se slučajno sortiraju. U tipičnom slučaju, prirodnog merge sortiranja mozda neće trebati toliko prolaza jer je manje pokretanja za spajanje.

U najboljem slučaju, ulaz je već sortiran (tj. jedan je potez), tako da prirodno sortiranje treba samo da prođe kroz podatke. U mnogim praktičnim slučajevima prisutni su dugi ciklusi, pa se iz tog razloga prirodna vrsta spajanja koristi kao ključna komponenta „Team sort-a“.

```
Start        :  3  4  2  1  7  5  8  9  0  6
Select runs  : (3  4)(2)(1  7)(5  8  9)(0  6)
Merge        : (2  3  4)(1  5  7  8  9)(0  6)
Merge        : (1  2  3  4  5  7  8  9)(0  6)
Merge        : (0  1  2  3  4  5  6  7  8  9)
```

3. Implementacija algoritama za sortiranje u programskom jeziku Python

3.1. Bubble sort (Python code)



Slika 9 Bubble sort program u PyCharm-u

```
C:\Users\Ivan\PycharmProjects\pythonP
Sortirani niz je:

11
12
22
25
34
64
90

Process finished with exit code 0
```

Slika 10 Rezultat Slike 9

i = 0	j	0	1	2	3	4	5	6	7
	0	5	3	1	9	8	2	4	7
	1	3	5	1	9	8	2	4	7
	2	3	1	5	9	8	2	4	7
	3	3	1	5	9	8	2	4	7
	4	3	1	5	8	9	2	4	7
	5	3	1	5	8	2	9	4	7
	6	3	1	5	8	2	4	9	7
i=1	0	3	1	5	8	2	4	7	**9**
	1	1	3	5	8	2	4	7	
	2	1	3	5	8	2	4	7	
	3	1	3	5	8	2	4	7	
	4	1	3	5	2	8	4	7	
	5	1	3	5	2	4	8	7	
i=2	0	1	3	5	2	4	7	**8**	
	1	1	3	5	2	4	7		
	2	1	3	5	2	4	7		
	3	1	3	2	5	4	7		
	4	1	3	2	4	5	7		
i=3	0	1	3	2	4	5	**7**		
	1	1	3	2	4	5			
	2	1	2	3	4	5			
	3	1	2	3	4	5			
i=4	0	1	2	3	4	**5**			
	1	1	2	3	4				
	2	1	2	3	4				
i=5	0	1	2	3	**4**				
	1	1	2	3					
i=6	0	1	2	**3**					
		1	**2**						

Slika 11 Slikoviti prikaz zadatka

3.2. Selection sort (Python code)

main.py

```
# Selection sort u Pythonu
# importujemo funkciju sys da pristupimo dodatnim varijablama itd
import sys

A = [64, 25, 12, 22, 11]

# Pomocu for funkcije prolazimo kroz svaki el u nizu
for i in range(len(A)):

    # Unosimo uslov za najmanji element
    min_index = i
    for j in range(i+1, len(A)):
        if A[min_index] > A[j]:
            min_index = j

    # Vrsimo zamjenu najmanjeg pronadjenog elementa sa prvim el
    A[i], A[min_index] = A[min_index], A[i]

print("Sortirani niz je: \n")
for i in range(len(A)):
    print("%d" %A[i])
```

Slika 12 Selection sort u PyCharm-u

```
C:\Users\Ivan\PycharmProjects\pytho
Sortirani niz je:

11
12
22
25
64

Process finished with exit code 0
```

Slika 13 Rezultat Slike 12

3.3. Insertion sort (Python code)

main.py

```
# Zapocinjemo tako sto pravimo nasu funkciju za sortiranje unosom
def insertion_sort(arr):

    for i in range(1, len(arr)):

        key = arr[i]

        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j+1] = arr[j]
            j -= 1
        arr[j + 1] = key

arr = [12, 11, 13, 5, 6]

insertion_sort(arr)
print("Sortirani niz je: \n")
for i in range(len(arr)):
    print("%d" %arr[i])
```

Slika 14 Insertion sort u PyCharm-u

```
C:\Users\Ivan\PycharmProjects\python
Sortirani niz je:

5
6
11
12
13

Process finished with exit code 0
```

Slika 15 Rezultat Slike 14

ZAKLJUČAK

Algoritmi predstavljaju veliki značaj u informatičkom svijetu sa kojima se nesvjesno susrećemo. Pomoću algoritama mogu se vršiti planovi u biznisu ili u predviđanju ishoda događaja.

Razvojem računara i programskih jezika, algoritmi imaju sve veći značaj i njigova primjena je sve češća.

Kombinacijom algoritama i vještačke inteligencije bi moglo dovesti do izvanrednih softverskih rješenja.

4. POPIS SLIKA

5. LITERATURA

1. “The Art Of Computer Programming-Addison-Wesley Professional”, Donald E. Knuth, 2006
2. “GeeksForGeeks”, URL: https://www.geeksforgeeks.org/data-structures/
3. “Wikipedia”, URL: https://sh.wikipedia.org/wiki/Struktura_podataka

   https://sh.wikipedia.org/wiki/Lista_(informatika)

   https://sh.wikipedia.org/wiki/Struktura_podataka

   https://hr.wikipedia.org/wiki/Red_(struktura_podataka)