Vizualizacija i usporedba algoritama za pronalazak najkraćeg puta

Todorović, Sebastijan

SVEUČILIŠTE U ZAGREBU
FILOZOFSKI FAKULTET
ODSJEK ZA INFORMACIJSKE I KOMUNIKACIJSKE ZNANOSTI
Ak. god. 2023./2024.

Sebastijan Todorović

Vizualizacija i usporedba algoritama za pronalazak najkraćeg puta

Završni rad

Mentor: izv. prof. dr. sc. Vedran Juričić

Zagreb, rujan 2024.

Izjava o akademskoj čestitosti

Izjavljujem i svojim potpisom potvrđujem da je ovaj rad rezultat mog vlastitog rada koji se temelji na istraživanjima te objavljenoj i citiranoj literaturi. Izjavljujem da nijedan dio rada nije napisan na nedozvoljen način, odnosno da je prepisan iz necitiranog rada, te da nijedan dio rada ne krši bilo čija autorska prava. Također izjavljujem da nijedan dio rada nije korišten za bilo koji drugi rad u bilo kojoj drugoj visokoškolskoj, znanstvenoj ili obrazovnoj ustanovi.

_____

(potpis)

Sadržaj

1. Uvod

Pronalazak najkraćeg puta jedan je od najvažnijih i najutjecajnijih problema u polju teorije grafa. Iako je problem koji je postojao i prije modernog digitalnog doba, dolaskom digitalizacije, polje teorije grafa brzo je napredovalo u svojem značaju i važnosti, te se pokazalo kao vrlo važno polje za rješavanje mnogih problema vezanih uz računala, što sa softverske, što hardverske strane.

U mnogim profesijama javljala se kroz godine potreba za efikasnim pronalaskom putanje između dvaju točaka koje mogu predstavljati fizičke lokacije, pojmove u lingvistici ili neke druge pojmove koji mogu biti međusobno povezani, bilo da je to problem koji se javlja u računalnim mrežama video igrama, neuronskim mrežama, i ili pak u navigacijskim i logističkim sustavima. Primjena algoritama u tim situacijama omogućila je brži razvoj, i općenito olakšanje mnogih poslova i problema na koje se može naići. Od izračuna ruta za transportna vozila kojima se olakšava protok na cesti i izbjegava prometni zastoj čime se ubrzava protok robe, čime se samim time povećava ekonomska dobit, do razvoja boljih i imerzivnijih likova u video igrama.

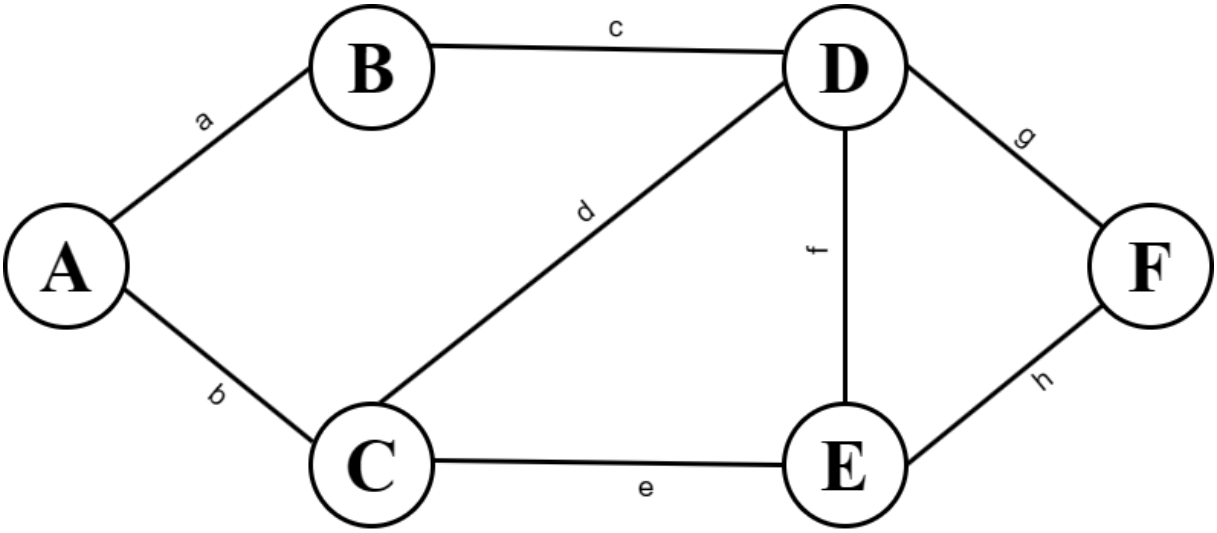
Ovaj rad podijeljen je na teorijski i praktični dio. U teorijskom dijelu bit će objašnjeni osnovni pojmovi iz teorije grafa, uključujući grafove, vrste grafova, čvorovi koji tvore te grafovi i bridovi koji ih povezuju. Nakon pojašnjenja glavnih dijelova teorije grafa koji se tiču problema pronalaska najkraćeg puta, bit će predstavljeni problemi koji se pokušavaju riješiti, te načini na koji se oni rješavaju. Bit će definirani i objašnjeni Dijkstrin algoritam, A*, HPA* i D* Lite algoritam. Način rada svakog od tih algoritama bit će objašnjen i vizualno svakim korakom do rješenja.

U praktičnom dijelu rada bit će definirana implementacija svakog od algoritama, te će se u istim uvjetima usporediti vrijeme potrebno za izračun rute svakoga od tih algoritama.

2. Teorija grafa

2.1. Graf

Kako bi se mogli razumjeti algoritmi za pronalaske najkraćeg puta, prvo je potrebno upoznati se s pojmom grafa, njegovim elementima i kako su povezani. Jednostavan graf G = (V, E) možemo definirati kao ne-prazan konačan skup čvorova V = {v_0, v_1, v_2, …}, i konačnog broja bridova E = {e_0, e_1, e_2, …} koji su povezani s tim čvorom. (Wilson, 1979, str. 8) Temeljni su dakle elementi grafa čvor i brid. Kako bi se napravio najjednostavniji i najmanji upotrebljivi graf, potrebna su dva čvora, i jedan brid koji ih povezuje. Time dobivamo informaciju o povezanosti dvaju čvorova što tvori graf.



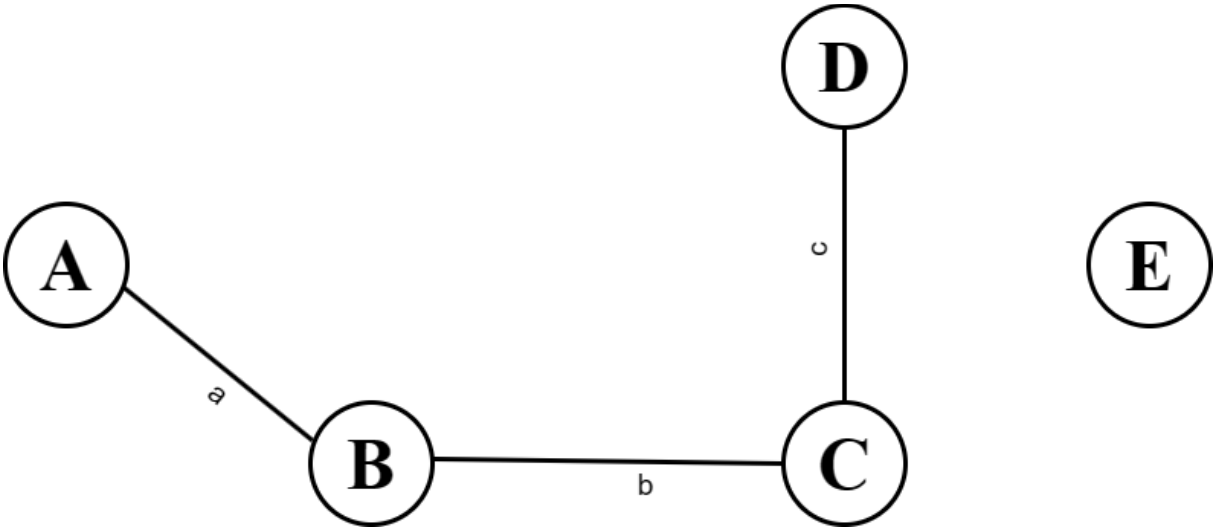
Slika 1 Običan neusmjereni graf

Vidljivo na primjeru slike 1. neki čvorovi (označeni velikim slovima) povezani su s drugima, te se ta veza opisuje bridovima (označeni malim slovima). Ova je slika primjer običnog grafa koji je konstruiran i sadrži najosnovnije informacije potrebne za izgradnju jednog grafa.

Čvor u grafu vrlo često ne nosi važnu informaciju, te služi za iskazivanje početka i kraja jednog brida koji u sebi nosi sve potrebne informacije. Primjer jedne takve interpretacije grafa može se pronaći u kontekstu primjene grafa u sustavima za navigaciju. Čvorovi će označavati raskrižja, a bridovi ceste, te s obzirom na to da je za navigaciju potrebna informacija o cesti kroz koju se tvori put za navigaciju, može se uvidjeti kako je u tome slučaju čvor manje važna informacija koja služi za lakšu interpretaciju brida kao osnovnog nositelja informacija.

2.2. Vrste čvorova

Postoje dvije vrste čvora, izolirani i krajnji. Izolirani je onaj koji nije povezan ni s jednim drugim čvorom, to jest, onaj koji nema niti jedan brid povezan s njime. Takvome je čvoru stupanj jednak nuli. Krajnji čvor je onaj čiji je stupanj jednak jedan. Ako je graf tvoren od čvorova od kojih niti jedan nema barem jedan brid, takav graf se naziva praznim grafom. (Deo, 2017, str. 8-9)

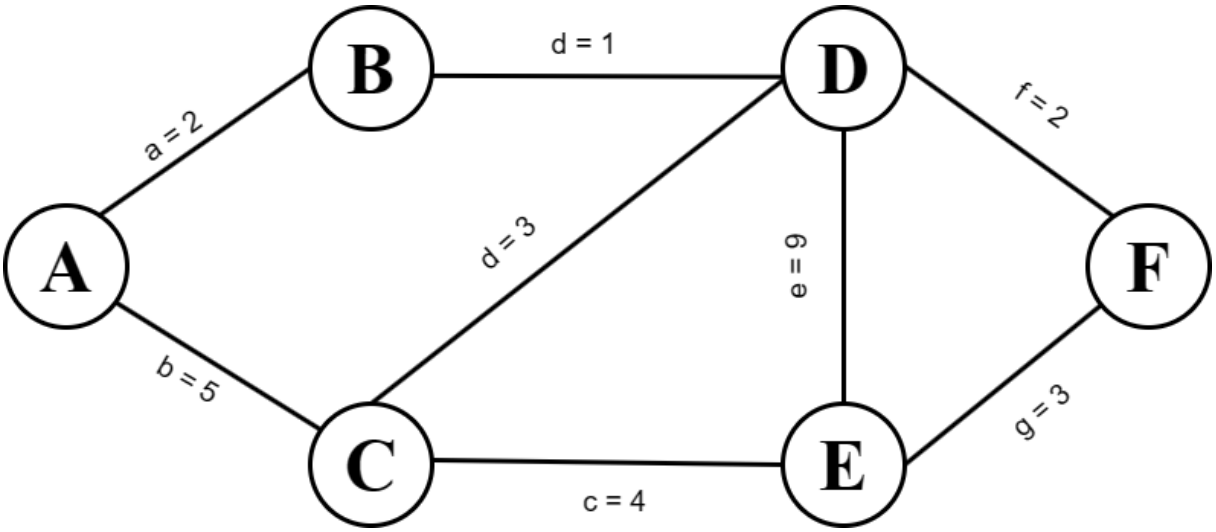


Slika 2 Tipovi čvorova

Kao što je vidljivo na slici 2, čvorovi *A* i *D* imaju stupanj jedan, svaki od njih ima samo jedan brid povezan s njima, dakle ta dva čvora su krajnji čvorovi. Čvor E je izolirani čvor, ovaj čvor nema niti jedan brid povezan s njime. Čvorovi *B* i *C* imaju stupanj jednak dva, čvor *B* sa sobom ima povezane bridove *a* i *b*, a čvor *C* je povezan s bridovima b i c.

