Binarna gomila

Red – primene i ograničenja

Primene reda:

- **uređivanje procesa** koji traže procesorsko vreme
- **simulacija događaja**
- **izbor sledeće web stranice** za pretraživanje

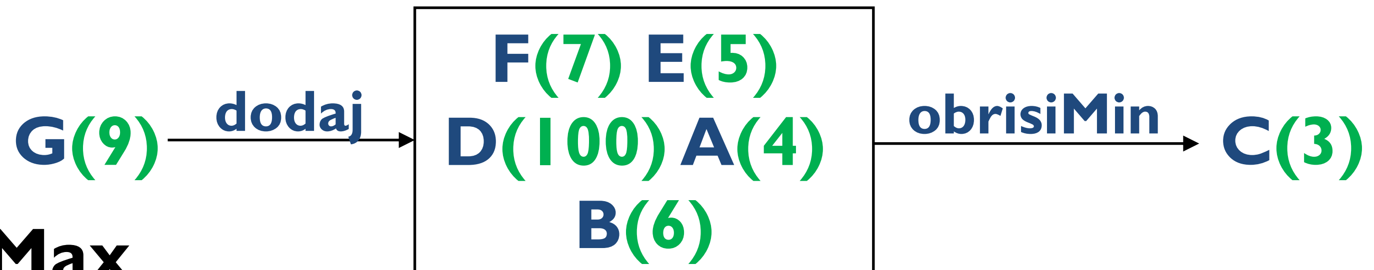
Problemi:

- kraći procesi bi trebalo da se **obrade prvi**
- najraniji događaji (u simuliranom vremenu) bi trebalo da se **smeste prvi**
- veb stranice koje najviše "obećavaju" trebalo bi da se **pretražuju prve**

Red sa prioritetom – ATP

Operacije:

- kreiraj
- uništi
- **dodaj**
- **obrišiMin/Max**



Svojstvo reda sa prioritetom – za dva elementa u redu *x* i *y*, ako *x* ima nižu (ili višu – zavisi kako koncipiramo red) vrednost prioriteta od *y*, onda će *x* biti obrisan iz reda sa prioritetom pre *y*

Red sa prioritetom – primene

Obrada poslova za štampu po redosledu dužine

Prosleđivanje paketa u mrežnim ruterima po redosledu hitnosti

Simulacija događaja na vremenskoj liniji

Selekcija simbola za kompresiju

Sortiranje brojeva (algoritam *heapsort*)

Bilo koji problem koji se može rešavati pohlepnom (engl. *greedy*) strategijom

Binarna gomila

Binarna gomila (engl. *heap*) je binarno stablo koje čuva vrednosti u internim čvorovima i koje ima sledeće osobine:

1. **(Svojstvo uređenja gomile** – engl. *heap-order***)** Ključ roditelja je manji ili jednak (veći ili jednak) od ključeva dece

2. **(Svojstvo oblika** – kompletno binarno stablo**)** Ako je *h* visina gomile, tada za $i = 0, 1, ..., h-1$ postoji 2^i na dubini *i*. Na dubini $h-1$, interni čvorovi su levo od eksternih čvorova

Max stablo je stablo u kojem **vrednost ključa** u svakom **čvoru nije manja** od vrednosti **ključeva njegove dece**. **Max heap** je **kompletno binarno stablo** koje je i **max stablo**.

Min stablo je stablo u kojem **vrednost ključa** u svakom čvoru **nije veća** od vrednosti **ključeva njegove dece**. **Min heap** je **kompletno binarno stablo** koje je i **min stablo.**

Binarna gomila

Glavne operacije:

- **kreirajGomilu** – kreiranje prazne gomile
- **dodaj** – dodavanje novog elementa u gomilu
- **obrišiMin/Max** – brisanje najmanjeg/najvećeg elementa iz gomile

Algoritam sortiranja pomoću gomile (engl. *heapsort*)

John Williams (1964) – prvi put uveden pojam gomile

- traženje min/max elementa $O(1)$
- brisanje min/max elementa $O(\log N)$
- traženje proizvoljnog elementa $O(N)$
- brisanje proizvoljnog elementa $O(N)$

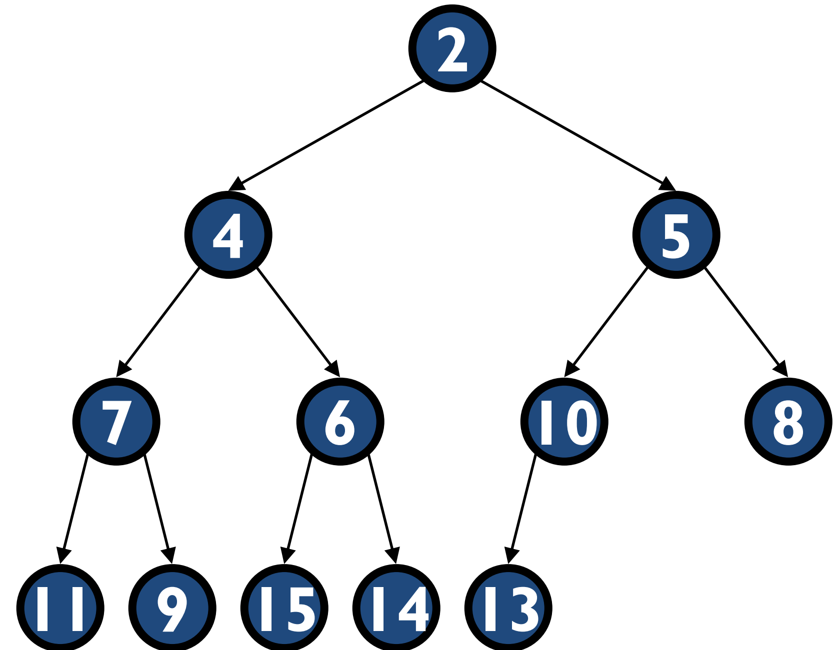
Binarna gomila

Svojstvo uređenja gomile:

- **Ključ roditelja je manji/veći od ključa dece**
- Rezultat: minimum/maksimum je uvek na vrhu (u korenu)

Svojstvo oblika:

- **Kompletno stablo – listovi** se nalaze što je **moguće više levo**
- Rezultat – dubina je uvek *O*(log *N*) – uvek se zna **sledeća slobodna lokacija**

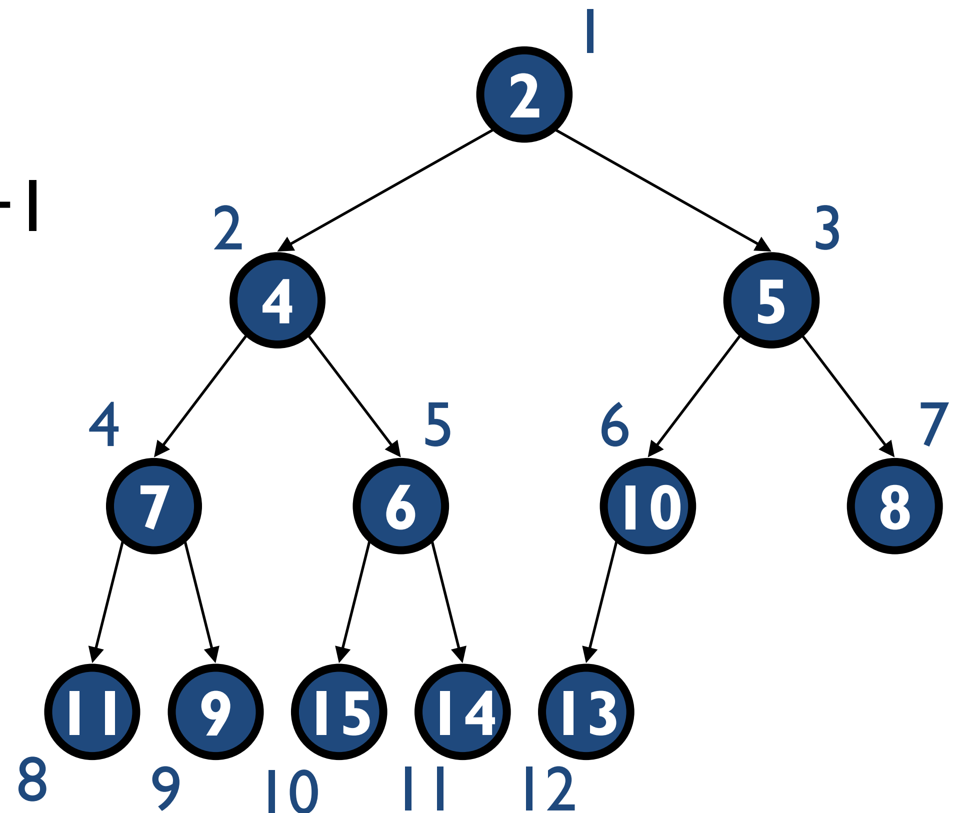


Kako se pronalazi minimum/maksimum?

Binarna gomila – memorijska reprezentacija

Izračunavanje indeksa:

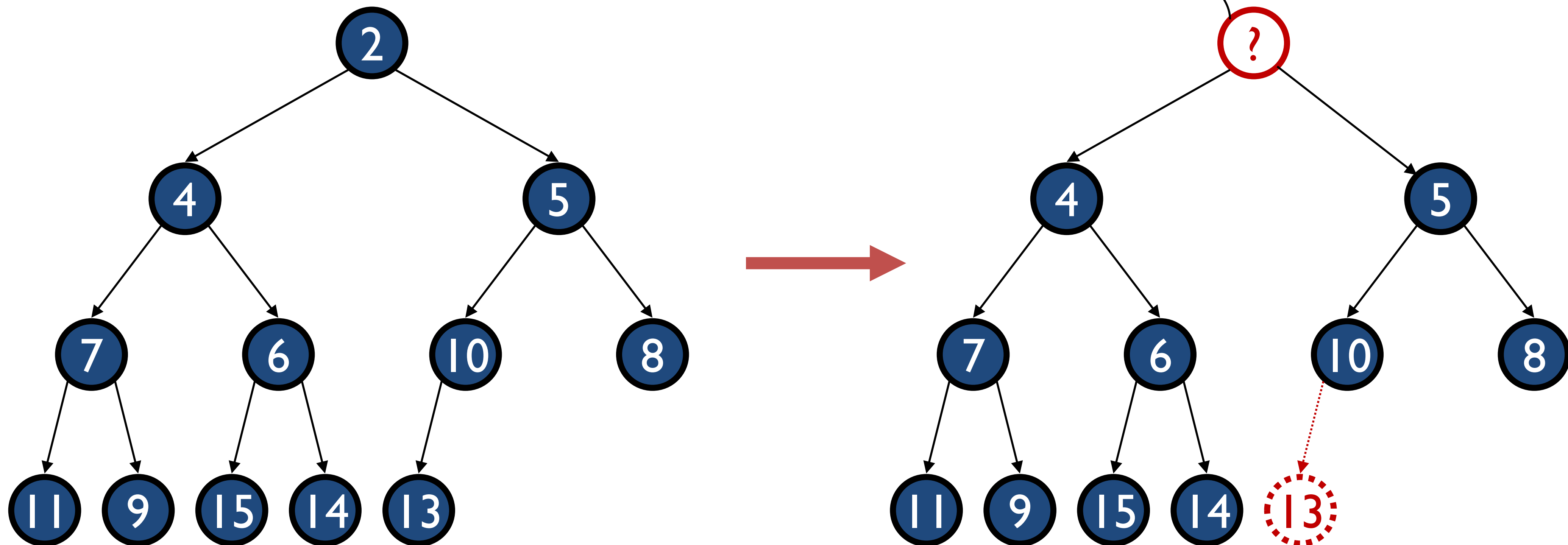
- deca: levo - 2**i*, desno 2**i*+1
- roditelj: *najmanje celo*(*i*/2)
- koren: 1
- sledeći slobodan: *dužina*+1



