FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
SVEUČILIŠTE U ZAGREBU

Grafovi i stabla: algoritmi pretraživanja i najkraćih putova

Računalna grafika

Samostalni projekt

Amalija Ramljak
0036501692

Opis ostvarenog **1**
Tehnički podaci 1
Obilazak stabla 1
Početak programa 1
Popunjavanje stabla 2
Stilovi čvorova i proces iscrtavanja 2
BFS algoritam 3
DFS algoritam 4
Prekidanje tijekom izvođenja algoritma 5
Najkraći put u grafu 5
Početak programa 5
Izgradnja grafa 6
Iscrtavanje grafa 7
Dijkstrin algoritam najkraćeg puta 8
Upute za pokretanje **9**

1. Opis ostvarenog

Kao dio projekta napravljene su dvije stvari: prikaz obilaska stabla BFS (*Breadth First Search*) i DFS (*Depth First Search*) algoritmima te prikaz najkraćeg puta u grafu pronađenog Dijkstrinim algoritmom najkraćeg puta. Vizualizacije algoritama vrlo su korisne pri učenju i shvaćanju rada nekog algoritma, no nisu uvijek jednostavne za izvedbu kao BFS i DFS, zbog čega je Dijkstrin algoritam izostavljen iz vizualizacije iteracija jer je malo zahtjevniji za kvalitetnu vizualizaciju korak po korak, no prikazan je konačan najkraći put kroz graf između odabranih čvorova. Sami grafovi i stabla kao strukture nad kojim se provode algoritmi odabrani su jer su učestali u svijetu u raznoraznim oblicima, a uostalom i struktura HTML datoteka te scena u svijetu računane grafike su napravljene kao stabla.

1.1. Tehnički podaci

Korišten je jezik JavaScript uz knjižnicu d3.js, a rješenje je izgrađeno u HTML datoteci i pripadnim HTML *tagovima,* te postoji i CSS datoteka dijeljenja između oba zadatka projekta.

1.2. Obilazak stabla

U datoteci `trees.html` implementirana je logika za vizualizaciju dva algoritma za traženje elemenata u grafu. U ovom slučaju korištena su samo stabla koja su podskup grafova, ali ispravno bi radilo i na grafovima zbog prirode algoritma koji je implementiran u cijelosti.

1.2.1. Početak programa

Kada se program otvori, korisnik vidi upravljački dio prikazan na Slici 1 u lijevom kutu, te se gore na sredini ekrana nalazi jedan čvor stabla označen brojem 0. Za popunjavanje stabla postoji samo mogućnost dodavanja nasumičnog čvora u stablo, o čemu će biti rečeno detaljnije malo kasnije. Osim dodavanja novog nasumičnog čvora u stablo, upravljačke opcije sadrže i polje za upisivanje broja te dva gumba, “Find by BFS” i “Find by DFS”, koji pokreću BFS ili DFS algoritam nad stablom u potrazi za čvorom čiji je broj jednak upisanom u polje. Ako nema upisanog broja, ništa se ne događa. Ispod gumbi još postoji jedan element gdje se ispisuju poruke po potrebi, kao, na primjer, kada se doda novi čvor ili tijekom traženja na kojem je trenutno elementu, te kada pronađe ili ne pronađe element u stablu. Primjeri nekih ispisa prikazani su na Slici 1.

Slika 1 - Upravljačke opcije za `trees.html` program uz primjere poruka.

1.2.2. Popunjavanje stabla

Prvi korak pri interakciji s programom je popunjavanje stabla. Na klik gumba "Add random node" poziva se funkcija `addnode()` koja kreće od korijena stabla te se spušta niz njega ovisno o vjerojatnosti te koliko djece već ima, a traži se roditelj za novi nasumični čvor. Kriteriji preskakanja su idući, povezani logičkim ILI:

