Merkleova stabla i njihove primjene u blockchain tehnologiji

Polutranko, Matej

Master's thesis / Diplomski rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Science / Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:217:734659

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-10-03**

Repository / Repozitorij:

Repository of the Faculty of Science - University of Zagreb

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET
FIZIČKI ODSJEK

Matej Polutranko

Merkleova stabla i njihove primjene u blockchain tehnologiji

Diplomski rad

Zagreb, 2022.

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET
FIZIČKI ODSJEK

INTEGRIRANI PREDDIPLOMSKI I DIPLOMSKI SVEUČILIŠNI STUDIJ
FIZIKA; SMJER NASTAVNIČKI

Matej Polutranko

Diplomski rad

Merkleova stabla i njihove primjene u blockchain tehnologiji

Voditelj diplomskog rada: doc. dr. sc. Andrej Novak

Ocjena diplomskog rada: _____

Povjerenstvo: 1. _____

2. _____

3. _____

Datum polaganja: _____

Zagreb, 2022.

Zahvaljujem se mentoru doc. dr. sc. Andreju Novaku na ukazanom povjerenju, pomoći i podršci pri izradi rada.

Sažetak

Merkleovo stablo je binarno hash stablo čiji čvorovi, za razliku od klasičnih stabala, spremaju hash vrijednosti. Najveću primjenu je pronašlo u blockchain tehnologiji gdje se koristi kao dio procesa sigurnog spremanja transakcija. Uz to što osigurava zaštitu od neovlaštenih izmjena ono omogućuje i sažimanje podataka, pa i transakcija, u jednu vrijednost zvanu Merkleov korijen.

Merkleovo stablo je ključan dio stvaranja kriptovaluta pa je opisana i osnovna ideja blockchaina na primjeru Bitcoina. Bitcoin je prva poznata izvedba decentraliziranog načina plaćanja u kojem korisnici sudjeluju u ravnopravnoj mreži. Način na koji se svi korisnici mogu složiti o stanju popisa transakcija se zove konsenzus, a u Bitcoinu je on ostvaren pomoću dokaza o radu. Dokaza o obavljenom radu se dobije rješavanjem kriptografskih zagonetki, a on omogućuje korisnicima spremanje novih blokova u blockchain.

Ključne riječi: Merkleovo stablo, Merkleov put, blockchain, dokaz o radu, hash funkcija

Merkle trees and their applications in blockchain technology

Abstract

Merkle tree is a binary tree in which every leaf is labelled with the cryptographic hash of a data block, while other nodes are labelled with the cryptographic hash of the labels of their child nodes. Its most popular use case is in blockchain where it is used as part of the process of securely storing transactions. In addition to providing protection against unauthorized changes, it also enables data, and by extent transactions, to be compressed into a single value called the Merkle root.

The Merkle tree is a key data structure used by cryptocurrencies like Bitcoin and Ethereum, and we use these platforms to demonstrate the basic ideas behind blockchain technology. Bitcoin is the first known implementation of a decentralized payment method in which users participate in a peer-to-peer network. The way in which all users can agree on the state of transactions is called consensus, and in the case of Bitcoin, it is achieved through a proof of work paradigm. Simply speaking, proof of work is obtained by solving cryptographic puzzles, and it is what allows miners to extend the blockchain with new blocks.

Keywords: Merkle tree, Merkle path, blockchain, proof of work, hash function

Sadržaj

1 Uvod

U radu je prikazna struktura podataka poznata kao Merkleovo stablo koju je osmislio i objavio Ralph Merkle još 1979. godine u svojoj doktorskoj disertaciji na Standfordu [1]. Ono je primjer binarnog stabla, a danas se takvo stablo i njegove varijante mogu pronaći u brojnim sustavima koji spremaju, manipuliraju i šalju podatke. Pokazalo se kao iznimno efikasan način provjere vjerodostojnosti podataka, a najveću primjenu ima dakako u kriptografiji.

Neke od primjena su i u sustavima za verzioniranje datoteka poput Gita, brojnim nerelacijskim bazama podataka, ali vjerojatno najpopularnija primjena je u blockchain tehnologiji. Upravo kroz primjer kriptovaluta Bitcoin i Ethereum je prikazana stvarna primjena Merkleovog stabla u nekom inženjerskom pothvatu.

U drugom i trećem poglavlju su opisani pojmovi koji čine osnovnu Merkleovog stabala, a to su hash funkcije i binarna stabla. U poglavlju četiri je definirano Merkleovo stablo, a u poglavlju pet je opisana osnovna ideja blockchaina i kako se u tu priču uklapa Merkleovo stablo. U šestom poglavlju je ukratko opisana generalizacija Merkleovog stabla na primjeru kriptovalute Ethereum.

1.1 Motivacija

Kad se radi sa skupovima podataka prirodno pitanje koje se javlja je: „Kako najbolje organizirati podatke za potrebe programa?”. Ako se u skup podataka rijetko dodaju novi elementi, a oni su organizirani u strukturu čija je najveća prednost to što se dodavanje izvršava u kratkom, konstantnom vremenu jasno da nije maksimalno iskorišteno računalo. Procesor će tijekom drugih, učestalijih operacija obavljati više rada zbog tako odabrane strukture.

Unatoč velikoj brizni modernih procesora i dalje je u cilju birati prave alate kako bi se smanjila prostorna i vremenska složenost programa. Poznavanje prostora problema omogućava povoljan odabir struktura podataka i algoritama, pa se time povećava efikasnost programa.

U radu sa skupovima podataka najčešće se ističu četiri operacije: dodavanje novih elemenata, traženje, brisanje i sortiranje postojećih elementa. Izbor odgovarajućih struktura omogućuje odabir brzih algoritama čime se smanjuje vremenska složenost programa. Povećanje efikasnosti učestalih operacija je lagan put k smanjenju broja

potrebnog računa.

Jedan klasični primjer strukture podataka je stablo traženja: Za veliki skup brojeva s n elemenata moguće je pronaći bilo koji broj, u prosjeku, u samo $\log(n)$ operacija.

Još jedan primjer stabla koje pomaže u efikasnijem računanju je hrpa. Ono je iznimno povoljna struktura kad se žele dodati novi element u skup podataka koji je organiziran tako da se može brzo pronaći najveći ili najmanji element, obično je on prvi ili zadnji, pa je jednu od svojih primjena našla u algoritmima za određivanje najkraćeg puta.

Osnovne strukture podataka su oslonac većine programskih rješenja, a njihova važnost se ne može dovoljno istaknuti. Stoga se u hrvatskim prirodoslovno-matematičkim gimnazijama one obrađuju kao obavezan dio informatičkog kurikuluma. Već se u trećem razredu spominje složenost algoritama, liste, baze podataka, a u četvrtom razredu su obrađene gotovo sve osnovne strukture: red, stog, stablo, graf i druge.

Spomenute strukture su i nezaobilazni element natjecanja iz informatike na kojima svake godine sudjeluje više tisuća učenika [2].

1.2 Notacija i kod

Svi primjeri su napravljeni u programskom jeziku C (99), a korištene su iduće pokrate za standardne primitivne tipove podataka koje se nalaze u biblioteci *stdint.h*:

```
typedef uint8_t  u8;
typedef uint16_t u16;
typedef uint32_t u32;
typedef uint64_t u64;
typedef int8_t  s8;
typedef int16_t s16;
typedef int32_t s32;
typedef int64_t s64;

typedef s32 bool32;
typedef float  f32;
typedef double f64;
```

2 Binarno stablo

Općeniti koncept stabla ima široku primjenu čak i izvan računarstva [3] [4] [5].

Za potrebe rada se koristi posebna vrsta uređenog n-stabla za $n = 2$ koje se zove binarno stablo. Definirano je rekurzivno [6]:

> Binarno stablo T je konačan skup podataka istog tipa koje zovemo čvorovi i vrijedi:
>
> - T je prazan skup, ili
> - Postoji istaknuti čvor r koji se zove korijen od T, a ostali čvorovi grade uređeni par (T_L, T_R) disjunktnih binarnih stabala.

Primjer takvog stabla je na slici 2.1. Stablo je izgrađeno od čvorova, u ovom slučaju to su slova abecede, koji su međusobno povezani. Svaki čvor ima ne više od dva podčvora koja se zovu lijevo i desno dijete. Korijen cijelog stabla je čvor koji nema roditelja, a čvor koji nema niti jedno dijete zove se list.

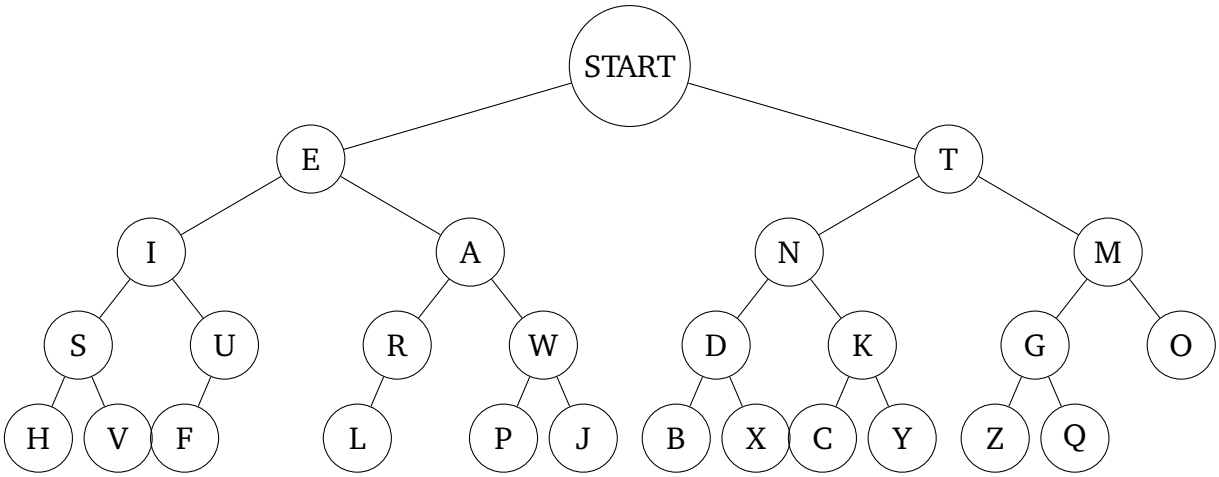
U stablu 2.1 čvorovi R i W su lijevo i desno dijete čvora A, a čvor A je njihov roditelj. Čvorovi imaju svoja podstabla pa je tako primjerice čvor A korijen dva podstabla. Svako podstablo čvora A ima svoje zasebne čvorove i tako dalje. Podstabla nemaju zajedničkih čvorova pa su ona disjunktna. Čvor A je i sam po sebi dio podstabla čiji je korijen čvor E.

Stabla su organizirana u jasnu hijerarhijsku strukturu. Čvorovi su podijeljeni na razine, a broj razina, izuzev samog korijena stabla, zove se visina stabla h. Visina stabla na slici 2.1 je $h = 4$.

2.1 Podjela i svojstva

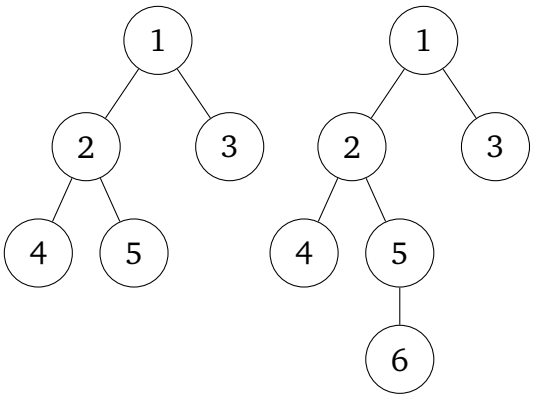
Ovisno o tome koliko su i kako razgranate grane binarnog stabla ono je svrstano u neku od tipičnih kategorija:

- *Savršeno*: binarno stablo u kojem svi unutarnji čvorovi imaju točno dva djeteta, a svi listovi su na istoj visini h. Stablo sa slike 2.1 bi bilo primjer savršenog binarnog stabla kad bi se nadopunila četiri lista koja nedostaju. Savršeno stablo visine h ima $n = 2^{h+1} - 1$ čvorova, a $l = 2^h$ listova.



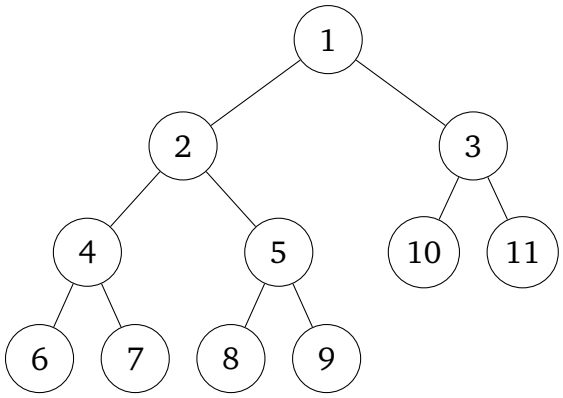
Slika 2.1: Morseov kod prikazan pomoću binarnog stabla. Ako se želi dobiti prikaz neke riječi pomoću Morsevog koda sve što je potrebno je silaziti niz stablo slovo po slovo. Ako se pomakne lijevo zapisuje se točka, a ako se pomakne desno zapisuje se crtica: STABLO = ... $-$. $-$ $-$... . $-$ .. $-$ $-$ $-$

- *Balansirano*: binarno stablo u kojem je razlika visina Δh lijevog i desnog podstabla svakog čvora manja ili jednaka 1. Primjer se vidi na slici 2.2.
- *Puno*: binarno stablo u kojem svaki čvor ima ili 0 ili 2 djeteta.
- *Potpuno*: savršeno binarno stablo kod kojeg je obrisan jedan ili više listova, a svi preostali listovi su pomaknuti u lijevo. Primjer se vidi na slici 2.3. Svako potpuno binarno stablo s n čvorova će imati isti raspored čvorova.



Slika 2.2: Primjer balansiranog i nebalansiranog binarnog stabla.

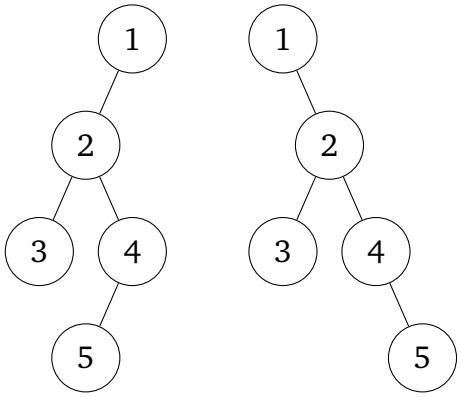
Način na koji se stabla spremaju u memoriji ovisi o samoj vrsti stabla. Primjerice, ako svi čvorovi stabla imaju samo jedno dijete, takvo se stablo zove degenerirano, ono će se efektivno ponašati kao vezana lista (struktura koja je kratko objašnjena na kraju trećeg poglavlja). Ako je u pitanju potpuno stablo ono se možemo vrlo efikasno prikazati pomoću jednostavnog polja. U tako spremljenom stablu korijen se nalazi



Slika 2.3: Primjer potpunog binarnog stabla.

na poziciji 0 u polju, a njegovo lijevo i desno dijete na pozicijama 1 i 2. Općenito, za i-ti čvor lijevo dijete će se nalaziti na poziciji $2i + 1$, a desno dijete $2i + 2$.

Stablo kod kojeg je važno je li neki čvor lijevo ili desno dijete zove se uređeno stablo. Na slici 2.4 je primjer dva stabla s naizgled istim rasporedom čvorova, ali kako su to uređena stabla onda ta stabla nisu ista.



Slika 2.4: Dva različita, uređena, binarna stabla.

2.2 *Osnovna implementacija pomoću pokazivača*

Detalji strukture stabla ovise o problemu koji se rješava pa je u ovom potpoglavlju prikazan samo osnovni način kako konstruirati binarno stablo.

Temeljna struktura koja se koristi za izgradnju stabla je čvor *(engl. node)*:

```
typedef struct
{
    struct node * Left;
    struct node * Right;
    s32 Data;
} node;
```

Struktura čvora je jednostavna: dva pokazivača na druga dva čvora, lijevo i desno dijete, i dodatni spremnik za vrstu podataka koju stablo sprema. U ovom primjeru čvorovi su spremljeni u dinamičku memoriju:

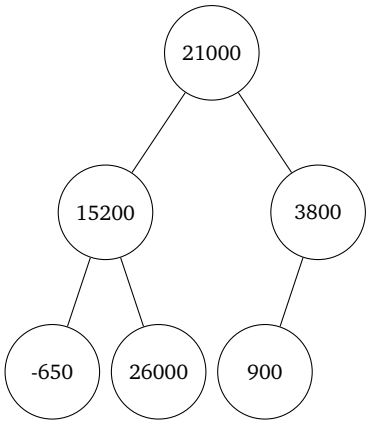
```
static void *
CreateNode(s32 Data)
{
    node * Result = malloc(sizeof(node));
    if(Result)
    {
        Result->Data  = Data;
        Result->Left  = 0;
        Result->Right = 0;
    }
    return (Result);
}
```

Pomoću dobivenih čvorova se konstruira stablo na način (slika 2.5):

```
// Korijen
node * Root = CreateNode(21000);

if(Root)
{
    // 1. Razina
    Root->Left  = CreateNode(15200);
    Root->Right = CreateNode(3800);

    if(Root->Left && Root->Right)
    {
        // 2. Razina
        ((node*)Root->Left)->Left   = CreateNode(-650);
        ((node*)Root->Left)->Right  = CreateNode(26000);
        ((node*)Root->Right)->Left  = CreateNode(900);
    }
}
```

2.3 Operacije

Za zadano binarno stablo T osnova operacija je obilazak stabla. Pod obilazak stabla se obično misli posjećivanje svakog čvora samo jedanput.



Slika 2.5: Binarno stablo iz primjera koda.

Stabla se mogu obilaziti na razne načine, a ovdje je naveden primjer iz grupe algoritama za obilazak u dubinu *(engl. depth-first)*. Još jedan tipični primjer je i obilazak u širinu, odnosno razinu po razinu.

Osnovna ideja je slična za sve algoritme: za dani čvor se rekurzivno poziva funkcija obilaska za oba podstabla, a ovisno u kojem je trenutku pozvana funkcija koja obavlja rad dobije se drugačiji redoslijed (pre-order, in-order, post-order).