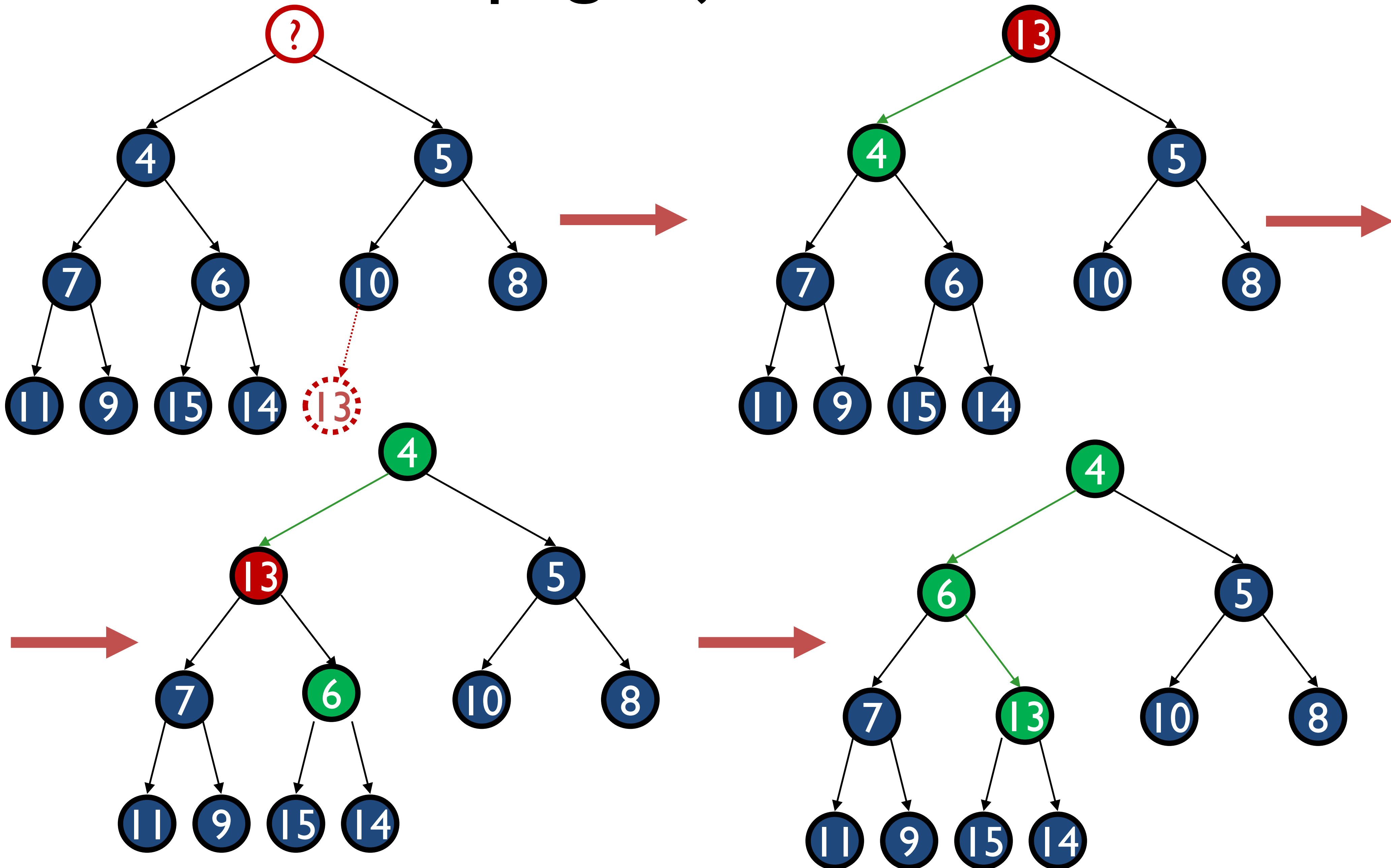
0	1	2	3	4	5	6	7	8	9	10	11	12	13
13	2	4	5	7	6	10	8	11	9	15	14	13	

Prolaskom kroz polje po redosledu indeksa dobijamo obilazak po nivoima

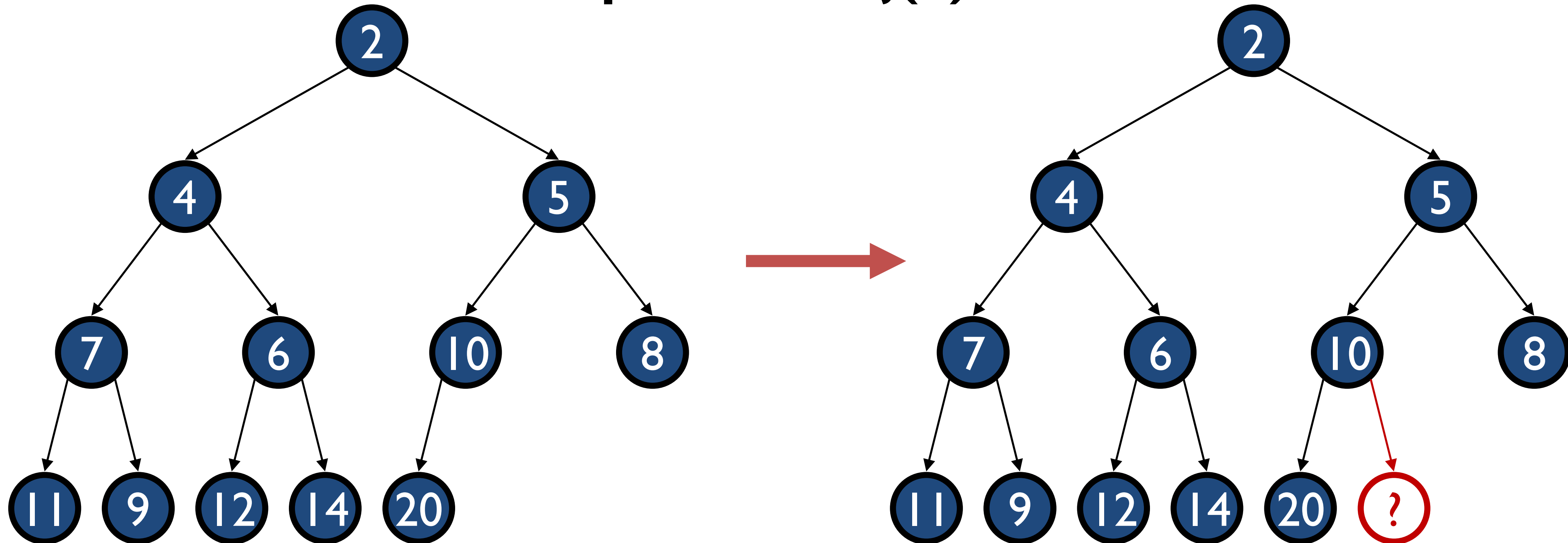
Binarna gomila – obrišiMin



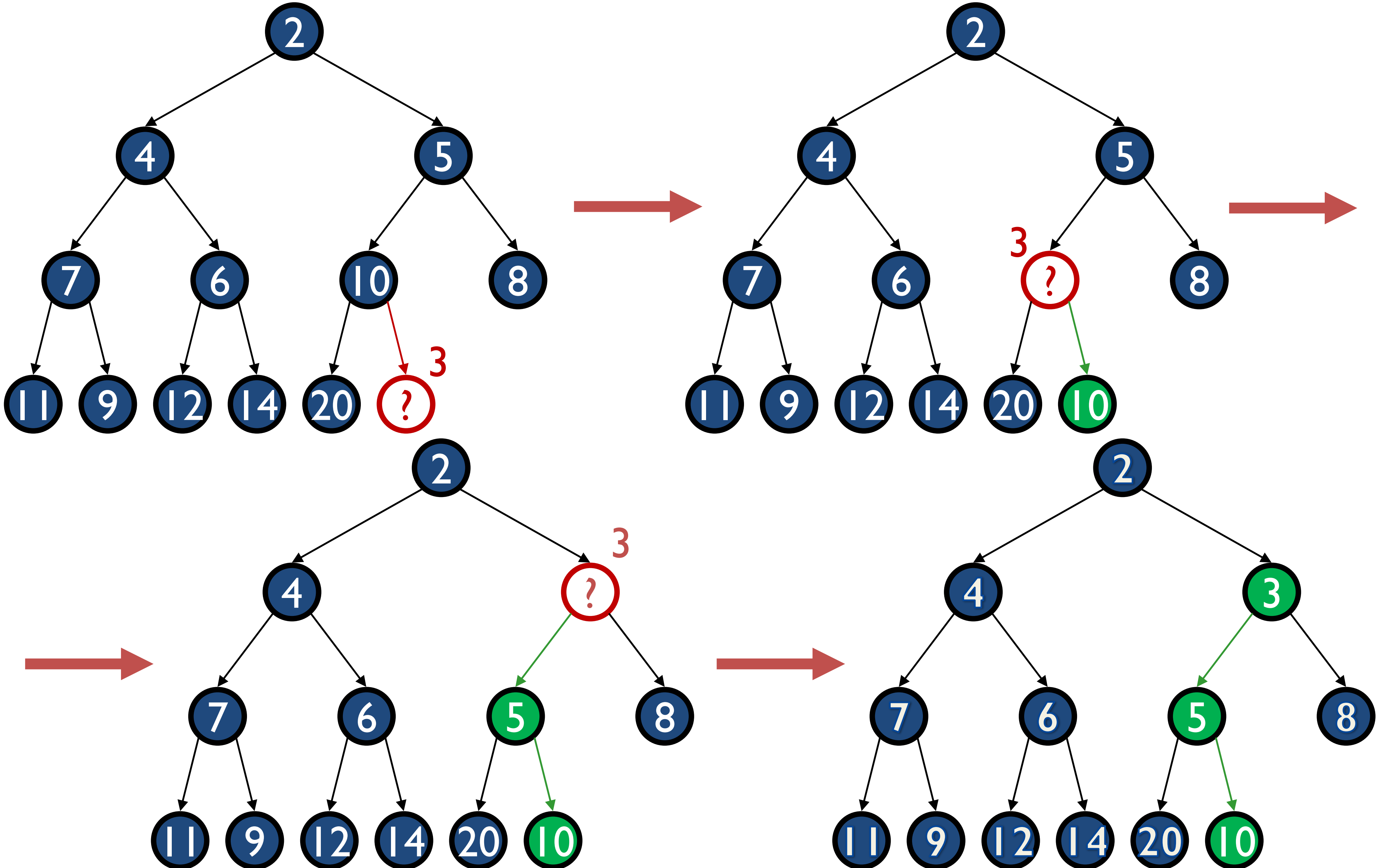
Propagacija nadole



Binarna gomila – dodaj



Propagacija nagore



Promenljivi prioriteti

U mnogim primenama, **prioritet** objekata smeštenih u red sa prioritetom može se **promeniti u toku vremena**