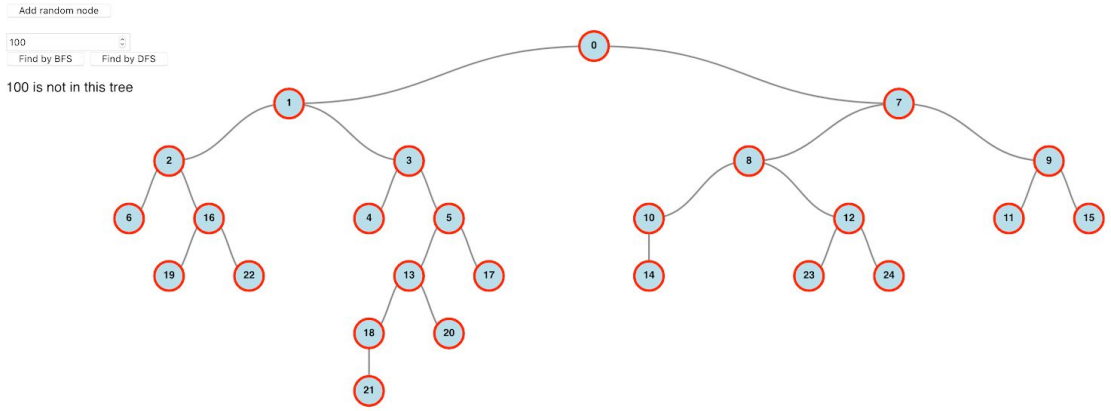
- ostvarena vjerojatnost od 75% za prelazak na neko dijete u slučaju da čvor ima djece - nasumičnim generatorom dobiven je broj manji od 0.75 ili
- čvor ima dvoje djece u slučaju da uopće ima djece.

U slučaju da je određeno preskakanje trenutnog čvora, idući čvor bira se od njegove djece: ako ima samo jedno dijete svakako će se spustiti na to dijete, no ako ima dvoje djece nasumično će se odabrati jedno dijete za idućeg kandidata za roditelja. Ta petlja spusta definira ograničenje od dvoje djece po čvoru kako bi se spriječila prenapučenost nekih čvorova dok drugi nemaju nijedno dijete, a napravljeno je isključivo iz vizualnih razloga - algoritmi korišteni na ovako generiranom binarnom stablu jednako bi dobro radili na bilokakvim stablima s bilokojom količinom djece jer su općeniti i zapravo su algoritmi za traženje u grafu koji nikako nemaju ograničenja na broj susjeda.

Dodavanje djece jednom kada je roditelj određen je trivijalno: samo se u polje djece trenutnog čvora doda novi čvor u formatu određenom knjižnicom d3.js, a zbog načina na koji JavaScript funkcionira s referencama to je moguće kroz varijablu u koju je spremljen trenutni čvor koji je određen kao roditelj novom čvoru.

1.2.3. Stilovi čvorova i proces iscrtavanja

Izgled čvorova koji se trenutno provjeravaju i koji su već provjereni te čvora koji je tražen, ako se nalazi u stablu, definiran je kao zajednički za BFS i DFS algoritme. Na Slikama 3 i 4 vidljiv je primjer već provjerenih čvorova koji nisu jednaki traženom čvoru s crvenim obrubom te čvora koji se trenutno provjerava s narančastim obrubom uz ispis odgovarajuće poruke ispod upravljačkih opcija. Niže na Slici 5 vidljiv je i način označavanja čvora koji se tražio zelenim obrubom. Na Slici 2 vidljiv je rezultat kada bilo koji od algoritama nije naišao na traženi čvor - svi čvorovi označeni su crvenim obrubom.



Slika 2 - Rezultat vizualizacije kada traženi čvor nije u stablu.

Kako bi se graf jasno iscrtavao iz koraka u korak namješteno je pauziranje izvođenja nakon svake iteracije jer se bez pauziranja nije uopće vidio redoslijed obilaženja zbog brzine izvođenja algoritma.

Za samo iscrtavanje različitih oznaka čvorova koriste se vlastiti atributi tipa boolean stavljeni u elemente čvorova koje u svojoj preddefiniranoj podatkovnoj strukturi za stabla knjižnica d3.js koristi za iscrtavanje grafa. Nakon svake promjene trenutnog promatranog čvora namještaju se odgovarajuće “zastavice” na trenutnom i prethodnom čvoru, točnije `checked` za prethodnog i `selected` za trenutnog, kako bi trenutni postao narančast, a prethodni crven. U slučaju da je trenutni čvor traženi čvor, preskače se korak gdje je iscrtan kao narančast i algoritam ga odmah zazeleni.

