

# Napredni algoritmi i strukture podataka

Stabla, Merkle stabla, Merkle dokaz, Anti-entropy popravke, Serijalizacija stabla

---



Univerzitet u Novom Sadu  
Fakultet Tehničkih Nauka

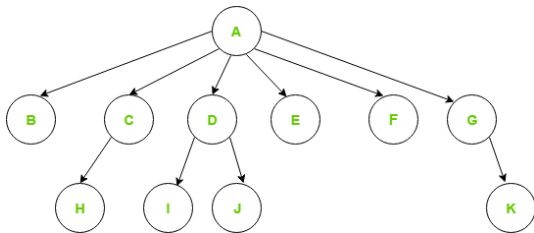
## Strukture podataka - stablo, kratak podsetnik

- ▶ Stablo predstavlja čvorove povezane ivicama — prosto gledano
- ▶ U računarskima naukama, stablo je široko korišćen apstraktni tip podataka koji prikazuje hijerarhijsku strukturu podataka
- ▶ Stablo sadrži korenski element (root) i podstabla dece sa roditeljskim čvorom, predstavljenim kao skup povezanih čvorova
- ▶ Struktura podataka stabla se može definisati *rekurzivno* kao kolekcija čvorova, gde je svaki čvor struktura podataka koja se sastoji od vrednosti, i liste referenci na druge čvorove

- ▶ Stablo se sastoji od čvorova koji su u vezi roditelj/dete
- ▶ Tačno jedan čvor nema roditelja — korenski element (root)
- ▶ Svaki čvor ima najviše jednog roditelja — osim korenskog
- ▶ Čvor ima nula ili više dece
- ▶ Čvor može da bude bez dece — list
- ▶ Čvor stabla i njegovi potomci — Podstablo
- ▶ Visina stabla — najveća dubina stabla

## N-arno stablo

- N-arno stablo — generičko stablo sa kolekcijom čvorova gde je svaki čvor struktura podataka koja se sastoji od zapisa, i liste referenci na svoje potomke (duplirane reference nisu dozvoljene)



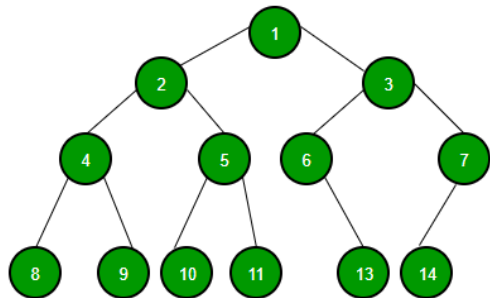
(Geeksforgeeks - Generic Trees(N-array Trees))

# Pitanje 1

Kako porediti N-arna stabla :) ?

# Binarno stablo

- ▶ Binarno stablo — stablo čiji elementi imaju najviše 2 deteta
- ▶ Pošto svaki element u binarnom stablu može imati samo 2 deteta, obično ih nazivamo *levo* i *desno* dete



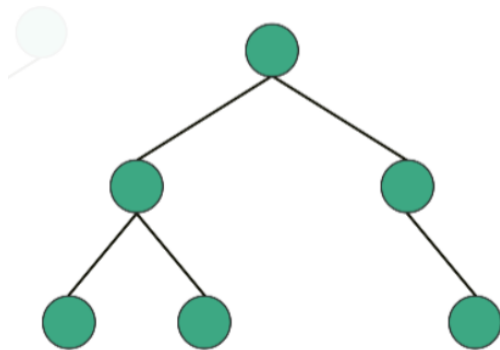
(Geeksforgeeks - Generic Trees(N-array Trees))

## Pitanje 2

Kako porediti binarna stabla :) ?