2.3. Težinski graf

Bridovi koji povezuju dva čvora ponekad mogu imati sebi pridružen ne negativni broj. Takav jedan graf naziva se težinski graf. (Wilson, 1979, str. 39) Težinska vrijednost svakoga brida ukazuje na „težinu“ prolaska tim bridom između dvaju čvorova koje on povezuje. Informacija o težini brida vrlo je korisna kada su u pitanju algoritmi za pronalazak najkraćeg puta. Težina jednog brida, ovisno o tipu grafa i kakvu informaciju on točno prezentira, može označavati više tipova prohodnosti. Stoga se u primjeni grafa za navigaciju vozila u prometu informacija o težini može označavati duljinu ceste ili vrijeme koje je potrebno kako bi vozilo prošlo kroz tu ulicu ili komad ceste.

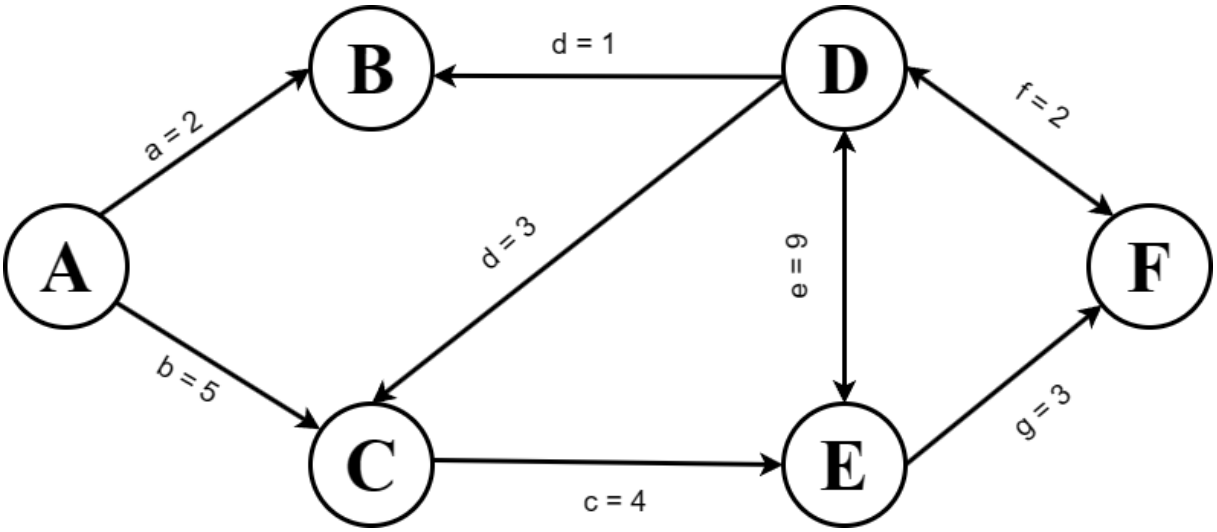


Slika 3 Težinski graf

Slika 3 prikazuje primjer jednog težinskog grafa. Kao što je vidljivo, svaki brid uz sebe ima pridruženu informaciju o težini. Bridovi koji izlaze iz čvora *A* imaju drugačije vrijednosti težine, brid *a* ima težinu 2, a brid *b* težinu 5. Stoga je dakle zaključivo kako je put između čvorova *A* i *B* kraći od puta između *A* i *C*. S ovom informacijom, moguće je naći najkraći put između dva čvora. Svaka kombinacija puta kroz povezane čvorove dati će valjanu rutu, kretajući se kroz bridove zbrajaju se težine tih bridova te se u konačnici dobiva ukupna težina puta. Put s najmanjom težinom bit će optimalni put kroz graf. Put između čvorova *A* i *F*, imat će ukupnu težinu vrijednosti 5, i kretat će se putem *A* -> *B* -> *D* -> *F*. Krene li se iz izvorišta *A*, prateći bridove s najmanjom težinom, uzimajući u obzir sve bridove između njih, dolazi se do već spomenutog rješenja koje je u ovome grafu optimalno za put između *A* i *F*.

2.4. Digraf

Digraf, ili usmjereni graf, je tip grafa u kojem bridovi između čvorova nose informaciju o smjeru. (Wilson, 1979, str. 100)



Slika 4 Usmjereni graf

Usmjereni graf nosi zadnju važnu informaciju koje je potrebna kako bi se mogao konstruirati graf za upotrebu u primjeni algoritama za pronalazak najkraćeg puta. Kao što je vidljivo na slici 4, svaki brid ima svoji smjer koji je ilustriran strelicom na kraju brida. Čvor *A* povezan je s bridovima *a* i *b*, te ti bridovi imaju svoj smjer od *A* prema *B* za brid b, i od *A* prema *C* za brid *b*. Ovi bridovi su jednosmjerni. Brid *c* koji povezuje čvorove *B* i *D* je dvosmjeran, dakle put između čvorova *B* i *D* je prohodan s obje strane. Put kroz čvorova *A* i *F* u ovakvome grafu imat će manji broj mogućih kombinacija puteva u usporedbi s grafom definiranim u slici 3. U grafu definiranom na slici 3, svaki brid je implicitno definiran kao dvosmjeran brid, a u usmjerenom grafu na slici 4 svaki brid ima definiran smjer, te nije svaki brid dvosmjeran. Optimalan put između čvorova *A* i *F* u ovome grafu neće biti isti kao optimalan put u grafu na slici 3. Brid *d* u ovome grafu nije dvosmjeran, te on vodi u smjeru isključivo iz čvora *D* u *B*, ali ne i obrnuto. Krene li se iz čvora *A*, put do *B* je konačan pošto je *B* krajnji čvor koji prema vani nije povezan ni s jednom drugim čvorom. Iz *C* moguće je kretati se isključivo prema *E*, a iz *E* je moguće kretati se prema *D* i *F*. Uzimajući u obzir kako je težina brida prema *F* manja nego prema *D*, put se usmjerava prema *F*. Čvor *F* je ciljni, te se stoga dolazi do optimalnog puta koji prolazi čvorovima *A -> C -> E -> F*, i bridovima *b* težine 5, *c* težine 4 *i g* težine 3. Optimalan stoga put od *A* do *F* u ovome grafu prolazi putem *A -> C -> E -> F*, te ima težinu 12. Ova informacija u smjeru brida, kada će se algoritmi koristiti u svrhu navigacije, moći će se interpretirati kao jednosmjerna i dvosmjerna ulica.

2.4.1. Primjena teorije grafa

Praktična primjena teorije grafa je vrlo široka, a jedna od važnijih primjena, uz onih opisanim detaljnije u ovome radu, je u komunikacijskim mrežama.
Komunikacijska mreža je skup terminala, veza i čvorova koji su povezani kako bi se omogućila telekomunikacija između korisnika terminala. Svaki terminal u mreži mora imati jedinstvenu adresu kako bi poruke ili veze mogle biti preusmjerene ispravnim primateljima. Kolekcija adresa u mreži naziva se adresni prostor. Svaka komunikacijska mreža ima tri osnovne komponente: terminale (ulazne i izlazne točke u mreži), procesore (koji daju funkcije za kontrolu prijenosa podataka) i prijenosne kanale. Komunikacijska mreža ima za cilj prijenos paketa podataka između računala, telefona, procesora, ili drugih uređaja. Izraz paket odnosi se na neke podatke fiksne veličine, najčešće duljine 256 bajtova ili 4096 bajtova. (Badwaik, 2020, p. 535)

Teorija grafa od pomoći je i u biologiji. Biomolekularni entiteti (kao što su kromosomi, proteini ili metaboliti) služe kao "čvorovi" u biološkim mrežama, dok "bridovi" koji povezuju čvorove predstavljaju interaktivnu, fizičku ili kemijsku interakciju između čvorova. Metaboličke mreže, i mreže protein-protein interakcije još su jedna od načina upotrebe teorije grafa. (Ali, 2021, p. 200)

3. Dijkstrin Algoritam

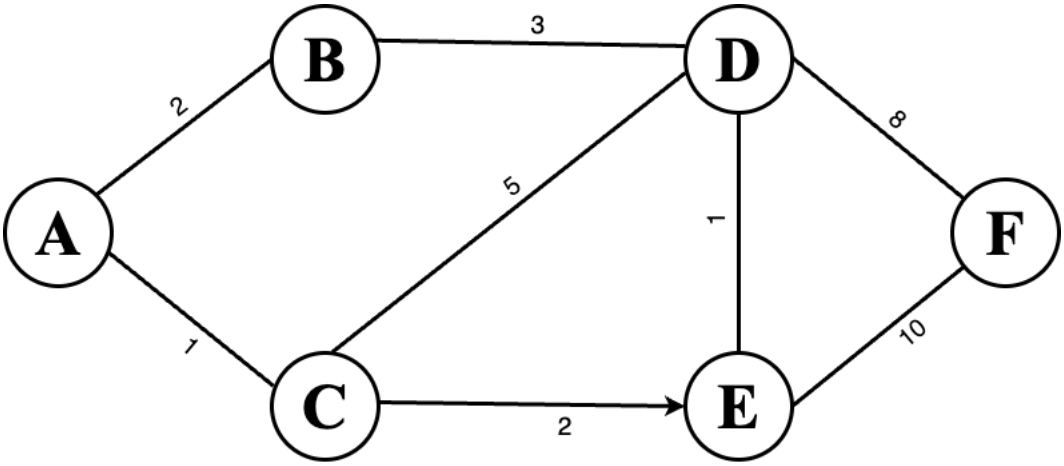
Jedan od najpoznatijih i najutjecajnijih algoritama za pronalazak najkraćeg puta je onaj Edsger Wybe Dijkstre. Svojim utjecajnim radom, Dijkstra je htio riješiti problem pronalaska najkraćeg puta između čvorova u grafu čije su međusobne težine bridova pozitivne vrijednosti. Dijkstrin algoritam jedan je od najvažnijih i najutjecajnijih algoritama, te je njegov algoritam, i svi ostali algoritmi koji su bazirani na njegovom, uvelike doprinijeli razvoju mnogih računalnih i ne računalnih polja kao što su telekomunikacija, navigacija, logistika, biologija, kemija i sl.