Primjerice, za *pre-order* redoslijed funkcija koja obavlja rad, u ovom slučaju *printf*, se izvršava prije rekurzivnih poziva (slično i za druge redoslijede):

```
static void
PreOrder(node * Root)
{
    if(Root)
    {
        printf("%d, ", Root->Data);
        PreOrder((node*)Root->Left);
        PreOrder((node*)Root->Right);
    }
}
```

Obilaskom stabla sa slike 2.5 za tri različita redoslijeda dobijemo:

```
PreOrder(Root);  // 21000, 15200, -650, 26000, 3800, 900
InOrder(Root);   // -650, 15200, 26000, 21000, 900, 3800
PostOrder(Root); // -650, 26000, 15200, 900, 3800, 21000
```

Osim obilaska stabla često se želi i pronaći zadani čvor u stablu. Kako na osnovnom binarnom stablu nisu definirana pravila za operaciju dodavanja novih elementa zadani čvor se može naći na bilo kojem mjestu. To povlači da se u stablu T s n ele-

menata traženje zadanog čvora u najgorem slučaju obavlja u n koraka. Vremenska složenost traženja je $\mathcal{O}(n)$. Primjer kako pronaći neki čvor pomoću malo izmijenjenog algoritma obilaska stabla:

```
static node *
FindNode(node * Root, s32 Data)
{
    node * Result = 0;
    if(Root)
    {
        if(Root->Data == Data)
        {
            Result = Root;
        }
        else
        {
            Result = FindNode((node*)Root->Left, Data);

            if(!Result)
            {
                Result = FindNode((node*)Root->Right, Data);
            }
        }
    }
  return (Result);
}
```

Primjer korištenja funkcije pronalaksa čvora:

```
node * SomeNode = FindNode(Root, 900);
if(SomeNode)
{
  printf("SomeNode->Data: %d", SomeNode->Data);
   // SomeNode->Data: 900
}
```

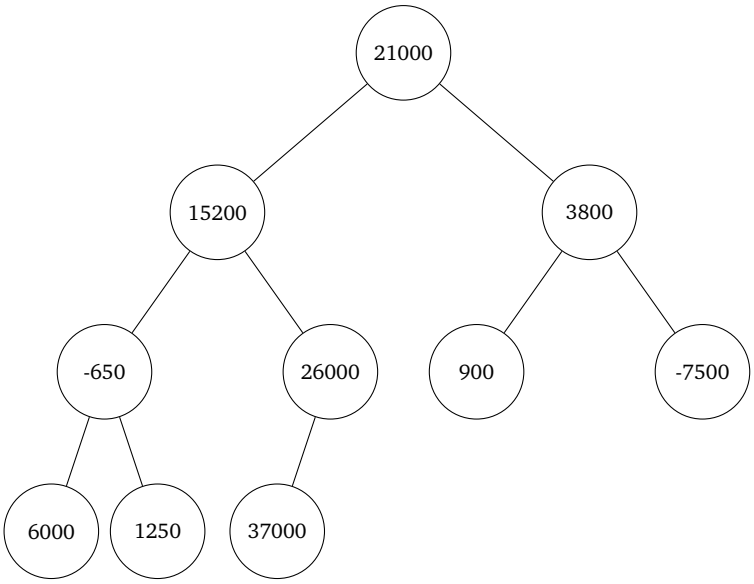
Osnovni način dodavanja novih čvorova u binarno stablo je na prvo slobodno mjesto, odnosno na mjesto lijevog ili desnog djeteta odgovarajućeg čvora. „Prvi" čvor ima drugačije značenje za različite vrste stabla. Primjerice, prvi čvor u potpunom stablu je list koji se nalazi, geometrijski gledano, skroz lijevo ili čvor koji ima samo jedno dijete (u potpunom stablu postoji samo jedan takav čvor).

Stablo sa slike 2.5 je izgrađeno točnim odabirom pozicija svakog čvora. Ako je važno sačuvati balansiranost stabla jednostavnija metoda dodavanja novih čvorova je pomoću algoritma za obilazak prvo u širinu *(engl. breadth-first)*. Ovo je jedan primjer kako zadati takvu funkciju pomoću reda:

```
static void
InsertNode(node * Root, s32 Data)
{
    u32 NumNodes     = CountNodes(Root, 0);
    node_queue Queue = CreateNodeQueue(NumNodes);
    Enqueue(&Queue, Root);

    while(Queue.Count)
    {
        node * Node = Dequeue(&Queue);
        if(!Node->Left)
        {
            Node->Left = CreateNode(Data);
            break;
        }
        else Enqueue(&Queue, (node*)Node->Left);

        if(!Node->Right)
        {
            Node->Right = CreateNode(Data);
            break;
        }
        else Enqueue(&Queue, (node*)Node->Right);
    }

   DeleteNodeQueue(&Queue);
}
```

Na slici 2.6 se vidi kako će stablo izgledati nakon što su mu dodana četiri nova čvora:

```
InsertNode(Root, -7500);
InsertNode(Root, 6000);
InsertNode(Root, 1250);
InsertNode(Root, 37000);
```



Slika 2.6: Balansirano binarno stablo sa slike 2.5 nakon što smo dodali četiri nova čvora.

Prilikom dodavanja novih čvorova se ne može na trivijalan način odrediti mjesto gdje će se dodati novi čvor, osim ako se ne naruši potpunost stabla. Za dodavanje novog čvora je nužno pretražiti stablo sve dok se ne pronađe prvo povoljno slobodno mjesto. Ako stablo imam n čvorova ponovno će vremenska složenost iznositi $\mathcal{O}(n)$.

Osim dodavanja, čvorovi stabla se mogu i uklanjati, a na mjesto uklonjenog čvora dolazi njegovo dijete. Algoritam brisanja čvorova je sličan algoritmu traženja čvorova: Prvo se u stablu pronađe odgovarajući čvor te ako taj čvor ima samo jedno dijete on se i ukloni. Ako čvor ima više od jednog djeteta nije jednoznačno određeno koje će dijete zauzeti njegovo mjesto pa brisanje takvih čvorova nije dopušteno. Slično kao i kod dodavanja novog čvora, nema trivijalnog načina pronalaska čvora pa u najgorem slučaju to će biti upravo n koraka.

2.4 Osvrt

Binarno stablo je struktura podataka koja omogućuje prikaz podataka na hijerarhijski način. Ovisno o načinu na koji su složena razlikuje se više vrsta osnovnih binarnih stabla od kojih su najkorisnija potpuna i savršena. Osnovne operacije poput obilaska, dodavanja, brisanja i traženja čvorova su sve redom vremenske složenosti $\mathcal{O}(n)$.

Merkleovo stablo je primjer binarnog stablo, ali točnije binarnog hash stabla. U idućem poglavlju je opisano što su hash funkcije i koja njihova svojstva su korisna, a zatim su u četvrtom poglavlju spojene hash funkcije i binarna stabla kako bi se definiralo Merkleovo stablo.

3 Hash funkcija

Hash funkcija h predstavlja preslikavanje između dva skupa A i B:

$$h : A \to B.$$

Elementi skupa A se zovu ključevi, a elementi skupa B hash vrijednosti. Skup A je proizvoljan i može biti: $\mathbb{Z}$,$\mathbb{R}^n$, $\mathbb{N}^n$ i tako dalje. Najčešće se za ključeve uzima niz znakova $A \subset \mathbb{N}^n$. Hash vrijednosti su zadane fiksne dimenzije, obično je to $B \subset \mathbb{N}$.

Za nas, osnovno svojstvo hash funkcije je pridruživanje prirodnog broja nizu znakova. Primjerice, za jednostavnu hash funkciju $h(x) = |x|$:

$$h\,(\text{”Neka rečenica.”}) = 14,$$
$$h\,(\text{”Neka druga rečenica.”}) = 20.$$

Iz primjera se vidi kako je ključna ideja hash funkcije sažimanje:

$$h : 2^{\text{VELIKI BROJ}} \to 2^{\text{MALI BROJ}}$$

Posljedica koja se može uočiti zbog sažimanja velikog skupa u mali je „gubljenje” informacija. Za hash funkciju iz prijašnjeg primjera i $x =$ „Ovo je stvarno isto?”:

$$h\,(\text{„Ovo je stvarno isto?”}) = 20.$$

Preslikavanje različitih ključeva u istu hash vrijednost se zove sudar. Kako je skup A znatno veći od skupa B ovaj problem se javlja i kod netrivijalnih hash funkcija. Upravo je jedna mjera „dobrote” hash funkcije koliko ona smanjuje broj sudara. Za hash funkcije je važno koliko se brzo i efikasno može izračunati hash vrijednost, a one su u pravilu i determinističke, odnosno hash vrijednost je uvijek ista za zadani ključ.

3.1 Hash tablica

Kad se radi s poljima koja sadrže podatke koji nisu sortirani čak i najbrže metode pretraživanja ovise linearno o broju elemenata u polju. Za manja polja cijelih brojeva takve metode su u praksi često dovoljne, no ako polja sadrže elemente koji nisu samo cijeli brojevi nego nizovi brojeva ili znakova *(engl. strings)* metode linearnog pristupa mogu značajno usporiti program.

Upravo u takvim slučajevima hash tablica može značajno ubrzati proces pristupanja; Hash tablica omogućuje vrlo brz pronalazak odgovarajućeg niza elementa. Takva prednost nije potpuno bez mane, a to je najčešće povećanje prostorne složenosti. Kako je danas memorija praktično neograničena takav kompromis je gotovo uvijek prihvatljiv.

Nedavno smo mogli vidjeti i primjer stvarne primjene koji je obišao digitalni svijet. Naime, programer pod jednostavnim imenom T0ST je objavio članak [9] u kojem je prikazao kako je u popularnoj igri GTA online skratio vrijeme učitavanja za čak 70%, s otprilike $6\ \text{min}$ na $2\ \text{min}$, bez direktnog pristupa izvornom kodu. Upravo je središnja popravka bila uvođenje hash tablice umjesto direktnog provjeravanja svakog elementa pojedinačno.

Pojmovi hash tablica i hash funkcija su usko povezani. Pojednostavljeno, hash funkcije se koriste kako bi se konstruirale hash tablice. Hash tablica je na neki način poopćenje indeksiranja polja. Elementi polja se ne dohvaćaju kao:

```
1 u32 Indeks = 0;
2 ...
3 s32 NekiBroj = Polje[Indeks];
```

nego:

```
1 u8 * Kljuc = 0;
2 ...
3 u32 Indeks   = HashFunkcija(Kljuc);
4 s32 NekiBroj = Polje[Indeks];
```

Hash tablica se sastoji od dva dijela: hash funkcije i polja podataka. Kako je prednost spremanja podataka u polje to što je ono jedan kontinuirani blok memorije često će polje podataka u hash tablici biti fiksne duljine, odnosno postojat će ograničeni broj podataka koje se u polje možemo spremiti.

Ako je polje zadane fiksne duljine n onda je potrebno na neki način prilagoditi

izlaznu vrijednost hash funkcije jer bi u suprotnom dobiveni indeks i mogao biti veći nego veličina polja. Najjednostavnija metoda je pomoću operacije modulo: za indeks $i > n$ vrijedi $i' = i - n\lfloor\frac{i}{n}\rfloor$.

```
u8 * Kljuc = 0;
...
u32 Indeks   = HashFunkcija(Kljuc) % DuljinaPolja;
s32 NekiBroj = Polje[Indeks];
```

Središnji dio tablice je polje podataka pa će ona imati otprilike sljedeću strukturu:

```
typedef struct
{
    u32 Num;
    u32 Max;
    ht_entry * Entries;
} hash_table;
```

gdje je *ht_entry* neka proizvoljna vrsta podataka koja se upisuje u tablicu.

Omjer broja elemenata u tablici *Num* i kapaciteta tablice *Max* se zove faktor opterećenja. Faktor opterećenja je povezan s vjerojatnošću sudara prilikom dodavanja novog elementa, što je veći to je i vjerojatnost sudara veća. Ovisno o tome koji je način rješavanja sudara odabran faktor opterećenja govori i tome koliko često će se u predmemoriji nalaziti pogrešni podatci *(engl. cache miss)*.

3.2 Rješavanje sudara i osnovne operacije

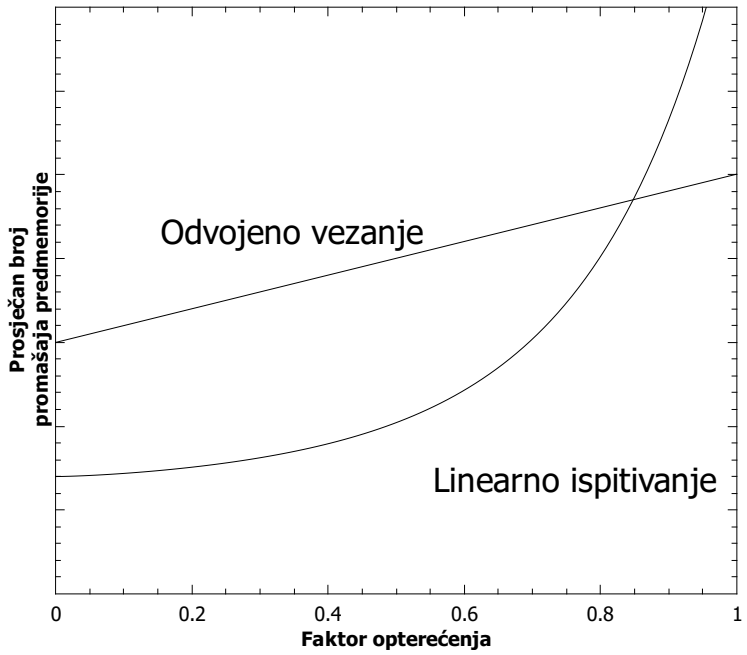
Idealna hash funkcija bi poprimala jedinstvene vrijednosti za sve različite ključeve unutar zadanog problema. No, u praksi je izvedba takve idealne hash funkcije nemoguća pa je potrebno iskoristiti neku od metoda rješavanja sudara. Dva su osnovna načina kako riješiti problem sudara: Prvi i preferirani način je iskoristiti jake hash funkcije. To su hash funkcije koje teoretski ne garantiraju jedinstveni izlaz za svaki mogući ulaz, ali je u praksi vjerojatnost sudara dovoljno mala ili se ulaznih podatci mogu jednostavno prilagoditi tako da se efektivno ponašaju kao idealne hash funkcije. Može se primijetiti kako kod takvih funkcija nije potrebna hash tablica. Podatci se mogu na jednostavan način povezati s jedinstvenim prirodnim brojem pa se dohvaćanje nekog elementa svodi samo na računanje njegove pripadajuće hash vrijednosti.

Drugi način je eksplicitno rješavanje sudara unutar hash tablice. Tri su klasične tehnike: odvojeno vezanje, korištenje dvije hash funkcije i otvoreno adresiranje. Za potrebe rada ćemo prikazati odvojeno vezanje i opisati otvoreno adresiranje.

Ideja otvorenog adresiranja je ispitivanje *(engl. probing)*: Ako za ključ k već postoji zapis na dobivenom mjestu $i = h(k)$ onda se polje pretražuje sve dok se ne pronađe slobodno mjesto. Polje se može pretraživati na razne načine. Osnovna metoda je linearno, jedno po jedno mjesto: $h(k) = (h(k) + 1)$. Često se koristi i metoda kvadratnog ispitivanja.

Kako su svi elementi tablice na jednom mjestu, u jednom kontinuiranom bloku memorije, metoda otvorenog adresiranja je iznimno povoljna za predmemoriju. Broj promašaja predmemorije linearno raste s brojem elementa u tablici, ali samo dok je faktor opterećenja ispod neke kritične vrijednosti.

Kad je faktor opterećenja iznad kritične vrijednosti broj promašaja predmemorije eksponencijalno raste kao što se vidi u primjeru na slici 3.1. Česti promašaji predmemorije dovode do značajnog smanjenja brzine programa pa se polje kod takve hash tablice morao povećati kad faktor opterećenja dosegne kritičnu vrijednost. Kako to uvodi dodatne komplikacije mi ćemo se zadržati samo na jednostavnijoj metodi rješavanja sudara, a to je odvojeno vezanje.



Slika 3.1: Graf prosječnog broja promašaja predmemorije ovisan o faktoru opterećenja za linearno ispitivanje i odvojeno vezanje.

Kako u ovom osnovnom primjeru tablica ima jednostavnu strukturu sve što je potrebno je odrediti maksimalan broj elemenata i dodijeliti pokazivač na blok memorije dovoljne veličine.

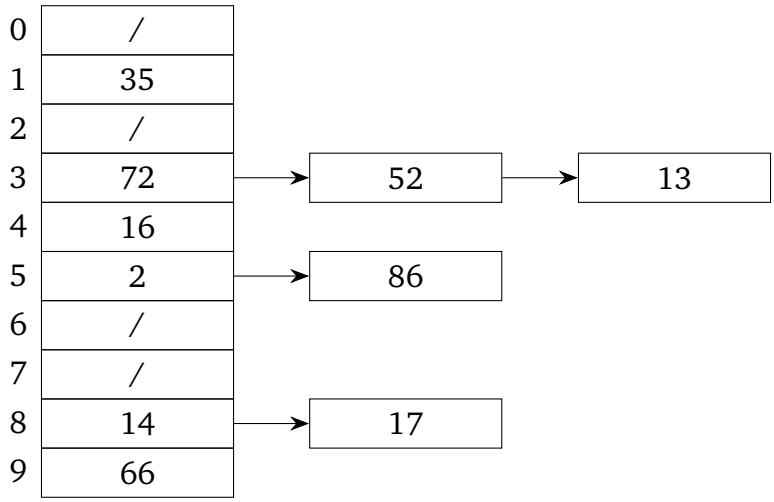
U tablicu će se upisivati imena gradova i broj stanovnika. Za imena gradova su potrebni nizovi znakova pa je prije svega nužno odlučiti gdje će se znakovi spremiti, direktno u tablicu ili u nekakav vanjski spremnik. Radi jednostavnosti svi znakovi će se upisati u tablicu no često je pogodnije napraviti jedan veliki spremnik koji će čuvati sve znakove na jednom mjestu. Kako se znakovi spremaju direktno u tablicu onda će *ht_entry* imati oblik:

```
typedef struct
{
    struct city * Next;
    u32 Population;
    char Name[MAX_NAME_LEN];
} city;
```

Odabrano je odvojeno vezanje kao način rješavanja sudara pa struktura *ht_entry* sadrži pokazivač *Next* koji omogućuje stvaranje vezane liste. Postupak unosa novih elementa je jednostavan. Prvo se izračuna hash vrijednost novog elementa te ako na dobivenom mjestu ne postoji već nešto tu se i spremi.