# Balansirano binarno stablo

- ▶ Balansirano binarno stablo — uravnoteženo po visini
- ▶ Definiše se kao binarno stablo u kome se visina levog i desnog podstabla bilo kog čvora razlikuje za **najviše 1**

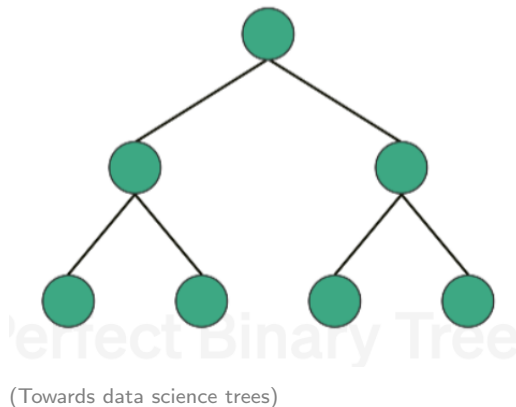


(Towards data science trees)



# Kompletno binarno stablo

- ▶ Kompletno binarno stablo — stablo koje ima sve nivoe u potpunosti popunjene čvorovima osim poslednjeg nivoa
- ▶ Na poslednjem nivou, svi čvorovi su što je moguće levo



## Pitanje 3

Kako formirati binarna/n-arna stabla :) ?

## Pitanje 4

Kako obilaziti binarna/n-arna stabla :) ?

# Problem 1

Zaposlili ste se u Dropbox-u (sweet), trebate da dodate funkcionanost provere integriteta podataka, da li ih slučajno Dropbox nije *oštetio* kada korisnik urad upload podataka, i obriše sa svog računara.

- ▶ Ne potrošimo previše resursa, ako je to moguće
- ▶ Kroz mrežu ne saljemo same podatke — nikako nije isplativo
- ▶ Procesna moć nije ograničavajući faktor

Ideje :)?

## Merkle stablo - uvod

- ▶ Merkle stablo ili **hash stablo** je stablo u kome je svaki čvor označen **hash** vrednošću
- ▶ Ova stabla omogućavaju efikasnu i bezbednu verifikaciju sadržaja velikih struktura podataka
- ▶ Merkle stablo je razvijeno 1979 godine, a 2002 je patent istekao (blago nama :))
- ▶ Ovo stablo je generalno **binarno** po prirodi
- ▶ Za razliku od uobičajenog postupka kod većine stabla, ovo stablo se formira od lista (dna) ka korenu (vrhu)

- ▶ Formalno gledamo, Merkle stabla uzimaju skup podataka  $(x_1, \dots, x_n)$  na ulaz
- ▶ Povratnu vrednost je **Merkel root hash**  $h = \text{MHT}(x_1, \dots, x_n)$
- ▶ MHT **collision-resistant hash funkcija**
- ▶ *Hash* funkcija je **collision-resistant hash funkcija** ako je teško pronaći dva ulaza koja *hash*-iraju isti izlaz
- ▶ Formalno, za ulaze  $a$  i  $b$ ,  $a \neq b$  ali  $H(a) = H(b)$

## Merkle stablo - formiranje

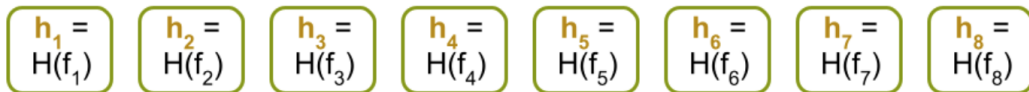
- ▶ Algoritam za formiranje Merkle stabla je relativno jednostavan
- ▶ Merkle stablo ima **bottom-top** pristup, zbog svoje specifičnosti
- ▶ Formiranje stabla počinje od dna tj. konkretizovanih podataka — **data block**
- ▶ Polako idemo do vrha, gradeći **Merkle root** element
- ▶ Prvi element koji gradimo je **list**
- ▶ Svaki podatak propustimo kroz *hash* funkciju, i tako formiramo prvi nivo — **list**

- ▶ Svaki **list** propustimo kroz *hash* funkciju, a svaka **dva susedna** elementa grade naredni nivo propuštajući njihove zajedničke hash vrednosti kroz hash funkciju
- ▶ Pošto radimo sa binarnim stablima, ako na nekom nivou nemamo odgovarajući čvor, možemo da dodamo *empty* element da bi algoritam mogao da se nastavi
- ▶ Kada propustimo poslednja dva čvora kroz hash funkciju dobijamo **Merkle root** element
- ▶ Time se algoritam za formiranje završava i formirali smo Merkle stablo