3.1. Kratki opis algoritma

Dijkstrin algoritam radi s dva seta čvorova. Prvi set sadrži posjećene čvorove dok drugi set sadrži one koje se tek mora posjetiti. Počevši sa startnim čvorom, uzimaju se svi njegovi susjedni povezani čvorovi koji su neposjećeni, te počevši s onim s kojim trenutni ima najmanju težinu brida, mjeri se zbroj dosadašnje težine i težine tog susjednog čvora. Ako se naiđe s drugog smjera na neki čvor za koji je već izračunata duljina iz nekog prethodnog smjera, i taj novi put bude kraći, ažurira se taj novi najkraći put do njega. Nakon što su obiđeni svi susjedi, zapisuje se u set posjećenih čvorova, te iz njega više ne izračunavaju putevi ako se ponovno na njega naiđe. (Dijkstra, 1959, str. 269-271)

3.2. Princip rada algoritma

S definiranim grafom na slici 5, uzimajući čvor *A* kao početni, i čvor *F* kao završni, postupak Dijkstrinog algoritma sa zadanim parametrima će biti sljedeći.



Slika 5 Graf za izračun Dijkstrinog algoritma

Definiraju se dva seta čvora, set posjećenih, i set neposjećenih. U set neposjećenih na samome početku se stavlja početni čvor *A.*

Nadalje, definira se tablica 1, s težinama između svakog čvora grafa s početnim. Pošto je A početni čvor, definirat će se njegova težinu prema samome sebi kao vrijednost 0. Sve ostale težine definirat će se u prvome koraku kao beskonačne.

Tablica 1 Težine između čvorova u izračunu Dijkstre

Čvor	Težina od A	Prethodni čvor
A	0	
B	∞	
C	∞	
D	∞	
E	∞	
F	∞	

3.2.1. Korak 1.

Posjećuju se svi čvorovi povezani s A, i zapisuju se njihove težine u tablicu težina, za *B* to će biti vrijednost **2**, a za *C* to će biti vrijednost **1**. Kao prethodni čvor s najmanjom težinom, objema čvorovima se zapisuje *A*. Čvor *A* se prebacuje iz seta neposjećenih u set posjećenih, a *B* i *C* se stavlja u set neposjećenih.

Tablica 2 Težine nakon konačnog izračuna čvora A

Čvor	Težina od A	Prethodni čvor
A	0	/
B	2	A
C	1	A
D	∞	
E	∞	
F	∞	

3.2.2. Korak 2.

U drugome koraku se uzima čvor iz seta neposjećenih koji ima najmanju težinu, to je *C*.
Čvor *C* je povezan u smjeru *D*. i *E*. Težina povezanosti *C* prema *D* je **5**, te za težinu *D* se uzima njegova težina prema C, i zbraja se s težinom *C*. Težina *D* bit će dakle **5 + 1 = 6.**
Težina između trenutnog čvora *C* i *E* iznosi **2**, zbrajajući ju s dosadašnjom težinom *C*, dobiva se iznos **3**.
C se prebacuje u set posjećenih, a *D* i *E* u set neposjećenih.

Tablica 3 Težine nakon konačnog izračuna čvora C

Čvor	Težina od A	Prethodni čvor
A	0	/
B	2	A
C	1	A
D	6	C
E	3	C
F	∞	

3.2.3. Korak 3.

U trećem koraku uzima se iz seta neposjećenih čvor s najmanjom težinom koji nije prisutan u setu posjećenih, a to je *B*.

Povezani čvorovi čvora *B* su *A* i *D*. Uzimajući u obzir kako je *A* u setu posjećenih, izračunava se težina *D*. Težina između čvorova *B* i *D* iznosi **3**, a dosadašnja najmanja težina *B* iznosi **2**, stoga nova vrijednost težine *D* u ovom smjeru iznosi **5**. Pošto je novo izračunata vrijednost težine manja od dosadašnje vrijednosti *D*, ažurira se nova težina **5**, i novi najkraći prethodni čvor *B*. Čvor *B* se stavlja u set posjećenih.

Tablica 4 Težine nakon konačnog izračuna čvora B

Čvor	Težina od A	Prethodni čvor
A	0	/
B	2	A
C	1	A
D	5	B
E	3	C
F	∞	

3.2.4. Korak 4.

Idući neposjećeni čvor s najmanjom težinom je *E*.

E je povezan s *C*, *D* i *F*. Čvor *C* nalazi se u setu posjećenih, stoga se računaju težine prema *D* i *F*. Težina između *E* i *D* iznosi **1**, a težina *E* prema početnom čvoru iznosi **3**, nova izračunata težina *D* iznosi **4**. Dosadašnja težina čvora *D* iznosi **5**, stoga se zamjenjuje s novom vrijednosti jer je kraća. Prethodni čvor najkraćeg smjera bit će *E*.

Čvor *E* i *F* imaju težinu povezanosti u vrijednosti **10.** Težina *F* od čvora *A* je stoga vrijednost njihove težine i težine najkraćeg puta od *A* i *E*. Težina čvora *F* od *A* iznosi **13**. *F* se dodaje u set neposjećenih, a *E* se prebacuje u set posjećenih.

Tablica 5 Težine nakon konačnog izračuna čvora E

Čvor	Težina od A	Prethodni čvor
A	0	/
B	2	A
C	1	A
D	4	E
E	3	C
F	13	E

3.2.5. Korak 5.

Čvor u setu neposjećenih s najmanjom vrijednosti težine je *D*.

Čvor *D* je povezan s B, C, E i F. Čvorovi B, C i E se nalaze u setu posjećenih te se stoga preskaču.

Težina između *D* i *F* iznosi **8**, a težina *D* prema početnom čvoru iznosi **8**. Njihovim zbrojem dobiva se vrijednost **12**. Nova izračunata težina za *F* manja je od prethodne se ažurira najmanja vrijednost i prethodni najkraći čvor prema početnom.

Čvor *D* se prebacuje u set posjećenih.

Tablica 6 Težine nakon konačnog izračuna čvora D

Čvor	Težina od A	Prethodni čvor
A	0	/
B	2	A
C	1	A
D	4	E
E	3	C
F	12	D

Jedini preostali je ciljni čvor *F*, no pošto je *F* povezan jedino s čvorovima koji su u setu već posjećenih, algoritam je završen.

U konačnici se dolazi do konačnog rezultata koji govori kako je najkraći put između *A* i *F* u zadanome grafu, težine **12**. Gledajući prethodne najkraće čvorove iz *F* u *A*, dolazi se i do najkraće putanje koja se kreće: ***A -> C -> E -> D -> F.***

3.3. Problemi i ograničenja algoritma

Prvi očiti problem Dijkstrinog algoritma je ograničenje algoritma na isključivo pozitivne vrijednost i težine bridova između čvorova. Ako bi se htjelo izračunati put između čvorova u grafu koji sadrži bridove negativnih vrijednosti, primjena Dijkstrinog algoritma u tom slučaju neće biti adekvatna.

Dijkstrin algoritam, po svojem principu rada, posjećuje svaki čvor koji se nalazi u setu čvorova za posjetu. Iako taj način rada garantira da će algoritam uvijek izračunati, ako je to moguće, najkraći put između početnog i krajnjeg čvora, ovisno o veličini grafa i povezanosti čvorova taj proces može vremenski biti vrlo neefikasan. Stoga je primjena Dijkstrinog algoritma vrlo ograničena i primjenjuje se u onim sustavima u kojima je prioritet točnost, a ne brzina.

Još jedno veliko ograničenje Dijkstrinog algoritma leži u činjenici da algoritam funkcionira isključivo sa statičkim podacima, grafovi u kojima se težine bridova mijenjaju tijekom izvođenja algoritma, neće biti mogući s Dijkstrinim algoritmom.

U konačnici, Dijkstrin algoritam vrlo je važan i znamenit rad koji je u svojoj primjeni, a i primjeni ostalih algoritama koji vuku podrijetlo iz njega, ostavio nemali trag u razvoju mnogih tehnologija, pogotovo zahvaljujući svojoj jednostavnosti i točnosti. Iako nije prikladan za svaku situaciju, mnogi algoritmi, neki koji će se analizirati u nastavku ovoga rada, se baziraju i unaprjeđuju na Dijkstrinom algoritmu, i koristeći ga kao bazu, specijaliziraju za mnoge vrste mogućih problema u pronalasku puteva kroz mnoge vrste grafova.