- Ako je posao (zadatak) čekao u redu za štampanje jako dugo vremena, treba povećati njegov prioritet
- Ako se za neki zadatak tj. proces utvrdi da je kritičan za dalji rad računara, treba povećati njegov prioritet

U ovim slučajevima, mora postojati posebna **operacija** kojom se u redu sa prioritetom **pronalazi pozicija objekta** čiji prioritet treba promeniti

- Kod gomile nema operacije traženja sa složenošću log*N* (kao kod binarnog stabla traženja)
- Heš mape (tabele) nude efikasnije rešenje ovog problema

Binarna gomila – ostale operacije

Ostale operacije:

smanjiKljuč (engl. *decreaseKey*) – ako je dat pokazivač na objekat u gomili, smanjiti njegovu vrednost prioriteta

povećajKljuč (engl. *increaseKey*) – ako je dat pokazivač na objekat u gomili, povećati njegovu vrednost prioriteta

obriši (engl. *remove*) – ako je dat pokazivač na objekat u gomili, ukloniti dati objekat

kreirajGomilu (engl. *buildHeap*) – ako je dat skup elemenata, kreirati gomilu

Algoritam heapsort

Složenost algoritma: u najgorem slučaju $O(N\log N)$, u najboljem slučaju $O(N)$, u prosečnom slučaju $O(N\log N)$

Algoritam u dva koraka:

1. Kreiranje max/min gomile
2. Uređeni niz se kreira uzastopnim uklanjanjem najvećeg (najmanjeg) elementa iz gomile (koren gomile) i ubacivanjem na odgovarajuće mesto u nizu. Gomila se ažurira nakon svakog brisanja kako bi se zadržalo svojstvo uređenja gomile

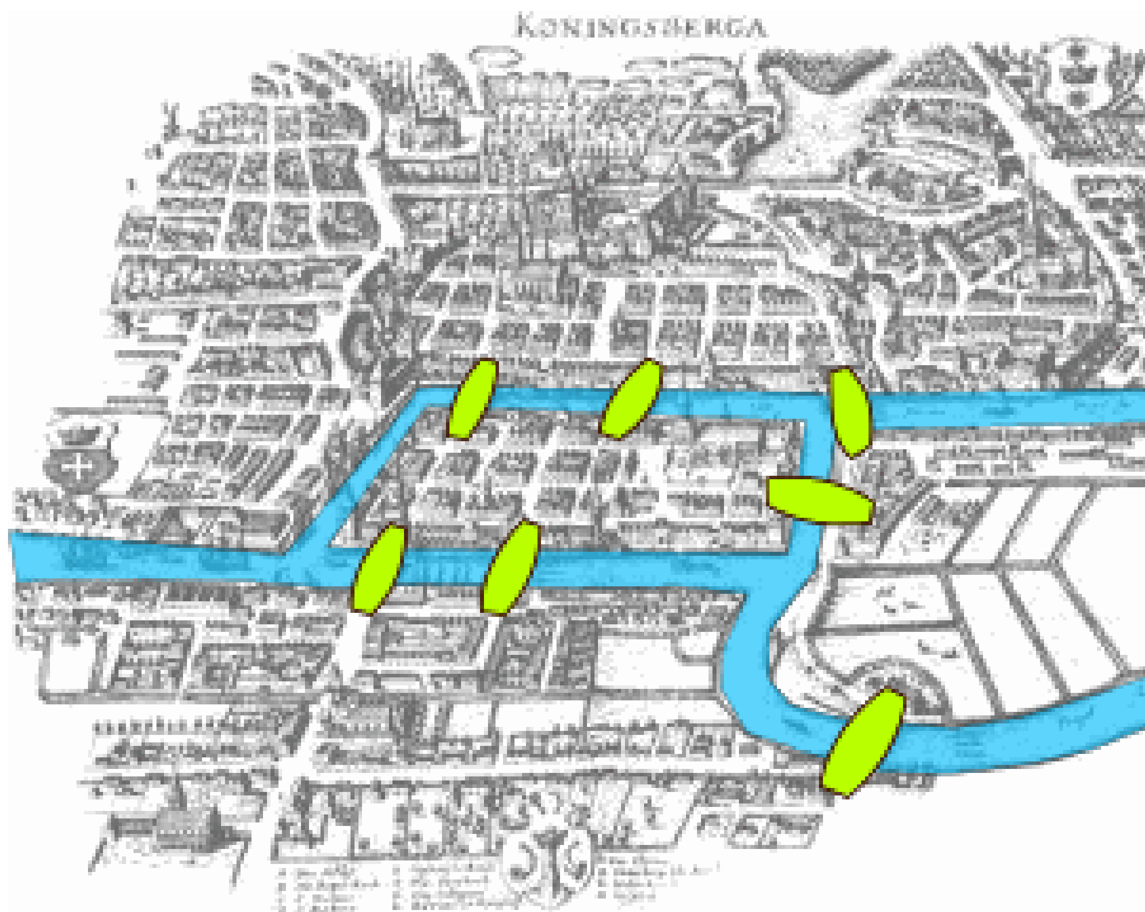
Algoritam heapsort

6 5 3 1 8 7 2 4

Izvor: https://en.wikipedia.org/wiki/Heapsort#/media/File:Heapsort-example.gif

Graf

Sedam mostova Keningsberga



Leonhard Euler
(1707-1783)

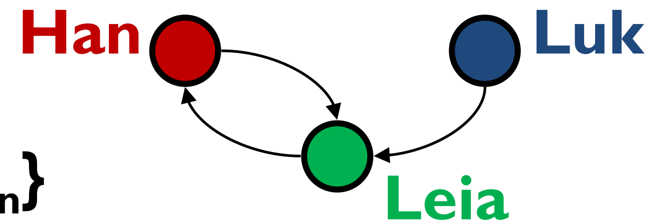
Izvor: https://en.wikipedia.org/wiki/Seven_Bridges_of_K%C3%B6nigsberg

Graf – ATP

Graf (engl. *graph*) je **apstraktni tip podataka** i **struktura podataka** pogodna za predstavljanje odnosa između entiteta

Graf **G** je **uređeni par G = (V, E)** gde je:

- **V** skup **čvorova** (engl. *vertices*): $\{v_1, \ldots, v_n\}$
- **E** skup **potega** (engl. *edges*): $\{e_1, \ldots, e_m\}$
  pri čemu svaki e_i povezuje dva čvora (v_{i1}, v_{i2})



V = {Han, Leia, Luk}
E = {(Luk, Leia),
(Han, Leia),
(Leia, Han)}

Operacije:

- iteracija kroz čvorove
- iteracija kroz potege
- iteracija kroz čvorove susedne određenom čvoru
- upit da li postoji poteg koji povezuje dva čvora

Graf – primene

Modelovanje problema koji su “grafovski” po svojoj prirodi

- udaljenost između gradova
- planiranje dostavnih ruta ili ruta avionskih linija
- relacije između ljudi – socijalne mreže (6 stepeni razdvajanja)
- rutiranje kroz računarske mreže

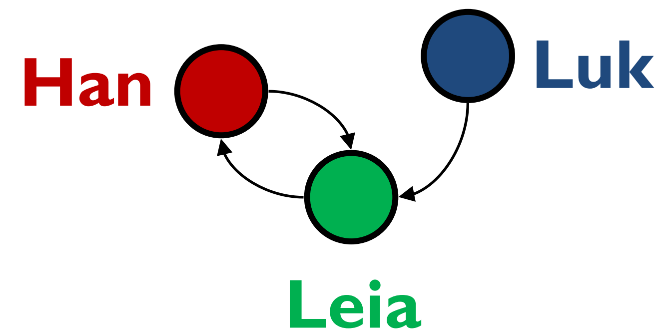
Programski prevodioci

- graf poziva (engl. *call graph*) – koje funkcije pozivaju koje
- graf zavisnosti (engl. *dependence graphs*) – koje promenljive se definišu i koriste u kojim naredbama

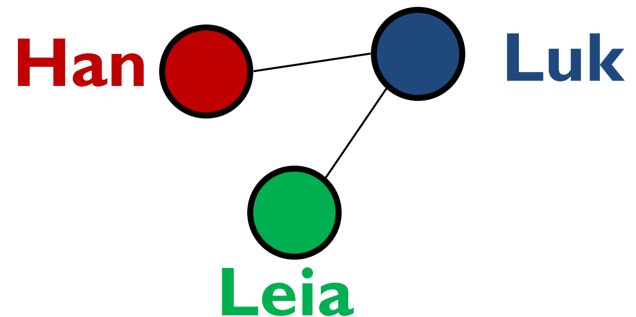
Skoro svi problemi koji uključuju **diskretne entitete**

Usmereni i neusmereni graf

Kod **usmerenih** (engl. *directed*) grafova, potezi imaju određen smer (digraf)



Kod **neusmerenih** (engl. *undirected*) grafova, potezi nisu usmereni (svi su dvosmerni)



Čvorovi **i** i **j** su **susedni** (engl. *adjacent*) ako je **(i, j)** ∈ **E**

Graf – totalno uređenje

Svojstva totalnog uređenja:

totalnost - ili *x* dolazi pre *y* ili *y* dolazi pre *x*

tranzitivnost – ako *x* dolazi pre *y* i *y* dolazi pre *z*, tada i *x* dolazi pre *z*