## Merkle stablo - formiranje, primer

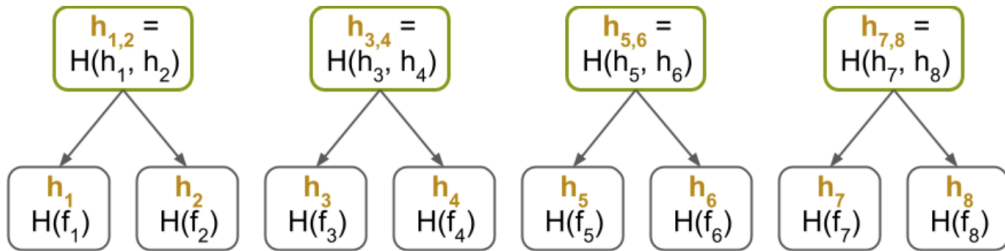
- ▶ Pretpostavimo da imamo 8 blokova podataka (fajlova)  $f = (f_1, \dots, f_8)$
- ▶ Svaki podataka  $f_i$  propustimo kroz hash funkciju  $H$  i dobijamo njegov *hash*
- ▶ Dobijamo hash vrednost za prvi nivo  $h_i = H(f_i)$ ,  $h_i = (h_1, \dots, h_8)$
- ▶  $H$  reprezentuje **collision-resistant hash** funkciju



(Decentralized Thoughts, Merkle trees)

## Merkle stablo - formiranje, primer

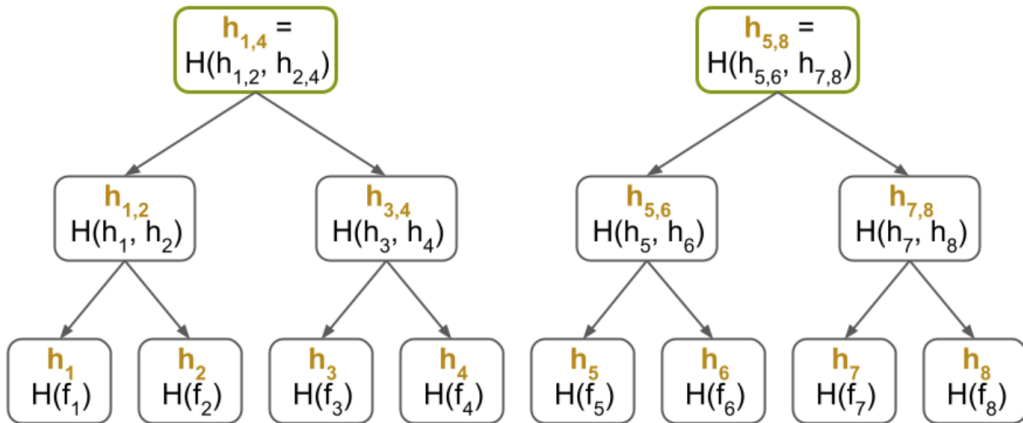
- ▶ Zabava se nastavlja u istom maniru, samo za naredne nivoe nam trebaju parovi :)
- ▶ Heširamo svaka dva susedna *hash*-a, da bi formirali sledeći nivo  
$$h_{k,m} = H(h_i, h_{i+1})$$
- ▶ Ako nam fali *hash*, da bi svako imao suseda :(), prosto napravimo prazan *hash* i nastavimo dalje (nećemo se stresirati oko gluposti)



(Decentralized Thoughts, Merkle trees)