Iscrtavanje se radi nakon svake iteracije kako bi se promjene vidjele na stablu, a to je omogućeno pozivom funkcije changetree() koja očisti trenutno nacrtano stablo iz HTML DOM-a (*Document Object Model*) i zamijeni ga novim

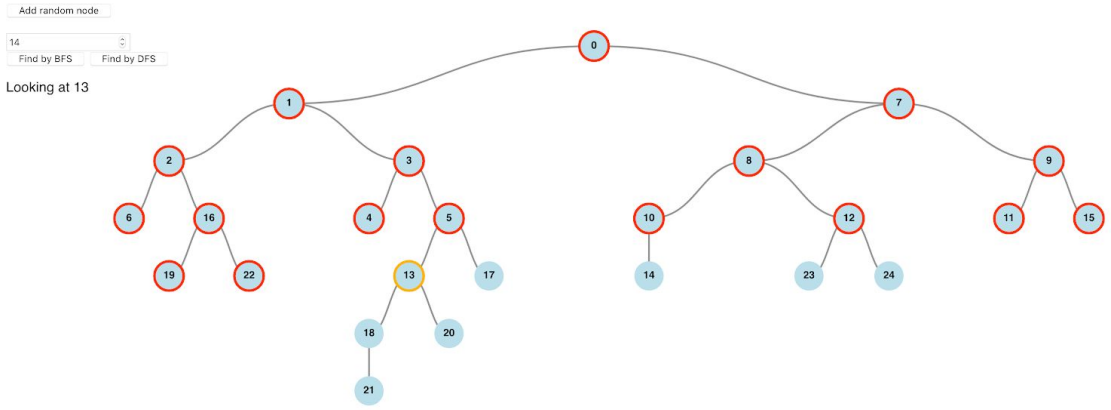
1.2.4. BFS algoritam

BFS algoritam temelji se na ideji da se prvo obiđu najbliži susjedi čvora u grafu, odnosno kod stabla to su djeca čvora, a tek se onda krene na susjede tih susjeda i tako dok nisu svi obiđeni. Koristan je kada je potrebno pronaći recimo najkraći put do nekog čvora u beštežinskom grafu, odnosno put koji prolazi najmanjim brojem čvorova - prvi put kojim se stigne do nekog čvora bit će najkraći put do njega upravo zbog pravila da se prvo obilaze najbliži čvorovi.

Pravilo obilaženja prvo najbližih čvorova implementira se strukturom red (eng. *queue*) za čuvanje liste idućih čvorova za obići, a kako algoritam ne bi zapeo u beskonačnoj petlji postoji i polje koje čuva informaciju o tome koji su čvorovi već posjećeni. To polje je bespotrebno za stabla, no ipak je sastavni dio BFS algoritma koji omogućava njegovu primjenu na grafovima te je zbog toga algoritam implementiran za ovaj projekt primjenjiv na bilokakva stabla ili grafove iako, u

trenutnom obliku, program za stabla ne podržava učitavanje preddefiniranih struktura ili generiranje biločega što nije binarno stablo.

Na Slici 3 vidljivo je da BFS algoritam prvo obilazi čvorove na višim razinama u stablu, pogotovo kada se usporedi s primjerom DFS obilaska istog stabla na Slici 4.

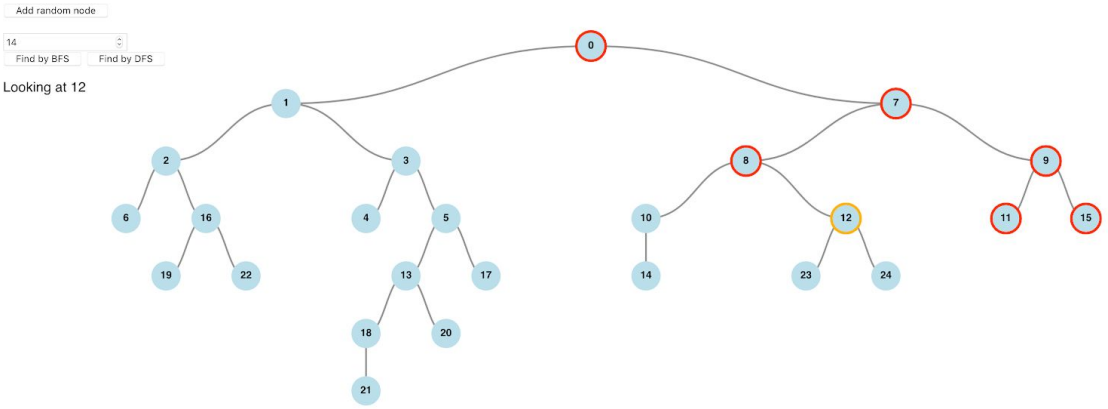


Slika 3 - Primjer prolaska BFS algoritma po stablu u potrazi za čvorom 14.