3.4. A* Algoritam

A* algoritam je algoritam koji se bazira na djelu E.W Dijkstre. Poput Dijkstrinog algoritma, A* služi pronalasku najkraćeg puta u grafu. Glavna odlika A* algoritma je heuristička funkcija kojom se algoritam usmjerava prema cilju. Tim usmjeravanjem, algoritam za razliku od Dijkstrinog algoritma ne prolazi kroz sve čvorove u grafu nego se ciljano kreće najbližim čvorovima dok ne stigne do cilja. (Russell & Norvig, 2016, str. 376)

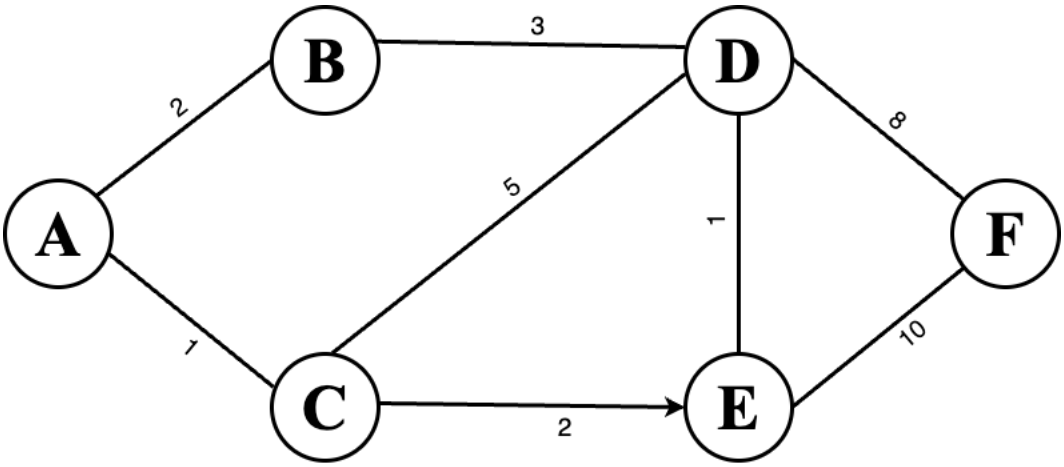
3.5. Kratki opis algoritma

Poput Dijkstrinog algoritma, A* algoritam čuva dva seta čvorova, posjećenih i onih koji tek trebaju biti posjećeni. Uz težinski podatak između čvorova u grafu, za izračun težine između njih se koristi heuristička funkcija. Tom funkcijom se dobiva novi iznos težine. Cilj te funkcije je usmjeravanje prema cilju. Uspješnost algoritma ovisi o efikasnosti te heurističke funkcije. Ako je ona premalo efikasna, i ne upotrijebi li se na optimalan način, algoritam će se ponašati uvelike poput Dijkstrinog algoritma, čime će se izgubiti potencijalna optimizacija i potencijalno ubrzanje. Ako funkcija premaši izračun težine, može se doći do krivog rezultata, što će učiniti algoritam beskorisnim. (Hart, Nilsson, & Raphael, 1968, str. 100-107)

3.6. Princip rada algoritma

Prije no što se opise rad algoritma, potrebno je definirati heurističku funkciju. Kao što je već navedeno, uspješnost algoritma uvelike ovisi o heurističkoj funkciji. U ovome primjeru bit će definirana heuristička funkcija *h* kao izračun najkraćeg puta od početnog čvora do trenutnog Dijkstrinim algoritmom. Težina čvora *d* će stoga biti zbroj težine između čvorova *g* i težine izračunate Dijkstrom *h*.

$$d = g + h(x)$$



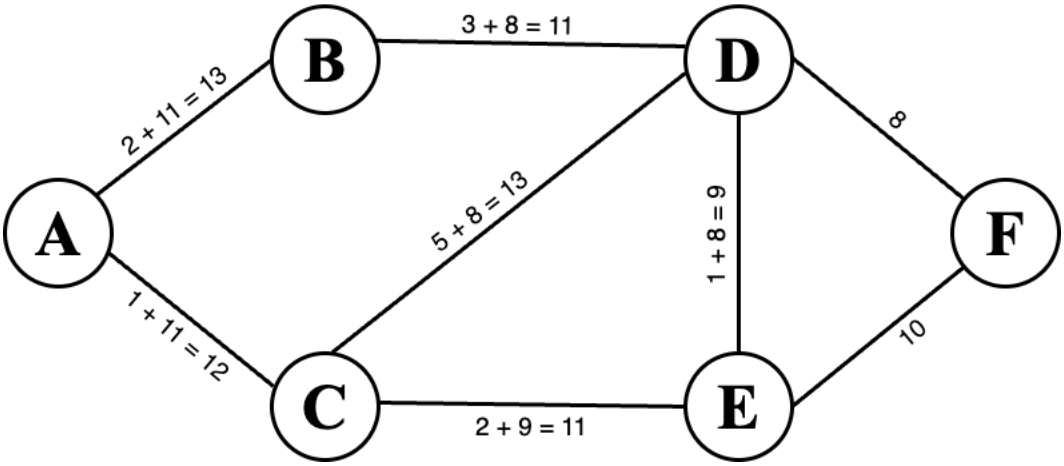
Slika 6 Graf za izračun A algoritma prije primjene heurističke funkcije*

Uzme li u obzir isti graf kao i u primjeru Dijkstre, dobiva se sljedeća tablica 7, s vrijednostima težina između čvorova. Kao početni čvor opet se uzima *A*, a kao ciljni uzima se *F*.

*Tablica 7 Vrijednosti težina između čvorova s heurističkom funkcijom za A**

Čvor	*h(čvor)*	*d*
A	0	0
A – B	2 + 11	13
A – C	1 + 11	12
B – D	3 + 8	11
C – D	5 + 8	13
C – E	2 + 9	11
D – E	1 + 8	9
D – F	8	8
E – F	10	10

Znajući nove vrijednosti težina, nastavlja se istim postupkom kao i kod Dijkstrinog algoritma.



Slika 7 Graf za izračun A algoritma s primijenjenom heurističkom funkcijom*

3.6.1. Korak 1.

Posjećuju se svi čvorovi povezani sa startnim čvorom *A*. Izračunavaju se težine između njih. Čvor *A* se stavlja u set posjećenih, a *B* i *C* stavljaju se u set neposjećenih.

Tablica 8 Težine nakon konačnog izračuna čvora A

Čvor	Težina od A	Prethodni čvor
A	0	/
B	13	A
C	12	A
D	∞	
E	∞	
F	∞	

3.6.2. Korak 2.

Uzima se čvor s najmanjom težinom u setu neposjećenih, a to je *C*.

Posjećujemo povezane čvorove *D* i *E*.

Težina između *C* i *E* iznosi **11**, a trenutna težina *C* od početnog iznosi **12**. Težina *E* od početnog stoga iznosi **11 + 12 = 23**. Prethodni čvor je *C*.

Težina između *C* i *D* je **13**. Zbrajanjem te težine s prethodnom vrijednošću *C* dobiva se težina **12 + 13 = 25**.

Tablica 9 Težine nakon konačnog izračuna čvora C

Čvor	Težina od A	Prethodni čvor
A	0	/
B	13	A
C	12	A
D	25	C
E	23	C
F	∞	

3.6.3. Korak 3.

U trećem koraku uzima se čvor s najkraćom težinom koji je u setu neposjećenih, a to je *B*.

Posjećuju se čvorovi povezani s *B*, a to su *D* i *A*. S obzirom na to da je *A* u setu posjećenih, uzima se *D*.

Težina između *B* i *D* je **11**, a težina između *B* i početnog *A* je **13**. Težina između početnog i čvora *D* iznosi **24**. Novi iznos manji je od prethodnog iznosa, te se ažuriraju vrijednosti, vidljivi u tablici 10. Novi prethodni čvor čvora *D* je *B*. *B* se stavlja u set posjećenih.

Tablica 10 Težine nakon konačnog izračuna čvora B

Čvor	Težina od A	Prethodni čvor
A	0	/
B	13	A
C	12	A
D	24	D
E	23	C
F	∞	

3.6.4. Korak 4.

U četvrtom koraku se uzima sljedeći čvor s najmanjom težinom, a to je *E*.
Čvor *E* povezan je s *D* i *F*. *F* je krajnji je čvor te je ovaj korak posljednji s obzirom na to da se nakon izračuna s krajnjim ciljem dolazi do konačnog rezultata.

Težina između *E* i *F* iznosi **10**, a težina između *E* i početnog čvora *A* iznosi **23**. Izračunom težine **33** do krajnjeg čvora se završava algoritam.

Tablica 11 Krajnje težine nakon konačnog izračuna čvora F i kraja algoritma

Čvor	Težina od A	Prethodni čvor
A	0	/
B	13	A
C	12	A
D	24	D
E	23	C
F	E	33

Putanja A* algoritma je **A -> C -> E -> D -> F**. Putanja je identična onom koja je izračunata Dijkstrinim algoritmom.

3.7. Važnost odabira heuristike

Odabir heurističke funkcije najvažniji je element efikasnosti A* algoritma. U ovome primjeru heuristička funkcija koja je definirana kao duljima čvora do krajnjeg čvora Dijkstrinim algoritmom nije previše efikasna. Postoje mnoge heurističke funkcije koje variraju u svojoj efikasnosti, te je njihov izbor odlučujući element u A* algoritmu.

3.8. Problemi i ograničenja algoritma

A* dijeli mnoge probleme i ograničenja s Dijkstrinom algoritmom. Jedna od prednosti je to što algoritam završava jednom kada naiđe na završni čvor. Time se štedi na vremenu koji Dijkstrin algoritam potroši na prelazak preko ostalih čvorova koji nisu bitni za izračun konačne rute. Druga prednost algoritma je ta što heuristička funkcija može imati izračun u dinamičkom okruženju, što ovisno o funkciji može biti i negativna karakteristika, ovisno o točnosti i efikasnosti funkcije. No s efikasnom funkcijom koje će statički izračunati vrijednosti težina, algoritam može višestruko ubrzati izračun rute u usporedbi s Dijkstrinim algoritmom.

4. D* Lite

D* Lite algoritam je algoritam baziran na A* algoritmu. Iako je nazvan D*, ne bazira se na D* algoritmu, no ponaša se na sličan način. D* Lite algoritam bazira se na inkrementalnoj uporabi heuristike, te nekoliko puta traži najkraći put. Vrijednosti koje se koriste pri izračunu udaljenosti između čvorova su vrijednosti *g* i *rhs*. Vrijednost *g* govori koliko je trenutni čvor udaljen od ***ciljnog***. Vrijednost *rhs* izračunava se uzimajući u obzir vrijednost *g* njegovih prethodnih čvorova koje definiramo kao one koji su usmjereni prema njemu. Algoritam traži put u zadanom radijusu od trenutnog, te sa svakom promjenom u grafu ponovno računa težine čvorova koji imaju promijenjenog kao prethodnika. Za razliku od A* algoritma, D* Lite algoritam se kreće od ciljnog prema startnom čvoru. (Sven, Maxim, & David, 2004, str. 93-146)

4.1. Kratki opis algoritma

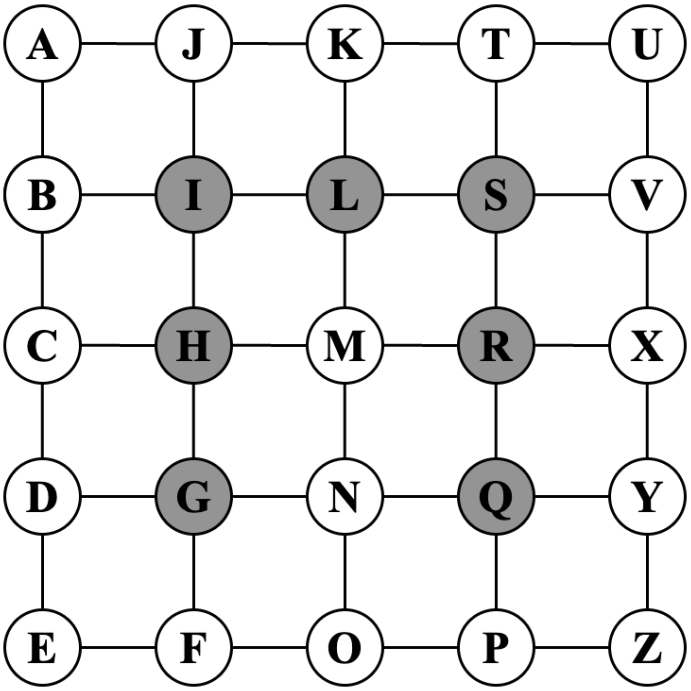
Algoritam se započinje s inicijalizacijom u kojoj postavljamo *rhs* vrijednost ciljnog čvora na 0, a *g* vrijednost, kao i *g* i *rhs* vrijednosti ostalih čvorova u grafu na beskonačnost.
Počinje se s prethodnim čvorovima ciljnog, te im se izračunavaju *g* i *rhs* vrijednosti. Koristi se prioritetni red kako bi se znalo koji se čvor idući treba izračunati. U prioritetni red se stavljaju oni čvorovi koji su nekonzistentni.
Konzistentnost se definira na sljedeći način:

- Ako je *g* vrijednost jednaka vrijednosti *rhs*, čvor je **konzistentan**
- Ako je *g* vrijednost različita od *rhs* vrijednosti, čvor je **nekonzistentan**
    - Ako je *g* vrijednost veća od *rhs* vrijednosti, čvor je **preko konzistentan**
    - Ako je *g* vrijednost manja od *rhs* vrijednosti, čvor je **pod konzistentan**

Ako dođe do promjena u grafu, ako se promijeni težina između čvorova, ili neki čvor koji se nalazi na putu između početnog i ciljnog postane neprohodan, izračunavaju se ponovno *g* i *rhs* vrijednosti svih onih koji su pogođeni. Pogođeni čvorovi su oni kojima su se *g* i *rhs* vrijednosti bazirale na vrijednostima promijenjenog čvora.

4.2. Princip rada algoritma

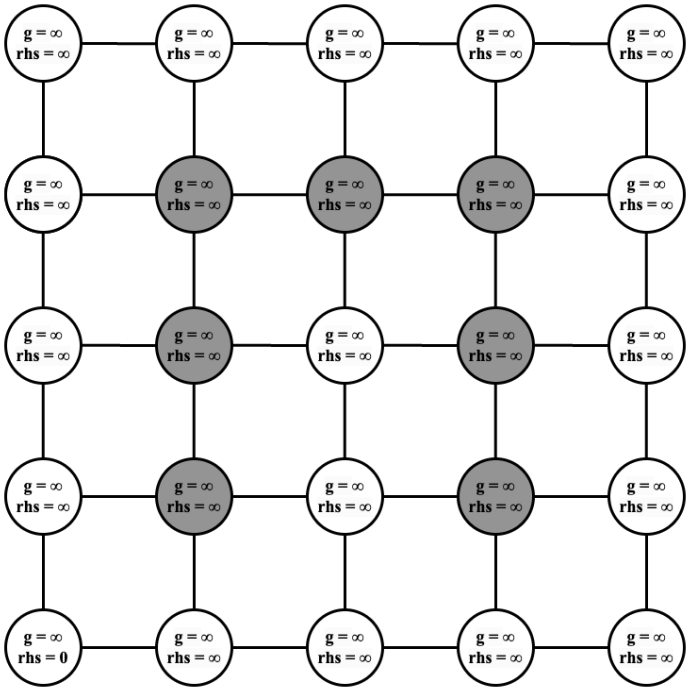
Za prikaz D* Lite algoritama, bit će korišten graf širine 5 polja i visine 5 polja. Svaki čvor povezan je sa susjednim u 4 smjera (gore, dolje, lijevo, desno), te je težina bridova između njih vrijednost jednaka 1. Čvorovi koji nisu prohodni bit će vizualno definirani tamnijom nijansom. Za početnu točku izračuna najkraćeg puta bit će uzeti čvor *U*, a za ciljni bit će uzet *E*.



Slika 8 Graf za izračun D Lite algoritma*

4.2.1. Korak 1.

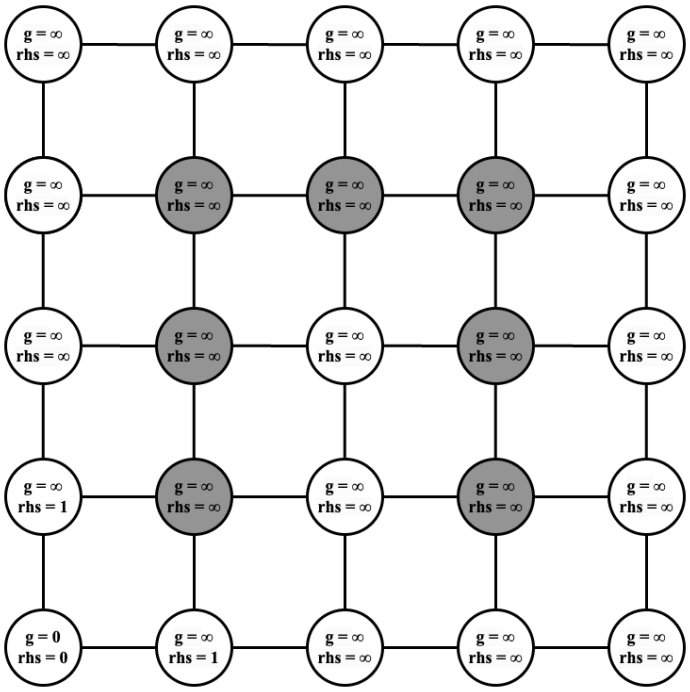
U prvome koraku se inicijaliziraju vrijednosti čvorova grafa. Za ciljnu točku, vrijednost *g* se za sada definira kao beskonačnost, dok se vrijednosti *rhs* dodjeljuje **0**, jer je trenutna pozicija ciljna, i udaljenost od samoga sebe je **0**. Provjerava se je li trenutni čvor konzistentan. S obzirom na to da se *g* i *rhs* vrijednost ne poklapaju, *E* se stavlja u prioritetni red.



Slika 9 Vrijednosti čvorova grafa nakon inicijalizacije

4.2.2. Korak 2.

Uzima se prvi čvor u prioritetnom redu. U ovome slučaju to je čvor E. Provjerava se jesu li vrijednosti preko konzistentne, ako jesu, postavlja se *g* vrijednost jednaku na vrijednost *rhs*. S obzirom na to da je *g* vrijednost (∞) veća od *rhs* vrijednosti (0), postavlja se $g = rhs$, $\boldsymbol{g = 0}$. Nakon izračuna vrijednosti prvoga čvora, sakupe se svi prohodni koji prethode trenutnom, a to su *D* i *F*, te im se za *rhs* vrijednost dodjeljuje udaljenost od trenutnog čvora *E*, u ovome slučaju to je vrijednost 1.

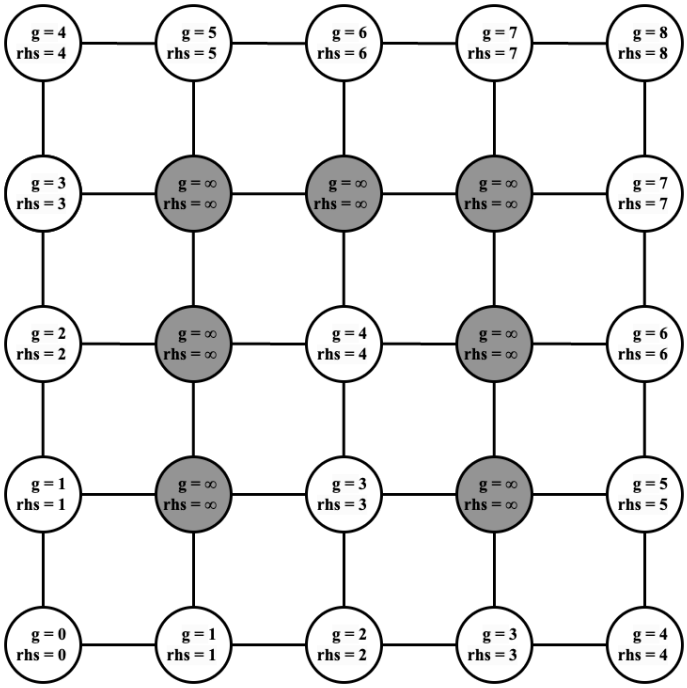


Slika 10 Vrijednosti čvorova grafa nakon izračuna vrijednosti prvog čvora

4.2.3. Korak 3.

Ponavlja se **korak 2.** za svaki čvor u prioritetnom redu dok se ne izračunaju sve vrijednosti svih čvorova i dok svi ne postanu konzistentni.

Kada su izračunati svi čvorovi dobiva se kompletni graf sa svim vrijednostima.



Slika 11 Vrijednosti grafa nakon izračuna vrijednosti svih čvorova

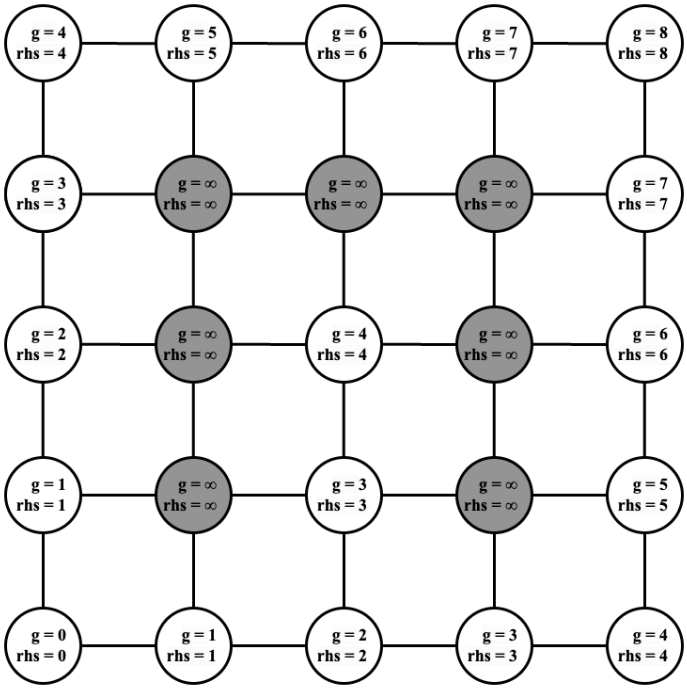
4.2.4. Korak 4.

Počinje se iz početnog čvora i kreće se prema susjednom s najmanjom *g* vrijednosti. Ako se dođe do ciljnog bez promjene vrijednosti nekog od čvora na putu prema cilju, algoritam je gotov i dobiva se krajnji put.

Ako se jedan od čvorova na putu do cilja promjeni i postane neprohodan, potrebno je ažurirati sve okolne vrijednosti prije no što se nastavi prema cilju.