Ako je na dobivenom mjestu već nešto zapisano onda se taj već zapisani element premjesti u dodatni spremnik, a novi element se upiše na njegovo mjesto u tablici. Pokazivač *Next* novog elementa se postavi na premještenu lokaciju starog. Tako je dodan novi element u tablicu i stvorena, ili produljena, vezana lista čiji su čvorovi *ht_entry*, odnosno *city*.

Kao što se vidi na slici 3.2, svi elementi tablice nisu više na jednoj lokaciji u memoriji pa su za hash tablicu potrebni dodatni spremnici. Često potraživanje memorije od operacijskog sustava dodatno usporava program, pa se memorija mora na neki način organizirati kako bi se osigurali dodatni spremnici koji su potrebni za premještanje elemenata prilikom sudara. Jedan od način kako organizirati dodatnu memoriju je da se na početku programa jednostavno zatraži veliki kontinuirani blok memorije u koji će biti premješteni svi potrebni elementi.



Slika 3.2: Slika pokazuje primjer rasporeda memorije kod hash tablice kod koje je korišteno odvojeno vezanje.

Primjer funkcije dodavanja elemenata u tablicu:

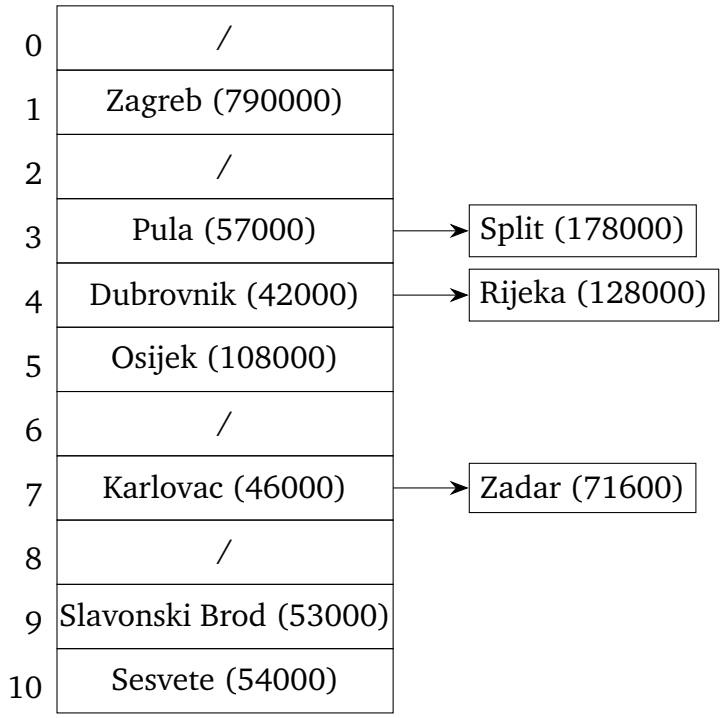
```
static bool32
InsertCity(hash_table* Table, void* Memory, char* Name, u32 Population)
{
    bool32 Result = 0;
    city NewCity  = {0};

    strcpy_s(NewCity.Name, MAX_NAME_LEN, Name);
    NewCity.Population = Population;
    u32 Hash  = HashFunction(Name) % Table->Max;
    u32 Index = Hash ? Hash : 1;

    if(Table->Entries[Index].Population != 0) {
        city * OldCity        = Memory;
        (city*)NewCity.Next   = OldCity;
        *OldCity              = Table->Entries[Index];
        Table->Entries[Index] = NewCity;
        Result = 1;
    }
    else {
        Table->Entries[Index] = NewCity;
        Table->Num++;
    }
    return (Result);
}
```

Za hash funkciju je korišten Fowler–Noll–Vo hash [8]:

```
static u32 HashFunction(u8 * Key)
{
    u32 Result = FNV_OFFSET_BASIS;
    u32 Strlen = strlen(Key);

    for(u32 i = 0; i < Strlen; i++)
    {
        Result ^= (u8)Key[i];
        Result *= FNV_PRIME;
    }
    return (Result);
}
```

gdje su *FNV_OFFSET_BASIS i FNV_PRIME* zadane konstante za 32-bitnu hash vrijednost. Na slici 3.3 je prikazana hash tablica nakon što je uneseno nekoliko elemenata:

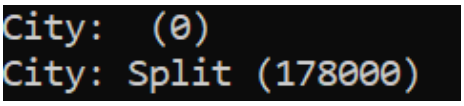
```
s32 Next = 0;
Next += InsertCity(&Table, (city *) Memory + Next, "Zagreb", 790000);
Next += InsertCity(&Table, (city *) Memory + Next, "Split",  178000);
Next += InsertCity(&Table, (city *) Memory + Next, "Rijeka", 128000);
...
```



Slika 3.3: Slika pokazuje primjer hash tablice s odvojenim vezanje za nekoliko unesenih vrijednosti.

Iz slike 3.3 i funkcije unosa novih elemenata se može uočiti kako niti jedan element ne može biti na prvom mjestu u polju, odnosno pod indeksom 0. To je čest obrazac kako bi se smanjio broj pogrešaka u programu. Tako je smanjen broj potrebnih provjera ispravnosti koda jer ako se zatraži element koji nije u tablici, a daljnji tijek programa ovisi o postojanju dohvaćenog elementa, program će i dalje nastaviti s normalnim radom bez prekida (neće se referencirati nul-pokazivač). Kaže se da je nulti element rezerviran. Primjer funkcije dohvaćanja elemenata:

```
static void *
GetCity(hash_table * Table, char * Name)
{
    void * Result = &Table->Entries[0];
    u32 Hash      = HashFunction(Name) % Table->Max;
    u32 Index     = Hash ? Hash : 1;

    if(Table->Entries[Index].Population)
    {
        if(Table->Entries[Index].Next == 0)
        {
            Result = &Table->Entries[Index];
        }
        else
        {
            city * Next = &Table->Entries[Index];
            do
            {
                if(Name && strcmp(Name, Next->Name) == 0)
                {
                    Result = Next;
                    break;
                }

                Next = (city*) Next->Next;
            } while(Next);
        }
    }
    return (Result);
}
```



Slika 3.4: Slika pokazuje primjer dohvaćanja elementa iz hash tablice za dva unosa „Pariz" i „Split". Možemo primijetiti kako program nije naglo prestao s radom kad nije uspješno pronađen element „Pariz" u tablici jer se nismo referencirali na pokazivač koji ne postoji već na rezervirani, nulti, element.

Iz zadane hash funkcije se vidi kako je unos novih podataka gotovo neovisan o broju postojećih elemenata u tablici. Ako hash funkcija, povezana s tablicom, ima relativno uniformnu raspodjelu (tako da se izbjegne scenarij u kojem se većina ključeva preslika u samo mali broj hash vrijednosti) pretraživanje tablice obavljat će se u konstantom vremenu. Upravo je ta činjenica najveća prednost hash tablice; Preslikavanje nekog velikog i rijetkog skupa koji ne stane u memoriju u manji skup podataka koji se može pospremiti i čiji se elementi mogu brzo i efikasno dohvatiti.

3.3 Kriptografska hash funkcija

Kriptografske hash funkcije se razlikuju od običnih hash funkcija po tome što se njihove hash vrijednosti, ili češće se kaže sažetci poruka, značajno razlikuju za čak i najmanje promjene ulaznih podataka. Primjerice, za neku kriptografsku hash funkciju h i dva slična ulaza:

$$h\,(\text{"Primjer nekog ulaza."}) = 214867,$$
$$h\,(\text{"Primjer nekog ulaza!"}) = 987821854867.$$

Osim kao preslikavanja hash funkcije se mogu promatrati i kao algoritmi. Algoritmi koji ulazne podatke transformiraju u neki prirodni broj zadane duljine. Obično je rezultat takvog algoritma 128-bitni ili 256-bitni broj. Osnovno svojstvo koje čini kriptografske hash funkcije korisnim je njihova uniformna raspodjela. One svaki jedinstveni ključ preslikaju u jedinstveni sažetak pa se za takve funkcije očekuje da na konačnim vremenskim i prostornim skalama nije moguće dobiti isti sažetak za različite poruke. Stoga su kriptografske hash funkcije otporne na sudare.

Kako su sažetci naizgled nasumični i jako osjetljivi na promjene pronalazak inverzne funkcije $h^{-1} : B \to A$ takve da $h^{-1}(h(x)) = x$ je također nemoguć u smislenom vre-

menu. Kriptografske hash funkcije su jednosmjerne; Lagano je izračunati sažetak poruke ali nemoguće je iz sažetka dobiti poruku.

Navedena svojstva su iznimno korisna za sigurnosne protokole. Kriptografske hash funkcije omogućuju razne načine ovjere podataka: Od digitalnog otiska do provjere vjerodostojnosti datoteka. Pomoću njih je moguće i dokazati posjedovanje rješenje bez da se to rješenje otkrije. Neke od poznatijih grupa kriptografskih hash funkcija su: MD, SHA, RIPMED i BLAKE.

Za konstrukciju Merkleovih stabla se najčešće koristi kriptografska hash funkcija SHA-256. Ona spada u grupu SHA-2 hash funkcija, a konstrukcija funkcije se temelji na originalnom radu Ralph Merklea [1]. Broj mogućih izlaza SHA-256 je naravno 2^{256}, broj koji je nezamislivo velik, pa se praktički za bilo koji niz ulaznih znakova, odnosno nula i jedinica, dobije jedinstveni broj. Obično se izlaz SHA-256 zapisuje u heksadecimalnom brojevnom sustavu:

$$\text{SHA256}(\text{"Je li netko vidio ovaj broj?"}) =$$
$$16df75ba9d230730bee643a238f5b8f173d823ca2a1e917e6c712c6e950fdf26.$$

Kriptografske algoritme je značajno sporije izračunati nego obične hash funkcije [10], ali se danas neki algoritmi poput SHA mogu izračunati na Intel, i ADM Zen, arhitekturama kroz niz intrinzičnih operacija što dovodi do ubrzavanja računa poruke [11].

4 Merkleovo stablo

Korisnik želi od poslužitelja preuzeti datoteku na svoje računalo. Kako će biti siguran da je preuzeta datoteka uistinu onakva kakvu je i zatražio? Očito rješenje je da se datoteka u cijelosti preuzme iz pouzdanog izvora, primjerice središnjeg servera.

Takvo rješenje je sasvim zadovoljavajuće sve dok središnji server može isporučiti tražene datoteke. Kako takav izvor ima ograničenu mrežnu propusnost on može istovremeno posluživati samo određen broj korisnika. Ako je tražena datoteka netrivijalne veličine vrlo brzo može doći do usporavanja preuzimanja kad veliki broj drugih korisnika želi preuzeti datoteke s danog servera. Također, takav središnji server uvijek mora biti spojen na mreži kako bi u svakom trenutku bio dostupan korisnicima.

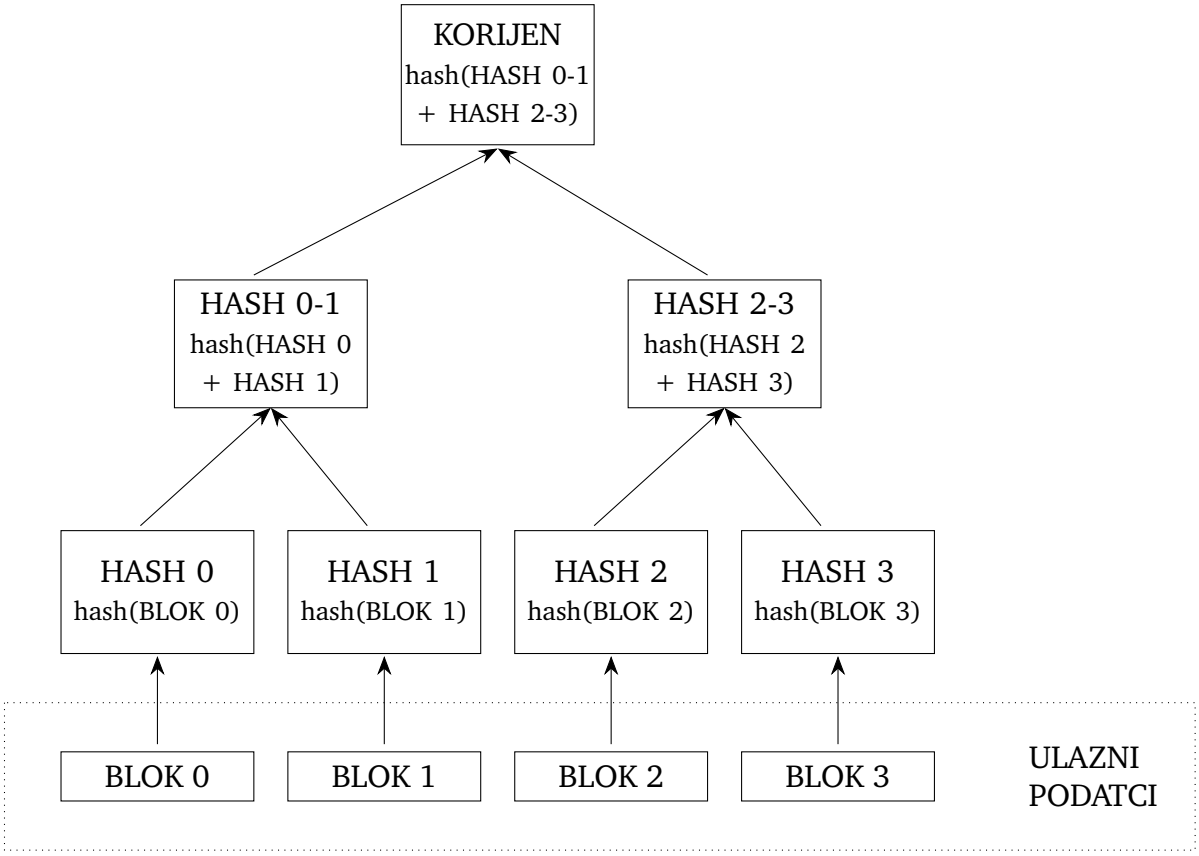
Način kako riješiti problem je da se sve korisnike organizira u ravnopravnu mrežu *(engl. peer-to-peer network)*. Svaki korisnik na mreži može svakom drugom korisniku slati datoteke, u cijelosti ili samo dijelove. Kako svi mogu svima slati potrebne datoteke više nema problem s nedostatnom mrežnom propusnosti jednog izvora. S takvom ravnopravnom mrežom je riješen jedan problem no ponovno se vratio onaj početni. Kako osigurati da ostali korisnici na mreži nisu izmijenili traženu datoteku prije nego što je isporuče drugim korisnicima?

Upravo se takav problem provjere ispravnosti podataka preuzetih iz nepouzdanih izvora može relativno lagano riješiti pomoću Merkleovog stabla.

4.1 Osnovna struktura

Merkleovo stablo je binarno hash stablo. Za razliku od klasičnih stabala koja spremaju općenite podatke u svoje čvorove, čvor Merkleovog stabla sprema hash vrijednosti. Primjer takvog stabla se vidi na slici 4.1. Može se uočiti kako čvorovi ne spremaju bilo kakve hash vrijednosti. Hash vrijednost nekog čvora se konstruira pomoću hash vrijednosti njegove djece.

Stablo se konstruira rekurzivno, od listova prema korijenu. Kako bi se pokrenulo rekurzivno šifriranje roditelja, početne hash vrijednosti listova dobiju se od ulaznih podataka. Šifrirani ulazni podatci se spremaju u listove stabla koji se zatim u parovima udružuju kako bi se izračunala hash vrijednost njihovih roditelja. Postupak udruživanja i računanja hash vrijednosti roditelja se ponavljamo sve dok se ne dođe do korijena Merkleovog stabla.



Slika 4.1: Primjer jednog Merkleovog stabla.

Može se odmah uočiti kako takva struktura rješava problem provjere ispravnosti podataka: Čvorovi su šifrirani pomoću kriptografske hash funkcije i to najčešće sa *SHA-256*. Ako se dogodi promjena u samo jednom bloku ulaznih podataka početna hash vrijednost odgovarajućeg lista više neće biti ista. Ta promjena će dovesti do promjene hash vrijednosti njegovog roditelja i tako dalje. Dolazi do kaskadnog efekta. Svi čvorovi iznad lista više neće imati istu hash vrijednost pa onda ni sam korijen Merkleovog stabla neće biti isti.