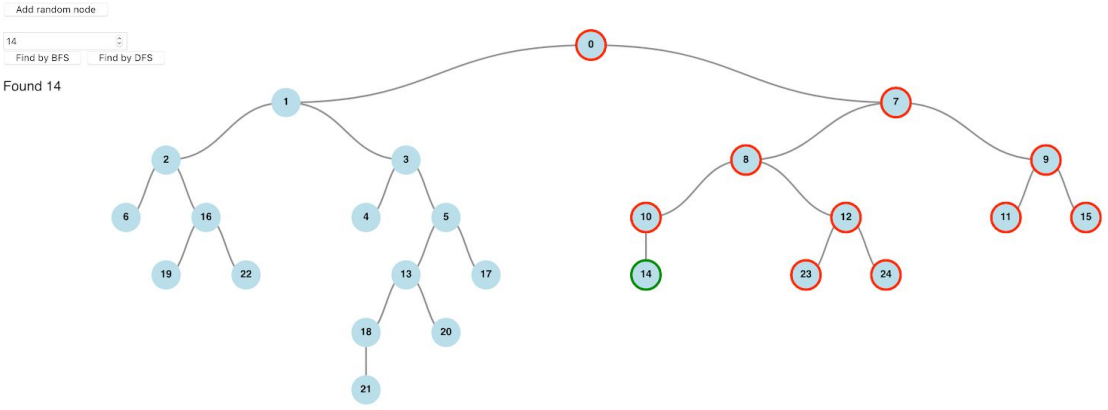
1.2.5. DFS algoritam

DFS algoritam razlikuje se od BFS-a po temeljnoj ideji, a to je da DFS ide u “dubinu” po grafu, kao što je rečeno u samom imenu algoritma. Ta ideja implementira se tako da se umjesto BFS-ovog reda koristi struktura stog, i to je zapravo jedina razlika između ovih algoritama - ostatak koda je u implementaciji dijeljen između ta dva algoritma.

Gdje BFS idućeg bira s početka polja, a stavlja na kraj, što je definicija strukture red, DFS stavlja i uzima s istog kraja svog polja, što je definicija strukture stog. Time je upravo postignut taj efekt obilaska u dubinu jer čim uzme novi čvor stavlja njegovu djecu na stog što znači da će mu idući čvor biti dijete trenutnog, dok je kod BFS-a općenit slučaj da je idući čvor dijete čvora čija djeca još nisu sva obiđena iz ranijih trenutaka algoritma. Na Slici 4 vidljiv je primjer provedbe DFS-a, a najbolje je usporediti tu sliku sa Slikom 3, na kojoj se u istom stablu traži isti čvor.



Slika 4 - Primjer prolaska DFS algoritma po stablu u potrazi za čvorom 14.



Slika 5 - Primjer izgleda pronađenog čvora 14 nakon provedbe DFS algoritma. Oznaka je jednaka nakon provedbe BFS algoritma.

1.2.6. Prekidanje tijekom izvođenja algoritma

Program podržava prekide algoritma na način da se doda novi čvor u stablo ili se ponovno pritisne jedan od gumbi za traženje uz, npr., izmijenjen traženi čvor. Bilo koji od tih događaja postavit će odgovarajuću zastavicu u kodu koja će zatim prekinuti petlju algoritma ako se ona vrti i ispisati odgovarajuću poruku o prekidu sparenu s porukom o dodanom čvoru. Kod pokretanja nove petlje algoritma prikazat će se informacija o prekidu, ali bit će kratkotrajna jer će ju prepisati informacije o trenutnom čvoru na kojem se nalazi algoritam.

1.3. Najkraći put u grafu

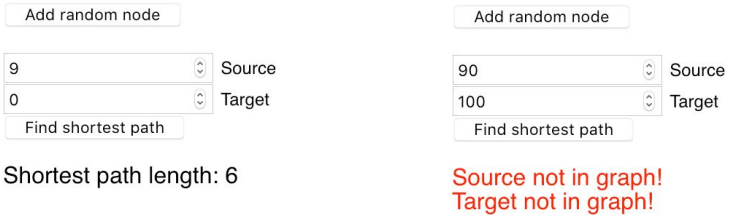
Datoteka `graphs.html` sadrži svu logiku za ovaj segment projekta. Ovdje su grafovi težinski jer je traženje najkraćeg puta na temelju broja bridova trivijalno, a zbog korištenja Dijkstrinog algoritma težine su ograničene na pozitivne brojeve.

