Blok 02
Struktura podataka i algoritmi

prof. Dr Vidan Marković

Univerzitet Singidunum
2023/2024

Sadržaj

1. Uvod u algoritme

2. Diskusija

Algoritmi sortiranja pregled

- Možda ćemo morati da sortiramo listu brojeva, pre nego što primenimo određeni skup uputstava na njega.
- U tom slučaju, napisali bismo jedan red pseudokoda, što ukazuje da sortiramo listu.
- Ali kakva bi bila složenost ove linije? Koliko je efikasno sortiranje? Pa, zavisi od algoritma koji koristimo za to.

Algoritmi sortiranja pregled: pohlepni *greedy* algoritmi

- Pre nego što počnemo da razmatramo pojedinačne algoritme, ukratko ćemo porazgovarati o dve velike kategorije algoritama.
- Prvi su takozvani pohlepni algoritmi. Glavna karakteristika ovde je da uvek naprave izbor koji je izgleda najbolji u datom trenutku.
- Oni su najednostavniji ali nisu uvek i najbolji.

Algoritmi sortiranja pregled: pohlepni *greedy* algoritmi

- Primer: doći do sume od 25c u najmanje iskorišćenih novčića, svakih ima neograničeno po kategoriji. Greedy algoritam bi pronašao najveći novčić koji je najbliži po vrednosti datoj sumi dok se ne dostigne željena ukupna vrednost.

- Kako ovo može da postane problem. Ako je potrebno 40 centi, šta bi pohlepni algoritam uradio? Videće da je najveći mogući novčić 25, pošto je manji od 40 i izabrao bi njega. Ovo bi nam ostavilo preostalih 15 centi. Zato biramo 10 centi i 5 centi i završimo sa tri novčića. Međutim, ako pogledate novčiće, možete videti da postoji efikasnije rešenje, koje koristi dva novčića od 20 centi.

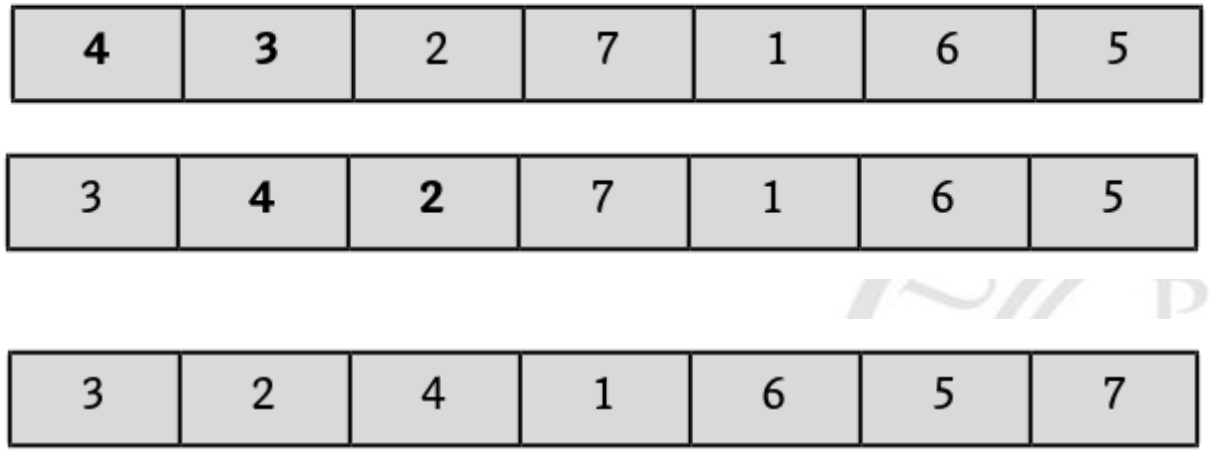
Algoritmi sortiranja pregled: podeli i osvoji

- Ono što oni u suštini rade je da dele instancu problema na manje probleme, deleći ih iznova i iznova, sve dok veličina problema ne bude toliko mala da je rešenje gotovo trivijalno. Ova pod-rešenja se zatim kombinuju u potpuno rešenje početnog problema.

- Najefikasniji algoritmi sortiranja kao što su ukršteno (merge) sortiranje i brzo sortiranje su algoritmi za deljenje i osvajanje.

Algoritmi sortiranja pregled: bubble sort

- Algoritam sortiranja mehurića uvek poredi dva susedna (znači susedna) elementa i zamenjuje ih ako nisu u pravom redu. To radi iznova i iznova dok se cela lista ne sortira.



I tako dalje dok se cela lista ne sortira

Algoritmi sortiranja pregled: bubble sort

- Medjutim očigledno se ne može sve završiti u jednoj iteraciji. Ono što je sigurno da najveći element zazme posle iteracije mesto na kraju. Što znači da ovo u najgorem slučaju treba da prolazimo n puta i da pri svakoj novoj iteraciji imamo jedan manje element za sortiranje.

function bubble_sort(list):

```
for i := 0 to size(list):
    for j := 0 to size(list) – i - 1:
        if list[j] > list[j+1]:
            swap list[j] and list[j+1]
```

```
for i := 0 to size(list):  (n times)
    for j := 0 to size(list) – i - 1: (n-i times)
        if list[j] > list[j+1]: (1 PO)
            swap list[j] and list[j+1] (1 PO)
```

$$2 * \sum_{i=1}^{n} (n - i) = n * (n - 1)$$

$$n * (n - 1) = n^2 - n \rightarrow \quad \theta(n^2)$$

Algoritmi sortiranja pregled: sortiranje selekcijom

- U suštini traži najmanji element na listi i zamenjuje ga elementom na prvoj poziciji.
- Zatim ponavlja isti proces za ostatak liste, pronalazi drugi najmanji element i zamenjuje ga elementom na drugoj poziciji.
- Kada ovo uradimo n puta, lista je definitivno sortirana.

```
function selection_sort(list):

    for i := 0 to size(list):
        min_value = list[i]
        min_index = i
        for j := i to size(list):
            if list[i] < min_value:
                min_value = list[j]
                min_index = j
        swap list[i] and list[min_index]
```

```
for i := 0 to size(list): (n times)
    min_value = list[i] (1 PO)
    min_index = i (1 PO)
    for j := i to size(list): (n-i times)
        if list[i] < min_value: (1 PO)
            min_value = list[j] (1 PO)
            min_index = j (1 PO)
    swap list[i] and list[min_index] (1 PO)
```