4.2.5. Korak 5.

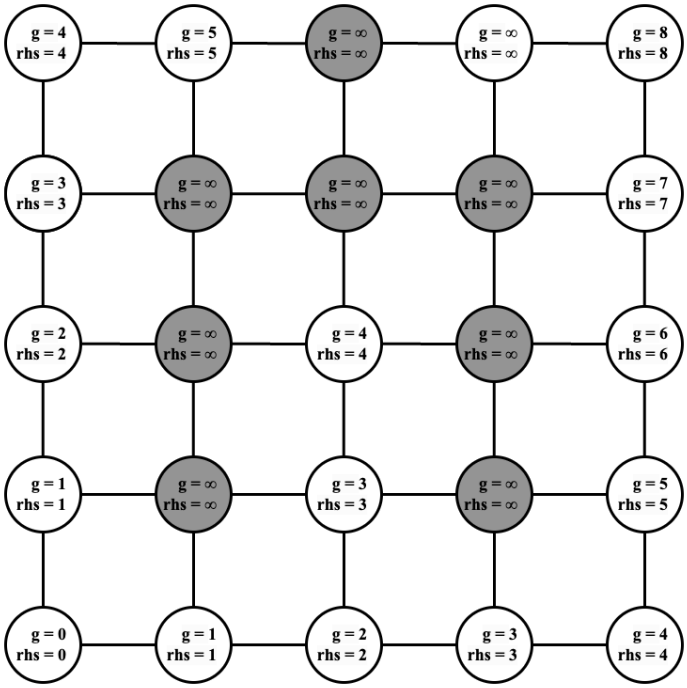
Kretanjem prema cilju gornjom rutom, čvor *K* postaje neprohodan. Počinje ažuriranje njegovih okolnih čvorova dok se ne dobiju nove vrijednosti. Njegova *rhs* vrijednost postavlja se na beskonačnost, postavlja se na prioritetni red, i ponovno se ponavlja **korak 2.** dok se ne isprazni prioritetni red.



Slika 12 Vrijednosti grafa nakon što je čvor K postao neprohodan

4.2.6. Korak 6.

Nakon što je **korak 2.** ponovljen dok se prioritetni red nije ispraznio, dobiva se graf s novim vrijednostima.



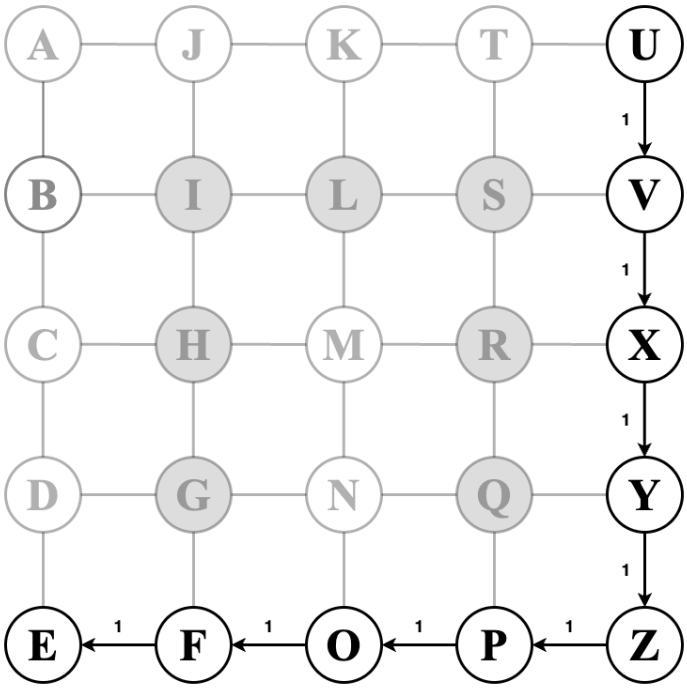
Slika 13 Krajnji graf nakon ažuriranja čvorova

4.2.7. Korak 7.

S ovim krajnjim stanjem grafa, počevši s početnim, prati se put kroz čvorove s najmanjom *g* vrijednošću. Krajnji put kroz ovaj graf bit će:

U -> V -> X -> Y -> Z -> P -> O -> F -> E

Uzimajući u obzir kako je u ovome primjeru duljina između svakog čvora bila jednaka **1**, *g* vrijednost početnog čvora govori i o duljini krajnjeg puta, koja iznosi **8**.



Slika 14 Krajnja putanja kroz graf

4.3. Prednosti Algoritma

Kako D* Lite napreduje, on iterativno usavršava svoja predviđanja funkcije troška i putanje koje generira, u konačnici konvergirajući prema optimalnom rješenju. To znači da iako nije zagarantirano da će D* Lite pronaći optimalni put pri svakoj kalkulaciji rute, neprestano se približava pronalasku kako se izvode daljnje iteracije. Ovo je značajna prednost u odnosu na algoritme s fiksnim aproksimacijama koji ne konvergiraju prema optimalnom rješenju.

4.4. Ograničenja Algoritma

Unatoč brojnim prednostima, D* Lite ima nekoliko potencijalnih nedostataka koji ga mogu učiniti neprikladnim za određene aplikacije traženja puta. Kada dođe do promjene prohodnosti nekoga čvora, broj čvorova koji se moraju ažurirati ponekad može jako brzo narasti, cime se algoritam višestruko usporava. Stoga iako vrlo efektivan, algoritam nije optimalan u situacijama u kojima se u grafu velike količine čvorova često mijenjaju.

4.5. Primjena

S obzirom na njegovu jedinstvenu mješavinu karakteristika i potencijalnih ograničenja, D* Lite je najprikladniji za niz aplikacija za pronalaženje puta gdje se optimalna rješenja mogu postići iterativnim usavršavanjem putanje. D* Lite algoritam privukao je pažnju u području robotike zbog svoje sposobnosti dinamičkog ponovnog izračunavanja putanja kao odgovor na promjene u okruženju i njegove konvergencije do optimalnog rješenja. Robotske platforme zahtijevaju učinkovite i fleksibilne metode pronalaženja puta za navigaciju i istraživanje okoline, posebno kada se pojave nepredviđene prepreke ili treba uzeti u obzir dinamičke uvjete, kao što su dinamičke karte troškova ili izbjegavanje gužve. U takvim slučajevima, D* Lite inkrementalno pretraživanje i srednja heuristika čine ga prikladnim za ove dinamičke scenarije.

5. HPA* Algoritam

HPA* (Hierarchical Path-Finding A*) algoritam je tehnika osmišljena za rješavanje izazova pronalaženja putanje u kartama u računalnim video igrama. Posebnost ovog algoritma leži u brzom dinamičnom izračunu puta. Algoritam se baš poput D* Lite algoritma baziran na inkrementalnom pronalasku puta, te je jako efikasan algoritam u dinamičkim grafovima. Veliki problemi koje je algoritam riješio su bili problemi brzog izračuna puta između dviju točaka velikog broj likova u video igrama, koji se svake sekunde mogu pojaviti u velikom broju na ekranu. (Harabor & Adi, 2008)

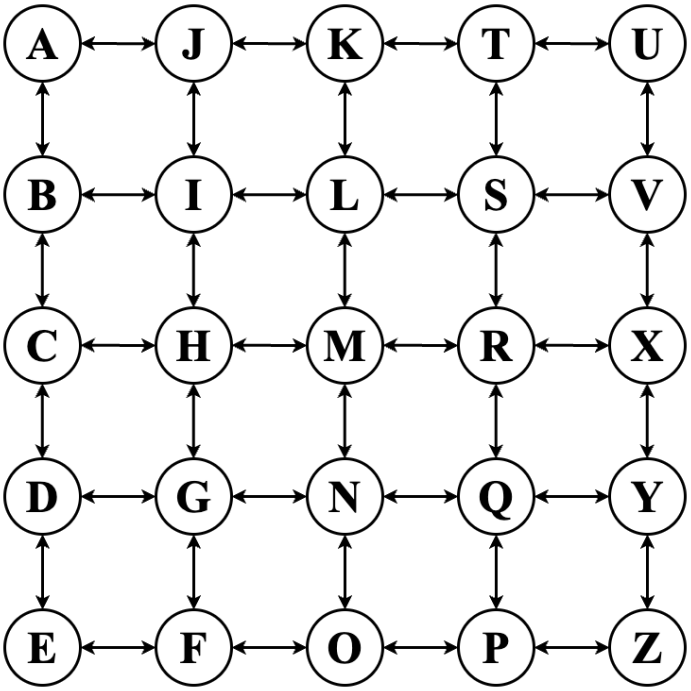
5.1. Kratki opis algoritma

HPA* algoritam dijeli mapu, tj. graf, na veće dijelove, odnosno klastere. Svaki klaster ima jednu ili više ulaznih i izlaznih točaka koje funkcioniraju kao čvorovi. Statički se graf podijeli na optimalni broj klastera, te se izračunaju optimalni putevi između svakog ulaznog i izlaznog čvora tog klastera, kao i između para ulaznih i izlaznih čvorova između dva susjedna klastera. Njihovu povezanost iskoristit će se kako bi se napravio apstraktni graf koji će za čvorove imati samo ulazne i izlazne čvorove klastera, a duljina puteva između njih bit će vrijednost težine između njih.

Pri izračunu puta algoritmom, prvo je potrebno povezati početnu točku s ulaznim i izlaznim točkama u klasteru u kojemu se početna točka nalazi. Nakon što se poveze početna točka, potrebno je na isti način povezati ciljnu s ulaznim i izlaznim točkama u njenom klasteru. Nakon što su ulazne i izlazne točke povezane s točkama u njihovim klasterima, ažurira se apstraktni graf s početnom i ciljnom točkom i njihovim putevima do izlaznih i ulaznih točaka njihovih klastera. Time je dobiven cjeloviti apstraktni graf koji se dalje može koristiti pri izračunu puta u grafu.