## Merkle stablo - formiranje, primer

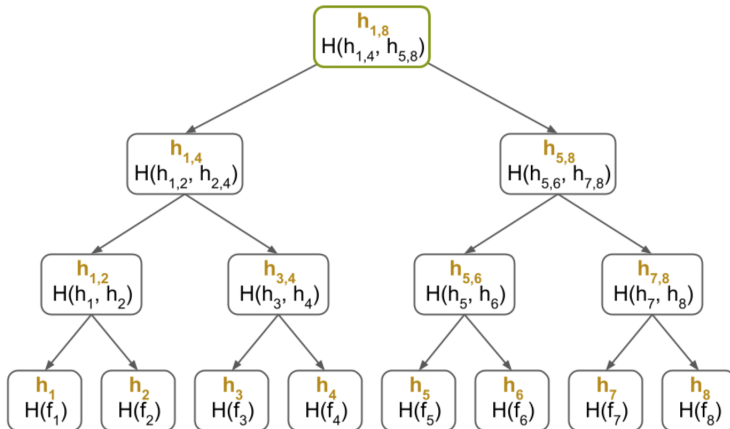
- Zabava se nastavlja isto kao i prethodno :)



(Decentralized Thoughts, Merkle trees)

## Merkel stablo - formiranje, primer

Idemo isto... i dobijamo  $h_{1,8} = H(h_{1,4}, h_{5,8})$



## Merkle stablo - napomena

- ▶ Ono što smo deobili na kraju  $h_{1,8}$  je **Merkle root hash**
- ▶ Obratiti pažnju da svaki čvor u stablu čuva **hash** vrednost
- ▶ Listovi čuvaju hash vrednost (blokova) podataka  $h_i = (h_1, \dots, h_8)$
- ▶ Čvorovi koji nisu listovi, i nisu **Merkle root hash**, čuvaju *hash* vrednost svoje dece — **internal node**

- ▶ Ako nam na nekom nivou fali par za neki element, prosto dodamo **prazan hash** da bi formirali par
- ▶ Može se lako generalizovati i izračunati Merkle stablo za bilo koji broj  $n$  podataka
- ▶ Formalno zapisano, prethodni primer se može zapisati kao  $h_{1,8} = \text{MHT}(f_1, \dots, f_8)$
- ▶ Merkel stabla se formiraju rekurzivno, od dna ka vrhu
- ▶ Ovaj proces može biti procesno zahtevan!
- ▶ To nikada nemojte izgubiti iz vida

## Merkle stabla - upotreba

- ▶ Merkle stabla se dosta koriste kod validacije podataka na različitim mestima
- ▶ Šta je potrebno da se sinhrnozuje, Merkle stablo može da nam kaže bez konkretnih podataka
- ▶ Danas se dosta koristi, i uglavnom se vezuje za *Blokchain* tehnologije
- ▶ **Merkle stabla  $\neq$  Blokchain**
- ▶ Merkle stablo nije jedino upotrebljeno za Blokchain
- ▶ Merkle stablo nije smišljeno za *Blokchain*
- ▶ Ali jeste osnova *Blokchain* tehnologije
- ▶ Pogledajte koliko Blockchain troši struje, pa će vam biti jasna konstatacija — *Ovaj proces može biti procesno zahtevan!*

## Pitanje 5

Zašto formiramo stablo, zar ne bi bilo jednostavnije da formiramo lanac, moža možemo izbeći rekurzije...

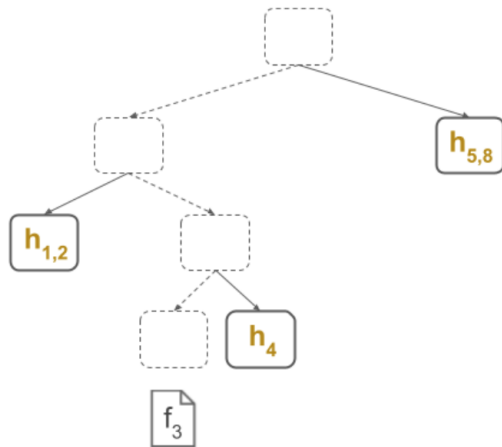
Ideje :)?