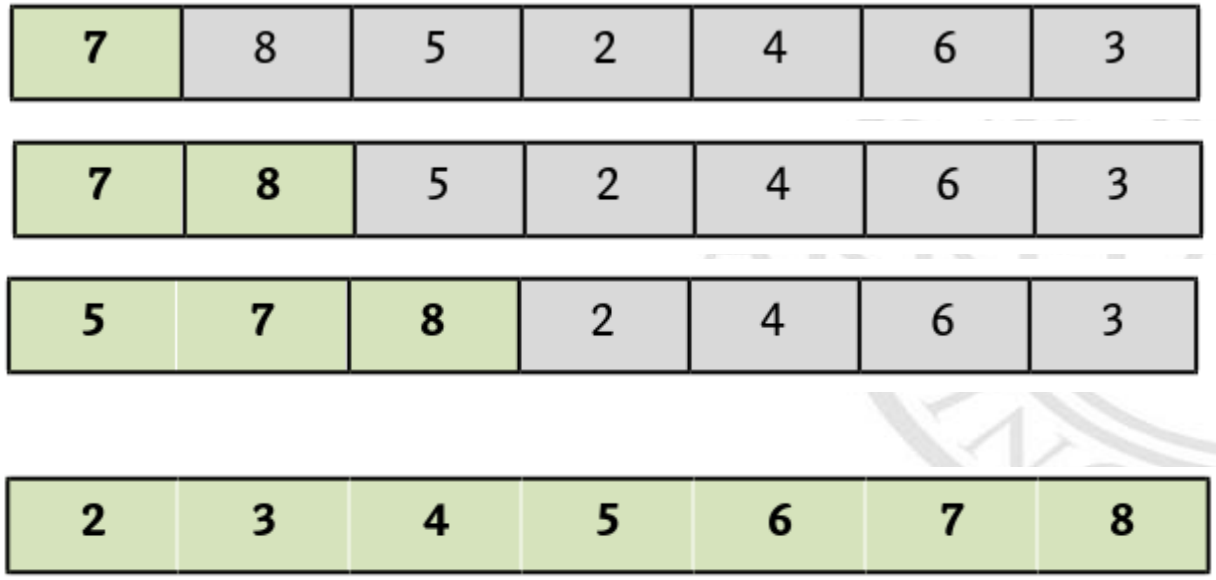
$$3n + 3 * \sum_{i=0}^{n} (n - i) = 3n + \frac{3n * (n+1)}{2}$$

$$3n + \frac{3n * (n+1)}{2} = 3n + \frac{3n^2 + 3n}{2} \rightarrow \theta(n^2)$$

Algoritmi sortiranja pregled: sortiranje ubacivanjem

- Osnovna ideja ovde je da se lista podeli na sortirani i neobrađeni deo liste.

- Primer: U početku samo izaberemo prvi element i kažemo da je sortiran. Dakle, sedmica je sortirani odeljak i svaki element desno pripada nesortiranom odeljku. Sada u sledećem koraku gledamo prvi nesortirni element i određujemo gde mu je mesto na sortiranom spisku. Pošto je osmica veća od sedam, možemo je ostaviti tamo i pomeriti označivač za sortirani odeljak za jedan.



Algoritmi sortiranja pregled: sortiranje ubacivanjem

```
function selection_sort(list):

    for i := 1 to size(list):
        value := list[i]
        j := i
        while j > 0 and list[j-1] > value
            list[j] = list[j-1]
            j = j – 1
        list[j] = value
```

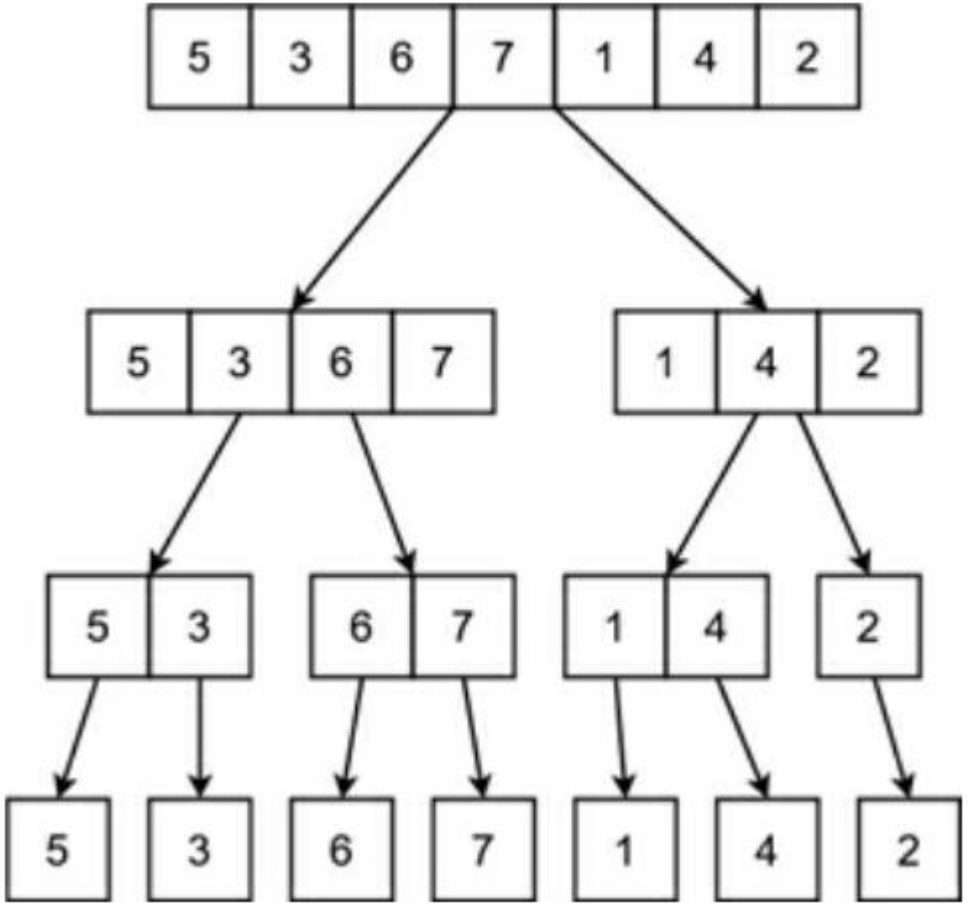
```
for i := 1 to size(list): (n-1 times)
    value := list[i] (1 PO)
    j := i (1 PO)
    while j > 0 and list[j-1] > value: (i times)
        list[j] = list[j-1] (1 PO)
        j = j – 1 (1 PO)
    list[j] = value (1 PO)
```