4.2 Primjer konstrukcije

Za jednostavan primjer konstrukcije korijena Merkleovog stabla mogu se uzeti imena gradova kao ulazni podatci:

$$\text{BLOK } 0 = \textit{"Zagreb"}$$
$$\text{BLOK } 1 = \textit{"Split"}$$
$$\text{BLOK } 2 = \textit{"Osijek"}$$
$$\text{BLOK } 3 = \textit{"Rijeka"}$$

Ako se izračunaju sve pripadajuće hash vrijednosti (zbrajanje hash je operacija dodavanje druge hash vrijednosti na kraj prve) pomoću SHA-256 funkcije:

$H(0) = H\,(\text{"Zagreb"}) =$
$ec0cf9e82c46ad97d04e16738f755921a7e9e83b6cc13673850c36d5d312fd85,$
$H(1) = H\,(\text{"Split"}) =$
$32afaa784333648025e24b162bece7474051bcaaa29e98e19922b400b4ceb04b,$
$H(2) = H\,(\text{"Osijek"}) =$
$af776334555d01b024244366766c2d48e7eed89dccf3d0301b70cabad30a0650,$
$H(3) = H\,(\text{"Rijeka"}) =$
$b8734fb05c3dc21300c940213d07d585c2083d8614b7361d259e9454fc098d4d,$
$H(H(0) + H(1)) =$
$32afaa784333648025e24b162bece7474051bcaaa29e98e19922b400b4ceb04b,$
$H(H(2) + H(3)) =$
$b8734fb05c3dc21300c940213d07d585c2083d8614b7361d259e9454fc098d4d,$
$H(H(H(0) + H(1)) + H(H(2) + H(3))) = H_{root}$
$715116125159a6787d65eb53b1d2ed48a471f2d4e0c038dc2cef4e80e0ba9237.$

Ako se umjesto „Zagreb” stavi „zagreb” i ponovno izračunaju sve pripadajuće hash vrijednosti:

$$H(0) = H\,(\text{”zagreb”}) =$$
$$4f6d3544216cc7e2562891eddd095c65c49cc9fcc90d4f564c5f0c5a0625dc19,$$
$$H(H(0) + H(1)) =$$
$$d308c49c9ea4b9442c6254255a1d641c74c353cfbd555ec5df5bef99ed23c371,$$
$$H(H(H(0) + H(1)) + H(H(2) + H(3))) = H_{root'}$$
$$b8734fb05c3dc21300c940213d07d585c2083d8614b7361d259e9454fc098d4d.$$

Ako se usporede dobivene hash vrijednosti korijena:

$$H_{root} = 715116125159a6787d65eb53b1d2ed48a471f2d4e0c038dc2cef4e80e0ba9237,$$
$$H_{root'} = b8734fb05c3dc21300c940213d07d585c2083d8614b7361d259e9454fc098d4d.$$

Za provjeru valjanosti se ne mora pretražiti svaki pojedinačni blok podataka već se samo usporede 256-bitni brojevi. Ako je očekivana vrijednost korijena Merkleovog stabla H_{root}, a izračunata je $H_{root'}$ korisnik može biti sigurni da ulazni blok podataka nije u potpunosti onakav kakvog je zatražio. Takva struktura omogućuje značajno smanjenje paketa podataka koje nekakav pouzdani, središnji izvor mora poslati. Sve što korisnik treba preuzeti sa servera su 256-bitni brojevi, a onda ostale konkretne datoteke može dohvatiti iz bilo kojeg nepouzdanog izvora. Sve dok se hash vrijednosti korijena, i čvorova, podudaraju može biti siguran da datoteke nisu bile mijenjanje od treće strane.

Osim brze provjere ispravnosti podataka može se i lagano pronaći koji je od ulaznih blokova promijenjen. Za dva različita korijena Merkleovih stabala potrebno je zatražiti hash vrijednost njihove djece. Usporedbom se utvrdi koji od dva čvora se razlikuje te se eliminira jedna cijela strana stabla. Postupak se ponavlja sve dok se ne dođe do neispravnog lista.

Kako se ne uspoređuje svaki čvor u stablu već samo dva para čvorova po jednoj razini ovakva operacija je vrlo brza. Za stablo s n čvorova pronalazak odgovarajućeg lista ovisi samo o visini stabla, odnosno obavlja se u $\mathcal{O}(\log_2(n))$ koraka, no takvo efikasno pretraživanje nije uvijek moguće. Ako se većina ili svi čvorovi razlikuju po-

trebno je obići svaku granu stabla pa je onda u najgorem slučaju složenost $\mathcal{O}(n)$. Postupak u kojem se traže i ispravljaju neispravni čvorovi se zove sinkronizacija stabla.

4.3 Osnovna implementacija

Ovisno o potrebama programa Merkleovo stablo može biti proizvoljne visine. Ovdje je prikazana konstrukcija stabla pomoću poznatog algoritma TREEHASH [13]. SHA-256 hash funkcija je preuzeta iz [12].

```
typedef struct {
    u32 Height;
    u256 Hash;
} leaf;

static void
GetMerkleRoot(void * Memory, u8 * Data, u32 DataSize, u32 MaxHeight) {
    u8 * Current = Data;
    u32 NumNodes = powf(2, MaxHeight + 1) - 1;
    u32 MaxLeafs = powf(2, MaxHeight);
    u32 LeafDataSize = DataSize / MaxLeafs;

    s32_stack Stack = CreateStack(MaxHeight + 1);
    leaf * Leafs    = Memory;
    u32 NumLeafs    = 0;

    for(;;)
    {
        s32 First  = Peek(&Stack, 0);
        s32 Second = Peek(&Stack, 1);
        bool32 Consolidated = 0;

        if((First != INVALID_VAL) && (Second != INVALID_VAL)) {
            leaf * Right = &Leafs[First];
            leaf * Left  = &Leafs[Second];

            if(Right->Height == Left->Height) {
                Pop(&Stack);
                Pop(&Stack);
```

```
30
31                  char String[128] = {0};
32                  CombineU256IntoString(Left->Hash, Right->Hash, String);
33
34                  leaf Parent = {0};
35                  Parent.Height = Left->Height + 1;
36                  Parent.Hash   = SHA256(String, 128);
37
38                  if(Parent.Height == MaxHeight) {
39                      PrintU256(Parent.Hash);
40                      break;
41                  }
42
43                  Push(&Stack, NumLeafs);
44                  Leafs[NumLeafs++] = Parent;
45
46                  Consolidated = 1;
47              }
48          }
49
50          if(!Consolidated) {
51              leaf NewLeaf   = {0};
52              NewLeaf.Hash   = SHA256(Current, LeafDataSize);
53              NewLeaf.Height = 0;
54
55              Push(&Stack, NumLeafs);
56              Leafs[NumLeafs++] = NewLeaf;
57              Current += LeafDataSize;
58          }
59      }
60      DeleteStack(&Stack);
61  }
```

Radi jednostavnosti je prikazan samo osnovni način kako napraviti stablo i on nije pogodan za ozbiljnije potrebe. Primjerice, za algoritam je u svakom trenutku potrebno ne više od *MaxHeight* + 1 spremljenih čvorova iako su u ovom primjeru spremljeni svi čvorovi. Također, zbog lakše provjere ispravnosti ključevi su šifrirani u obliku teksta, ali puno efikasniji način je šifrirati ih u binarnom obliku. Još jedan očiti nedostatak je to što se stablo može šifrirati samo za paran broj ulaznih blokova.

Svi problemi se mogu relativno lagano ispraviti, ali kako postoje kompleksniji algoritmi [13] koji šifriraju stablo u kraćem vremenu onda za tim nema potrebe u ovom jednostavnom primjeru.

Algoritam radi u dvije grane na sljedeći način:

- Ako su prva dva čvora u stoga na istoj visini oni se zajedno šifriraju kako bi se dobila vrijednost novog čvora, njihovog roditelja. Visina novog čvora se uveća za jedan u odnosnu na visinu njegove djece. Ako je novi čvor na maksimalnoj visini algoritam staje i dobivena je hash vrijednost korijena, a ako nije, čvor se stavlja na vrh stoga i algoritam se vraća na početak.
- Ako čvorovi nisu na istoj visini računa se hash vrijednost idućeg bloka ulaznih podataka. Dobivena vrijednost lista se sprema na vrh stoga i algoritam se vraća na početak.

Na slikama 4.2 i 4.3 se može vidjeti primjer kako će izgledati program nakon dvije, odnosno pet iteracija algoritma za ulazni niz podataka „ABCDEFGH" i zadanu maksimalnu visinu $h = 3$. Slika cijelog stabla se može vidjeti na 4.4.

0	1
H(A), 0	H(B), 0

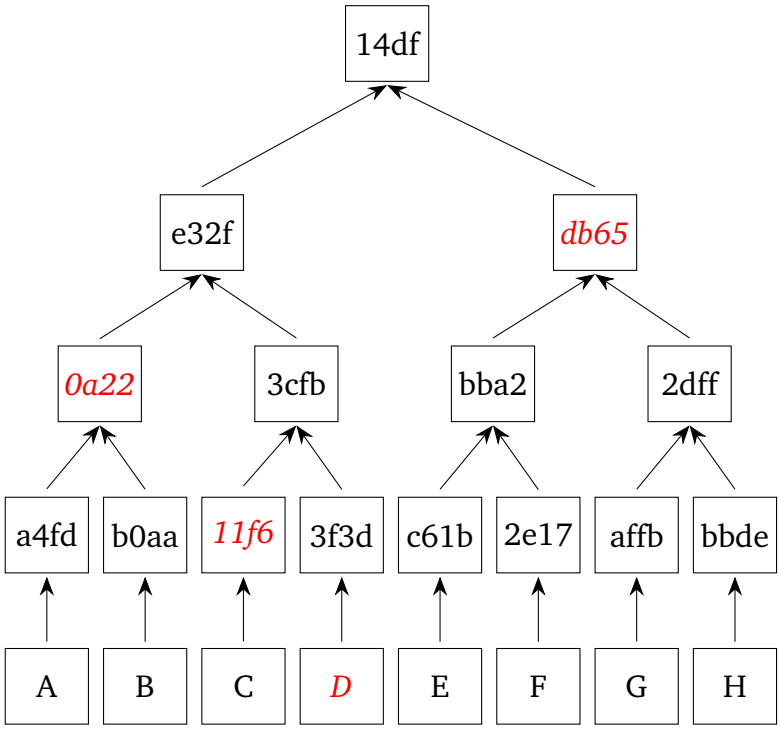
Slika 4.2: Slika prikazuje stog nakon dvije iteracije petlje. H(A) i H(B) predstavljaju hash vrijednosti prvog i drugog bloka, a broj predstavlja visinu čvora (lista). U ovom slučaju su oba čvora na visini 0 (gledajući od dna stabla).

0	1	2
H(H(AB)), 1	H(C), 0	H(D), 0

Slika 4.3: Slika prikazuje stog nakon pet iteracije petlje.

4.4 *Merkleov put*

Na slici 4.4 se vidi primjer Merkelovog stabla. Ako korisnik želi provjeriti je li „D" dio ulaznih podataka njemu su potrebni samo u kurzivu crvenom bojom označeni podatci. Korisnik ne treba računati hash vrijednost svakog čvora već može izračunati samo njemu bitne grane. Ako je dobio jednaku vrijednost korijena stabla može biti siguran da se traženi podatak nalazio u ulaznom bloku. U ovom primjeru korisnik bi morao izračunati samo četiri hash vrijednosti kako bi provjerio je li blok „D" uistinu dio ulaznih podataka. Takav minimalni skup čvorova, odnosno podataka, potrebnih za ponovno računanje korijena stabla se zove Merkleov put ili Merkleov dokaz. Mogućnost brze provjere prisustva nekih podataka u Merkleovom stablu je ono što omogućuje trgovcima pružanje plaćanja putem Bitcoina u stvarnom vremenu. Nešto više je o tome rečeno u idućem poglavlju.



Slika 4.4: Slika pokazuje Merkleovo stabla za ulazni niz podataka „ABCDEFGH" i visinu stabla $h = 3$. Radi kraćeg ispisa hash vrijednosti su veličine dva bajta.

5 Blockchain i Bitcoin

U svojoj osnovni blockchain je rješenje problema knjigovodstva: Kako bez direktne razmjene novca platiti za nekakvu uslugu ili stvar?

Takav problem je već riješen u svijetu bankovnih kartica i sličnih sustava. Problem koji blockchain rješava je kako provesti takva plaćanja bez potrebe za središnjim autoritetom (na primjer bankom). Blockchain uvodi sustav razmjene vrijednosti u kojem su sve transakcije javno dostupne na uvid, plaćanje je sigurno, a za plaćanje se ne mora tražiti dozvolu od treće strane. Omogućuje direktno plaćanje usluga bez čekanja potvrde banke, sve što je potrebno za plaćanje je dokaz o posjedovanju transakcija.

Reprezentacija blockchaina ovisi o specifičnim potrebama, odnosno o načinu izvedbe virtualne valute (kriptovalute), a sama ideja je objavljena 2008. u znanstvenom radu danas poznatom pod imenom „White paper" [14]. To je ujedno bila i prva izvedba blockchaina nazvana Bitcoin, a objavljena je godinu dana kasnije. Rad je objavio nepoznati autor, ili više autora, pod pseudonimom Satoshi Nakamoto, čiji identitet još uvijek nije točno potvrđen.

5.1 Virtualna valuta

Osnovni problem koji se odmah javlja ako se želi razmjenjivati novac elektroničnim putem je kako znati tko ima koliko novca, odnosno, gdje spremiti korisnički „račun" ako ne postoji središnji autoritet? Gdje novac živi?

Jedan način na koji se to može riješiti je da digitalna valuta postane knjiga transakcija. Korisnici nakon svake transakcije upisuju u svoje knjige detalje transakcije sa svojim potpisom, a te informacije o transakciji proslijede svim ostalim korisnicima. Na kraju određenog ciklusa (jednog dana/tjedna/mjeseca/itd.) svi korisnici zbroje svoje knjige te pošalju adekvatni iznos stvarnog novca drugim korisnicima. Ako su Marko, Ivan i Petar korisnici na takvoj mreži i obavljali su nekakve transakcije, knjiga bi otprilike mogla izgledati kao:

[0.] Marko plaća Petru $35\ \mathrm{kn}$ *Marko*

[1.] Ivan plaća Petru $50\ \mathrm{kn}$ *Ivan*

[2.] Petar plaća Marku $5\ \mathrm{kn}$ *Petar*

Očekuje se da na kraju ciklusa Marko Petru pošalje $30\ \mathrm{kn}$, a Ivan Petru pošalje $50\ \mathrm{kn}$.

Tako je riješen problem korisničkih računa, ali pojavilo se par novih problema:

- Ivan je platio Petru $50\ \mathrm{kn}$ digitalnim putem, ali kako će Petar biti siguran da će mu na kraju ciklusa knjige Ivan stvarno poslati taj novac?
- Svatko može upisati u svoju knjigu što želi, a kako je knjiga digitalna, što sprječava Petra u kopiranju drugog reda u kojem piše da mu Ivan plaća $50\ \mathrm{kn}$ i tako primi dvostruki iznos?
- Ako nastane više različitih knjiga, kako znati koja od primljenih knjiga je ispravna? Ako neki korisnici nisu bili prisutni za vrijeme transakcije kako će znati detalje transakcije koja se tada dogodila? Kako će znati koja se transakcija dogodila prije koje?

Prvi problem se lagano riješi pomoću jedne vrste početnog uloga *(engl. buy-in)*. Svi korisnici na početku mreže razmijene stvarni novac za željenu količinu digitalne valute (dv), a te se uplate zapišu na početak knjige:

[0.] Marko prima $50\ \mathrm{dv}$

[1.] Ivan prima $200\ \mathrm{dv}$

[2.] Petar prima $200\ \mathrm{dv}$

ako sad neki korisnik želi potrošiti određenu količinu sve što ostali korisnici trebaju provjeriti je lista transakcija. Ako zbroje sva primanja i davanja za nekog korisnika mogu se lagano uvjeriti ima li taj korisnik stvarno dovoljno preostalog novca.

Drugi problem kopiranja transakcija se također može relativno lagano riješiti

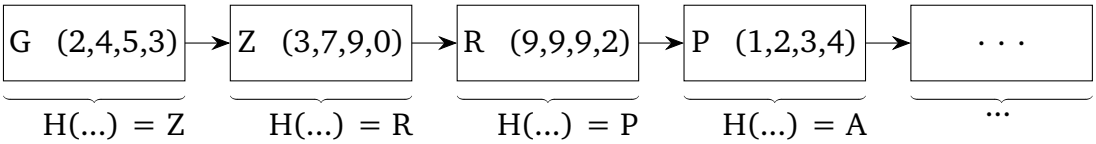
pomoću digitalnog potpisa. Digitalni potpis je ukratko objašnjen na kraju poglavlja.

Treći problem, problem sinkronizacije, je onaj najvažniji i najteži. I prije razvoja Bitcoina je postojala ideja o ravnopravnoj mreži u kojoj korisnici mogu direktno plaćati jedan drugome, ali nikako se nije uspio riješiti problem sinkronizacije. Upravo je rješenje tog problema ključna ideja blockchaina.

5.2 Osnovna struktura blockchaina

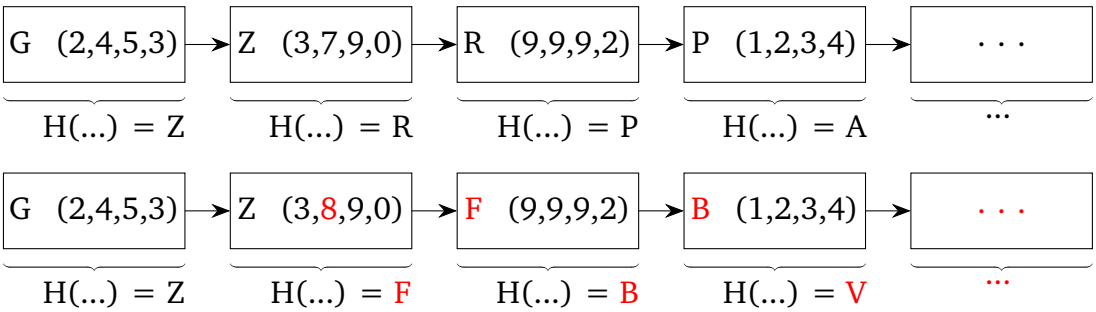
Već iz imena, ali i sa slike 5.1, je moguće naslutiti nešto o strukturi blockchaina. On je ništa drugo nego niz blokova podataka, a ti blokovi su na neki način povezani jedan za drugim, spojeni su u lanac.

Osnovna ideja kako spojiti blokove je da se svaki blok nekako pozove na prijašnji blok, a u slučaju blockchaina to je preko imena bloka. Svaki blok sadrži ime prijašnjeg bloka i za taj blok specifične podatke. Ime n-tog bloka se dobije tako što se zajedno šifriraju podatci unutar tog bloka i ime prijašnjeg, $n-1$, bloka.



Slika 5.1: Slika pokazuje jednostavan primjer blockchaina. Primjerice, drugi blok sadrži ime prijašnjeg bloka „Z" i nekakve svoje podatke $(3, 7, 9, 0)$. Te dvije informacije se mogu zajedno šifrirati kako bi se dobilo ime bloka „R". Zatim se ime bloka „R" koristi u stvaranju idućeg bloka, i tako dalje.

Blockchain omogućava svakom korisniku spremanje podataka u blokove, a povezani lanac takvih blokova osigurava zaštitu od neovlaštenih izmjena. Jednostavan primjer zaštite podataka se vidi na slici 5.1: nakon što je korisnik spremio podatke $(3, 7, 9, 0)$ u drugi blok on može biti sigurni da će ti podatci uvijek ostati takvi kakvi jesu. Ako netko želi retroaktivno izmijeniti njegove podatke u drugom bloku, on mora prepraviti i sve druge blokove kako bi se imena blokova podudarala. Ako se imena blokova ne podudaraju korisnik može biti sigurni da to više nisu isti blokovi. Kako su imena blokova rezultati kriptografskih hash funkcija takav zadatak je gotovo nemoguć. Jednostavan primjer se vidi na slici 5.2.