5.2. Princip rada algoritma

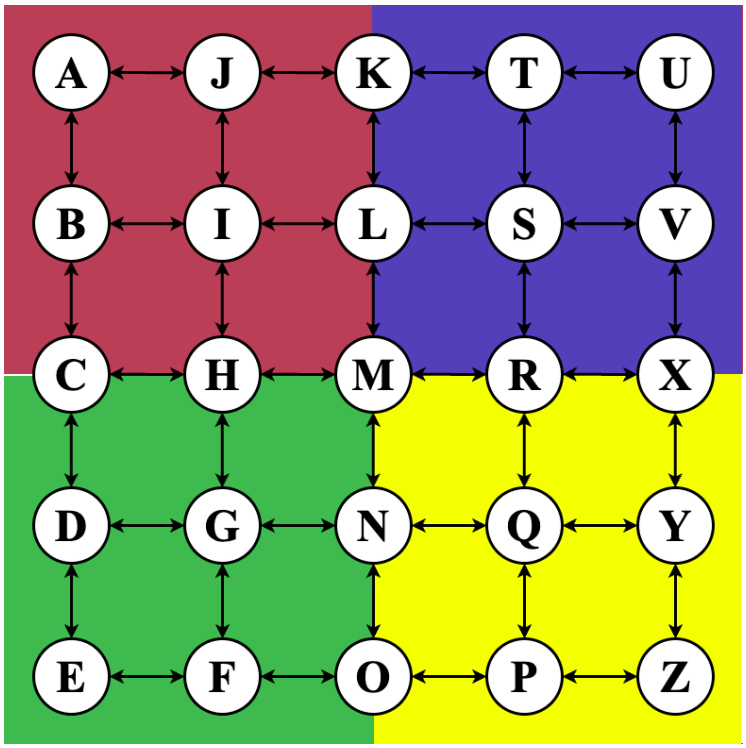
HPA* algoritam najbolje funkcionira i optimiziran je za one grafove kojima su čvorovi jednake udaljenosti, i koji su uniformno raspoređeni. S obzirom na to kako je ovaj algoritam napravljen s ciljem rješavanja problema pronalaska najkraćeg puta u kontekstu video igara, u kojima je mapa najčešće u kvadratnome obliku, jedan takav graf, u kojemu će težina između čvorova biti vrijednosti **1**, bit će iskorišten prikaz rada algoritama.



Slika 15 Graf za izračun HPA algoritma*

5.2.1. Korak 1.

Prvo je potrebno identificirati način izrade i organizacije klastera. Ovisno o veličini i kompleksnosti grafa s kojim se radi, način određivanja razdjele i povezivanja klastera može biti raznovrstan. U ovome primjeru, uzimajući u obzir veličinu grafa, kao i činjenicu kako je graf izrađen u savršenom kvadratnom obliku gdje su svi čvorovi povezani sa susjednim, graf će biti podijeljen u četiri klastera, ravno po sredini po vertikalnoj i horizontalnoj osi.



Slika 16 Klasteri grafa za izračun HPA algoritma*

Na grafu na slici *16* kojemu su dodijeljeni klasteri, vidljivo je kako granica klastera leži na postojećim čvorovima, stoga ti čvorovi pripadaju objema klasterima, te će oni biti uzeti u obzir pri odabiru izlaznih i ulaznih čvorova susjednih klastera, postupak koji će biti objašnjen u sljedećem koraku.

5.2.2. Korak 2.

U drugome koraku potrebno je odabrati odgovarajuće ulazne i izlazne čvorove za granicu i prijelaze klastera. Pošto granica između klastera leži direktno na čvorovima, parovi ulaznih i izlaznih čvorova između klastera bit će jedan čvor.

Bit će definirana tablica prijelaznih čvorova klastera, gdje definiramo gornje-lijevi klaster kao klaster ***A***, gornje-desni klastera kao klaster ***B***, donje-lijevi klastera kao klaster ***C***, i donje-desni klaster kao klaster ***D***.

Tablica 12 Klasteri u D Lite algoritmu i njihovi prijelazni čvorovi*

Granica klastera	Prijelazni čvor
A - B	
A - C	
B- D	
C - D	

Za granicu i prijelazni čvor između klastera ***A*** i ***B***, uzimaju se *K*, *L* i *M*.

Definira se izraz validan čvor, koji opisuje graničan čvor koji je povezan sa svim susjednim u oba klastera koje povezuje.

Pošto su na granici ovih klastera svi čvorovi validni, odabir prijelaznog je arbitraran, te će u ovome primjeru biti izabran *L*.

Ažurirana je tablica povezanosti klastera, te sada sadrži informaciju o prijenosnom čvoru između klastera ***A*** i ***B***.

Tablica 13 Prijelazni čvor klastera A i B

Granica klastera	Prijelazni čvor
A - B	L
A - C	
B - D	
C - D	

Istim postupkom odredit će se i prijelazni čvor između klastera ***A*** i ***C***.
Čvorovi koji leže na prijelazu klastera ***A*** i ***C***, su čvorovi *C*, *H* i *M*. Svi troje čvorova su validni čvorovi, te i pri ovome izboru će se izabrati središnji čvor, u ovome slučaju to će biti čvor *H*. Novonastala informacija reflektirat će se i na tablicu prijelaznih čvorova.

Tablica 14 Prijelazni čvor klastera A i C

Granica klastera	Prijelazni čvor
A - B	L
A - C	H
B - D	
C - D	

Na prijelazu klastera ***B*** i ***D*** nalaze se *M, R* i *X*. Svi troje prolaze definiciju validnog čvora, te se svi uzimaju u obzir pri odabiru prijelaznog čvora između spomenutih klastera. Bit će odabran R, te će se ažurirati tablica prijelaznih čvorova s tim čvorom.

Tablica 15 Prijelazni čvor klastera B i D

Granica klastera	Prijelazni čvor
A - B	L
A - C	H
B - D	R
C - D	

Uzima se zadnji par klastera, ***C*** i ***D***, te se od troje validnih čvorova *M, N* i *O* uzima srednji *N*. Tablica prijelaznih čvorova klastera je sada upotpunjena, te se s njom kreće na treći korak.

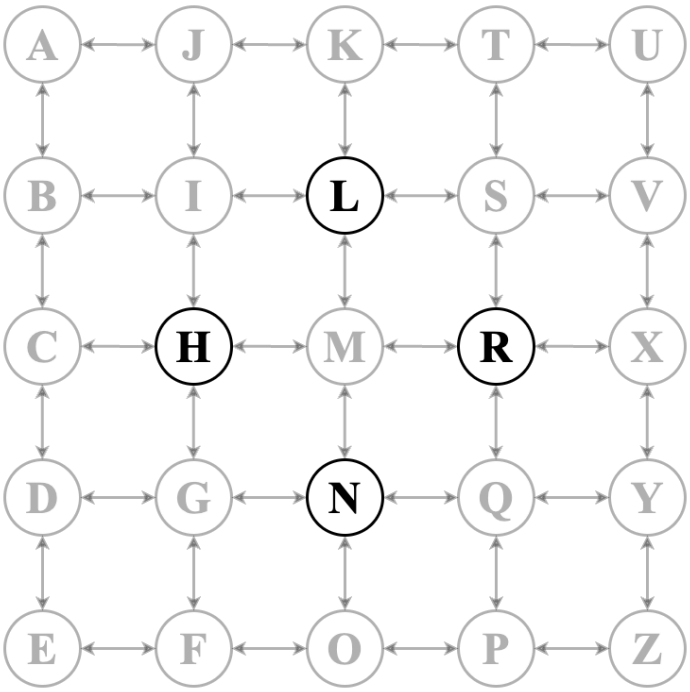
Tablica 16 Konačno stanje povezanosti klastera

Granica klastera	Prijelazni čvor
A - B	L
A - C	H
B - D	R
C - D	O

5.2.3. Korak 3.

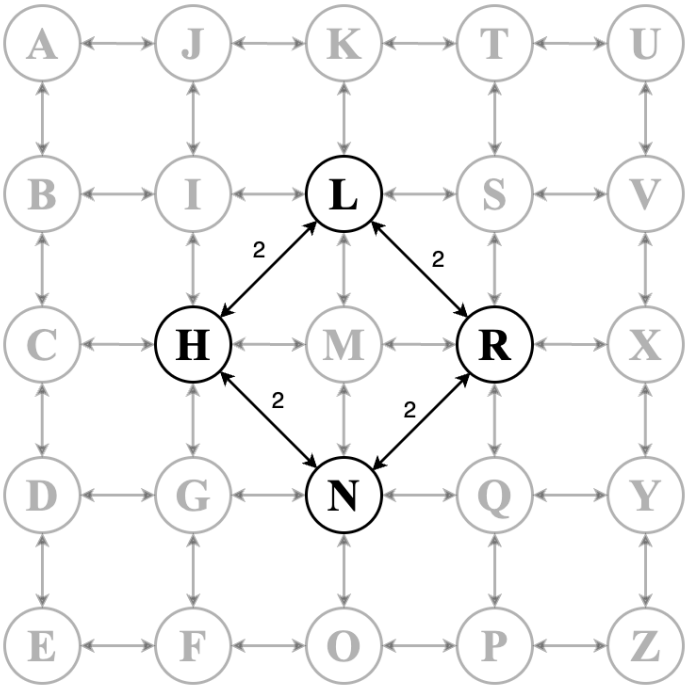
Kada je poznata informacija o povezanosti klastera i njenim prijelaznim čvorovima, stvara se apstraktni graf koji će sadržavati samo prijelazne čvorove, i težine između njih bit će duljina puta između njih.

Prvo definiramo prazan apstraktan graf bez ikakve povezanosti.



Slika 17 Nepovezan apstraktni graf za izračun HPA algoritma*

Nakon što je definiran apstraktni graf, vidljiv na slici 17, potrebno je povezati točke toga grafa. Metoda izračuna nije definirana, važno je uspješno povezati čvorove, ako je to moguće, s točnim vrijednostima kako bi daljnje prolaženje kroz graf bilo što brze i točnije. Ako putevi između čvorove klastera nisu najkraći putevi, iako će sâm izračun biti brzi, neće nužno biti optimalan. Stoga je za optimalnije korištenje HPA* algoritma potrebno izračunati što kraću rutu između čvorova klastera. Nakon što izračunamo putanje između čvorova povezanih klastera, dobivamo povezani apstraktni graf.



Slika 18 Apstraktni graf s povezanim graničnim čvorovima

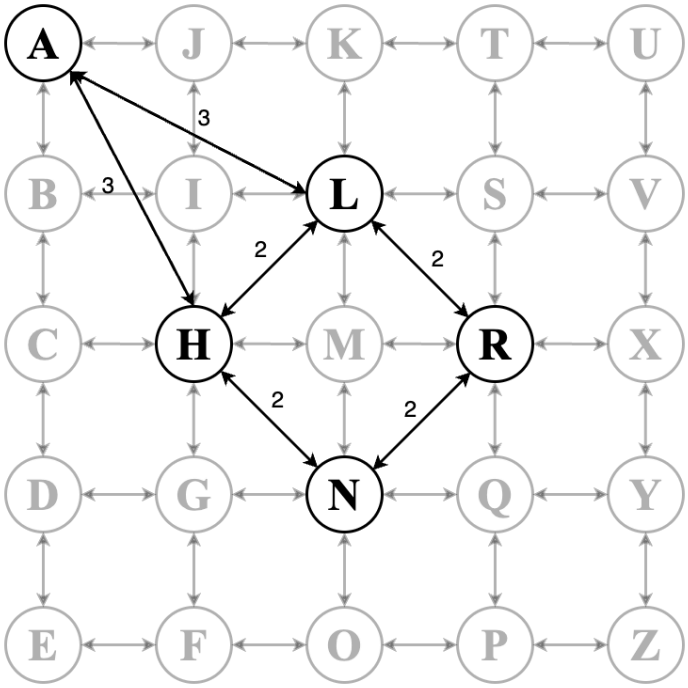
Kada se apstraktni graf upotpunjen, može se početi koristit za izračun najkraćeg puta.