asimetričnost - ako *x* dolazi pre *y*, onda *y* ne dolazi pre *x*

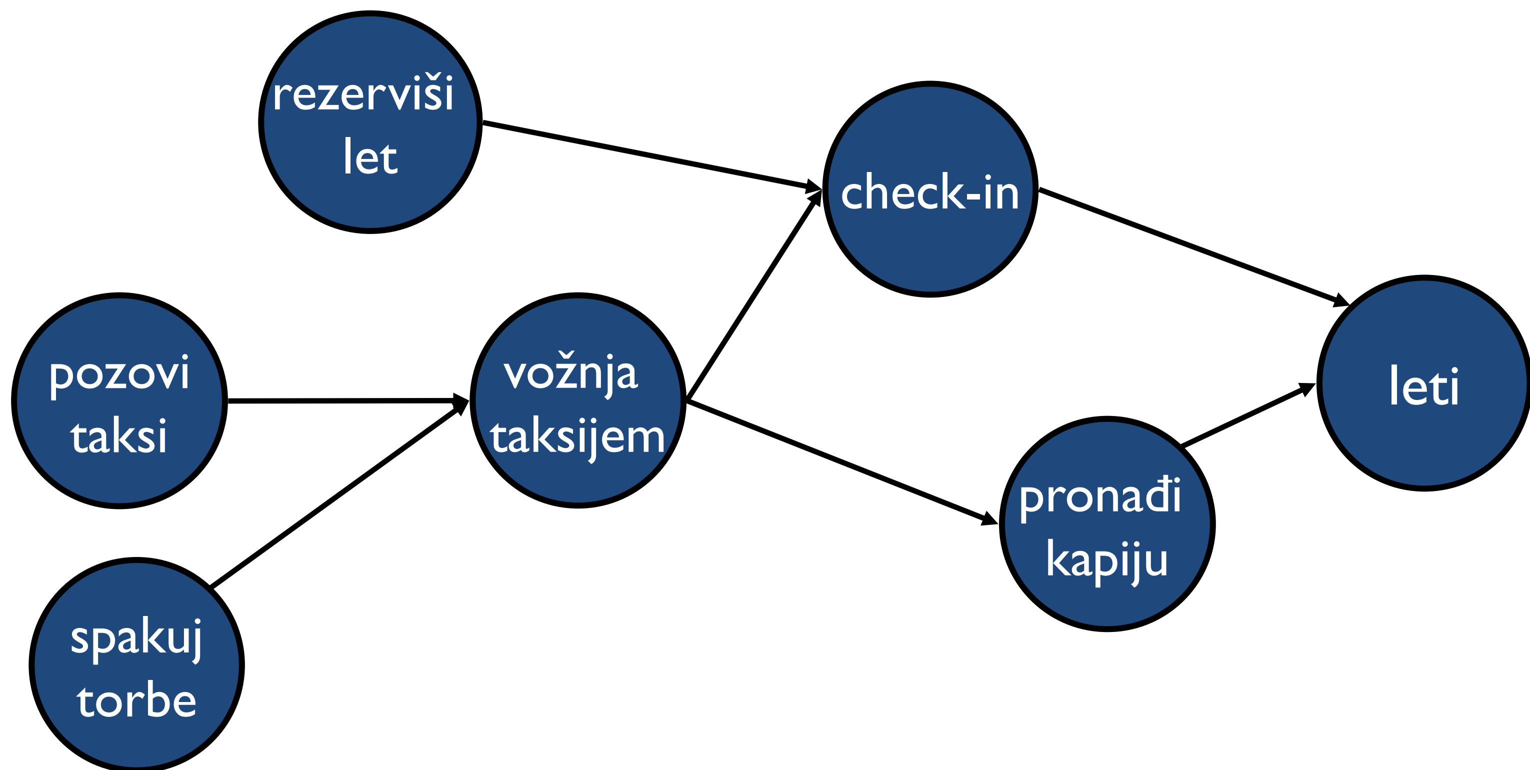
znači da A mora doći pre B

Parcijalno uređenje – planiranje puta

Kod **parcijalnog uređenja ne zahteva** se **totalnost**

Topološko uređenje

Ako je dat graf **G = (V, E)**, prikazati sve čvorove iz **V** tako da se nijedan čvor ne prikaže pre bilo kog drugog čvora koji ima poteg ka njemu



Topološko uređenje – 1. pristup

1. Obeležiti ulazni stepen (engl. *in-degree*) svakog čvora (broj ulaznih potega)
2. Dok ima preostalih čvorova:
   - Izabrati čvor sa ulaznim stepnom nula i prikazati ga
   - Smanjiti ulazni stepen čvorova koji su mu susedni
   - Ukloniti čvor iz liste čvorova

Topološko uređenje – 2. pristup

1. Označiti ulazni stepen svakog čvora
2. Smestiti sve čvorove sa ulaznim stepenom nula u red
3. Dok u redu ima čvorova:
    - Uzeti čvor *v* sa ulaznim stepenom nula i prikazati ga
    - Smanjiti ulazni stepen svih čvorova susednih čvoru *v*
    - Smestiti nove čvorove sa ulaznim stepenom nula u red
    - Ukloniti *v* iz reda

Graf – reprezentacije

Elementi grafa – **lista čvorova** i **lista potega**

Implementacija grafa:

Sekvencijalna – matrica čvorova u kojoj elementi predstavljaju potege

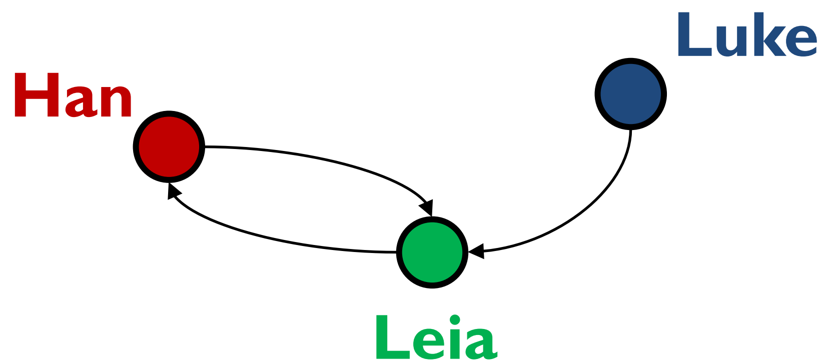
- **Matrica susedstva** (engl. *adjacency matrix*) – element a_{ij} je 1 kada postoji poteg od čvora *i* do čvora *j*, a nula u suprotnom

Spregnuta – lista čvorova u kojoj su za svaki element zabeleženi susedni čvorovi

- **Lista susedstva** (engl. *adjacency list*)

Graf – matrica susedstva

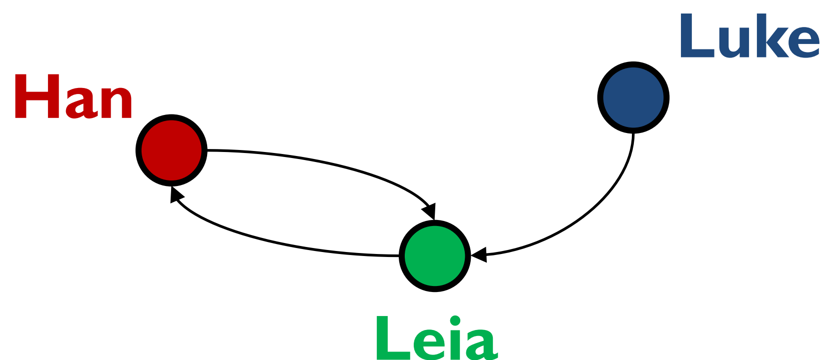
Matrica dimenzija **|V| x |V|** u kojoj je svaki element **(i, j)** **istinit (1)** ako i samo ako postoji poteg od **i** do **j**

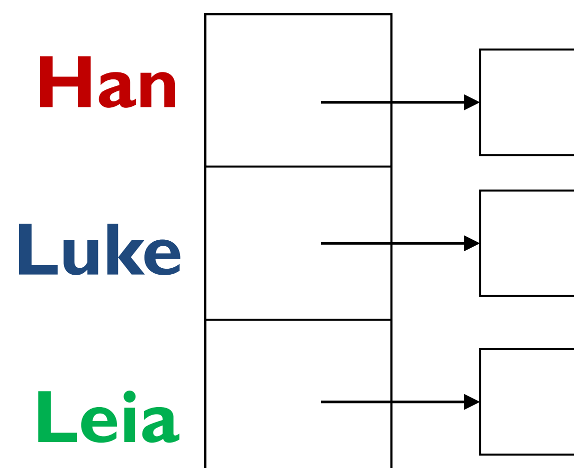


	Han	Luke	Leia
Han	0	0	1
Luke	0	0	1
Leia	1	0	0

Graf – lista susedstva

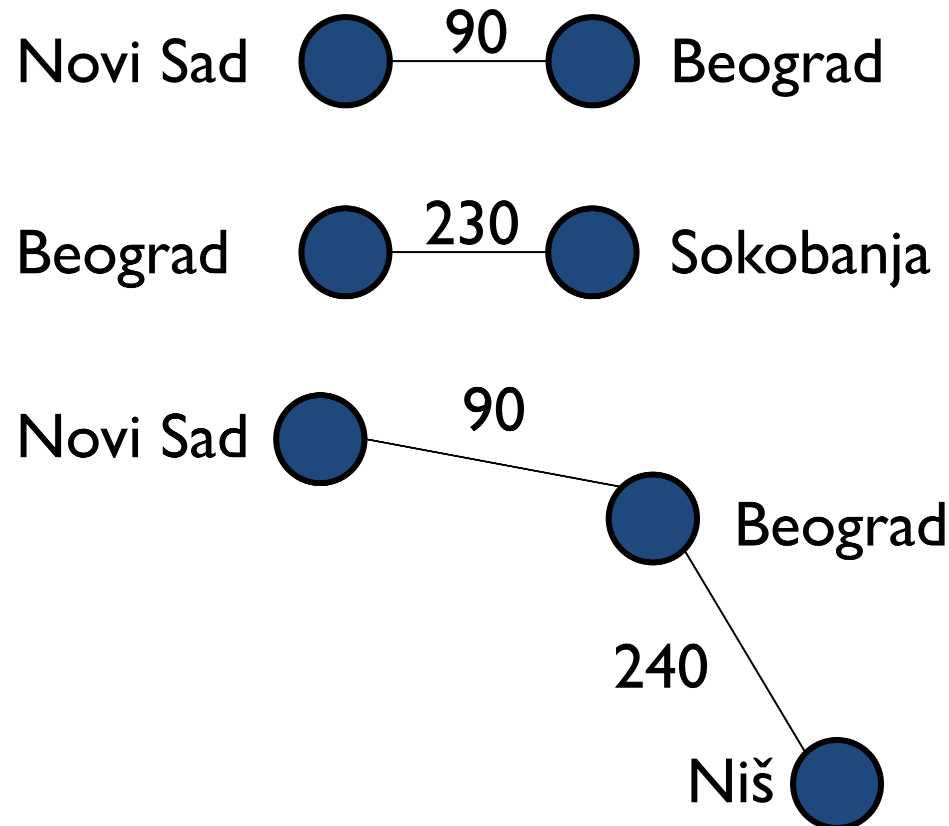
Lista sa **|V|** elemenata u kojoj svaki element čuva **spregnutu listu** svih susednih čvorova (ili potega)





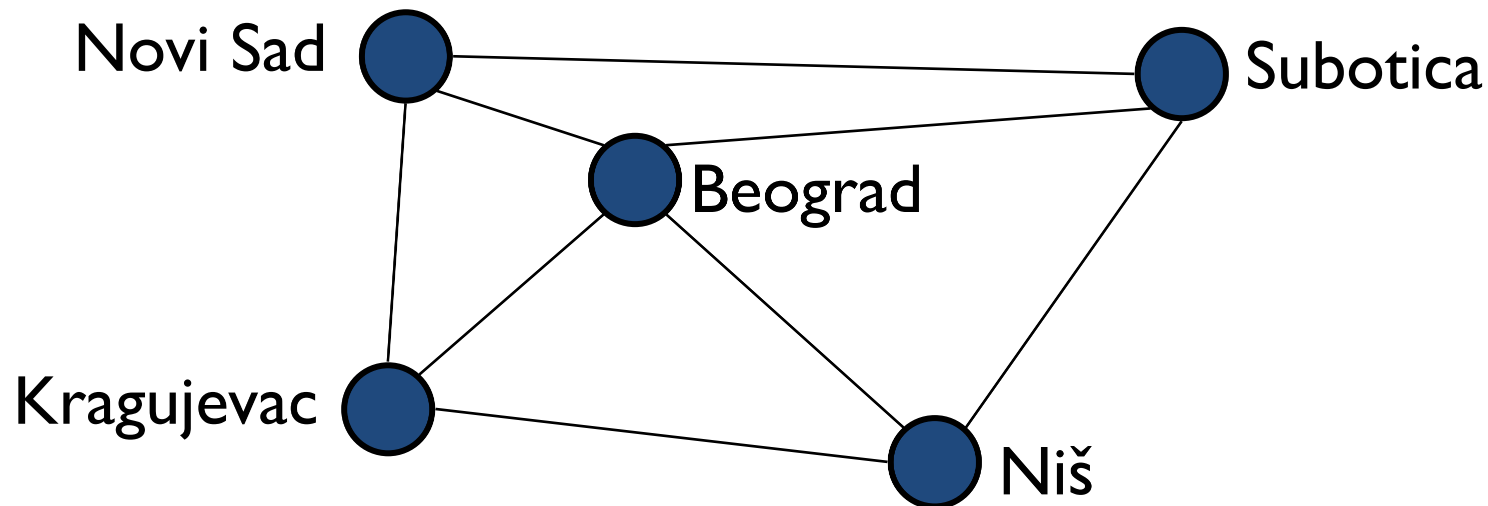
Težinski graf

Kod težinskog (engl. *weighted*) grafa, svakom potegu je pridružena odgovarajuća težina (engl. *weight*) ili cena (engl. *cost*)