1.3.1. Početak programa

Slično kao i kod `trees.html`, kada korisnik otvori datoteku `graphs.html` prvo što vidi je upravljački dio programa koji stoji u gornjem lijevom kutu. Doduše, u ovom

zadatku je početno stanje prazan graf te se kod dodavanja čvorova kreće od broja 0, dok je za stabla taj korijen odmah definiran i prikazan.

Na Slici 6 prikazan je izgled upravljačkih opcija, a sastoji se od gumba za dodavanje nasumičnih čvorova u graf generiranih prema pravilima koja su kasnije objašnjena, zatim od dva polja za unos brojeva gdje se unose identifikacijski brojevi izvornog čvora te krajnjeg čvora između kojih se traži najkraći put, a ispod toga nalazi se gumb koji pokreće traženje te ispod toga element u koji se na kraju algoritma ispisuje duljina pronađenog puta između željenih čvorova ili po potrebi greška pri unosu izvornog i krajnjeg čvora jer ovdje nije dopušteno da jedan od njih ne postoji u grafu jer zaista nema smisla tražiti put između postojećeg čvora i nekog kojeg uopće nema u grafu ili, još gore, između dva čvora od kojih se nijedan ne nalazi u grafu.



Slika 6 - Sučelje kroz koje korisnik dodaje nasumične elemente grafu i odabire dvije točke grafa između kojih se traži put. Lijeva slika prikazuje tekst nakon završetka algoritma, a desna prikazuje greške koje se prijavljuju korisniku kada neki od odabranih čvorova ne postoji u grafu.

1.3.2. Izgradnja grafa

Kao kod izgradnje stabla od nasumičnih čvorova, i za grafove je omogućena izgradnja pomoću nasumičnih novih čvorova spojenih s nasumičnim susjedima, a funkcija `addnode()` poziva se na pritisak gumba "Add random node", kao i kod `trees.html`.

Kod grafa su pravila stvaranja novog čvora malo drugačija. Prvo se napravi novi čvor koji se odmah doda među čvorove (nije nimalno nužno), a zatim se određuje broj veza kojima će inicijalno biti povezan u grafu. Taj broj veza određuje se formulom `min(1 + floor(random() * 3), broj_starih_cvorova)`, a rezultat koji daje je broj između 1 i 3 jer je broj starih čvorova uvijek minimalno 1 jer se u dio programa za stvaranje veza ulazi samo ako postoji već barem 1 čvor u grafu.

Za svaku stvorenu vezu nasumično se traži čvor grafa s kojim će se novi čvor povezati. Osim što se nasumično bira čvor s kojim će se novi čvor povezati, postoje i dodatne provjere koje odbacuju nasumično odabran čvor ako ne zadovoljava određena svojstva. Uvjeti prihvaćanja nasumično odabranog čvora kao susjeda novog čvora su sljedeći:

- novi čvor nije već u nekoj prethodnoj iteraciji povezan s odabranim čvorom ili
- odabrani čvor ima manje od 5 veza.

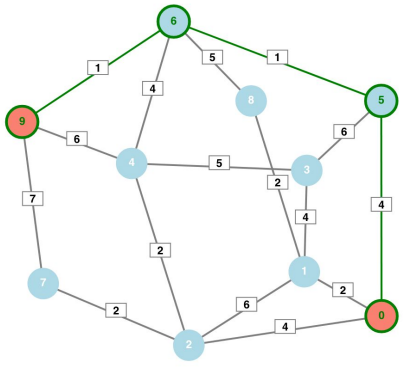
Dupliciranje veza je problematično u ovim grafovima jer ne postoji implementacija razdvajanja takvih preklapajućih veza, pa se veze prekrivaju i može lako doći do toga da konačan put izgleda slomljeno i krivo jer je odabrana veza koja je manje težine, a prekrivena je vezom veće težine. Što se tiče ograničenja od 5 veza po čvoru, ono je najviše kozmetički jer bi grafovi lako postali previše nepregledni za demonstracije, no broj veza jednog čvora nije nimalno bitan samom algoritmu traženja najkraćeg puta.

Nakon dodavanja veza novom čvoru i dodavanja nasumičnih težina svakoj vezi u intervalu cijelih brojeva [1, 7], graf se počisti i ponovno iscrta s novom strukturom grafa.