$$3(n-1) + 2 * \sum_{i=1}^{n-1} i = 3n - 3 + n * (n-1)$$

$$3n - 3 + n * (n-1) = n^2 + 2n - 3 \rightarrow \theta(n^2)$$

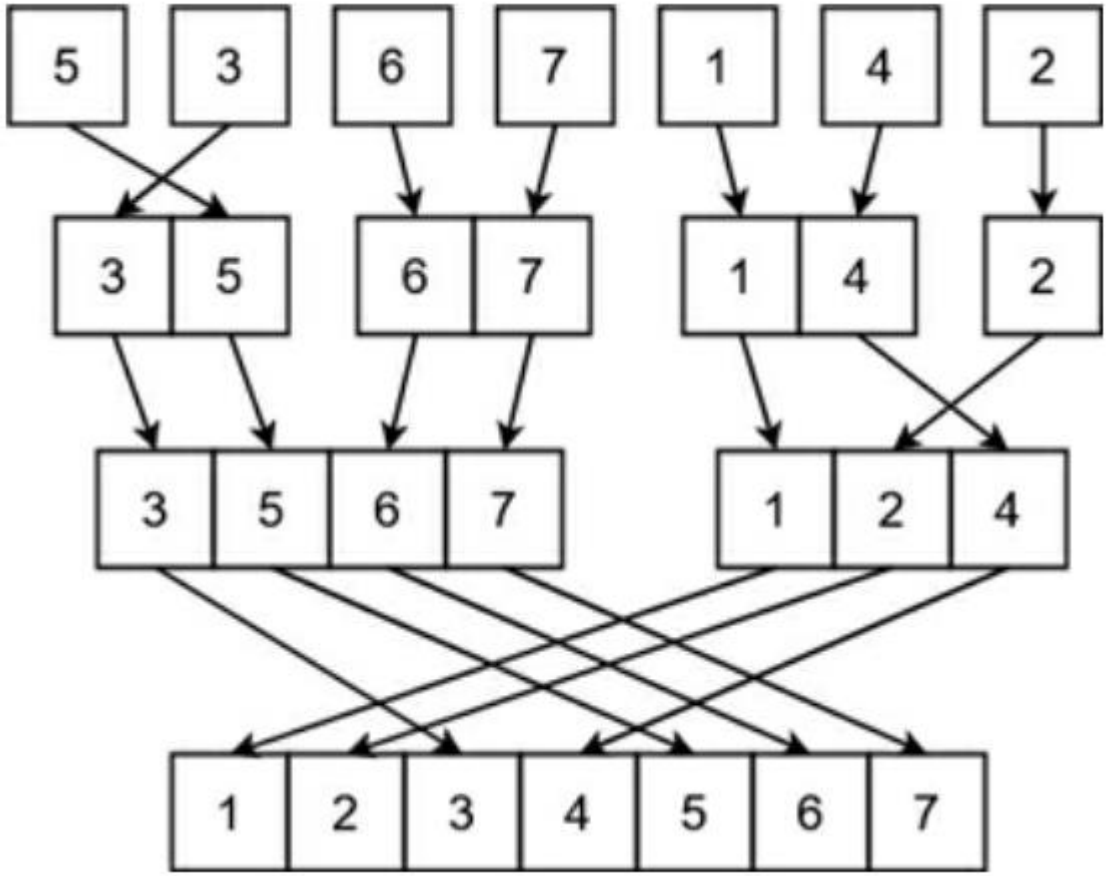
Algoritmi sortiranja pregled: sortiranje spajanjem

- Najzad idemo u zonu "efikasnih" algoritama sortiranja – algoritmi za podelu i osvajanje.
- Prvi je takozvani sort spajanja. Kao što ime sugeriše, ona deli listu na pod-liste iznova i iznova, sortira ih, a zatim objedinjuje pod-rešenja u potpuno sortiranu listu.



Algoritmi sortiranja pregled: sortiranje spajanjem

- Pošto su sve pod-liste (u početku one dužine jedna) već sortirane, sve što treba da uradimo je da uporedimo najmanji element jedne liste sa najmanjim elementom druge liste i izaberemo manji.



Algoritmi sortiranja pregled: sortiranje spajanjem

- U suštini, sortiranje objedinjavanja se može rezimirati u tri koraka:

    1. Deljenje date liste na pod-liste
    2. Poziv koji se ponavlja za sortiranje
    3. Objedinjavanje rezultata

- Deljenje liste može se obaviti u konstantnom roku, tako da za sada možemo da zanemarimo ovaj korak.

- Objedinjavanje rezultata se obavlja u linearno vreme, pošto moramo da objedinimo n elemenata.

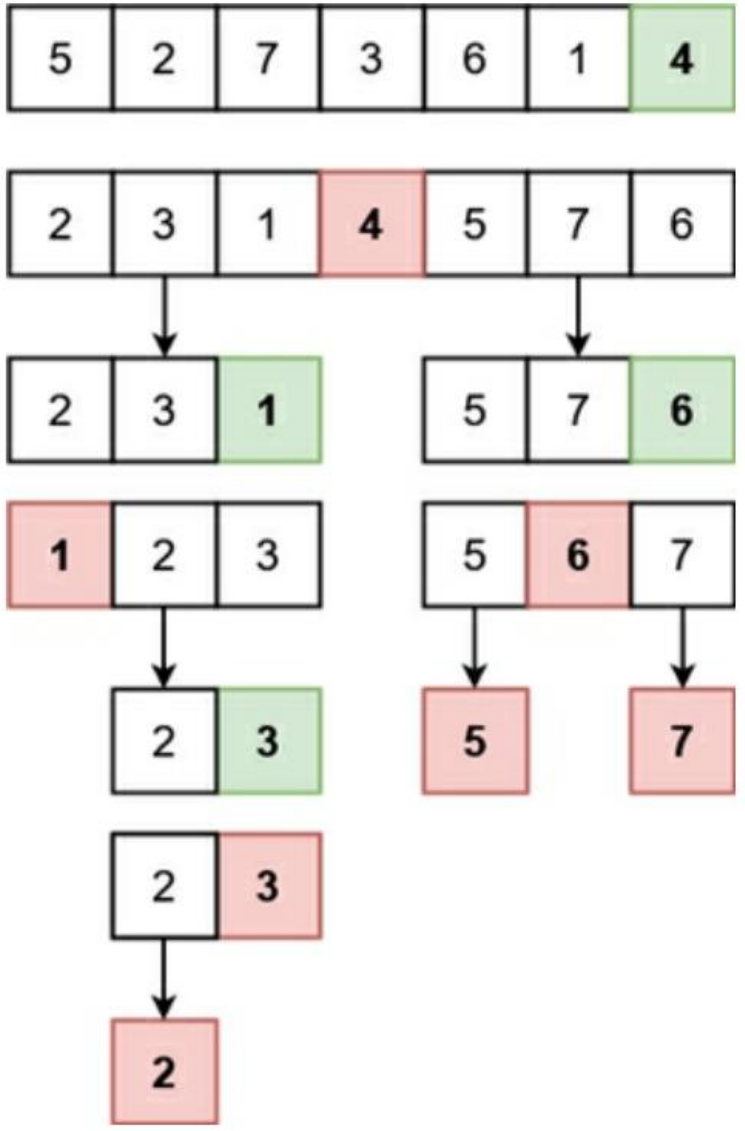
- Zbog koraka tri, već imamo bar linearno vreme. Ali sada se postavlja pitanje: Koliko puta treba da se spojimo? Za ovo moramo da pogledamo drugi korak. Prepolovili smo veličinu problema sa svakim ponavljanjem – logaritamsko vreme baze 2!