Put u grafu

Put (putanja) je lista čvorova $\{\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v}_N\}$ takva da je $(\mathbf{v_i}, \mathbf{v_{i+1}}) \in \mathbf{E}$ za svako $\mathbf{0} \leq \boldsymbol{i} < \boldsymbol{N}$

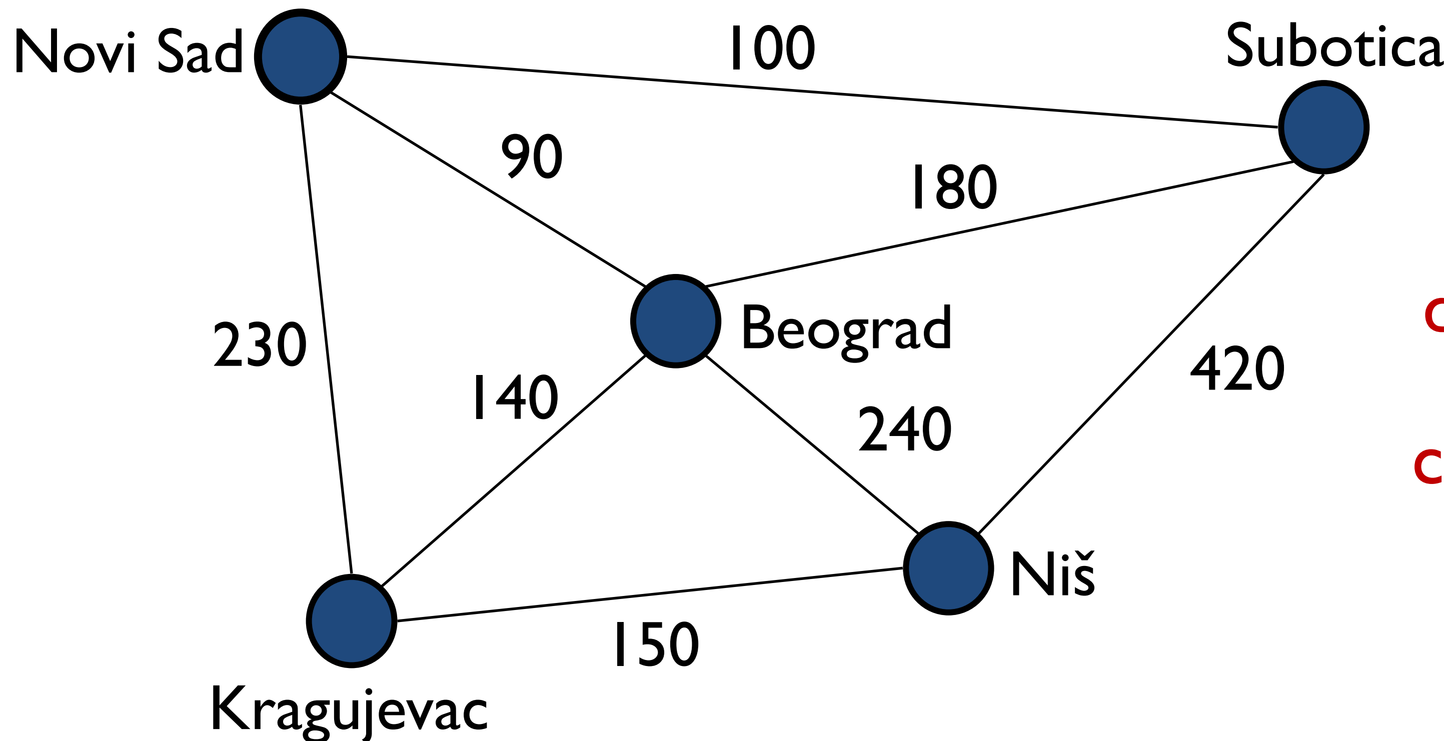


p = {NS, SU, BG, NI, KG, NS}

Put u grafu – dužina i cena

Dužina puta (engl. *path length*) – broj potega na putu

Cena puta (engl. *path cost*) – ukupna suma težina svih potega



dužina(p) = 5

cena(p) = 900

p = {NS, SU, BG, NI, KG, NS}

Prosti putevi i ciklusi

Prost put (engl. *simple path*) je put u kome nema ponovljenih čvorova (osim prvog koji može biti i poslednji):

- p = {Novi Sad, Beograd, Niš, Kragujevac}
- p = {Novi Sad, Subotica, Beograd, Kragujevac, Niš}

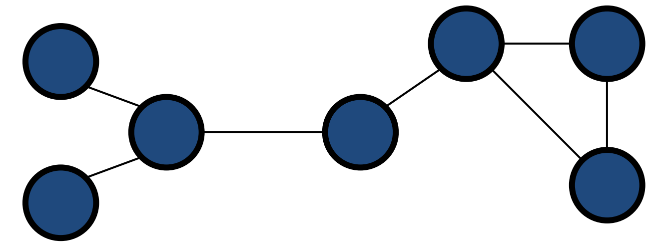
Ciklus (engl. *cycle*) je putanja koja počinje i završava se u istom čvoru:

- p = {Novi Sad, Subotica, Beograd, Niš, Kragujevac, Novi Sad}

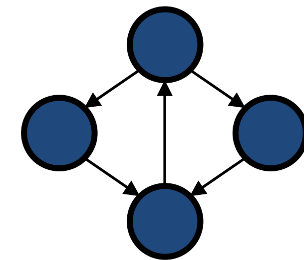
Prost ciklus (engl. *simple cycle*) je ciklus u kome se ne ponavlja nijedan čvor, osim prvog koji je i poslednji (kod neusmerenih grafova, nijedan poteg se ne sme ponoviti)

Povezanost

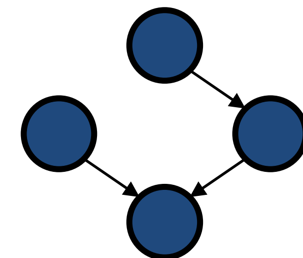
Neusmereni grafovi su **povezani** (engl. *connected*) ako u njima postoji putanja između bilo koja dva čvora



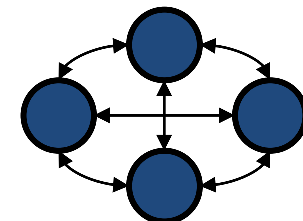
Usmereni grafovi su **jako povezani** (engl. *strongly connected*) ako postoji putanja između bilo koja dva čvora



Usmereni grafovi su **slabo povezani** (engl. *weakly connected*) ako postoji putanja između bilo koja dva čvora, zanemarujući usmerenje



Kompletan graf sadrži poteg između svakog para čvorova



Gustina grafa (engl. *graph density*)

Graf može biti **redak** (engl. *sparse*) ili **gust** (engl. *dense*)

Redak graf ima $O(|V|)$ potega

Gust graf ima $\Theta(|V|^2)$ potega

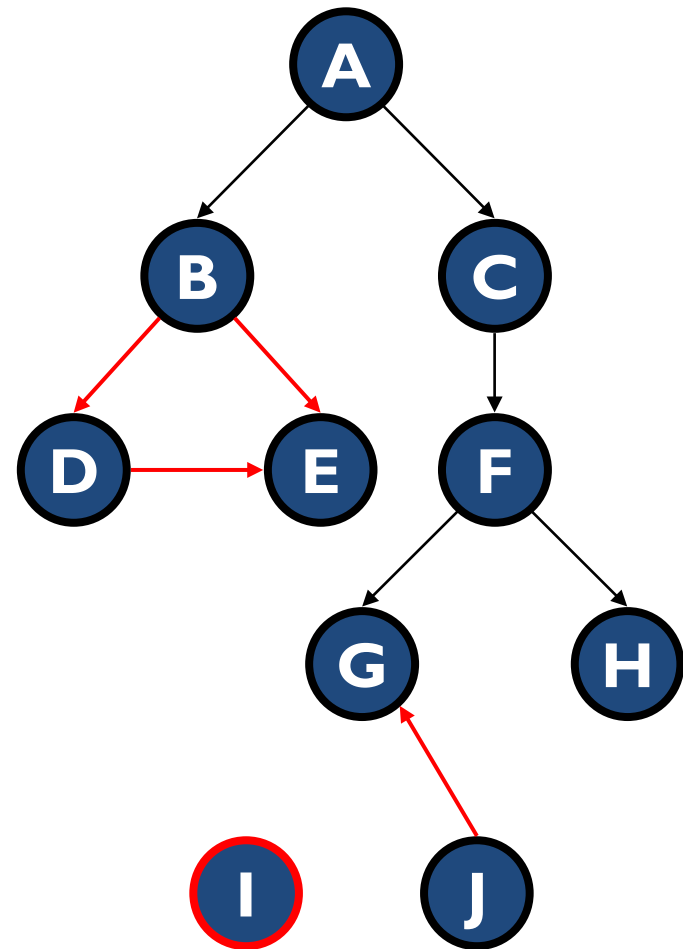
Bilo koji graf čiji se broj potega nalazi negde između smatra se **pretežno retkim** (engl. *sparsish*) ili **pretežno gustim** (engl. *densy*), zavisno od konteksta u kome se razmatra

Po pitanju **memorijskih zahteva**, matrica susedstva je povoljnija kod gustih grafova, a lista susedstva kod retkih (ili treba koristiti odgovarajuće kompaktne reprezentacije retkih matrica)

Stablo kao graf

Svako stablo je graf sa određenim ograničenjima:

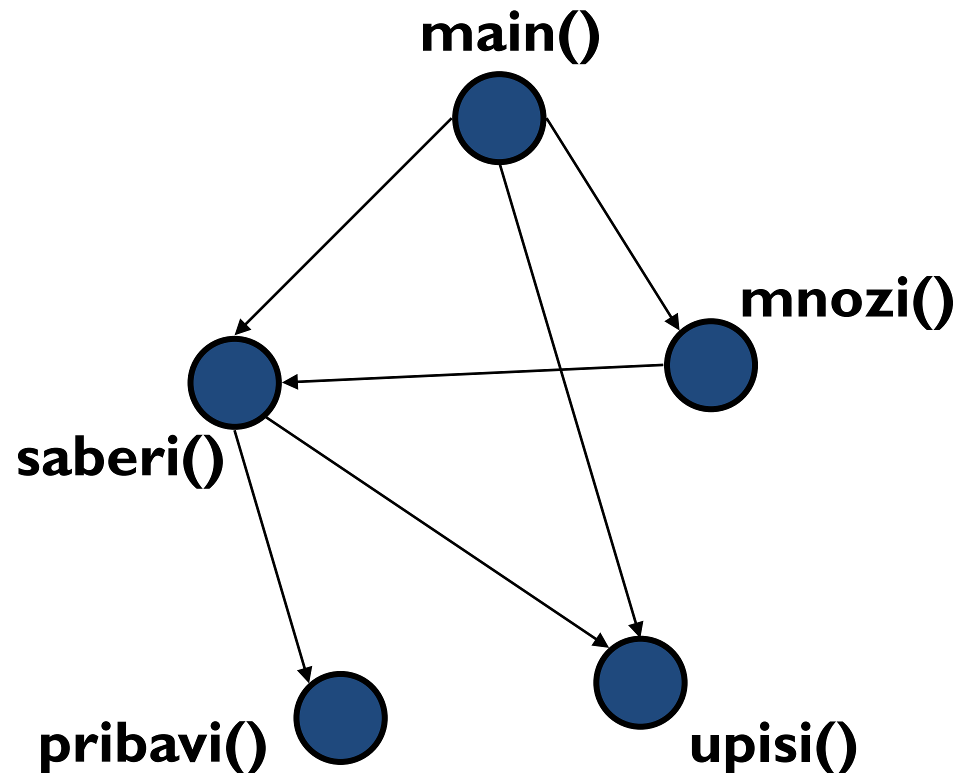
- stablo je **usmereno**
- **nema ciklusa** (usmerenih ili neusmerenih)
- postoji **usmereni put od korena do svakog čvora**