1.3.3. Iscrtavanje grafa

Za iscrtavanje grafa koriste se mogućnosti JavaScript knjižnice d3.js kao i za stabla. Stabla su statična između promjena jer je za njih lakše odrediti fiksne pozicije djece, no za grafove je potreban drugačiji pristup iscrtavanja, a on uključuje simuliranje sile između čvorova koja ih onda razmješta po njihovom koordinatnom sustavu svaki puta kada se iscrtavaju, a taj proces traje nekoliko sekundi za svako novo iscrtavanje. To simuliranje sile dolazi već s knjižnicom d3.js pa nije bilo potrebe samostalno implementirati funkciju za to.

Sam graf prikazan je kao čvorovi spojeni ravnim crtama preko kojih je pozicioniran broj koji predstavlja težinu brida grafa. Iscrtavanje konačnog rješenja događa se tek kada je algoritam završio, a algoritam za označavanje čvorova i bridova/veza da su na putu prima cijeli put koji je pronađen. On prolazi po tom putu i pronalazi pripadne čvorove i veze te ih prikladno označi preddefiniranim imenima za zastavice, pa pozove funkciju iscrtavanja koja uz pauziranje iscrtava korak po korak svaki čvor po putu u stilu koji je definiran za čvor na putu te vezu između tog čvora i prethodnog, a stilovi su vidljivi na Slici 7. Jedino su, radi preglednosti, početni i krajnji čvor obojani crvenkastom bojom kako bi se lakše vidjelo koja dva čvora spajamo.



Slika 7 - Primjer iscrtanog grafa s označenim najkraćim putem između čvorova 0 i 9.

1.3.4. Dijkstrin algoritam najkraćeg puta

Za traženje najkraćeg puta implementiran je Dijkstrin algoritam zbog kojeg su težine definirane kao pozitivne vrijednosti jer taj algoritam zbog svoje prilično pohlepne prirode može beskonačno zapeti na negativnim bridovima.

Sam algoritam je jednostavan. Na početku se za sve čvorove osim početnog postavi udaljenost na beskonačno te taj broj predstavlja udaljenost tog čvora od početnog čvora. U red se na početku stavljaju svi susjedi početnog čvora te se ide sličnim principom kao BFS algoritam. Međutim, ono što Dijkstra radi drugačije od BFS-a je da iz svog reda bira čvor koji trenutno ima najmanju udaljenost od početnog čvora, a ta je informacija kodirana u element koji se čuva u redu. Taj red se u biti sortira rastuće prema toj vrijednosti, a zove se prioritetni red (eng. *priority queue*).

Zapravo, ova implementacija i nije pravi Dijkstra jer je izostavljeno to sortiranje, odnosno korištenje prioritetnog reda, jer nije korištena prava referenca na algoritam već je samo izveden funkcionalan algoritam najkraćeg puta preko BFS-a koji se prilično preklapa s Dijkstrinim algoritmom. Glavna razlika između pravog algoritma i ove implementacije su performanse jer Dijkstra svojom pohlepnošću brže dođe do rješenja i odbaci sve ostalo u svom redu jer nije bolje od dosadašnjeg, dok ova implementacija sporije nalazi bolje rješenje pa sve više i više puni red i traži bolje rješenje na svaku promjenu najbolje udaljenosti. Međutim, i dalje postoji uvjet koji sprječava algoritam da se razleti i previše poveća svoj red, a to je da preskače čvorove i njihove susjede s novim udaljenostima ako udaljenost čvora od početnog čvora nije bolja od do sad pronađene za taj isti čvor. Nema smisla tražiti dalje od tog čvora jer će sve daljnje udaljenosti za njegove susjede biti sve veće i veće te veće od njihovih do sad pronađenih najmanjih udaljenosti od početnog čvora.

Implementacija nije pravi Dijkstra zbog želje da se isproba izvesti funkcionalan algoritam s težinama pomoću BFS algoritma i poznavanja Dijkstrinog algoritma i njegove povezanosti i djelomične sličnosti s BFS algoritmom.

2. Upute za pokretanje

Za pokretanje dovoljno je otvoriti odgovarajuću HTML datoteku u internetskom pregledniku po izboru. Datoteka `graphs.html` otvorit će sučelje za grafove, a `trees.html` sadrži sučelje za stabla. Sučelja su opisana u ranijem poglavlju i pripadnim potpoglavljima vezanim uz odabranu datoteku te su intuitivna za korištenje.