- Ukratko, moramo da izvedemo linearnu količinu operacija na svakom nivou i količina nivoa je logaritamska. Zbog toga je složenost objedinjavanja u izvršavanju **pseudo-linearna.**

$$levels = \theta(\log_2 n) \rightarrow \quad \theta(n \log n)$$

Algoritmi sortiranja pregled: quick sort

- Kada proveravamo na Google-u koji algoritam sortiranja je najbolji, većina stranica će vam reći da je to brzo sortiranje.
- Iako možda postoje neki bolji izbori (kao što su kombinacije algoritama sortiranja), ovo važi u većem delu.
- Brzo sortiranje je još jedan algoritam za deljenje i osvajanje koji je super efikasan i iako ima istu kompleksnost izvršavanja u prosečnom slučaju (nije najgori slučaj) kao sortiranje, obično se smatra efikasnijim jer su njegovi konst. faktori manji.



Algoritmi sortiranja pregled: quick sort

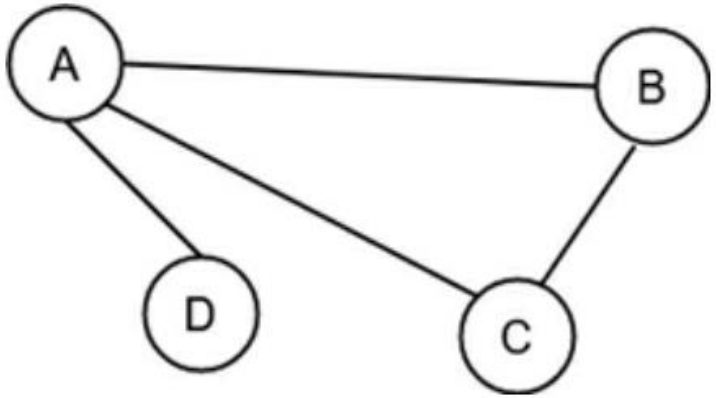
- Algoritam brzog sortiranja koristi takozvani izvedeni element (**pivot**) da bi sortirao listu.
- Možemo slobodno da definišemo koji element će biti izabran za ovo.
- Na datom primeru, odlučili smo da uvek odaberemo poslednji element liste. Takođe možemo da izaberemo prvu ili onu u sredini.
- Nakon odabira ovog pivot elementa, prolazimo kroz celu listu i stavljamo sve manje elemente na njenu levu stranu i sve veće elemente desno. Redosled ovih elemenata među sobom ne menja. To znači da je algoritam sortiranja stabilan.
- Kada to uradimo, našli smo pravo mesto za naš pivot element. Nakon toga delimo listu na dva dela i ponavljamo isti proces dok ne dobijemo ispravnu poziciju za sve elemente.

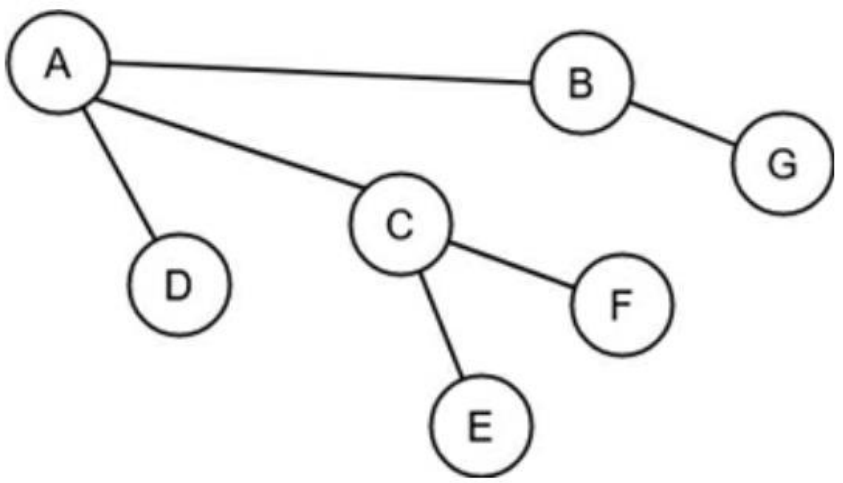
Algoritmi sortiranja pregled: quick sort

- Lako je videti zašto ovaj algoritam ima pseudo-linearnu kompleksnost prosečnog slučaja.
- U suštini menjamo položaj elemenata u linearno vreme i to radimo onoliko puta koliko postoji nivoa. Obično će ovo biti logaritmično.
- Međutim, najgora složenost ovog algoritma je kvadratna. Iako je malo verovatno, mogli bismo da naiđemo na listu koja je strukturirana na takav način, da posle svakog razdvajanja, pivot element završi na poslednjoj ili prvoj poziciji. Količina nivoa bi onda bila linearna, što znači da bismo završili sa kvadratnom kompleksnošću.
- Ipak, quicksort se smatra najefikasnijim algoritmom sortiranja (ne računajući kombinovane algoritme ili algoritme zasnovane na ne-poređenju), s obzirom da se kvadratna složenost gotovo nikada ne dešava u stvarnosti.
- U poređenju sa sortiranjem objedinjavanja, u većini slučajeva, brzo sortiranje je brže i potrebno mu je manje operacija.

Algoritmi sortiranja pregled: graf teorija i algoritmi

- Šta je zapravo grafik. U osnovi, to je samo struktura podataka koja se sastoji od temena i ivica.
- Graf može biti usmeren i neusmeren.
- U usmerenom grafikonu, ivice imaju uputstva, što znači da to što mogu da idu od A do B ne podrazumeva da možete da idete od B do A. Ako grafikon nema krugove ili cikluse, zovemo ga drvo.