Usmereni aciklični graf

Usmereni aciklični graf (engl. *directed acyclic graph – DAG*) je usmereni graf koji ne sadrže cikluse

stabla ⊂ usmereni aciklični grafovi ⊂ grafovi



Grafovski algoritmi

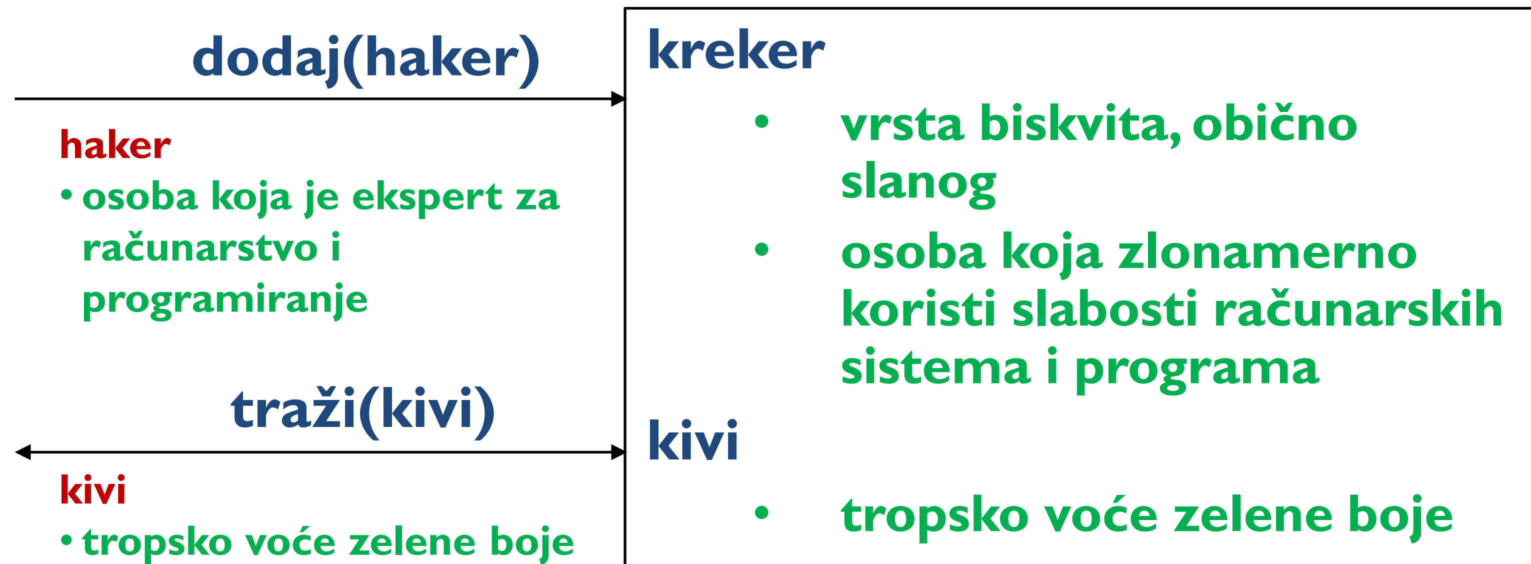
- Topološko uređivanje
- Obilazak po širini i dubini (engl. *breadth-first search – BFS* i *depth-first search – DFS*) – po širini red kao pomoćna struktura, a po dubini stek
- Detekcija ciklusa
- Traženje najkraćeg puta u grafu (algoritmi – Dijkstra, Bellman-Ford, Floyd-Warshall...)
- Provera povezanosti
- "Teški" problemi (problem trgovačkog putnika (engl. *Travelling Salesman Problem*), bojenje grafa, Hamiltonova putanja...)
- Minimalna sprežna stabla (algoritmi – Prim, Kruskal, Boruvka...)
- Backtracking algoritmi (praćenje unazad) – lavirinti, 8 kraljica...

Heš mapa

Rečnik i ATP za traženje

Operacije:

- kreiraj
- uništi
- dodaj
- traži
- obriši



Rečnik (engl. *dictionary*) ili **mapa** (engl. *map*) ili **asocijativni niz** (engl. *associative array*) čuva **vrednosti** (engl. *values*) pridružene **ključevima** (engl. *keys*) specificiranim od strane korisnika – **uređeni par (k, v)**

- **ključevi** mogu biti bilo kog homogenog uporedivog tipa
- **vrednosti** mogu biti bilo kog homogenog tipa
- implementacija – **stablo traženja** (engl. *search tree*) ili **heš mapa/tabela** (engl. *hash map/table*), polje za podatke je struktura sa dva dela – ključ i vrednost

ATP za traženje (engl. *search ADT*) – **ključevi = vrednosti**

Implementacija rečnika

	neuređeno polje	**uređeno polje**	**spregnuta lista**	**binarno stablo traženja**
dodaj (sa/bez duplikata)	traži + O(1)	O(*N*)	traži + O(1)	O(*d*)
traži	O(*N*)	O(log *N*)	O(*N*)	O(*d*)
briši	traži + O(1)	O(*N*)	traži + O(1)	O(*d*)

BST ima **odlične performanse** za **"plitka"** (engl. *shallow*) **stabla**, tj. kada je **dubina *d* mala** (log *N*), **u suprotnom BST** su **jednako loši** kao **spregnute liste!**

Heš mapa

Heš mapa ili **heš tabela** (engl. *hash map – table*) je struktura podataka koja **implementira ATP rečnik** (mapa, asocijativni niz) tj. mapira **ključeve** na **vrednosti**

Kod heš mapa se koristi **heš funkcija** (engl. *hash function*) za izračunavanje **indeksa** u nizu

Lokacije u ovakom nizu nazivaju **kofe** (engl. *buckets*) ili **prorezi** (engl. *slots*) i u njima se nalaze odgovarajuće vrednosti

Heš mapa

Heš mapa je struktura podataka koja omogućava **direktan pristup podacima izračunavanjem** njihovog **položaja** u mapi **na osnovu ključa**

Heš mapa je **generalizacija običnog polja** (koje se indeksira celobrojnim indeksima) u vidu **polja indeksiranog proizvoljnim ključevima** (koji mogu biti brojevi, nizovi znakova i sl.)

Cilj: smeštanje/dodavanje i brzo pretraživanje velike količine podataka

Osnovna ideja: da se **ulazni skup** veličine *N* (koji može biti **jako veliki** – veći od raspoložive memorije) svede na **manji broj stavki**. Takvu **kompresiju** radi **heš funkcija**

Heš mapa – osobine

Obezbeđuje brz pristup i dodavanje podataka, najčešće reda složenosti $O(1)$

Relativno lako se programira (u odnosu na stabla i grafove)

Implementira se putem polja, s tim da pozicija elementa u polju zavisi od aritmetičke transformacije ključa

Pošto se implementacija zasniva na poljima, postoji problem ekspanzije (širenja) polja

Ne postoji podesan način za obilazak elemenata heš mape u proizvoljnom redosledu

Cilj heš mape – ključ kao indeksi

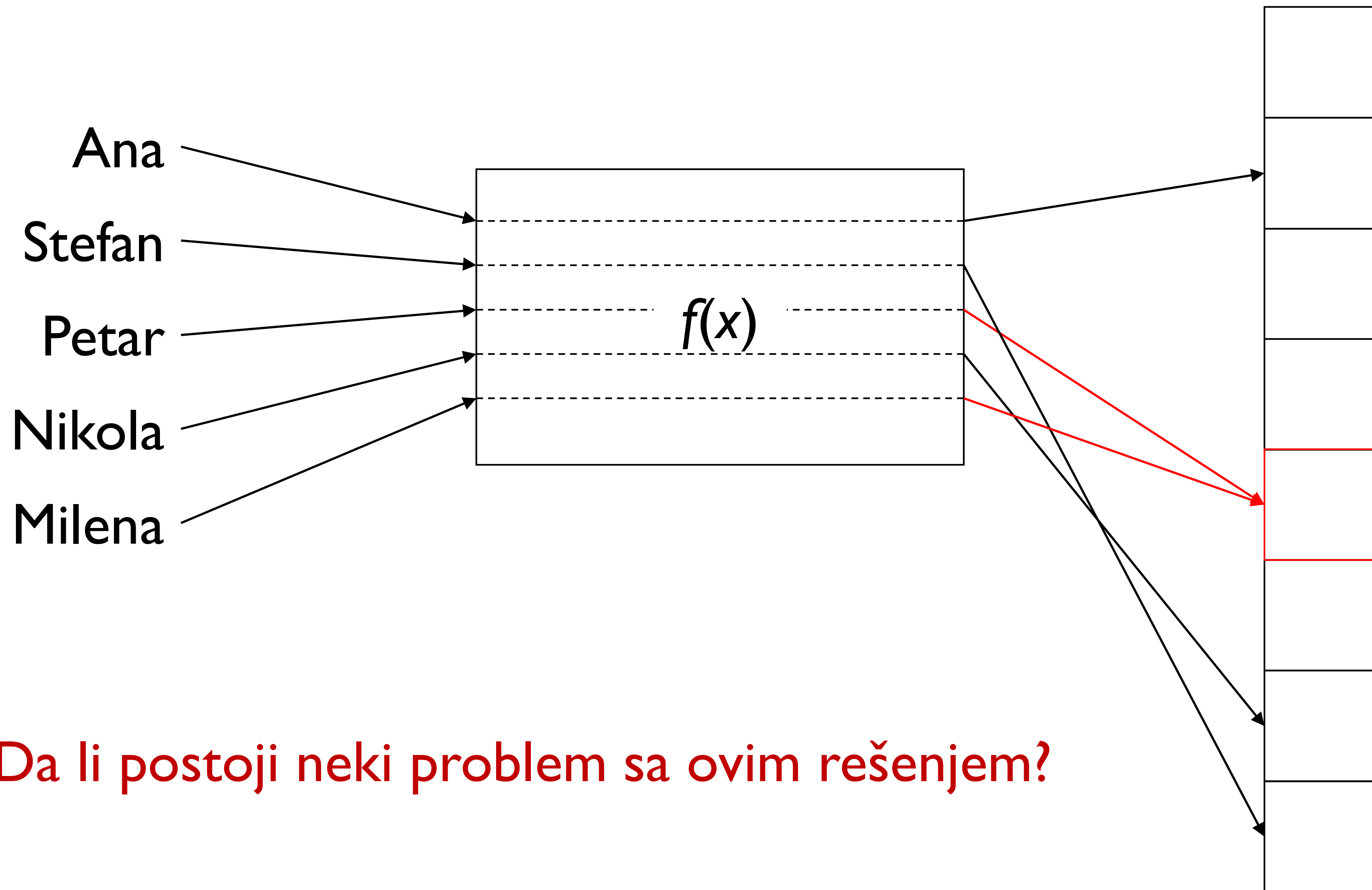
Elementima polja možemo pristupati putem operatora indeksiranja – npr. a[5] nas dovodi do "Ana"

Cilj je pristupati elementima polja putem ključa – npr. **a["Ana"]**

	...
2	**Nikola** **NBA košarkaš**
3	**Stefan** **violinista**
5	**Ana** **C++ guru**
	...