Slika 5.2: Slika pokazuje primjer dva blockchaina kad se u jednom od blokova malo promijene bloku specifični podatci.

5.3 Blockchain i problem sinkronizacije

S knjigom transakcija je formaliziran proces plaćanja, ali ono što nedostaje je način kako organizirati transakcije kako bi sve knjige bile jednake. Kako iskoristiti blockchain u rješenju problema sinkronizacije?

Mjesto blockchaina u rješenju problema je već očito. Kako se u blok unutar blockchaina mogu upisivati proizvoljni podatci u njega se mogu upisati i knjige transakcija. Ciklus knjiga je zamijenjen nizom povezanih blokova. Više nema prekida spisa transakcija jer jednom kad se knjiga popuni ona se spremi u blockchain. Kako su blokovi u blockchainu povezani sve obavljene transakcije su trajno i nepromjenljivo tamo zapisane, ukratko, stvorio se neprekidni lanac knjiga transakcija.

Za riješiti ostaje još samo središnji dio problema, što učiniti u slučaju ne poklapanja dvije knjige? Što učiniti ako se želi pospremiti knjiga u blockchain, ali knjige nekolicine korisnika se ne podudaraju s ostalim korisnicima? Ne mogu se odjednom pospremiti sve različite verzije knjige jedna za drugom jer bi tako došlo do umnožavanja gotovo svih transakcija, a ne mogu se ni izbaciti samo neke transakcije jer nije jasno kojoj knjizi vjerovati. Potrebno je uvesti mjeru koja će dati garanciju ispravnosti svake knjige.

Ime bloka unutar blockchaina nije proizvoljan niz znakova već je rezultat kriptografske hash funkcije, a u praski je to najčešće 256-bitni broj. Jedan od načina kako uvesti mjeru točnosti knjige je zadavanjem dodatnih uvjeta na ime bloka. Jedan od najjednostavnijih uvjeta koji je moguće zadati je da ime bloka mora započeti s određenim brojem nula. Primjerice, ako se očekuje ime bloka koje sadrži $n = 20$

vodećih nula može se vidjeti primjer jednog ispravnog i neispravnog imena bloka:

$$000000000000000000092d17ec38780f66586d165f5f145ebda428b00e25087,$$
$$004030eb0ee00d001aef0944492d17ec38780f66586d165f5f145ebda428b00e.$$

Takav uvjet se ne čini kao nešto korisno. Ako se u blok želi pospremiti zadana knjiga transakcija s imenom prijašnjeg bloka kako je moguće da će kriptografska hash funkcija izbaciti rezultat koji ima baš točan broj nula za ulazne podatke? Iako je rezultat kriptografske hash funkcije nasumičan on je uvijek isti za iste podatke, pa uz zadane transakcije i ime prijašnjeg bloka nikako se ne može garantirati da će hash vrijednost takvih podataka započeti s nekim zadanim nizom nula.

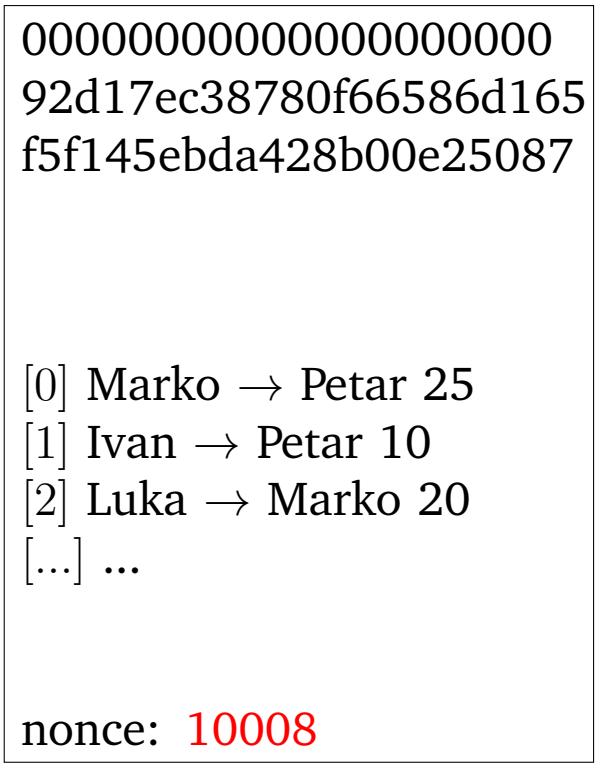
Rezultat kriptografske hash funkcije je nasumičan što dovodi do velikih promjena izlaznih vrijednosti za samo male promjene ulaznih podataka. Ako se popisu transakcija i imenu prijašnjeg bloka pridruži i brojač *(engl. nonce)*, može se bez promjene ulaznih podataka samo jednostavnim uvećavanjem brojača dobiti potpuno druga hash vrijednost bloka.

Sve što je potrebno napraviti za pronalazak odgovarajuće hash vrijednosti je uvećavati nonce za jedan dok se ne dođe do hash vrijednosti koja započinje s traženim brojem nula. Kako je raspodjela uniformna, svaki broj je jednako moguć, a očekuje se da će se to dogoditi za 2^N pokušaja, gdje je N broj traženih vodećih nula. Jedan takav primjer se može vidjeti na slici 5.3.

Traženje pravog broja koji „rješava" blok je zapravo nekakva vrsta kriptografske zagonetke. Traži se broj za koji će hash vrijednost bloka započeti s određenim brojem nula. Jednom kad se takav broj pronađe ime bloka će biti zadovoljavajuće pa se on može smjestiti unutar blockchaina.

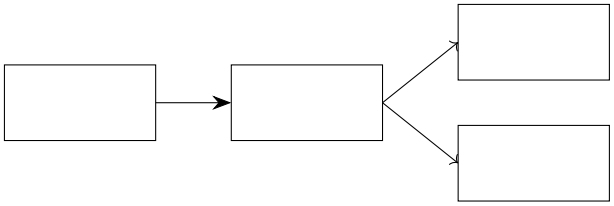
Upravo je taj postupak rješavanja kriptografske zagonetke ono što se zove rudarenje *(engl. mining)*. Kako nije moguće predvidjeti rezultat kriptografske hash funkcije ne postoji bolja metoda traženja od traženja na silu *(engl. brute-force)*. Inverzni postupak je trivijalan: Kako bi se provjeri neki broj sve što je potrebno je izračunati hash vrijednost bloka i ako ona započinje s traženim brojen vodećih nula onda je to pravi broj. Takav broj se zove dokaz o obavljenom radu *(engl. proof of work)*.

Svaki blok mora započeti s određenim brojem nula, no i dalje nije jasno kako to iskoristiti za rješenje ne slaganja dvije knjige transakcije. Na slici 5.4 se može uočiti



Slika 5.3: Slika pokazuje neki blok unutar blockchaina koji sadrži ime prijašnjeg bloka, knjigu transakcija i dodatni broj. Kad je $nonce = 10008$ kriptografska hash vrijednost bloka je broj koji započinje s 20 nula.

kako su nakon dva složena bloka došle dvije knjige koje se razlikuju. Netko je u jednu od knjigu upisao lažne transakcije. Kako bi se riješilo ne slaganje na mjestu novog bloka su stvorene dvije grane, jedna s pravom knjigom i jedna s lažnom ili neispravnom.

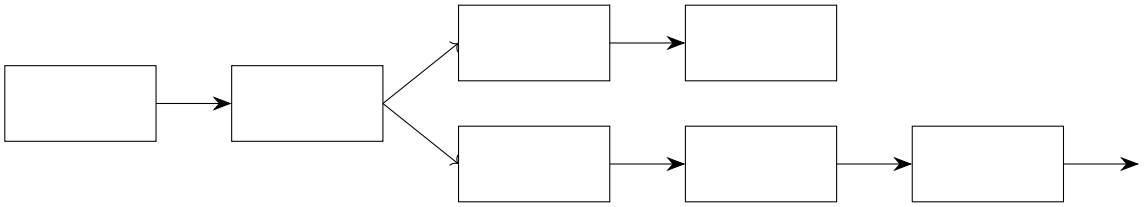


Slika 5.4: Slika pokazuje blockchain u kojem se dogodilo grananje.

Kako stalno pristižu nove transakcije tako stalno nastaju novi blokovi. Osnovna ideja kako riješiti ne slaganje je da se jednostavno vjeruje lancu knjiga (blokova) nad kojim je obavljeno najviše računskog rada, odnosno, da se nove blokove uvijek dodaje na kraj najduljeg lanca.

Ako je i samo jedna transakcija u samo jednom bloku lažirana, onaj tko je napravio takvu lažnu transakciju mora moći dovoljno brzo praviti nove blokove kako bi njegov lanac bio što dulji, odnosno onaj u koji dolaze nove transakcije. Ako je

na mreži dovoljno dobronamjernih korisnika, više od 50% računalne moći, lako se uvjeriti da će bilo kakvo lažiranje biti gotovo nemoguće. Blokovi napravljeni od dobronamjernih korisnika će u konačnici stvoriti dulji lanac, a time će svi loši blokovi otići u zaborav. Primjer se vidi na slici 5.5.



Slika 5.5: Slika pokazuje blockchain u kojem se dogodilo grananje nakon nekoliko dodatnih blokova.

Ovakav pristup nije jedini način na koji se može riješiti problem sinkronizacije, ili se nekad kaže konsenzusa, a jedan od popularnijih načina je princip zasnovan na dokazu o ulogu *(engl. proof of stake)*. Njegova osnovna ideja je da korisnici koji su zaduženi za provjeru ispravnosti transakcija moraju i sami posjedovati određenu količinu digitalnog novca. Dokaz o ulogu je znatno energetski povoljnija metoda od dokaza o radu, ali svi detalji još uvijek nisu do kraja razrađeni kako bi se neke od najpoznatijih kriptovaluta prebacile na takav model. Trenutno vodeća, od popularnijih, kriptovaluta za koju se očekuje da će implementirati ovakav model je Ethereum (Ethereum 2.0).

5.4 Primjer blokova unutar Bitcoin blockchaina

Na web stranici `https://www.blockchain.com` se mogu pregledati do sad svi složeni blokovi za razne kriptovalute. Primjerice, može se pogledati kako izgleda 100001. blok (danas ih ima nešto manje od 700000) unutar Bitcoin blockchaina `https://www.blockchain.com/btc/block/100000`:

Hash	000000000003ba27aa200b1cecaad478d2b00432346c3f1f3986da1afd33e506
Confirmations	596,364
Timestamp	2010-12-29 12:57
Height	100000
Miner	Unknown
Number of Transactions	4
Difficulty	14,484.16
Merkle root	f3e94742aca4b5ef85488dc37c06c3282295ffec960994b2c0d5ac2a25a95766
Version	0x1
Bits	453,281,356
Weight	3,828 WU
Size	957 bytes
Nonce	274,148,111
Transaction Volume	53.01000000 BTC
Block Reward	50.00000000 BTC
Fee Reward	0.00000000 BTC

Slika 5.6: Primjer jednog stvarnog bloka (na visi 100000) unutar Bitcoin blockchaina.

Može se primijetiti kako hash vrijednost bloka:

$$000000000003ba27aa200b1cecaad478d2b00432346c3f1f3986da1afd33e506$$

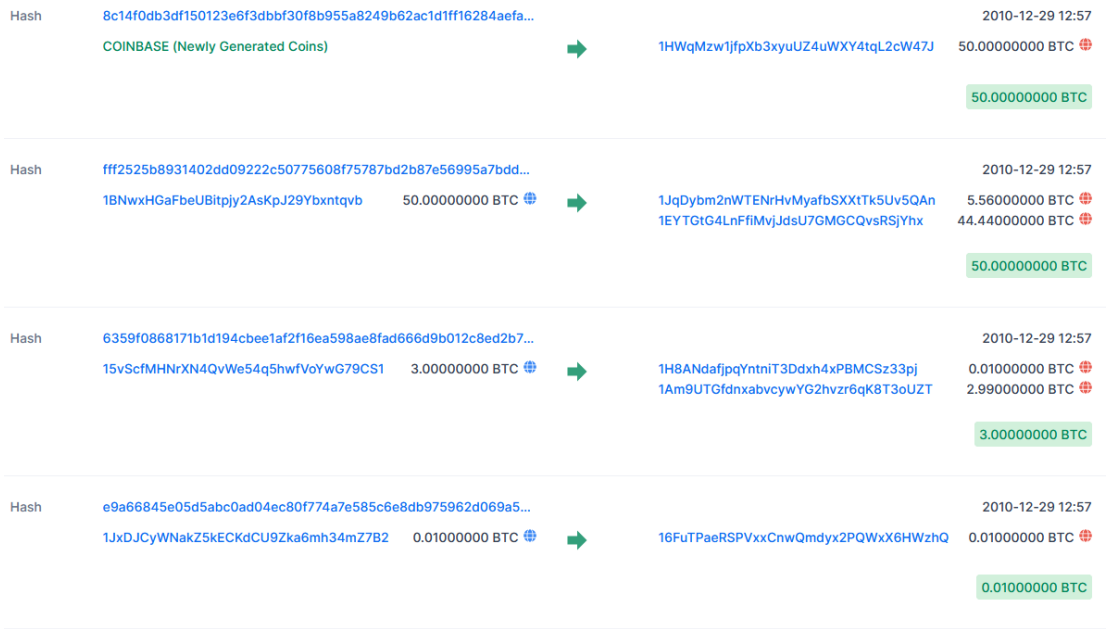
započinje sa samo 11 nula. U vrijeme kad je taj blok nastao (2010. godine) Bitcoin još uvijek nije imao globalnu popularnost. Broj korisnika koji su se bavili rudarenjem je bio znatno manji pa je i težina pronalaska pravog broja koji rješava blok bila znatno lakša. Jedna od ideja je da broj složenih blokova u nekoj jedinici vremena bude otprilike konstantan pa ako je na mreži više korisnika koji se bave rudarenjem trebat će im i teža zagonetka. Težina stvaranja novih blokova u Bitcoin blockchainu je takva da se novi blok stvara otprilike svakih desetak minuta.

Uz hash vrijednost bloka mogu se vidjeti još neke zanimljive vrijednosti. Primje-

rice, nonce iznosi 274148111, ako se to usporedi s očekivanom vrijednošću $2^{11} = 2048$ može se primijetiti kako korisnik koji je složio taj blok nije imao puno sreće.

Vidi se i još jedna zanimljiva vrijednost, a to je nagrada za stvaranje bloka *(engl. block reward)* koja iznosi 50 BTC (možda taj korisnik i nije toliko nesretan jer današnja vrijednost 50 BTC iznosi preko 14 milijuna kuna). Nagrada je osnovni način kako nastaju novi Bitcoini, ali i poticaj na sudjelovanje u rješavanju kriptografske zagonetke. Ako ne bilo nikakve nagrade za mukotrpno računanje i traženje pravog broja nitko ne bi ni sudjelovao u takvoj mreži. Kao i stvarni novac Bitcoin je kvantiziran, najmanja jedinica se zove Satoshi i iznosi $0,00000001$ BTC.

Mogu se pogledati i sve transakcije unutar bloka:



Slika 5.7: Popis svih transakcija unutar 100001. bloka za Bitcoin blockchain.

Uočava se vrlo slična struktura kao i u primjeru prve knjige transakcija. Na početku popisa transakcija je stvoreno novih 50 BTC (kao nagrada za slaganje bloka), što je slično redu „Marko prima 50 dv". Jedna bitna razlika je u tome što nije eksplicitno napisano ime korisnika koji razmjenjuju novac nego su ona zamijenjena s adresama Bitcoin novčanika *(engl. Bitcoin wallet)*.

Blok unutar blockchaina se sastoji od dva dijela: zaglavlja bloka i popisa transakcija. U zaglavlju bloka se nalaze meta informacije o bloku: verzija bloka, nagrada za stvaranje bloka, vremenska oznaka *(engl. time stamp)* kad je složen, hash vrijednost prijašnjeg bloka i tako dalje. U zaglavlju se nalazi i još jedna zanimljiva vrijednost, a to je hash vrijednost Merkleovog stabla transakcija. Upravo je ona jedna od ključnih

ideja koja je omogućila postojanje blockchaina u današnjem obliku. Bez postojanja Merkleovih stabala rukovanje sa svim podatcima o transakcijama bi bilo iznimno nepraktično.

5.5 Čvorovi i osnovne transakcije

Kad korisnik želi napraviti transakciju ta transakcija prvo mora biti pospremljena u blok, a taj blok se mora nadovezati na blockchain. Uz to, sažetak bloka mora sadržavati određeni broj vodećih nula.

Kako takav proces sadržava previše koraka koje bi svaki korisnik morao provesti prije nego što bi mogao napraviti i najjednostavnije uplate Bitcoina, mreža organizira korisnike u nekoliko grupa ili tipova čvorova. Dvije osnovne vrste čvorova su oni korisnici koji samo primaju ili plaćaju pomoću Bitcoina i korisnici koji se bave rudarenjem.

Prva grupa korisnika zahtjeva samo osnovni Bitcoin novčanik, odnosno, neki jednostavan način na koji korisnici vode brigu o tome s koliko Bitcoina raspolažu. Bitcoin novčanik je ukratko objašnjen u idućem potpoglavlju, ali jedna iznenađujuća činjenica je da niti jedan korisnik zapravo ne posjeduje Bitcoin novčiće, nego samo dokaz o mogućnosti trošenja istih.

Druga grupa korisnika su oni koji se natječu u što bržem rješavanju kriptografske zagonetke kako bi zaradili nagradu za stvaranje bloka. Njihov posao je slušanje transakcija korisnika prve grupe, skupljanje tih transakcija u popis transakcija te traženju čarobne vrijednosti brojača koji rješava blok.

Često su korisnici u ovakvim grupama udruženi u zajednički bazen *(engl. mining pool)* pa je tako primjerice kineska skupina „F2pool" zaslužna za čak 16% svih složenih blokova [15]. Takvi veliki bazeni su uspjeli složiti čak sedam blokova za redom pa se prodavačima preporučuje čekanje barem sedam blokova prije nego što se usluga isporuči.