## Merkel dokaz

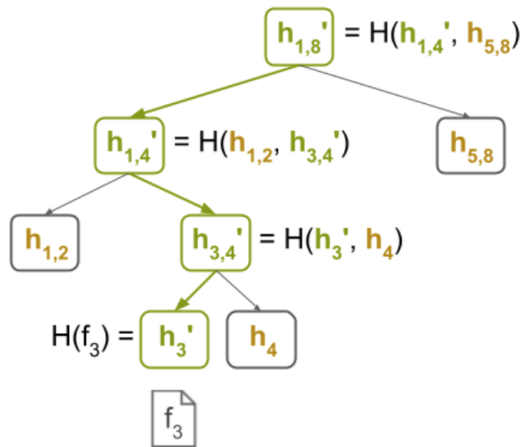
- ▶ Ako bi formirali *hash-eve* kao lanac, vrlo je moguće da bi utrošili manje resursa
- ▶ Proces bi verovatno bio znatno jednostavniji, rekurzije mogu biti nezgodne
- ▶ **ALI** taj proces ima jednu nezgodnu osobinu — manu
- ▶ Ako bi trebali da proverimo integritet podataka, ili da li podataka pripada nekom skupu morali bi da sačuvmo/poredimo kompletan skup
- ▶ Ako su nam podaci na više čvorova, to znači da bi morali da prebacujemo ceo skup podataka kroz mrežu
- ▶ U najmanju ruku, to nije lepo i nije kulturno :)
- ▶ Zato se (binarno) stablo pokazalo boljim izborom

- ▶ Ako se vratimo na problem integriteta podataka koji su prebav ceni negde na skladištenje (npr. Dropbox)
- ▶ Ključna ideja je da, nakon što preuzmete podatak  $f_i$ , tražimo mali deo Merkle stabla — **Merkle dokaz**
- ▶ **Merkle dokaz** nam omoguććava da proverimo da li preuzeti podatak  $f_i$  nije slučajno ili zlonamerno izmenjen
- ▶ Ako imamo podatak  $f_3$ , zanima nas da li je on deo većeg skupa podataka



(Decentralized Thoughts, Merkle trees)

- ▶ Da bi to odredili, treba nam samo mali deo Merkle stabla, da bi formirali Merkle root
- ▶  $h_3$  možemo izračunati propuštajući podatak kroz *hash* funkciju
- ▶ Delovi koji nam trebaju su  $h_4$ ,  $h_{1,2}$ ,  $h_{5,8}$
- ▶ Sa ovim delovima možemo stići do Merkle root-a
- ▶ Ova ideja se dosta koristi kod Torrent sistema



(Decentralized Thoughts, Merkle trees)

- ▶ Merkle dokaz pokšava da nam kaže da li podatak pripada Merkle stablu ili ne
- ▶ Da se nedvosmisleno dokaže valjanost podataka koji su deo skupa podataka, bez skladištenja celog skupa podataka i dobacivanja kroz mrežu
- ▶ Da bi se osiguralo da je skup podataka deo većeg skup podataka bez otkrivanja kompletnog skupa podataka ili njegovog podskupa
- ▶ S obzirom na to da su jednosmerne *hash* funkcije namenjene da budu algoritmi bez kolizija, nijedna dva *hash-a* ne mogu da budu ista
- ▶ Ove osobine mogu biti primamljive za razne tipove aplikacija (ne samo *Blokchain*)

## Problem 2

Zaposlili ste se u Amazonu (lepo), razvijate nov sistem za skladištenje podataka i od vas se očekuje da razvijete sistem za efikasnu verifikaciju sadržaja velikih skupova podatak. Pred vama su sledeća ograničenja:

- ▶ Ne potrošimo previše resursa ako je moguće
- ▶ Kroz mrežu ne saljemo same podatke — nikako nije isplativo

Ideje :)?