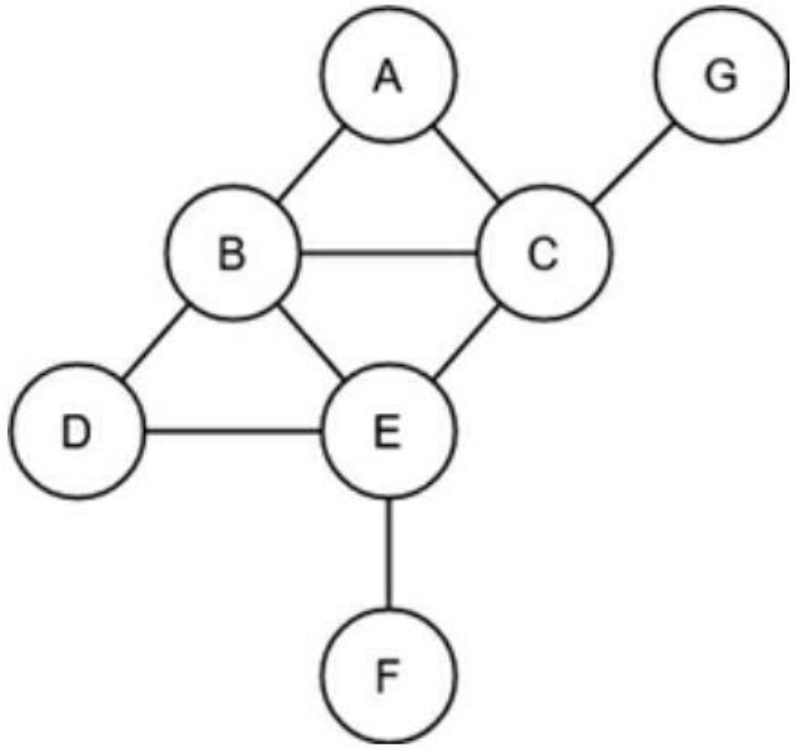


Drvo

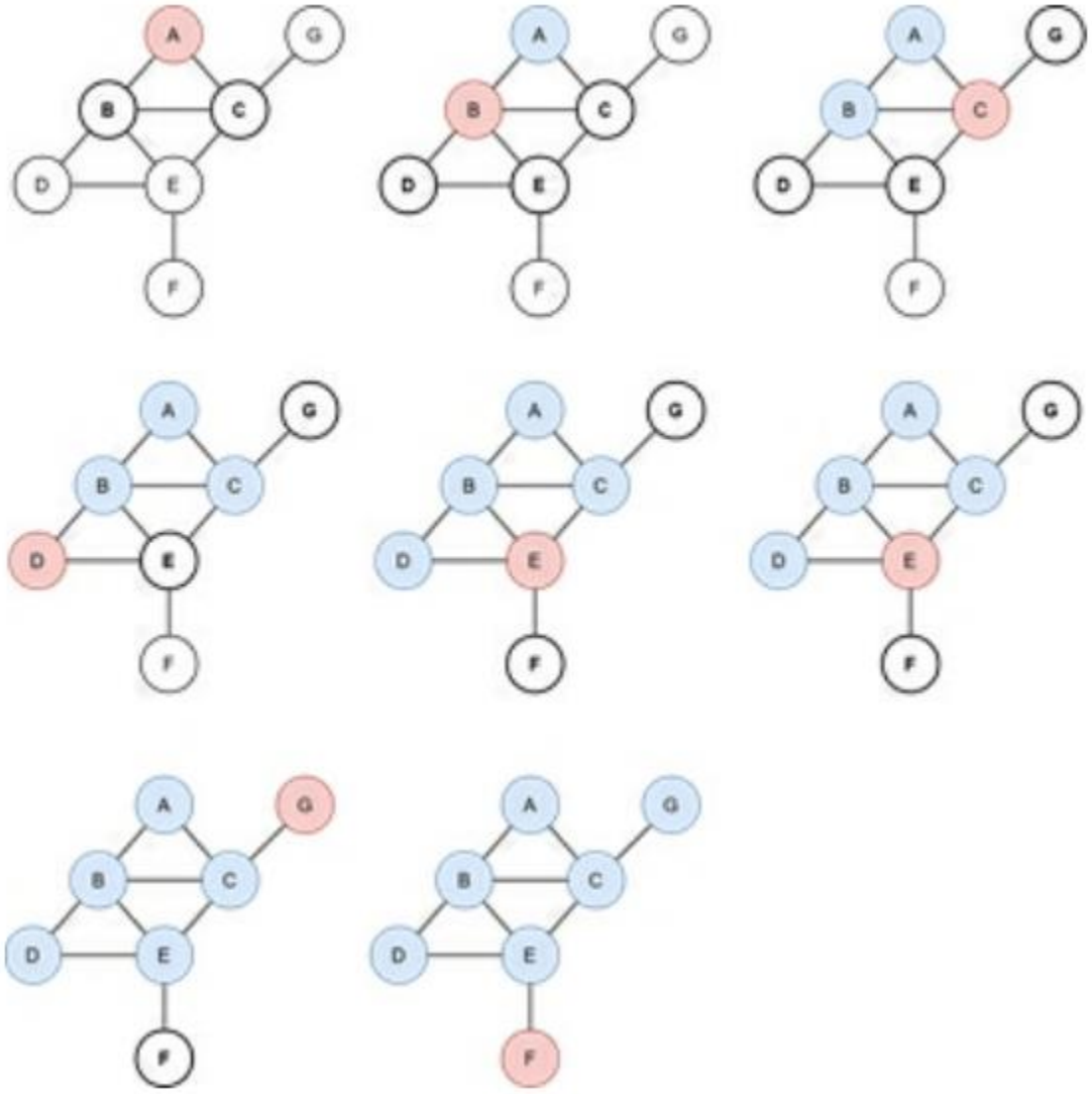
Algoritmi sortiranja pregled: pretraživanja prvo po širini

- Grubo govoreći u BFS-u, više volimo širinu nego dubinu. To znači da ćemo prvo pogledati sve komšije našeg početnog verteksa pre nego što nastavimo da idemo dublje u grafikon. Onda ćemo pogledati sve komšije i tako dalje.

- Primer:



Algoritmi sortiranja pregled: pretraživanja prvo po širini

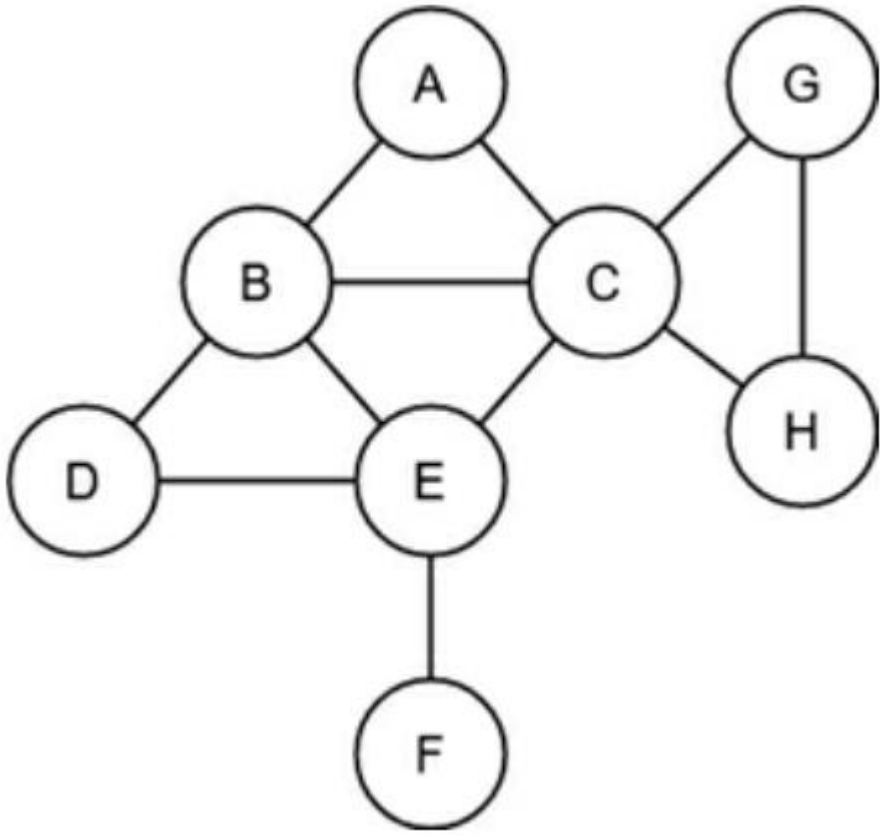


- Algoritamska složenost BFS-a zavisi od zbira količine temena i količine ivica.
- Pošto količina ivica može da varira od O(1) do $O(V^2)$, ovde ne možemo zaista navesti kompleksnost izvršavanja osim V + E.

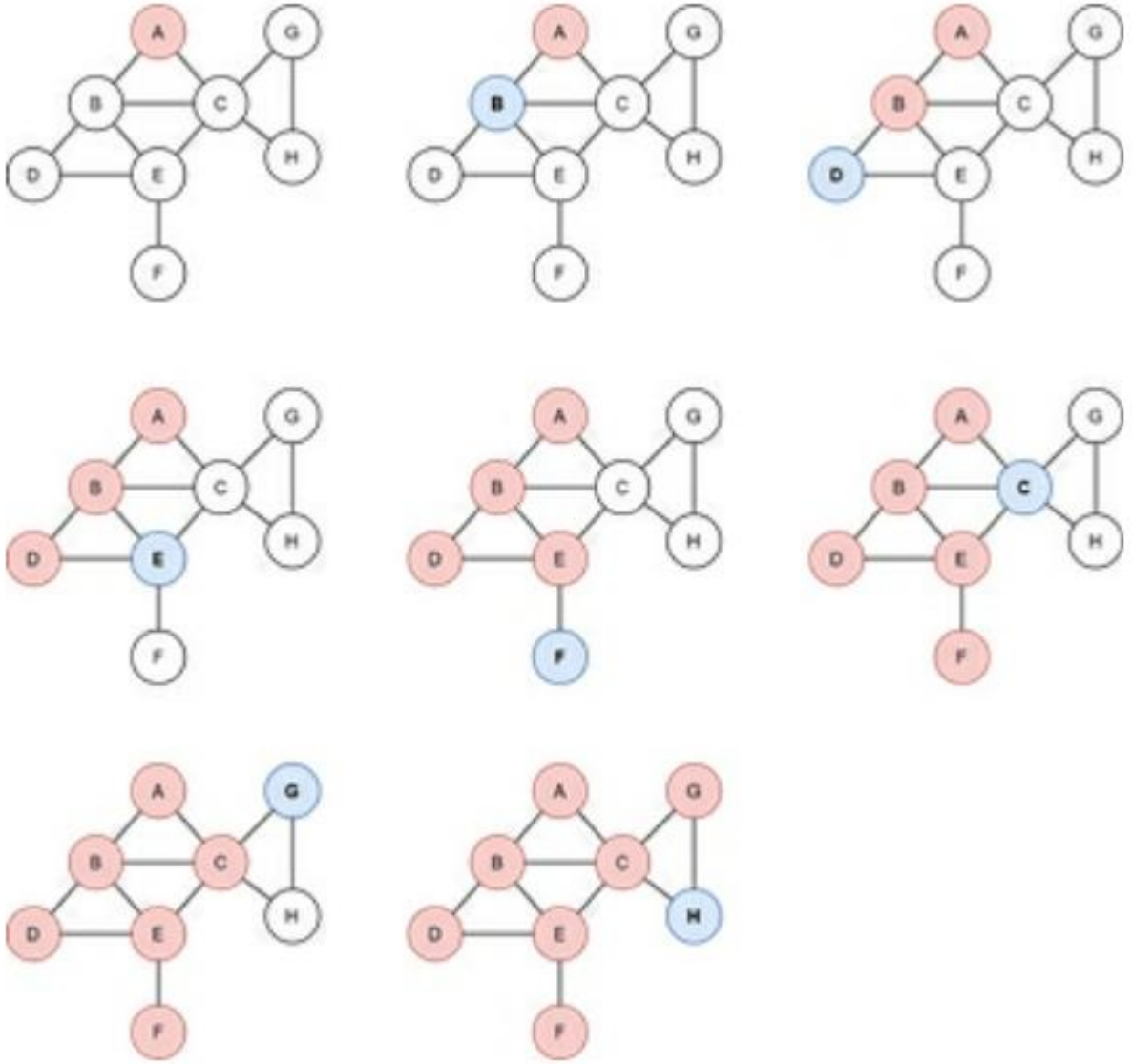
$$O(V + E)$$

Algoritmi sortiranja pregled: pretraživanja prvo po dubini

- Pretraga prvo po dubini (DFS), kao što ime već sugeriše, ideja je suprotna od BFS- a.
- Primer:



Algoritmi sortiranja pregled: pretraživanja prvo po dubini

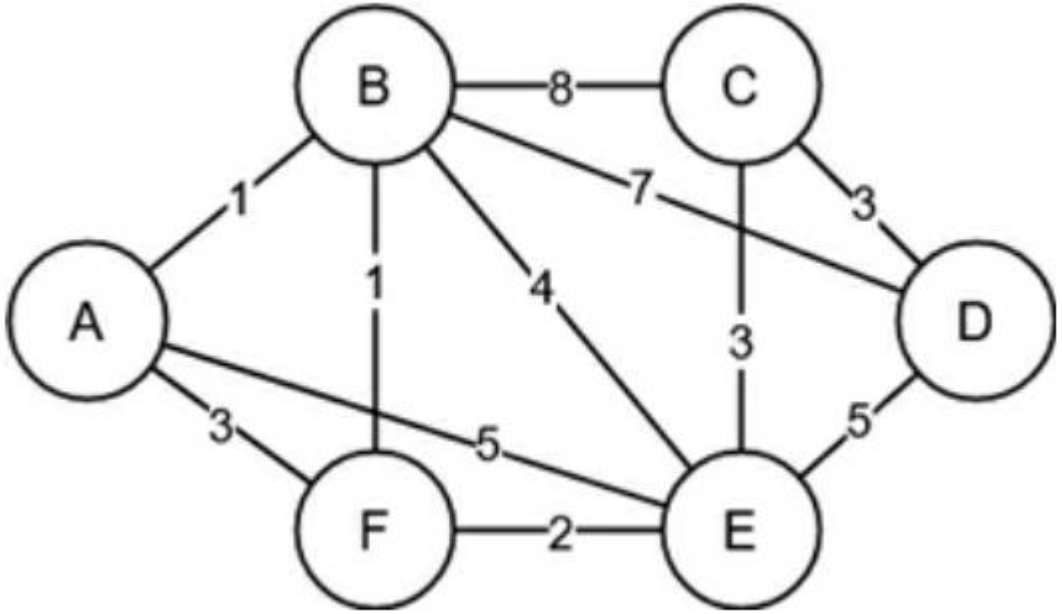


- Složenost BFS-a je ista kao i kompleksnost DFS-a.
- Međutim, postoje neke razlike. Na primer, BFS koristi strukturu podataka Reda čekanja u svom algoritmu, dok DFS koristi Stek strukturu podataka.
- U zavisnosti od toga koji problem pokušavate da rešite, potrebno je da razmislite koji bi prelazni pristup mogao da bude prikladniji.

$$O(V + E)$$

Algoritmi sortiranja pregled: Dijksta algoritam

- Pronalaženje najkraćeg puta od verteksa A do verteksa D. Sa BFS-om možemo da nađemo takav put ali samo ako su sve ivice podjednako dobre ili loše. Kada imamo težinu ivica stvari postanu teže.



Algoritmi sortiranja pregled: Dijksta algoritam

- Ono što u osnovi radimo je da izaberemo polaznu tačku i onda zapišemo koliko košta dostizanje svakog verteksa pomoću jednog koraka. Onda biramo nod sa najmanjim troškom i zapisujemo prethodni verteks, odakle smo mu pristupili.

Steps	A	B	C	D	E	F	Choice	Prev
1	-	1	∞	∞	5	3	B	A

- Jednim korakom možemo stići do B sa troškom jedan, E sa troškom od 5 i F sa troškom od tri. Cena svih ostalih čvorova je beskonačna, pošto uopšte ne možemo da ih dostignemo. Zato biramo B i beležimo da smo pristupili B iz A.

Steps	A	B	C	D	E	F	Choice	Prev
1	-	1	∞	∞	5	3	B	A
2			9	8	5	2	F	B

Algoritmi sortiranja pregled: Dijksta algoritam

- Pošto smo izabrali B, sada gledamo sva temena kojima možemo da pristupimo na korak od B. Svim čvorovima se može pristupiti, tako da zapišemo cenu za svaki od njih.

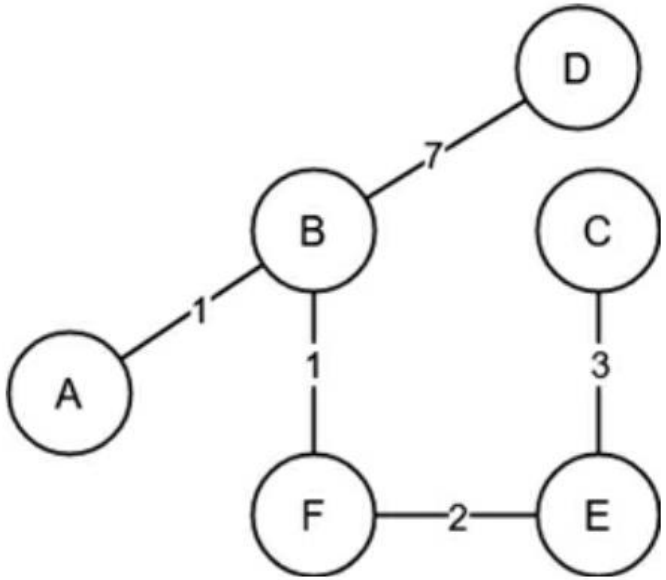
- Vertex C ima cenu od devet, jer košta jedan da ide od A do B i osam da bi se išlo od B do C. Kada završimo sa zapisima svih vrednosti, možemo videti da je trošak za F najmanji. Zato biramo F i zapišemo da smo stigli od B.

Steps	A	B	C	D	E	F	Choice	Prev
1	-	1	∞	∞	5	3	B	A
2			9	8	5	2	F	B
3			9	8	4		E	F

Algoritmi sortiranja pregled: Dijksta algoritam

Steps	A	B	C	D	E	F	Choice	Prev
1	-	1	∞	∞	5	3	B	A
2			9	8	5	2	F	B
3			9	8	4		E	F
4			7	8			C	E

Steps	A	B	C	D	E	F	Choice	Prev
1	-	1	∞	∞	5	3	B	A
2			9	8	5	2	F	B
3			9	8	4		E	F
4			7	8			C	E
5				8			D	B



$$O(V^2)$$

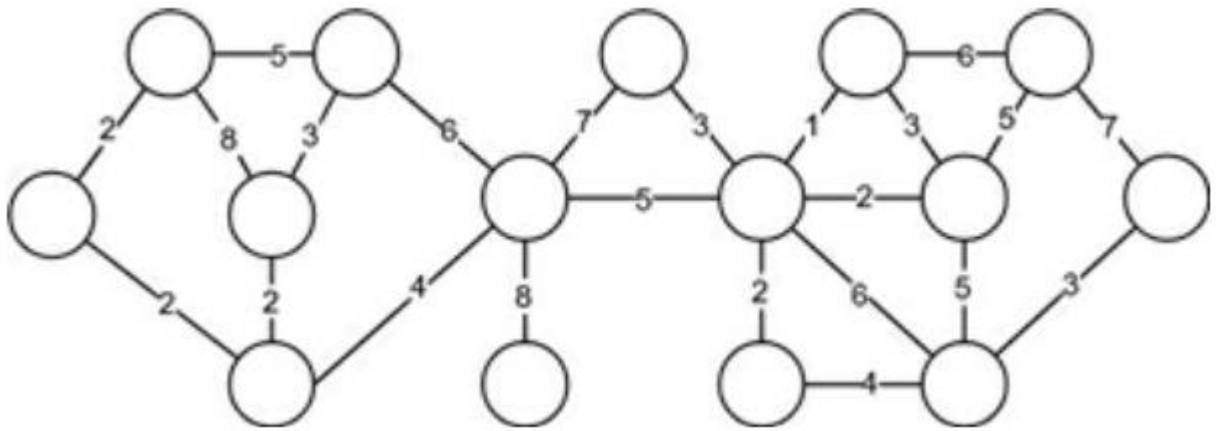
- Međutim, postoje implementacije ovog algoritma pomoću strukture podataka redosleda prioriteta, koja omogućava sledeću složenost izvršavanja:

$$O([V+E]*\log V)$$

Algoritmi sortiranja pregled: Kruskal

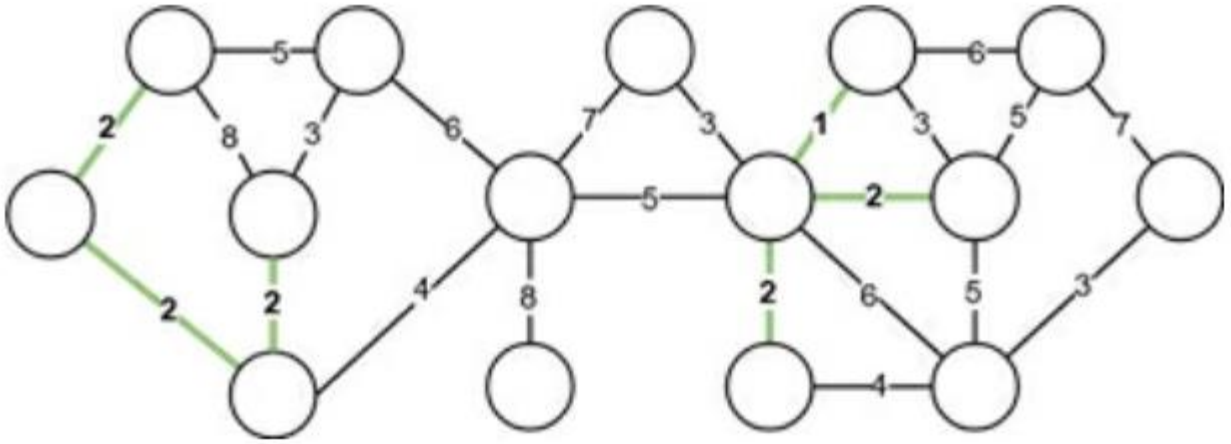
- Kurskal algoritam nam pomaže da pronađemo minimalnu i maksimalnu širinu drveta u grafu. Ovo znači da pronalazimo poddrvo koje će doći do svih temena uz najmanji (min) ili najveći (max) trošak.

- Primer:



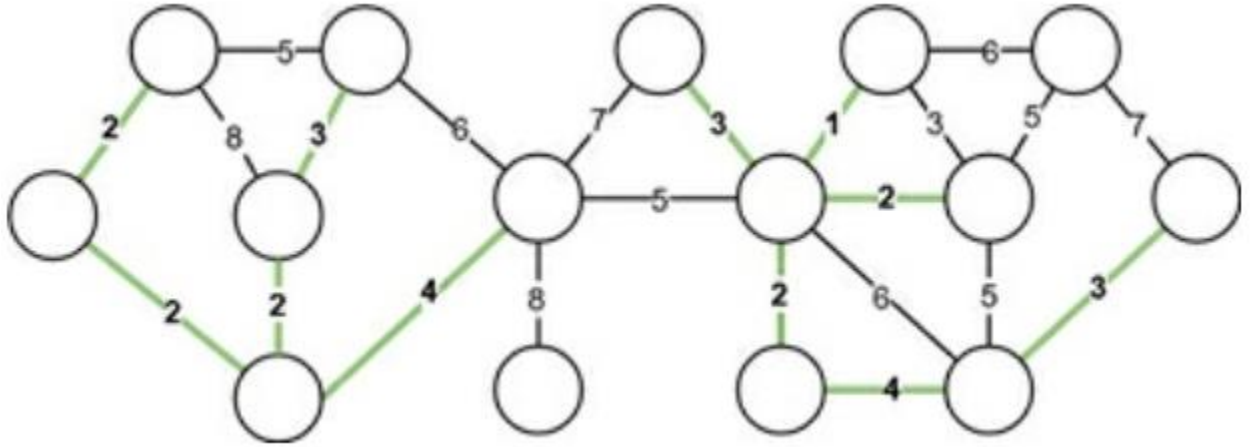
Algoritmi sortiranja pregled: Kruskal

- Prvo označavamo ivice sa najmanjom tižinom i bez kreiranja petlji.
- Ovo nastavljamo dalje sa 2, 3, 4 itd. dok ne označimo sve.

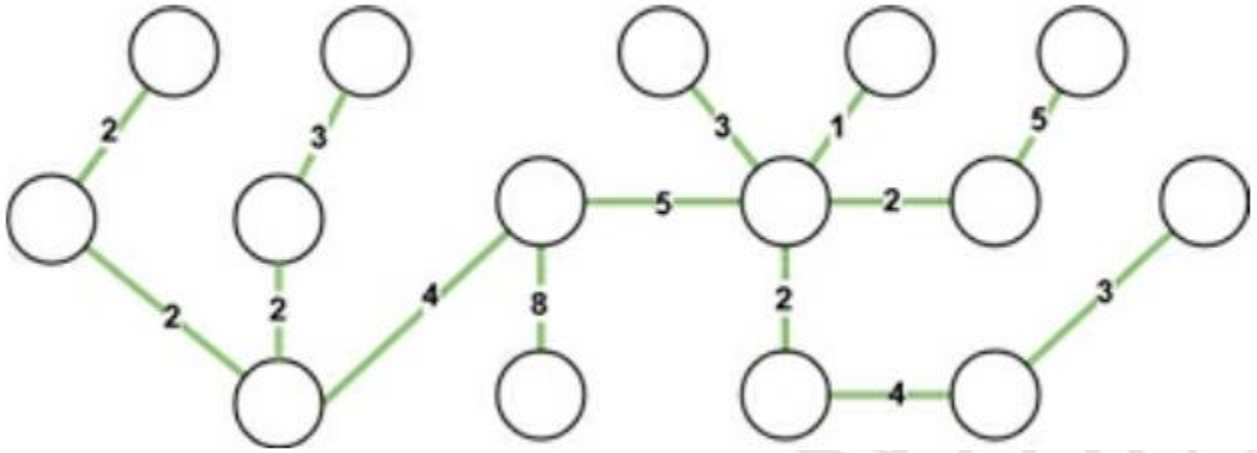


Algoritmi sortiranja pregled: Kruskal

- Prve četiri iteracije:



- Minimalna širina drveta:



$$O(E * \log(E))$$