Kao i težina zagonetke nagrada za stvaranje bloka se mijenja kroz vrijeme. Svakih 210000 blokova nagrada se prepolovi, a danas iznosi $12,5\ \mathrm{BTC}$. Uz direktnu nagradu za stvaranje bloka korisnici druge grupe mogu dobiti i dodatnu nagradu u obliku provizije za svaku transakciju.

5.6 Bitcoin novčanik i stvarne transakcije

Može se primijetiti kako je Bitcoin sastavljen od dvije jasne cjeline: Blockchain tehnologije i kriptografske tehnologije. Kriptografski dio je ostvaren upravo pomoću Bitcoin novčanika.

Bitcoin novčanik je najčešće računalni program pomoću kojeg korisnici mogu razmjenjivati Bitcoine. Postoje brojne izvedbe, a neke od poznatijih su: Bitcoin Core, Electrum, BitPay i drugi. Za razliku od klasičnog novčanika u koji se direktno sprema novac u Bitcoin novčanik se ne sprema nikakva valuta. On prvenstveno služi za stvaranje adresa pomoću kojih se mogu manipulirati transakcije. Iako svaki Bitcoin novčanik pokazuje količinu Bitcoina koju neki korisnik posjeduje to je samo pojednostavljeni prikaz.

Bitcoin, u smislu novca ili valute, se ne može nigdje spremiti jer je on ništa drugo nego popis transakcija složenih u blokove. Ono što Bitcoin novčanik sprema je popis svih transakcija čija adresa isporuke odgovara adresi novčanika. Ako Bitcoin novčanik pokazuje stanje računa od $5\ \mathrm{BTC}$ to zapravo znači da unutar blockchaina postoji jedna ili više transakcija čija je odredišna adresa upravo adresa tog novčanika i kad se zbroje sve te transakcije u njima je na adresu poslano točno $5\ \mathrm{BTC}$.

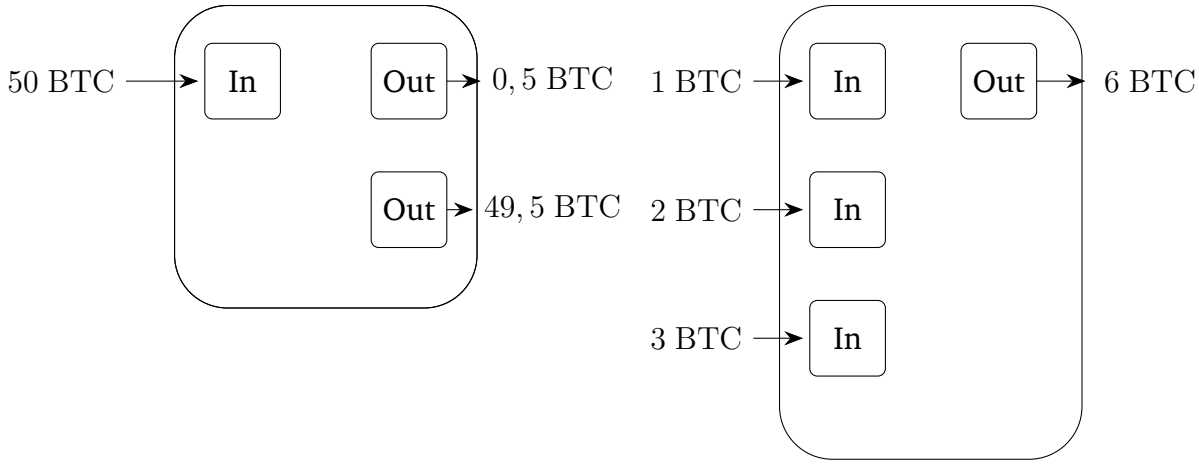
Prije nego je moguće iskoristiti određeni iznos potrebno je dokazati vlasništvo transakcije. Slično kao i kod digitalnog potpisa to se čini pomoću privatnog ključa. Takav privatni ili skriveni ključ je stvoren s adresom novčanika. Kako su adrese novčanika rezultati kriptografske hash funkcije *SHA-256* svaki novčanik može sadržavati praktično neograničeni broj adresa.

Jedno zanimljivo svojstvo Bitcoina koje dolazi od činjenice da se u blockchain ne sprema novac u klasičnom smislu je razdjeljivanje valute na manje jedinice. Ako korisnik posjeduje transakciju u vrijednosti $1\ \mathrm{BTC}$, a želi potrošiti $0,1\ \mathrm{BTC}$ on ne može iskoristiti samo dio te transakcije, ona se ne može dijeliti na manje dijelove. Transakcija se mora iskoristiti u cijelosti, no to ne znači da će korisnik izgubiti $0,9\ \mathrm{BTC}$. Korisnik može iskoristiti svoju transakciju od $1\ \mathrm{BTC}$ kako bi napravio dvije nove: jednu transakciju na adresu korisnika kojem želi poslati $0,1\ \mathrm{BTC}$, a ostalih $0,9\ \mathrm{BTC}$ može poslati na svoju vlastitu adresu.

Korisnik je tako iskoristio svoju originalnu transakciju od $1\ \mathrm{BTC}$ i izgubio pravo na njeno daljnje korištenje. Sad posjedujemo pravo na trošenje nove transakcije u vrijednosti $0,9\ \mathrm{BTC}$. Svi dokazi o posjedovanju transakcija su također provjerni od

strane korisnika koji te transakcije spremaju u blockchain.

Transakcije su podijeljene na dvije jasne strane: ulazne *(engl. transaction inputs)* i izlazne *(engl. transaction ouputs)* transakcije. Na ulaz se upisuju imena, odnosno adrese, svih transakcija koje se koriste, a na izlaz se zapisuju sve adrese novčanika i način na koji se želi raspodijeliti ulazne transakcije. Primjer se vidi na slici 5.8.



Slika 5.8: Primjer ulaznih i izlaznih transakcija unutar nekog Bitcoin bloka.

5.7 Blockchain: Merkleovo stablo

Na slici 5.6 se može vidjeti kako je unutar blok zapisana i njegova veličina koja iznosi nešto manje od $1\ \mathrm{KB}$, no danas su blokovi i do tisuću puta veći. Iako se to ne čini kao značajan broj ($\approx 1\ \mathrm{MB}$) zbog velikog broja blokova ukupna veličina Bitcoin blockchaina iznosi otprilike $350\ \mathrm{GB}$.

Ako trgovac želi svojim kupcima omogućiti plaćanje putem Bitcoina on bi morao preuzeti cijeli blockchain na svoje servere kako bi mogao samostalno potvrditi svaku transakciju. On ne može znati koliko duboko u blockchainu se nalazi neka transakcija pa mora imati sve blokove brzo dostupne.

Kako preuzimanje cijelog blockchaina često i nije najbolje rješenje uz dvije nabrojane vrste čvorova unutar Bitcoin mreže postoje i lagani čvorovi *(engl. light nodes)*. Lagani čvorovi ne preuzimaju cijeli blockchain već samo zaglavlje blokova. Veličina zaglavlja iznosi otprilike $80\ \mathrm{bajtova}$, a kako trenutno postoji otprilike 700000 blokova ukupna veličina zaglavlja iznosi $80\ \mathrm{bajtova} \times 700000 = 56\ \mathrm{MB}$, što je značajno manje od $350\ \mathrm{GB}$.

Jednom kad prodavač posjeduje zaglavlja svih blokova on može u direktnoj komunikaciji s korisnicima, Bitcoin novčanicima, riješiti transakciju. Prodavač više ne

mora čekati rudare za potvrdu i prihvaćanje transakcije. Kako više nema čekanje za provjeru transakcija on svoje usluga može obavljati stvarnom vremenu. Trgovac na takav način prihvaća mali rizik, jer više ne čeka da se složi dovoljno blokova u nizu niti da dovoljan broj drugih korisnika potvrdi transakciju, za male transakcije kako bi poboljšao kvalitetu usluge (naravno za veće transakcije je bitnija sigurnost isplate). Upravo takav princip jednostavnih provjera ispravnosti transakcija je moguć zahvaljujući Merkleovom stablu.

Način na koji lagani čvorovi provjeravaju transakcije je identičan onom opisanom u potpoglavlju o Merkleovom putu. Lagani čvor zatraži transakciju i sve druge njemu potrebne hash vrijednosti (taj broj je značajno manji od ukupnog broja čvorova). Zatim izračuna sve pridružene grane Merkleovog stabla te na kraju i sam korijen stabla. Ako se korijeni poklapaju može biti siguran kako je priložena transakcija stvarno unutar očekivanog bloka.

Iz navedenog se vidi kako Merkleovo stablo ima dvostruku korist unutar blockchaina: osigurava transakcije od neovlaštenih izmjena i omogućuje sažimanje transakcija radi brze provjere.

5.8 Kritike na blockchain tehnologiju i dokaz rada

Prema podatcima sa Sveučilišta u Cambridgeu [17] godišnja iskorištena energije na održavanje Bitcoin mreže iznosi otprilike $92\ \mathrm{TWh}$. Većina te energije odlazi na rješavanje kriptografske zagonetke, odnosno, na slaganje blokova. Ako se taj broj usporedi s godišnjim potrebama energije Republike Hrvatske [18] koji otprilike iznosi $18\ \mathrm{TWh}$ može se primijetiti kako se u jednoj godini održavanja Bitcoin mreže iskoristi više energije nego što to mogu iskoristiti četiri milijuna stanovnika u pet godina.

Kako se za računanje dokaza rada ponavlja jedan te isti postupak, odnosno ne obavlja se nikakav koristan rad, sva ta energije se efektivno pretvara samo u toplinu. Kad bi model konsenzusa bio drugačiji velika količina energija bi se mogla bolje iskoristiti. Primjerice, kad bi se umjesto traženja pravih hash vrijednosti kao dokaz koristili rezultati simulacija savijanja bjelančevina ili se dokazivale matematičke tvrdnje ili se tražili rezultati raznih simulacija iz područja astronomije ili elementarnih čestica možda bi se mogao uočiti brži napredak tehnologije.

Trenutna tržišna kapitalizacija Bitcoina iznosi otprilike $1,2$ bilijuna američkih dolara, a broj Bitcoina je nešto manji od 19 milijuna. Bitcoin je započeo kao način plaćanja, ali danas se on prvenstveno koristi kao način ulaganja novca, poput dionica ili sirovina. Zbog svoje velike popularnost na financijskom tržištu je podložan čestim promjenama tržišne cijene što ga čini neprikladnim za svakodnevno korištenje u smislu novca, odnosno valute.

Iako su danas kriptovalute sveprisutne njihova rasprostranjenost je znatno manja u odnosnu na klasične metode plaćanja. Prema podatcima tvrtke Crypto.com [19] ukupan broj korisnika kriptovaluta iznosi otprilike 106 milijuna. Taj broj nije zanemariv, ali je i dalje dovoljno nizak kako bi kriptovalutama omogućio određenu slobodu. Zbog relativno malog broja aktivnih korisnika kriptovalute trenutno nisu pretjerano zanimljive zakonodavnoj vlasti. Jednom kad broj korisnika postane dovoljno velik nije nemoguće zamisliti situaciju u kojoj će vlade kriptovalutama nametnuti razne lokalne regulacije.

5.9 Dodatak: Digitalni potpis

Digitalni potpis se na prvu ruku čini kao apsurdna ideja. Kako god se netko na računalu potpisao to je i dalje samo niz nula i jedinica, niz koji bilo tko drugi može vrlo jednostavno kopirati i zalijepiti. No zapravo je digitalni potpis značajno bolja verzija od klasičnog, ručnog potpisa. On se temeljni na metodama asimetrične kriptografije. Metoda je zasnovana na šifriranju i verifikaciji poruka pomoću para ključeva, jedan privatni i jedan javni ključ. Osnovni problem koji rješava digitalni potpis je dokazivanje identiteta; Kako prilikom slanja nekakvih dokumenata ili podataka druga strana može potvrditi da smo to uistinu mi.

U pojednostavljenoj verziji [16], ako Ivan želi uvjeriti Marka da je on stvarno taj koji mu je poslao poruku $m =$ „Dobar dan Marko.” sve što on treba je šifrirati tu poruku (obično sa SHA256) sa svojim tajnim ključem sk tako da dobije:

$$\sigma \leftarrow \mathrm{Sign}_{sk}\left(m\right).$$

Kad Marko na svojoj strani primi potpis, i prije svega poruku, sve što on treba napraviti je ubaciti dobiveni potpis s Ivanovim javnim ključem u algoritam za verifi-

kaciju:

$$\mathrm{Vrfy}_{pk}\,(m, \sigma) = 1 \text{ ili } 0.$$

Algoritam za verifikaciju je moguć jer se temelji na činjenici da su javni i privatni ključ na neki način povezani. Nisu samo dva nasumična broja, ali su takvi da je vezu između njih, u praksi, nemoguće pronaći. Takva dva broja, odnosno ključa, su povezani pomoću eliptične krivulje:

$$y^2 = x^3 + ax + b.$$

Skup točaka krivulje zajedno sa specifično zadanom operacijom zbrajanja čini grupu, a grupa je i komutativna. Stoga, za točku na krivulji g i $g + g + ... + g = n \cdot g$ je na krivulji. Za dovoljno velike krivulje i zadanu točku $g' = n \cdot g$ u praksi je nemoguće iz samo točke g' odredite broj n u razumnom vremenu. Takav broj n se zove privatni ključ.

Može se vidjeti otkud dolazi tvrdnja da je digitalni potpis značajno bolji od klasičnog: Za svaku jedinstvenu poruku digitalni potpis će izgledati drugačije. Nemoguće ga je kopirati s jedne poruke na drugu.

6 Merkle-Patricia stablo - Ethereum

Blockchain se može na apstraktni, ili filozofski, način promatrati kao jedinstvenim izvorom istine (u kontekstu zatvorenog okruženja). Informacije spremljene unutar blockchaina su one koje su točne i istinite. Ako je neka transakcija odobrena i zapisana na blockchain onda je ona nepovratno izvršena, nemoguće ju je više obrisati. Sve što je u blockchain zapisano se nepobitno dogodilo. Upravo je to jedna od temeljnih zamisli Ethereuma.

Poput Bitcoina, baza Ethereuma je blockchain tehnologija, ali on ne staje samo na jednostavnim transakcijama već pravi konceptualni korak iznad Bitcoina. Ethereum nije samo kriptovaluta već cijeli decentralizirani ekosustav. Ethereum blockchain služi kao računalna platforma *(engl. computing platform)* i izvor nedvojbenih informacija.

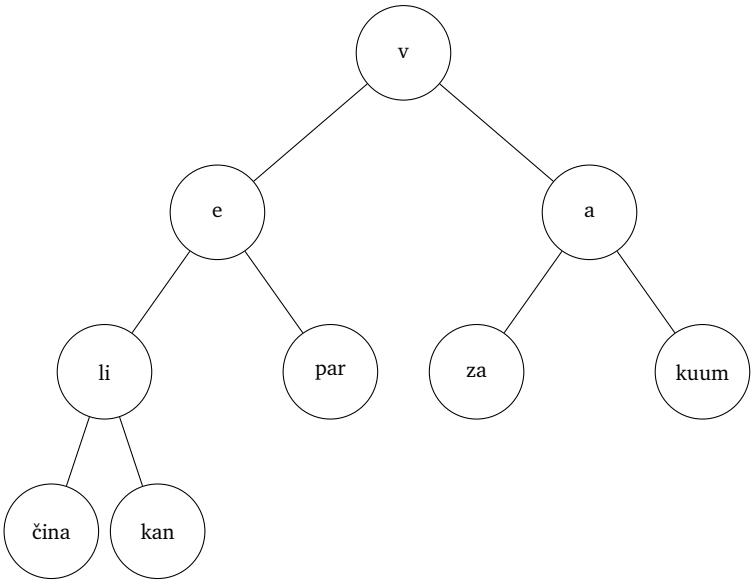
Jedna od osnovnih jedinica Ethereuma su programi, ili pametni ugovori *(engl. smart contracts)*, koje decentralizirana mreža izvršava. Za razliku od Bitcoina, jezik (Solidity) koji se koristi za pisanje programa je Turing potpun što omogućuje stvaranje puno zanimljivijih aplikacija.

Knjiga transakcija je bila korisna analogija za opis Bitcoin blockchaina, ali kako je Ethereum nadogradio, ili iskoristio u većem kapacitetu, ideju blockchaina takva analogije ovdje nije primjenjiva. Ethereum blockchain se može zamisliti kao decentralizirani konačni automat. Stanje automata je veliko Merkle-Patricia stablo korisničkih računa, programa i ostalih podataka.

6.1 Merkle-Patricia stablo

PATRICIA je akronim za algoritam koji omogućuje efikasno spremanje i dohvaćanje nizova znakova, ili pojednostavljeno nula i jedinica, za proizvoljno velike datoteke [20]. Pozadinska struktura je vrsta kompaktnog prefiksnog stabla za koje je $radix >= 2$. Način na koji je stablo organizirano omogućuje brz pronalazak podnizova čije vrijeme izvršavanja ovisi samo o duljini podniza, a neovisno je o duljini teksta (niza). Jednostavan primjer se može vidjeti na slici 6.1. Stablo je kompaktno jer su čvorovi koji bi imali samo jedno dijete udruženi, sa svojim djetetom, u jedan čvor.

Kako na Ethereum mreži postoji koncept korisničkih računa onda je za fleksibilno spremanje podataka potrebna nešto kompliciranija struktura od običnih binar-



Slika 6.1: Primjer kompaktnog prefiksnog stabla za $r = 2$. Sve zapisane riječi: vakuum, vaza, vepar, velikan, veličina.

nih hash stabala. Ako se izmjeni neka grana stabla, odnosno promjene se podatci u korisničkom računu, ne bi bilo efikasno ponovno računati hash cijelog stabla. Za rješavanje tog problema se upravo koriste Merkle-Patricia stabla. Ta struktura spaja dvije osnovne ideje: prefiksno stablo kao strukturu koja omogućuje lokalne izmjene podataka i brzo traženje podnizova i ideju šifriranja cijelog stabla kao zaštitu protiv neovlaštenih izmjena.

Zaglavlje bloka unutar Ethereum blockchaina sadrži tri Merkle-Patricia korijena: korijen stanja, korijen transakcija i korijen računa (u smislu potvrda) [21]. Nakon što je blok dodan na blockchain zadnja dva korijena, odnosno podatci u njima, se nikad ne mijenjaju. Kako prvi korijen sadrži korisničke račune i pametne ugovore on se konstantno mijenja.