## Anty-entropy - ideja

- ▶ Merkle stabla se mogu koristiti za sinhronizaciju podataka i proveru ispravnosti kopija u sistemima sa više čvorova (peers) u distribuiranom sistemu
- ▶ Ne moramo da poredimo čitave podatke da bismo shvatili šta se promenilo
- ▶ Možemo samo da uporediti *hash* stabala
- ▶ Kada shvatimo koji listovi su promenjeni, odgovarajući komad podataka može da se pošalje preko mreže i sinhronizuje na svim čvorovima — znatno jednostavnije
- ▶ Ova ideja se dosta koristi kod velikih sistema za skladištenje podataka (Amazon DynamoDB, Cassandra, ScyllaDB, ...)

## Anty-entropy - algoritam

- ▶ Algoritam je relativno jednostavan i odvija se u tri koraka
  1. Napraviti Merkle stablo za svaku repliku (čvor) koja čuva kopiju podataka
  2. Uporedite Merkle stablo da bi otkrili razlike
  3. Razmenite šta je potrebno od selova podataka da svi imaju isti skup kopija
- ▶ Izgradnja Merkle stabla je resursno intenzivna operacija, opterećuje disk I/O i koristi dosta memorije :(
- ▶ Time plaćamo manje slanje podataka kroz mrežu — **Nema besplatnog ručka**
- ▶ Ovaj proces se često odbija u pozadini, da ne bi blokirali ostatak sistema

- ▶ Provera kreće od vrha stabla, ako je *root hash* identičan, nema potrebe za popravkama
- ▶ Ako to nije slučaj, prelazimo na **levo dete**, zatim na **desno dete** — obilazimo stablo
- ▶ Postupak se nastavlja dok ne stignemo do bilo kakve razlike u podacima
- ▶ Kada ustanovimo šta je različito, samo taj deo skupa podataka treba da se popravi — pošalje kroz mrežu
- ▶ Kreće proces razmene podataka
- ▶ Odgovor na pitanje svih pitanja: **Ne, nećete razmenjivati podatke sa drugim čvorovima/replikama za projekat. Ali formiraćete Merkle stablo vaših podataka :)**



## Pitanje 5

Ako imamo formirano binarno stablo (Merkle ili ne), kako ćemo ga serijalizovati/zapisati u datoteku...

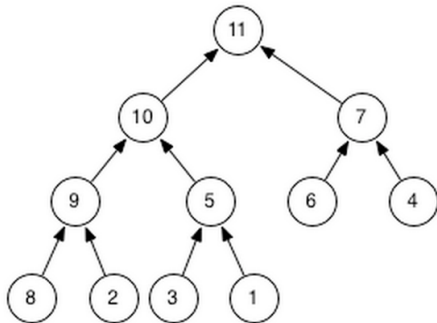
Ideje :)?

## Serijalizacija stabla

- ▶ U nekim situacijama, treba da zapišemo stablo u fajl i da kasnije radimo nešto sa njim
- ▶ Idealan primer nam je **Anty-entropy**, kasnije moramo da poredimo stabla da vidimo razlike i prebacujemo podatke kroz mrežu
- ▶ Postoji dosta načina kako bi mogli da stablo zapišemo u fajl
- ▶ Pogledajte samo koliko ima tehnika obilazaka stabla

## Serijalizacija stabla — jedan primer

- ▶ Ako imamo stablo kao sa slike
- ▶ Treba da idemo kroz njega, nekim od poznatih algoritama
- ▶ Jedna opcija je da idemo po nivoima:
  - ▶ [11 10 7 9 5 6 4 8 2 3 1]
- ▶ Treba voditi računa ako na nekom nivou imamo manjka elmenata, treba da zapišemo nekakav marker da nam bude jasan znak za kasnije!
- ▶ Ovo neće biti problem kod Merkle stabala, ali u opštem slučaju treba voditi računa



(Ritambhara, Storing Binary Tree in a file)

## Dodatni materijali

- ▶ Merkle tree original paper
- ▶ Merkle tree easier paper
- ▶ Amazon Dynamo db paper
- ▶ Decentralized Thoughts
- ▶ Cassandra Manual repair: Anti-entropy repair
- ▶ Understanding Merkle Trees - Why Use Them, Who Uses Them, and How to Use Them
- ▶ Merkle tree and Golang

## Merkel stabla - Pitanja

Pitanja :) ?