	...
Nikola	**Nikola** **NBA košarkaš**
Stefan	**Stefan** **violinista**
Ana	**Ana** **C++ guru**
	...

Heš mapa – pristup



Da li postoji neki problem sa ovim rešenjem?

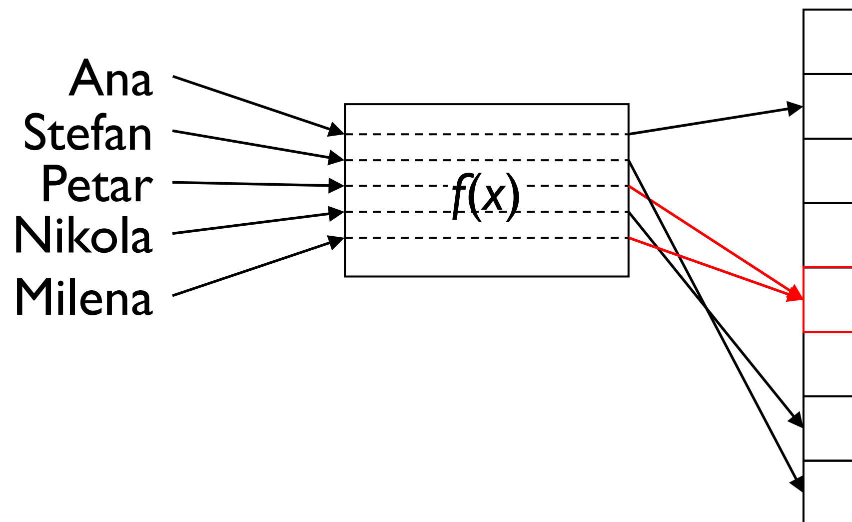
Heš mapa – struktura podataka za rečnik

Heš funkcija mapira ključeve na cele brojeve

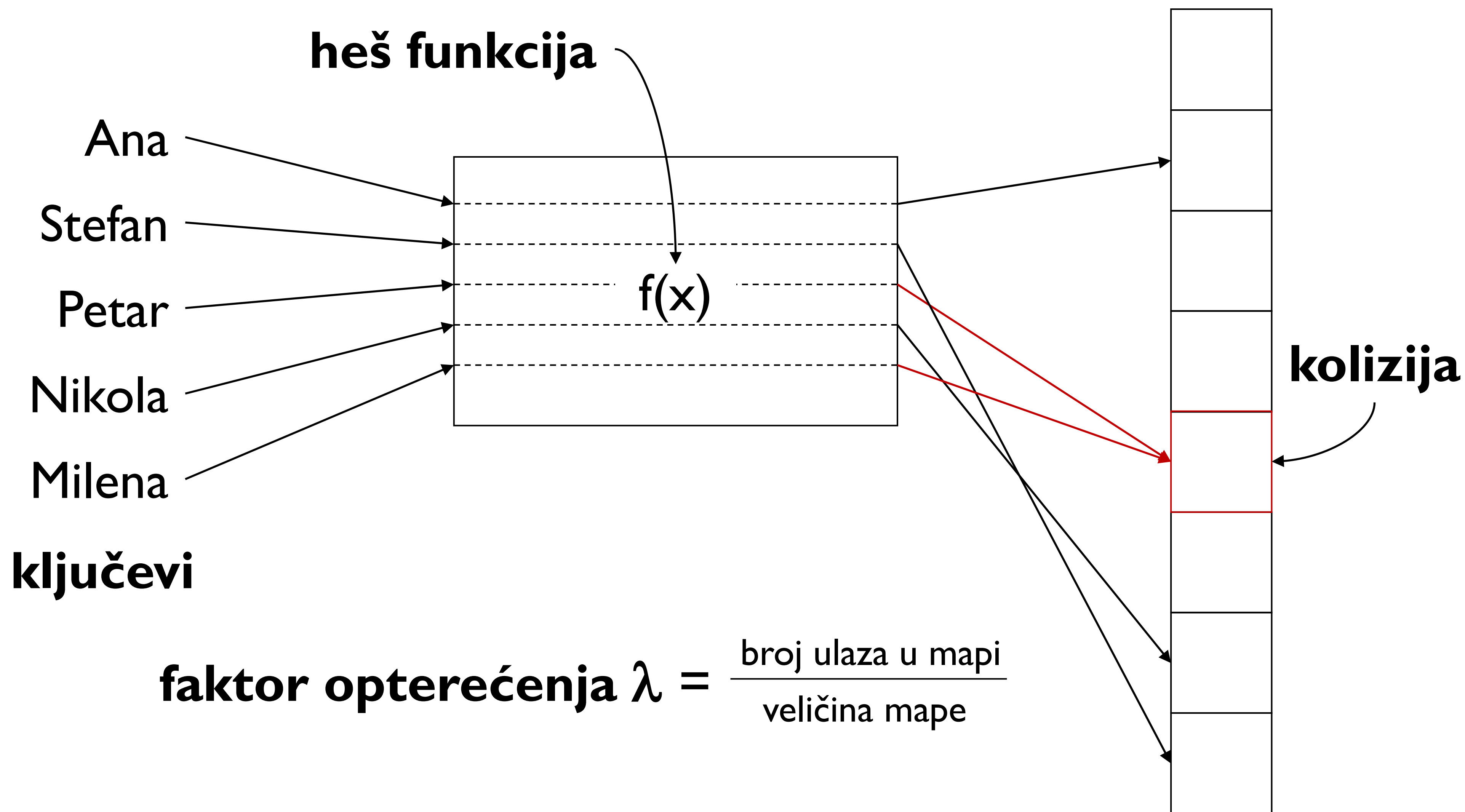
- **Možemo** brzo da pronađemo lokaciju za dati ulaz

Neuređene i retko-popunjene mape

- **Ne možemo** efikasno obići sve ulaze
- **Ne možemo** efikasno naći min, max ili uređene opsege



Heš mapa – pojmovi



Heš mapa – osnovni pojmovi

Heš mapa (tabela) – struktura podataka

Heš funkcija – funkcija za preslikavanje ključa u adresu (indeks) mape

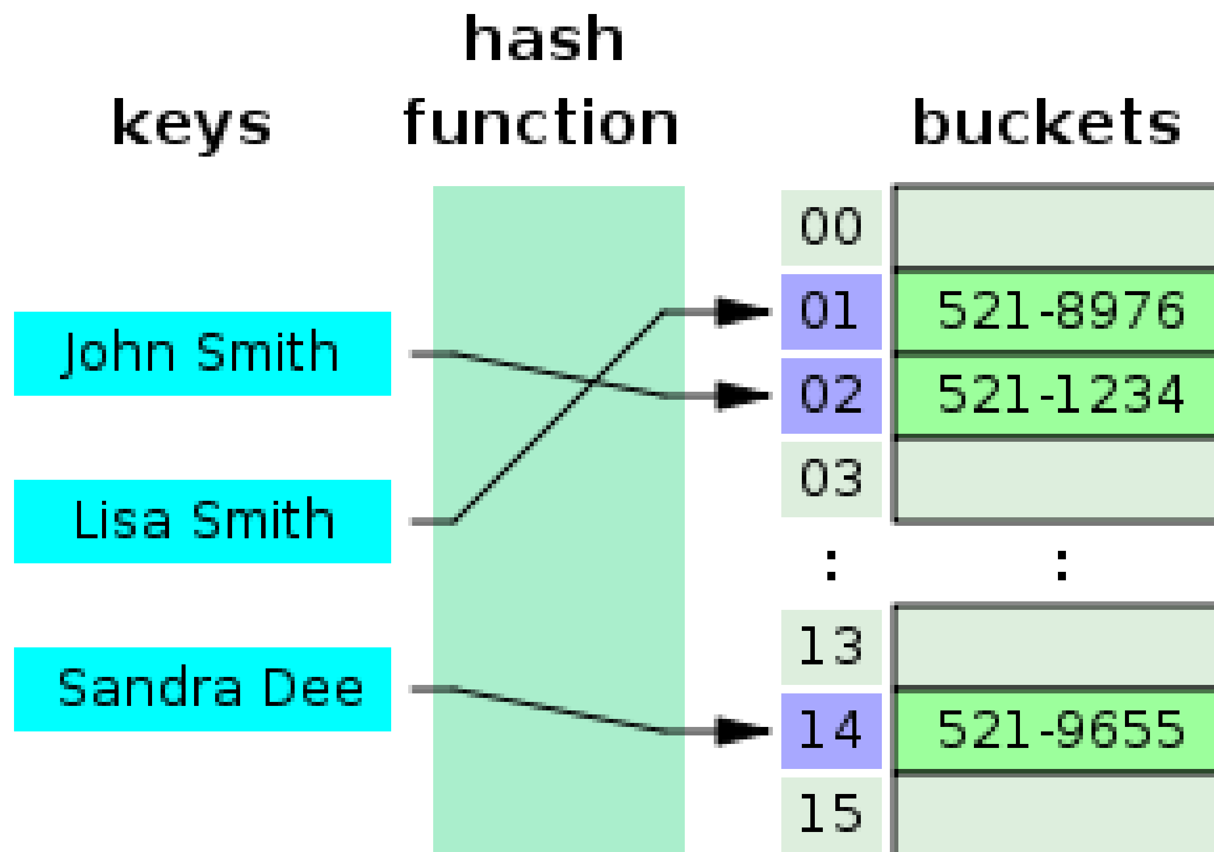
Ključ – vrednost na osnovu koje se određuje adresa podatka

Kolizija (engl. *collision*) – situacija kada se dva ključa preslikavaju u istu adresu u mapi

Sinonimi (engl. *synonyms*) – podaci sa različitim ključevima koji se preslikavaju u istu adresu u mapi