Pojednostavljeno, Merkle-Patricia stablo je proširena ideja hash tablice. Ono omogućuje preslikavanje ključeva u hash vrijednosti uz povećanu sigurnost i blisko grupiranje sličnih vrijednosti. Stabla se grade pomoću deterministčkog algoritma pa će svako Merkle-Patricia stablo za zadane parove ključ-vrijednost imati isti oblik.

7 Zaključak

U radu je prikazana struktura podataka zvana Merkleovo stablo. Ona omogućuje brzu verifikaciju podataka, a svoju najveću primjenu je našla u sustavima građenim na ravnopravnoj mreži *(engl. peer-to-peer network)*. Proces sigurnog preuzimanja datoteka iz nepoznatih izvora je znatno olakšan upravo zahvaljujući takvom hash stablu. Kod preuzimanja datoteka iz nepouzdanih izvora korisnici mogu potvrditi njihovu valjanost samo usporedbom korijena Merkleovog stabla. Ako se korijeni stabala podudaraju korisnik može biti siguran da su datoteke onakve kakve je zatražio.

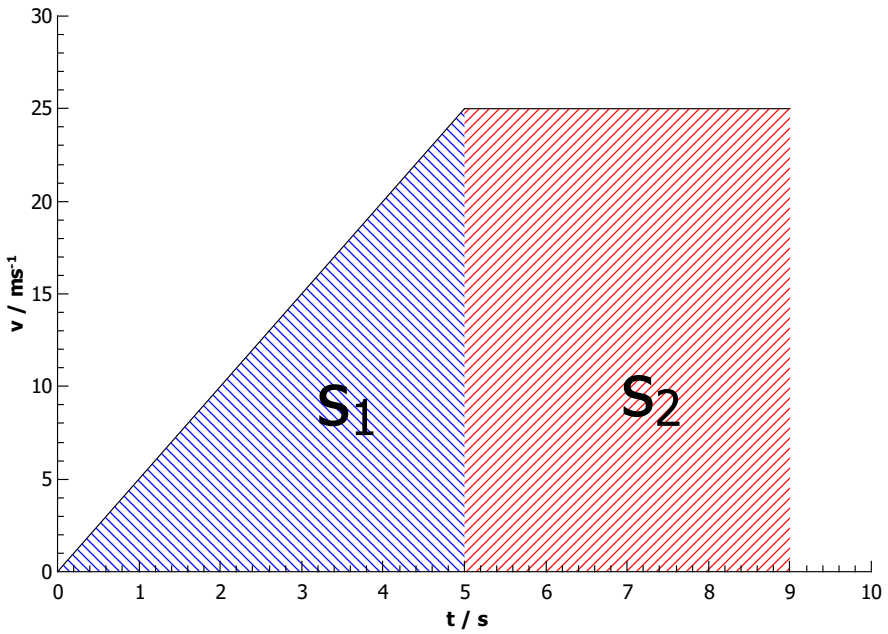
Drugu primjenu Merkleovo stablo je našlo u blockchain tehnologiji. Ono je središnja struktura koja osigurava zaštitu od neovlaštenih izmjena, a unutar Bitcoin mreže omogućuje laganim čvorovima mogućnost verifikacija transakcija. Način na koje je hash stablo organizirano dopušta sigurno šifriranje i spremanje transakcija unutar blockchaina. Kad se korijen Merkleovog stabla transakcija spremi unutar zaglavlja bloka sve transakcije prikupljene u bloku su trajno i nepromjenjivo zapisane u blockchain. Bilo koje retroaktivno mijenjanje transakcije zahtjeva nemoguće velike računske napore.

U radu je uz Merkleovo stablo opisana i ideja blockchaina, odnosno jedna specifična, i prva, izvedba u obliku Bitcoina. Organizacija mreže u kojoj korisnici mogu razmjenjivati novac na siguran i transparentan način nije bila moguća sve do pojave ideje blockchaina. Blockchain je sustav u koji korisnici mogu dodavati *(engl. append)* podatke, ali jednom spremljeni podatci se više ne mogu izmijeniti. Takav način organizacije podataka omogućen je zahvaljujući konsenzusu između svih korisnika mreže. Konsenzus se postiže tako što se od korisnika, od onih koji žele dodavati nove blokove u blockchain, traži rješavanje teške kriptografske zagonetke. Kad korisnik riješi zagonetku i priloži *nonce* za koji hash vrijednost bloka sadrži određen broj vodećih nula on dobiva nagradu, a novonastali blok se sprema u blockchain. Kako se rješavanje zagonetke svodi na pogađanje niti jedan individualni korisnik ne može dodati novi blok, odnosno niz blokova, u blockchain s kojim se ostali korisnici ne slažu. Za takvu manipulaciju korisniku je potrebno barem 51% ukupne računalne moći mreže.

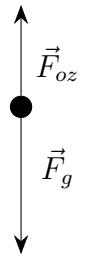
8 Metodički dio

U ovom dijelu rada je predstavljen članak o energiji i stupčastim dijagramima autora Alan Van Heuvelena i Xuele Zou [22]. Ukratko je opisano kako i zašto se stupčasti dijagrami mogu, i trebaju, koristiti u nastavi fizike te je kroz par primjera klasičnih zadataka prikazana njihova konstrukcija.

Uspješno rješavanje problemskih zadataka iz fizike se vrlo rijetko svodi na direktno prevođenje tekstualno zadanog zadatka u apstraktni jezik matematike. Za razumijevanje problema često je potrebno fizičke procese prikazati na više od jednog načina. Reprezentacija problema pomoću skica, dijagrama, grafova ili drugih kvalitativnih prikaza olakšava proces dolaska do točnog, kvantitativnog rješenja. Tipični primjeri takvih prikaza se mogu vidjeti na slikama 8.1 i 8.2.



Slika 8.1: Slika pokazuje primjer v-t grafa iz područja kinematike.



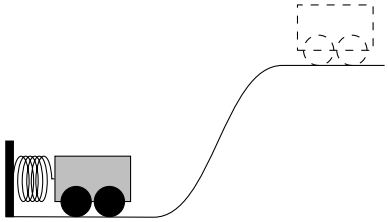
Slika 8.2: Slika pokazuje primjer dijagrama sila iz područja dinamike.

Prikazi sila pomoću dijagrama kao na slici 8.2 su intuitivni, no za apstraktnije kon-

cepte poput energije nije jednostavno osmisliti dobar način prikaza fizikalne veličine koji će pomoći razvoju dubljeg razumijevanja. Jedna ideja koja se pokazala jako uspješnom u tom zadataku su upravo stupčasti dijagrami.

Otkriveno je kako stupčasti dijagrami energije pomažu u boljoj analizi fizičkih procesa koji su povezani s radom i energijom. Način na koji su vizualno prikazani omogućuje lakše pretvaranja riječi u matematičke simbole. Iz takvih dijagrama je lakše iščitati temeljni zakon očuvanja energije. Oni služe kao jedna vrsta mosta između apstraktnog svijeta riječi i apstraktnog svijeta matematike.

Na slici 8.3 se može vidjeti primjer jedne česte problemske situacije s energijom u kojoj se mogu iskoristi stupčasti dijagrami u svrhu lakše analize problema.



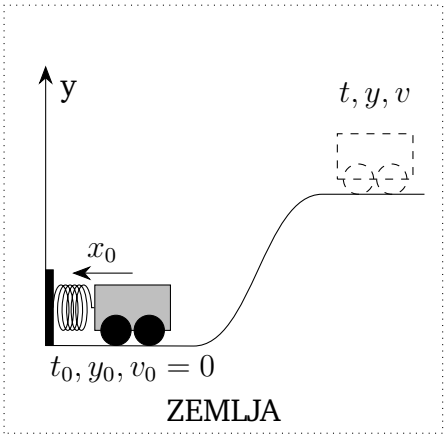
Slika 8.3: Slika pokazuje primjer problemske situacije u kojoj se traži računanje određenih fizikalnih veličina pomoću energije i rada.

8.1 Primjer jednostavnog procesa

Za analizu problema pomoću energijskih dijagrama prvi korak je odabir sustava. Sustav čini tijelo ili više tijela odvojenih od okoline zamišljenom površinom. Odabir sustava je ključan korak u određivanju promjene energije i obavljenog rada. Ako se pogleda zakon očuvanja energije $E_0 + W = E_1$ uočava se važnost odabira sustava. Ako je odabran sustav u kojem nema vanjskih tijela, odnosno sila, neće biti ni obavljenog rada $W = 0$, a ako se neka tijela nalaze izvan sustava energije E_0 i E_1 za dva sustava više neće biti iste.

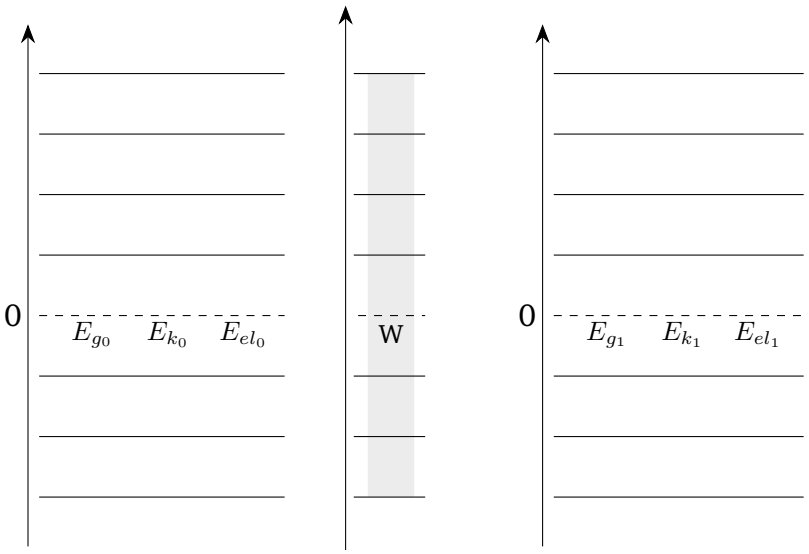
Za zadani fizikalni proces sustav se može odabrati proizvoljno i analiza svakog odabranog sustava će dovesti do istog rješenja. No, prepoznavanje odgovarajućih veličina i lakoća s kojom se dolazi do rješenja nije ista za svaki sustav. Kako se u srednjoškolskoj nastavi fizike stavlja naglasak na gravitacijsku potencijalnu energiju preporučeno je Zemlju uključiti kao dio sustava jer u suprotnom sustav ne bi imao tu energiju. Stoga je u primjeru odabran sustav kojeg čine: kolica, opruga i Zemlja.

Uz sustav je potrebno odabrati i dva vremenska trenutka. Za početni trenutak je odabran trenutak u kojem kolica miruju, a konačni trenutak je kad su kolica na vrhu brijega. Potrebno je odabrati i položaj nulte razine potencijalne gravitacijske energije, a u primjeru je odabrano dno brijega. Na slici 8.4 je prikazan odabrani sustav.



Slika 8.4: Slika pokazuje odabrani sustava za primjer sa slike 8.3.

Na slici 8.5 je prikazan primjer stupčastog dijagrama koji nije popunjen. Nakon odabira sustava, vremenskih trenutaka i nulte razine moguće je prepoznati promjene svih vrsta različitih energija unutar sustava te do kraja ispuniti stupčasti dijagram za oba trenutka.



Slika 8.5: Slika pokazuje prazni stupčasti dijagram energije.

8.2 Konstrukcija stupčastog dijagrama energije

Osnovna ideja kako popuniti dijagram je jednostavna:

- Ako u početnom trenutku tijelo ima brzinu $v \neq 0$ na dijagram se pod stupac E_{k_0} ucrta jedna, ili više, kućica, a ako tijelo miruje stupac ostaje prazan.
- Ako se u početnom trenutku tijelo nalazi iznad, ili ispod, nulte razine potencijalne gravitacijske energije crtaju se kućice u E_{g_0} stupac, u suprotnom, stupac ostaje prazan.
- Ako se u sustavu nalazi opruga na dijagram se ucrtava i stupac E_{el_0} s elastičnom potencijalnom energijom. Ako je opruga sabijena kućice se crtaju iznad nule.
- Konačni trenutak se analizira na sličan način.
- Stupac rada se crta na zasebni dijagram jer rad ne pripada niti jednom trenutku već je on proces koji se odvija između dva trenutka. Usporedbom smjera djelovanja vanjske sile i smjera u kojem se giba promatrano tijelo kućice se crtaju iznad za pozitivan rad ili ispod nule za negativan rad.
- Broj kućica za svaku pojedinu energiju ne mora biti proporcionalan iznosu energije već može biti proizvoljan. Dijagram za sustav sa slike 8.4 se može vidjeti na slici 8.6.

Očito je kako se iz dobivenog dijagrama može lakše naglasiti zakon očuvanja energije:

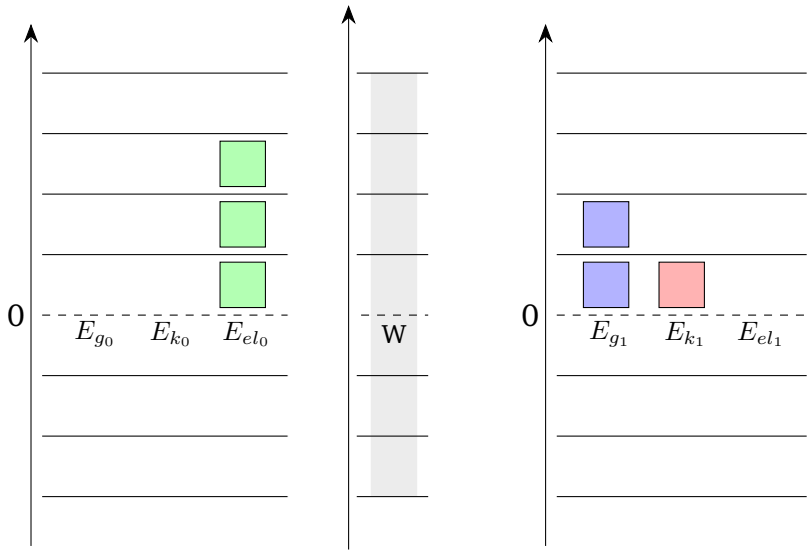
$$\textit{početna energija} + \textit{rad} = \textit{konačna energija}.$$

Ukupan broj kućica u početnom trenutku zbrojen s kućicama rada mora biti jednak ukupnom broju kućica energije u konačnom trenutku, energija je očuvana. Može se postaviti općenita jednadžba:

$$E_{g_0} + E_{k_0} + E_{el_0} + W = E_{g_1} + E_{k_1} + + E_{el_1}$$

iz koje je jednostavnije doći do jednadžbe očuvanja energije za specifični problem:

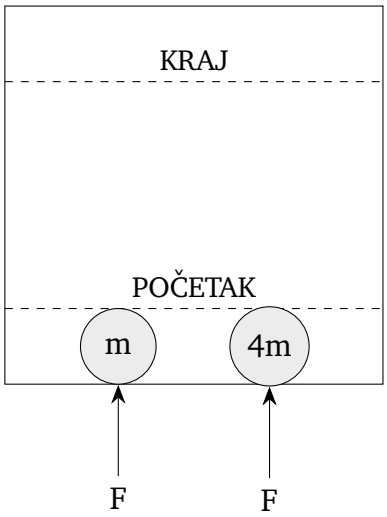
$$\begin{aligned} mgh_0 + \frac{1}{2}mv_0^2 + \frac{1}{2}kx_0^2 = mgh_1 + \frac{1}{2}mv_1^2 + \frac{1}{2}kx_1^2 \Rightarrow \\ \frac{1}{2}kx_0^2 = mgh_1 + \frac{1}{2}mv_1^2. \end{aligned}$$



Slika 8.6: Slika pokazuje stupčaste dijagrame energije za odabrani sustava sa slike 8.4.

8.3 Istraživanje uspješnosti stupčastih dijagrama

U istraživačkom radu predstavljenom u članku [22] uspješnost primjene stupčastih dijagrama je ispitana pomoću primjera na slici 8.7. Zadatak je zadan za dvije grupe studenata inženjerstva: 56 studenata koji su tijekom nastave fizike upoznati sa strategijom stupčastih dijagrama i 147 studenta koji su zadatke rješavali do tada klasičan način.



Slika 8.7: Slika pokazuje problemski zadatak zadan studentima Sveučilišta savezne države Ohio. Od studenata se traži usporedba konačnih kinetičkih energija dva tijela mase m i $4m$. Na tijela djeluje jednaka sila na jednakom putu.

Pokazalo se kako su studenti koji su sudjelovali u nastavi u kojoj je korišten pristup višestrukog kvalitativnog predstavljanja fizikalnih procesa bili znatno uspješniji u rješavanju zadanog zadatka. Od 56 studenata njih otprilike 60% je odgovorilo točno. Od 147 studenta druge grupe samo 20% je uspješno riješilo zadatak. Rezultat od 60% uspješnosti je bio na razini već diplomiranih asistenata, ali i usporediv s postotkom riješenosti među profesorima fizike.

Uz ispitivanje uspješnosti primjene stupčastih dijagrama 67 studenta je ispitano i o subjektivnom osjećaju znanja kojeg su stekli te o ulozi takvih kvalitativnih metoda prikaza fizikalnih procesa u rješavanju zadataka i boljem razumijevanju koncepata vezanih uz energiju i rad.

Čak 92% studenta je imalo pozitivan stav prema stupčastim dijagramima. Od njih, 62% se izjasnilo kako su im stupčasti dijagrami direktno pomogli u rješavanju zadatka tako što su im omogućili lakšu vizualizaciju problemske situacije i lakše pretvaranje riječi u prave jednadžbe. Primjerice, jedan student je napisao kako su im oni omogućili da bolje *vidi* koje su sve vrste energije prisutne u zadanom problemu.

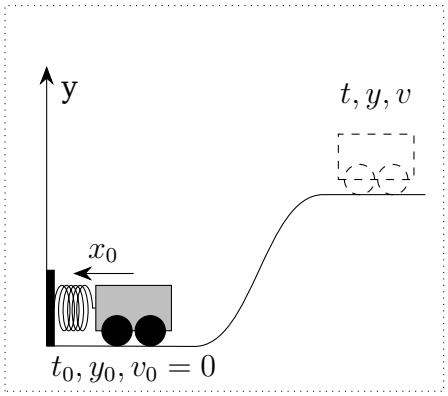
84% studenta se izjasnilo kako smatraju da su im strategije višestrukog predstavljanja fizikalnih procesa pomogle u boljem razumijevanju koncepata vezanih za energiju i kako je to dobra metoda za učenje fizike studenata iz raznih pozadina.

Oko polovice ispitanih studenata je reklo kako su im stupčasti dijagrami pomogli na početku učenja/studija, ali kako su postajali vještiji u rješavanju zadataka tako im se i potreba za crtanjem dijagrama smanjivala. Jednom kad su stekli bolje intuitivno poznavanje koncepata vezanih za energiju proces crtanja su internalizirali. Nije im više bio potreban eksplicitno nacrtan dijagram kako bi iskoristili prednosti koje on omogućuje.

8.4 Drugi odabir sustava

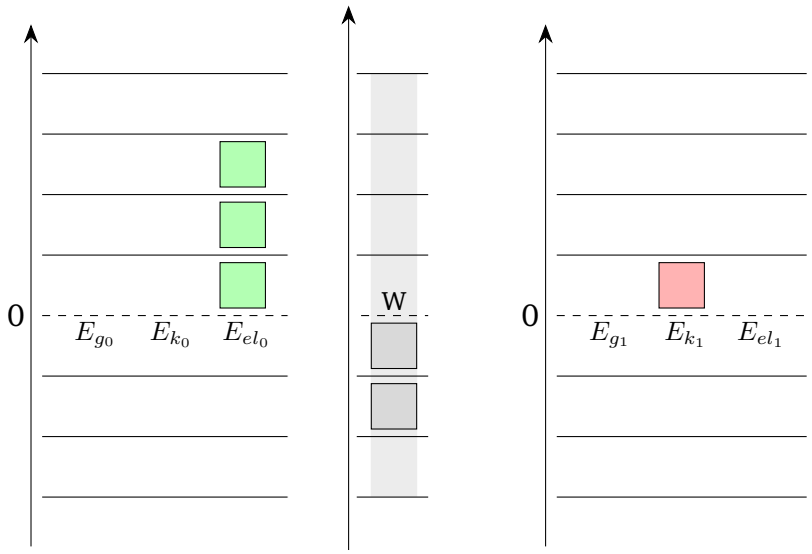
U primjeru sa slike 8.4 je odabran sustav u kojem su uključena kolica, opruga i Zemlja. Tako odabran sustav vodi do jednadžbe $\frac{1}{2}kx_0^2 = mgh_1 + \frac{1}{2}mv_1^2$. Kako je sustav proizvoljan moguće ga je odabrati i na drugačiji način. Primjerice, na slici 8.8 su kolica i opruga dio sustava, a Zemlja se nalazi izvan sustava.

Kako je Zemlja izvan sustava na kolica će djelovati vanjska gravitacijska sila, a smjer njezinog djelovanja je suprotan smjeru gibanja tijela pa se na stupčasti dijagram



Slika 8.8: Slika pokazuje odabrani sustav kolica + opruga za primjer sa slike 8.3.

u stupcu rada kućice ucrtavaju ispod nule jer je rad na sustav negativan. Pripadajući stupčasti dijagram se može vidjeti na slici 8.9.



Slika 8.9: Slika pokazuje stupčaste dijagrame energije za sustav kolica + opruga.

Ponovno se iz dijagrama može jednostavno iščitati zakon očuvanja energije:

$$E_{g_0} + E_{k_0} + E_{el_0} + W = E_{g_1} + E_{k_1} + E_{el_1} \Rightarrow$$
$$\frac{1}{2}kx_0^2 - mgy = \frac{1}{2}mv_1^2$$

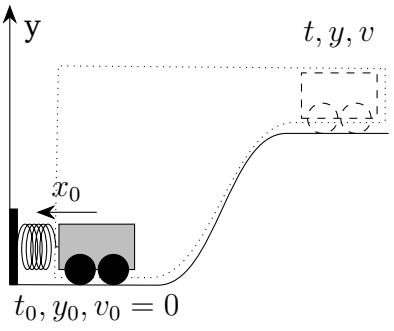
Usporedbom dobivenih jednadžbi za dva različita sustava može se uočiti kako su u osnovni one jednake, ali njihova interpretacija i značenje nije isto. U prvom slučaju, kad su u sustavu kolica, opruga i Zemlja, dolazi do promjene potencijalnih energija

zbog relativne promjene položaja ili oblika tijela. Gravitacijska potencijalna energija se povećava jer se kolica uspinju uz brijeg, povećava se udaljenost između Zemlje i kolica, a promjena oblika opruge dovodi do promjene elastične potencijalne energije. U drugom slučaju na sustav djeluje vanjska sila. Ne događa se promjena potencijalne gravitacijske energije već se nad sustavom vrši negativan rad što dovodi do smanjenja ukupne energije sustava.

Osim kao u prva dva slučaja sustav se može odabrati i kao na slici 8.10. Kako su u sustavu samo kolica, nema promjene potencijalnih energija kad sustav dođe iz početnog u konačni položaj. Nad sustavom se obavlja rad: rad elastične sile opruge i negativan rad gravitacijske sile Zemlje. Ponovno se može konstruirati pripadajući energijski dijagram kao na slici 8.11.

Iz dijagrama se može iščitati jednadžba: $W_{opruga} - mgy = \frac{1}{2}mv_1^2$. Kako je u početnom trenutku opruga sabijena vanjska elastična sila će na tijelo djelovati u smjeru njegovog gibanja pa će rad elastične sile opruge biti pozitivan i iznosi $W_{opruga} = \frac{1}{2}kx_0^2$. Stoga će jednadžba očuvanja energije imati isti oblik kao i u prijašnja dva primjera:

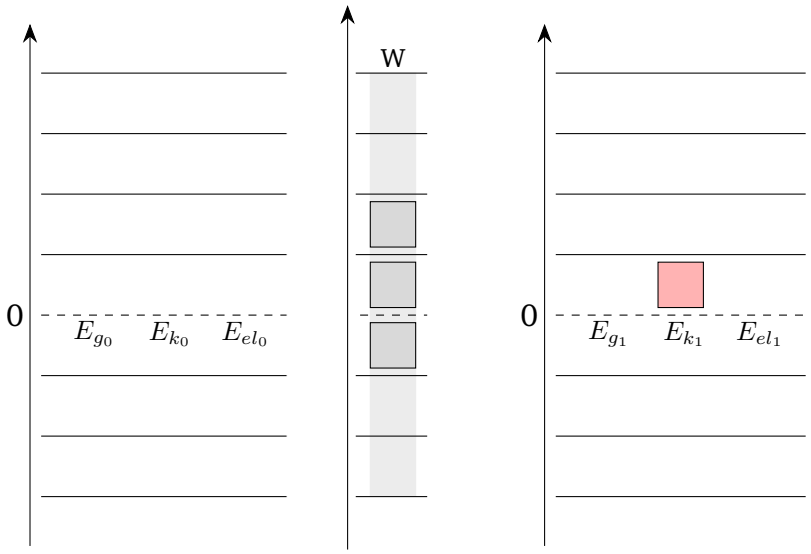
$$\frac{1}{2}kx_0^2 - mgy = \frac{1}{2}mv_1^2.$$



Slika 8.10: Slika pokazuje sustav u kojem se nalaze samo kolica.

8.5 Trenje

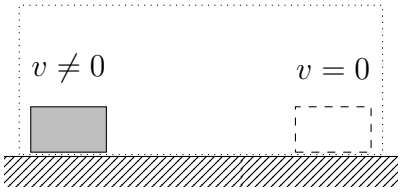
Česti problemski zadatci su i oni s trenjem. Način na koji trenje utječe na energiju tijela razlikuje se od ostalih sila u mehanici. Ako se ne radi samo o materijalnim točkama, kod trenja dolazi do deformacije kontaktnih površina što dovodi do povećanja temperature tijela pa stoga i povećanja unutarnje energije [23]. Kako bi



Slika 8.11: Slika pokazuje stupčaste dijagrame energije kad se u sustavu samo kolica.

se mogao analizirati proces u kojem se javlja trenje na stupčasti dijagram energije je nužno dodati stupac s promjenom unutarnje energije.

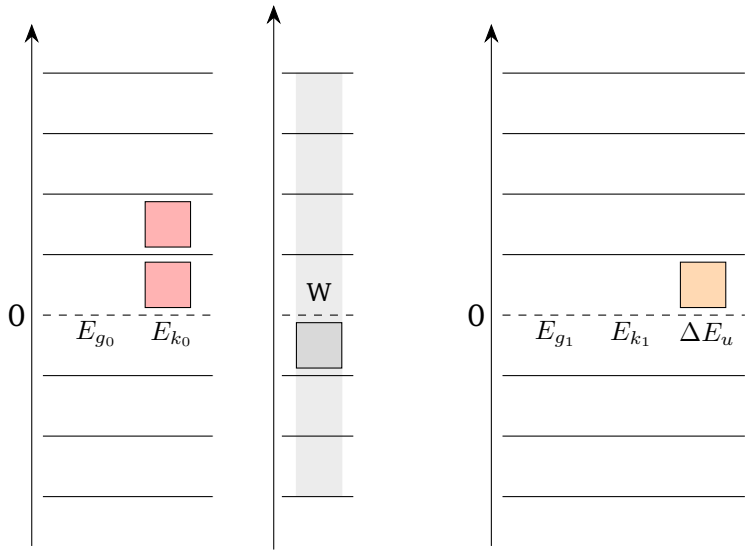
Jedan primjer tipične problemske situacije se može vidjeti na slici 8.12: Kvadar klizi po podlozi sve dok ga u potpunosti ne zaustavi trenje. Za analizu ovakvog problema se ponovno može provesti sličan postupak kao u prijašnjem primjeru.



Slika 8.12: Slika pokazuje primjer problemskog zadatka za sustav u kojem je samo kvadar.

Ako je sustav odabran kao na slici 8.12 sila trenja će vršiti negativan rad nad kvadrom sve dok se on u potpunosti ne zaustave. Uz to, kako su kotači stvarno tijelo, a ne samo materijalna točka, oni će se zbog djelovanja sile trenja i zagrijati. Stupčasti dijagram ovako odabranog sustava se može vidjeti na slici 8.13.
Ako se iz dijagrama očita zakon očuvanja energije:

$$E_{k_0} - W_{tr} = \Delta E_u,$$
$$\frac{1}{2}mv^2 + W_{tr} = \Delta E_u.$$

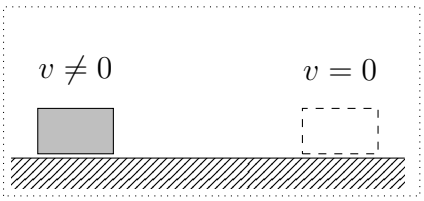


Slika 8.13: Slika pokazuje stupčaste dijagrame energije za primjer sa slike 8.12 kad je u sustavu samo kvadar.

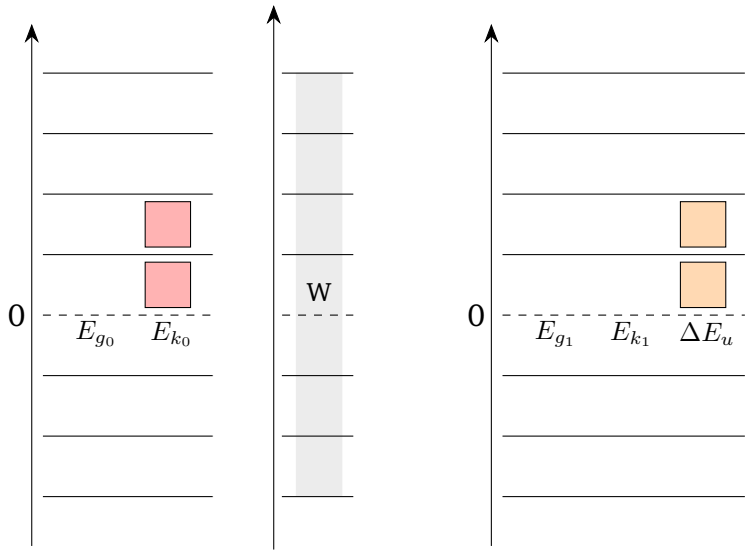
Iz jednadžbe se jasno vidi kako rad koji obavi sila trenja nije jednak samo početnoj kinetičkoj energiji. Ako je kvadar prešao put d rad sile trenja na kvadar nije $F_{tr}d = E_{k_0}$. Trenje je složen proces koji se javlja samo kad su izbočine tijela u kontaktu pa je efektivna udaljenost na kojoj djeluje sila trenja manja od d pa je i $|F_{tr}d_{ef}| < |F_{tr}d|$. [23]

Kako je za potpuni izračun potrebno poznavanje svih veličina efektivno se pojavila dodatna nepoznanica, rad sile trenja na kvadar, koja se ne može izračunati pa se i ne može nešto više reći o kvantitativnom rješenju problema. Stoga za problemske zadatke s trenjem preporučuje se odabir sustava kao na slici 8.14.

Kad su u sustavu i podloga i kvadar rad sile trenja će točno odgovarati promjeni ukupne unutarnje energije sustava jer tijela koja su u kontaktu se nalaze upravo u sustavu.



Slika 8.14: Slika pokazuje primjer problemskog zadatka za sustav kvadar + podloga.



Slika 8.15: Slika pokazuje stupčaste dijagrame energije za primjer sa slike 8.14 kad su u sustavu kvadar i podloga. Iz dijagram se očitava $\frac{1}{2}mv_0^2 = \Delta E_u$. Može se uočiti kako je jednadžba dobivena ovakvim odabirom sustava jednostavnija od prvog slučaja. Nema dodatnog člana koji se mora zasebno izračunati. Sva početna kinetička energija je pretvorena u unutarnju energiju sustava.

8.6 Zaključak

Iz navedenih primjera se vidi kako stupčasti dijagrami energije mogu pomoći pri analizi problemski zadataka. Prikazivanje problema na više od jednog kvalitativnog načina omogućuje brži razvoj intuitivnog osjećaja za apstraktne koncepte poput energije. Prikladni odabir sustava i pripadajućih veličina olakšava se put k lakšem rješenju zadatka.

Iz jednadžbi nije uvijek lagano iščitati pozadinske zakone posebice za učenike srednjih škola, ali pažljivim crtanjem dijagrama se može približiti osnovni zakon očuvanja energije. Usporedbom kućica energije u dva zadana trenutka jasno se vidi kako se energija ne može izgubiti ili stvoriti.

Bibliography

[1] Merkle, R. (1979) Secrecy, authentication, and public key systems, Ph.D., dostupno na: http://www.merkle.com/papers/Thesis1979.pdf 21.5.2021.

[2] Ministarstvo znanosti i obrazovanja, Natjecanje iz informatike, dostupno na: https://informatika.azoo.hr/o-natjecanju 7.9.2021.

[3] OpenStax, Chemistry. OpenStax CNX, dostupno na: https://opentextbc.ca/chemistry/chapter/20-1-hydrocarbons/ 13.6.2021.

[4] Bhavanari, S. Discrete Mathematics and Graph Theory. Prentice Hall India Learning Private Limited, 2014.

[5] Electrical4U, (2020). Trees and Cotrees of an Electric Network, dostupno na: https://www.electrical4u.com/trees-and-cotrees-of-electric-network/ 13.6.2021.

[6] Manger, R. Strukture podataka i algoritmi. Element d.o.o., 2014.

[7] Cormen, T.; Leiserson, C.; Rivest, R.; Stein, C. Introduction to Algorithms, 3rd Edition. The MIT Press, 2009.

[8] Noll, L. *Fowler–Noll–Vo hash function*, dostupno na: http://www.isthe.com/chongo/tech/comp/fnv/index.html. 27.5.2021.

[9] T0ST, (28.2.2021.) *How I cut GTA Online loading times by* 70%, dostupno na: https://nee.lv/2021/02/28/How-I-cut-GTA-Online-loading-times-by-70/ 27.5.2021.

[10] Bench Crypto, eBACS: ECRYPT Benchmarking of Cryptographic Systems, dostupno na: http://bench.cr.yp.to/. 29.5.2021.

[11] Intel, Intel SHA intrinsics, dostupno na:https://software.intel.com/content/www/us/en/develop/articles/intel-sha-extensions.html. 29.5.2021.

[12] Mosnier, A. *SHA-2 algorithm implementations*, dostupno na: https://github.com/amosnier/sha-2. 29.5.2021.

[13] Szydlo M. (2004). Merkle Tree Traversal in Log Space and Time. In: Cachin C., Camenisch J.L. (eds) Advances in Cryptology - EUROCRYPT 2004. EUROCRYPT 2004. Lecture Notes in Computer Science, vol 3027. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-24676-3_32 6.6.2021.

[14] Nakamoto, S. (2008.) *Bitcoin: A Peer-to-Peer Electronic Cash System*, dostupno na: https://bitcoin.org/bitcoin.pdf. 24.5.2021.

[15] Buy Bitcoin - Worldwide. Bitcoin Mining Pools, dostupno na: https://www.buybitcoinworldwide.com/mining/pools/. 13.9.2021.

[16] Katz, J. i Lindel, Y. Introduction to Modern Cryptography. 3rd edition: Chapman and Hall/CRC, 2020.

[17] University of Cambridge, Centre for Alternative Finance, dostupno na: https://cbeci.org/ 14.9.2021.

[18] U.S. Energy Information Administration. Electricity consumption, dostupno na: https://www.eia.gov/ 14.9.2021.

[19] Crypto.com.(2021.) On-Chain Market Sizing, dostupno na: https://crypto.com/eea/research/article?category=data&page=on_chain_market_sizing 14.9.2021.

[20] Morrison, D. PATRICIA — Practical Algorithm To Retrieve Information Coded in Alphanumeric. The Journal of the ACM. Vol. 15., Izdanje 4, Listopad 1968. Str 514-534. doi.org/10.1145/321479.321481. 18.9.2021.

[21] Ethereum. Modified Merkle Patricia Trie, dostupno na: https://eth.wiki/en/fundamentals/patricia-tree 18.9.2021.

[22] Van Heuvelen. A, i Zou, X. Multiple representations of work–energy processes. American Journal of Physics 69, 184 (2001); doi: 10.1119/1.1286662.

[23] Planinić, M. Skripta iz metodike nastave fizike. metodika.phy.hr/claroline