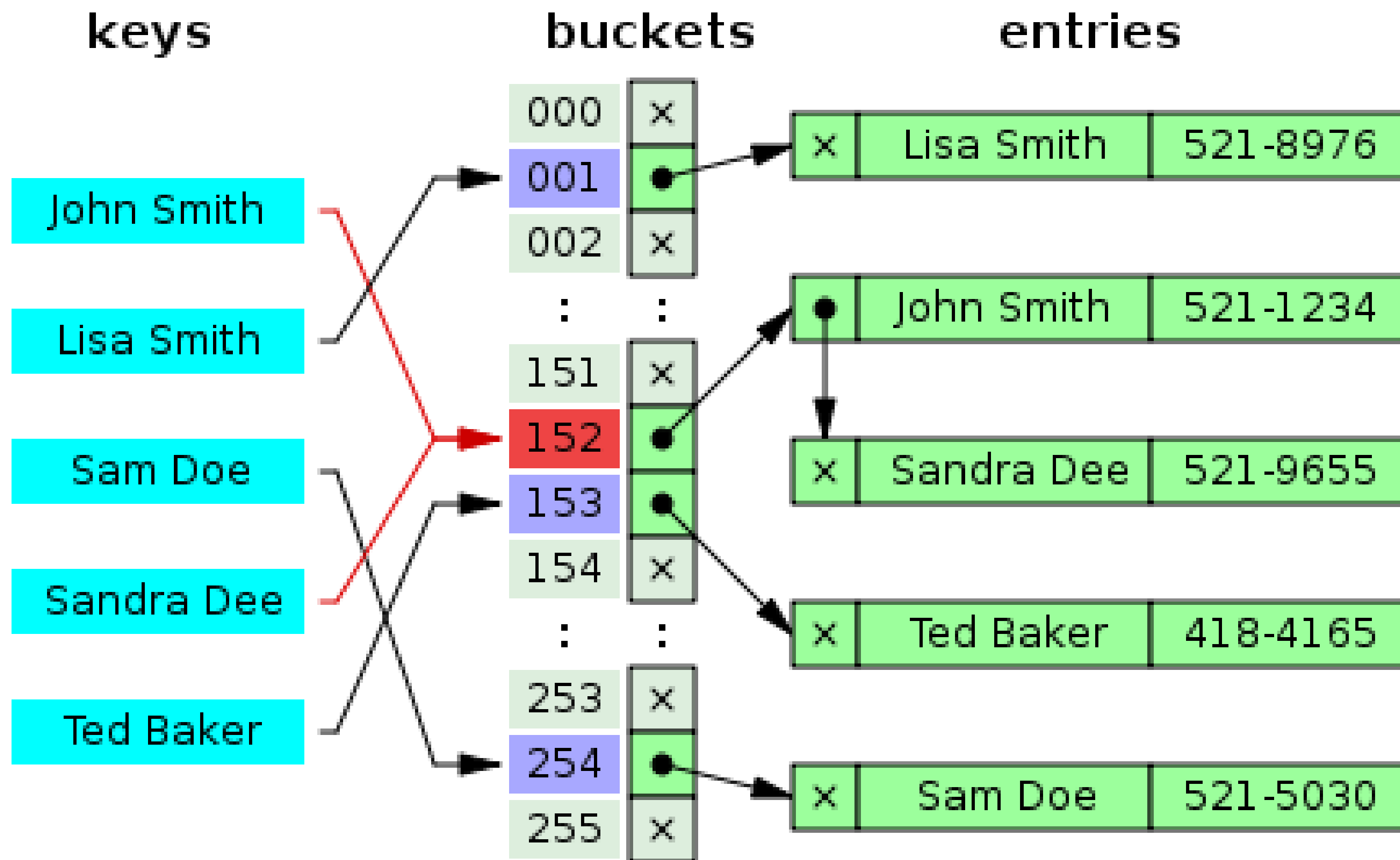
Faktor popunjenosti ili **opterećenja** (engl. *load factor*) – odnos broja elemenata i veličine mape

Heš mapa



Izvor: https://en.wikipedia.org/wiki/Hash_table

Heš mapa



Izvor: https://en.wikipedia.org/wiki/Hash_table

Projektovanje heš mape – odluke

Kako izabrati heš funkciju?

Kako odrediti pogodnu veličinu tabele?

Kako rešiti kolizije?

Heš funkcija

Funkcija koja **preslikava ključeve** (vrednosti na osnovu kojih se smeštaju podaci) **u cele brojeve**, tj. vrednosti indeksa polja, naziva se **heš funkcija**

Heš funkcija omogućava **mehanizam pristupa** koji **izbegava pretraživanje strukture** radi nalaženja elementa

Na osnovu podataka, vrši se izbor funkcije za preslikavanje njihove vrednosti u adrese, odnosno indekse polja

Izbor heš funkcije zahteva izvesno **predznanje o vrednostima podataka** koje se smeštaju u heš mapu

Osobine dobre heš funkcije

Brzo izračunavanje

- $O(1)$ i brzo u praktičnom smislu

Distribuira podatke ravnomerno

- heš(a) % *veličina* $\neq$ heš(b) % *veličina*

Koristi celu heš mapu

- za svako $0 \leq i <$ *veličina*, postoji neki k takvo da je heš(k) % *veličina* = i

Heš funkcije za celobrojne ključeve

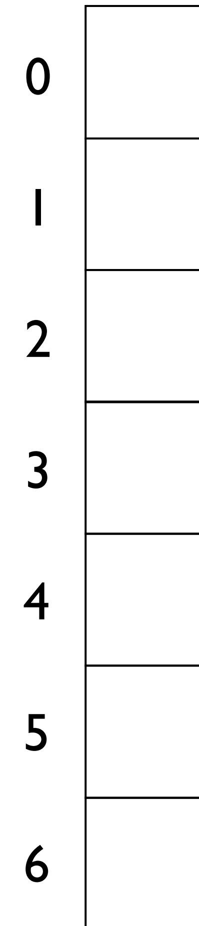
Izabrati:

- veličinu mape kao prost broj
- heš(*n*) = *n* % *veličina*
- **metod deljenja**

Primer:

- *veličina* = 7

dodaj(3)
dodaj(13)
nađi(19)
dodaj(8)
obriši(13)



Heš funkcije za stringove

Ako je $s = s_1 s_2 s_3 s_4 \ldots s_n$ izabrati:

- $\text{heš}(s) = s_1 + s_2 128 + s_3 128^2 + s_4 128^3 + \ldots + s_n 128^{n-1}$
- string se praktično posmatra kao broj u brojnom sistemu sa osnovom (engl. *radix*) 128

Problemi:

- heš("veoma, veoma dugačkog stringa") = veoma, veoma, veoma veliki broj
- heš("jedna stvar") % 128 = heš("druga stvar") % 128

Heširanje stringova – tehnike

Primer implementacije heš funkcije za stringove:

```
int hash(char[] s) {
        int h = 0, i;
        for (i = strlen(s) - 1; i >= 0; i--) {
                h += (s[i] + 128*h) % velicina;
        }
        return h;
}
```

Minimizacija kolizija

- Veličina tabele i osnova (*radix*) treba da su uzajamno prosti (uzajamno prosti su oni brojevi koji nemaju zajedničkog delioca većeg od 1)
- Tipično se vodi računa da veličina tabele ne bude umnožak od 128

Pojednostavljenje izračunavanja

- Upotreba Hornerovog pravila (za izračunavanje vrednosti polinoma)

Heš funkcije – metode za računanje

Najčešće korišćene metode za izračunavanje heš funkcija su:

- Metod deljenja
- Metod množenja
- Metod sredine kvadrata
- Fibonačijev metod
- Metod ekstrakcije
- Metod transformacije osnove

Dobro heširanje – metod množenja

Heš funkcija je definisana kao *veličina* plus parametar *A*

$$h_A(k) = \lfloor \mathit{veli\check{c}ina} \times ((k \times A) \bmod 1) \rfloor \text{ gde je } 0 < A < 1$$

Primer: $\mathit{veli\check{c}ina} = 10, A = 0.485$

$$h_A(50) = \lfloor 10 \times ((50 \times 0.485) \bmod 1) \rfloor$$

$$= \lfloor 10 \times (24.25 \bmod 1) \rfloor = \lfloor 10 \times 0.25 \rfloor = 2$$

- Nema ograničenja u veličini
- Kada se kreira statičko polje, može se probati više vrednosti parametra *A*
- Zahtevnije za izračunavanje od pojedinačne mod operacije

Heširanje – dileme

Ako se pretpostavi da **napadač zna korišćenu heš funkciju** i **može da bira koje ključeve šalje**, tada napadač može da **pošalje ključeve koji maksimiziraju kolizije kod izabrane heš funkcije čime se performanse svode na spregnutu listu!**

Zaključak: **Ne postoji jedna heš funkcija koja pruža zaštitu!**

Suočeni sa ovom situacijom, odlučujete da:

a) odustanete od heš mapa i koristite spregnutu listu za implementaciju vašeg rečnika

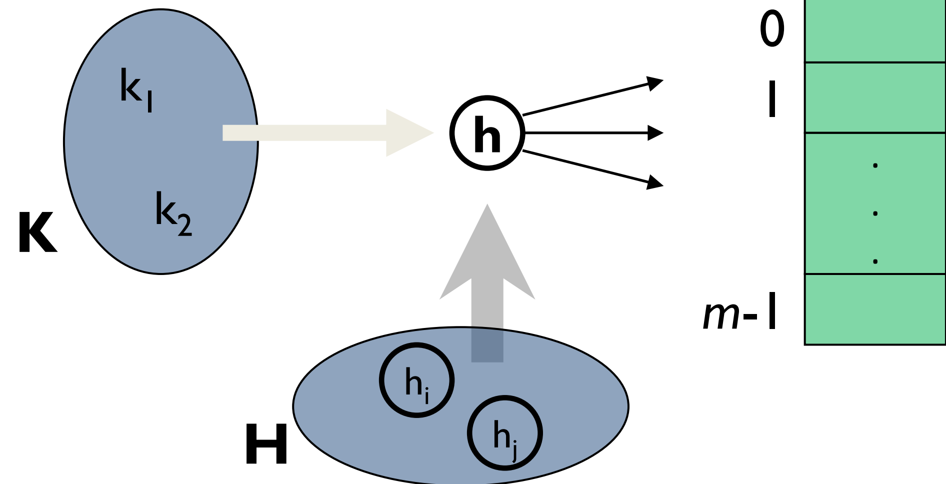
b) odustanete od studija informaconog inženjeringa i započnete karijeru u promociji proizvoda za ulepšavanje

c) prespavate ostatak časa, u nadi da pitanje neće biti na testu

d) sačekate do sledećeg slajda, u nadi da postoji rešenje ☺

Univerzalno heširanje

Pretpostavimo da imamo skup mogućih ključeva **K** i konačni skup heš funkcija **H** koje mapiraju ključeva na lokacije u heš mapi veličine *m*



H je **univerzalna kolekcija heš funkcija** ako i samo ako za bilo koja dva ključa k_1 i k_2 u **K**, postoji najviše |**H**|/m funkcija u **H** za koje je $h(k_1) = h(k_2)$

Ako **slučajno izaberemo heš funkciju** iz **H**, **verovatnoća kolizije nije ništa veća** nego kada bi **slučajno odabirali ulaze u heš tabelu**

Heš funkcija – izbor i projektovanje

Kako bi se izabrala dobra heš funkcija treba:

- znati koje se vrednosti ključeva očekuju
- analizirati distribuciju ključeva
- pokušati da se uključe sve važne informacije u ključu prilikom izračunavanja njegovog heša
- pokušati da se "susedni" ključevi heširaju u što je moguće više različite lokacije
- odbacivati svojstva korišćena za kreiranje heša dok izračunavanje ne bude dovoljno brzo, što značajno zavisi od konkretne primene

Heš funkcija – rezime

Ciljevi heš funkcije:

- ponovljivo mapiranje ključeva na indekse u tabeli
- ravnomerna distribucija ključeva u tabeli
- odvajanje ključeva koji se često pojavljuju
- brzo izračunavanje

Primeri heš funkcija:

- *h*(*n*) = *n % veličina*
- *h*(*n*) = *string kao broj sa osnovom 128 % veličina*
- Metod množenja
- Univerzalno heširanje