5.2.4. Korak 4.

S upotpunjenim apstraktnim grafom, definira se početni i ciljni čvor za koji se želi izračunati ruta.

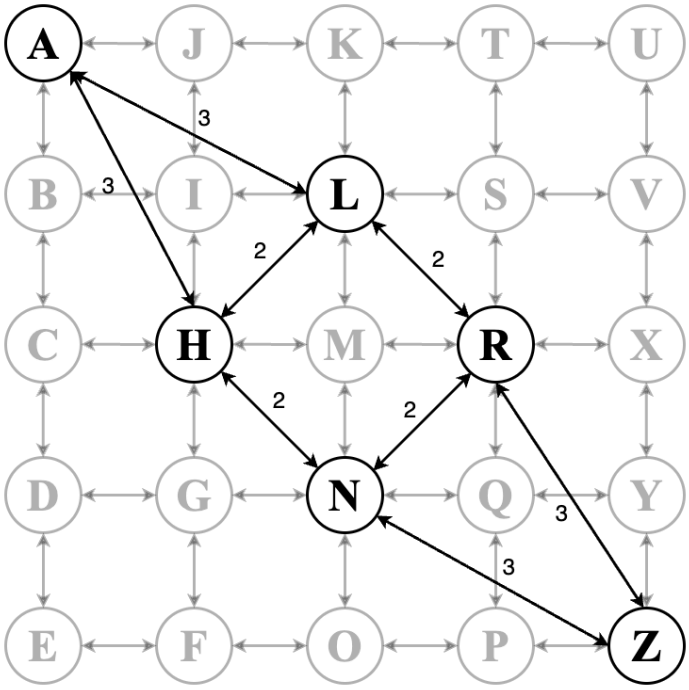
U ovome primjeru bit će odabrani ***A*** kao početni čvor, i ***Z*** kao ciljni.

Prvo je potrebno povezati ***A*** s graničnim čvorovima njegovog pripadajućeg klastera. Iz tablice povezanosti graničnih klastera može se iščitati povezanost klastera ***A*** s klasterima ***B*** i ***C***, odnosno vidljivo je kako su granični čvorovi klastera ***A*** čvorovi *L* i *H*. Stoga je potrebno povezati početni *A* s *L* i *H*.



Slika 19 Apstraktan graf s povezanim početnim čvorom

Nakon što je povezan početni čvor, potrebno je napraviti isti postupak i s ciljnim čvorom ***Z***. Čvor ***Z*** se nalazi u klasteru ***D***, čiji su prijelazni čvorovi *N* i *R*. Stoga se *Z* povezuje s *N* i *R*, te se ažurira apstraktni graf.



Slika 20 Apstraktan graf s povezanim početnim i ciljnim čvorovima

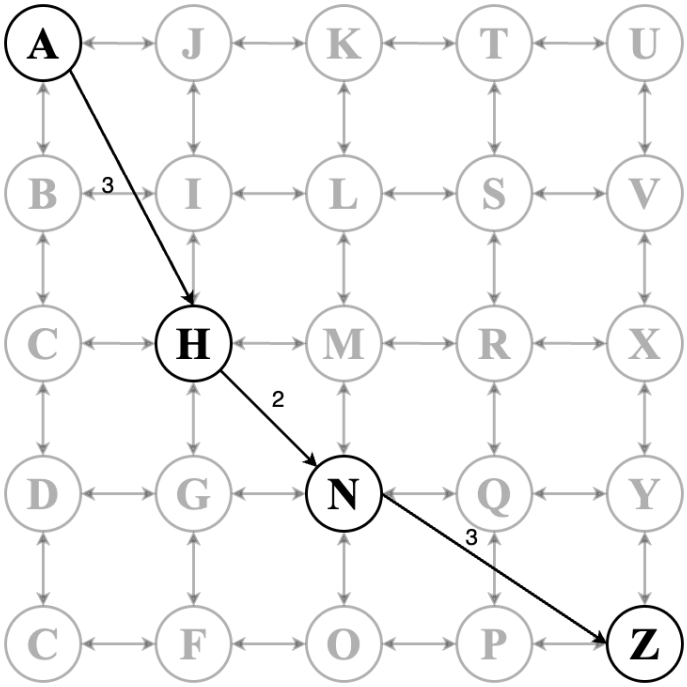
5.2.5. Korak 5.

U zadnjem koraku potrebno je A* algoritmom izračunati najkraći put između čvorova ***A*** i ***Z*** u apstraktnom grafu. Uzimajući u obzir težine između čvorova u apstraktnom grafu, moguće je dobiti dva duljinom jednaka puta.

Put *A* koji se kreće čvorovima **A** -> **H** -> **N** -> **Z**

ili put B koji se kreće čvorovima **A** -> **L** -> **R** -> **Z**

U ovome primjeru, bit će odabran put A koji prolazi putem **A** -> **H** -> **N** -> **Z**, i čija je duljina puta vrijednosti **8**.



Slika 21 Krajnja putanja kroz apstraktni graf

5.2.6. Korak 6.

Kada je poznata konačna putanja kroz apstraktni graf, preostaje puteve između čvorova apstraktnog grafa provuci kroz A* algoritam i povezati putanje. Krajnja duljina zbrojenih putanja između čvorova mora biti jednaka duljini puta kroz apstraktni graf.

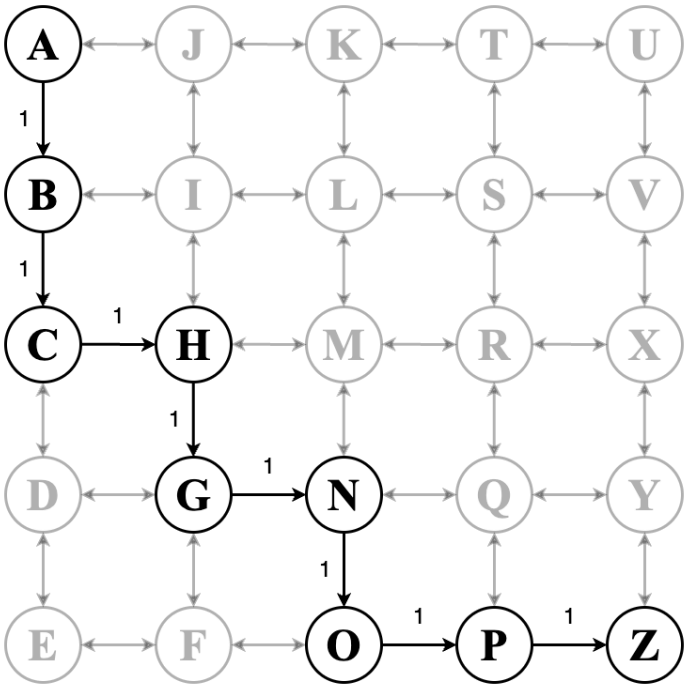
Ako se provuče putanja između čvorova *A* i *H* kroz originalni graf, dobivamo putanju **A -> B -> C -> H** duljine **3.**
Najkraći put u originalnom grafu između čvorova *H* i *N* dati će putanju čvorovima **H -> G -> N** duljine **2.**
Putanja kroz zadnji dio puta dobivenog na apstraktnom grafu između čvorova *N* i *Z* dati će putanju kroz
N -> O -> P -> Z duljine **3.**

Definira se tablica putanja između čvorova apstraktnog puta, i usporede se zbrojene putanje i njegove težine puta s duljinom apstraktnog puta. Dobiva se krajnja putanja kroz originalni graf koji nam pokazuje stvarnu konačnu putanju.

Tablica 17 Konačne putanje kroz grafove HPA algoritmom*

Apstraktna putanja	*Duljina apstraktnog puta*
A -> H -> N -> Z	**8**
Putanje na originalnom grafu	*Duljina puta*
A -> B -> C -> H	**3**
H -> G -> N	**2**
N -> O -> P -> Z	**3**
Krajnja putanja kroz originalni graf	*Duljina krajnjeg puta kroz originalni graf*
A -> B -> C -> H -> G -> N -> O -> P -> Z	**8**

Rješenje koje se dobije kroz HPA* algoritam između točaka ***A*** i ***Z*** je putanja **A -> B -> C -> H -> G -> N -> O -> P -> Z** duljine puta **8.**



Slika 22 Krajnja putanja kroz originalni graf

5.3. Prednosti algoritma

Pošto se graf dijeli u nekolicinu klastera, vrijeme i veličina grafa koji je potreban za izračun drastično se smanjuje. Time se znatno smanjuje potreban izračun, pogotovo ako se rute između čvorova apstraktnog grafa keširaju na efektivan način. Zbog svoje naravi, baš kao i D* Lite algoritam, HPA* algoritam je savršen u uvjetima u kojima se prohodnost grafa dinamički mijenja.

5.4. Problemi algoritma

Jedan od problema HPA* algoritma se može očitovati u koracima 4 i 5, a dotiče se količine podataka. Ako se više puta prolazi kroz neki apstraktni graf, keširanjem početnih i ciljnih točaka uštedjelo bi se na vremenu svih naknadnih izračuna puta koji uključuju iste te točke. No svakim novim izračunom apstraktni bi se graf povećavao, te bi se njegovim porastom u konačnici došlo do situacije u kojoj apstraktni graf sadrži previše čvorova i izgubio bi na svojoj korisnosti. Keširanje puteva između čvorova apstraktnog puta uštedjelo bi na vremenu izračuna puteva u koraku 6, no neefektivno keširanje moglo bi dovesti do situacije u kojoj se količina informacija u grafu doseže prevelike razine za sustav u kojem se koristi, npr. u navigacijskim sustavima u automobilima koji su najčešće vrlo limitirani uređaji.

6. Mjerenje i usporedba brzine izvedbe algoritama

6.1. Uvjeti mjerenja

Mjerenje je izvedeno na statičnim kvadratnim grafovima redova veličina: ***10x10, 20x20, 40x40, 80x80, 160x160 i 320x320***. Svaki čvor je prohodan, i svi su čvorovi povezani sa susjednim čvorovima u 8 smjerova. Mjerenje je izvedeno u 100 iteracija za svaki algoritam s istim početnim i ciljnim čvorovima. D* Lite algoritam je izuzet iz mjerenja jer prednost spomenutog algoritma lezi u promjenjivim grafovima, te je mjerenje u statičkom grafu beskorisno za prikaz boljitka algoritma.

Za mjerenje su stoga uzeti ***Dijkstrin*** algoritam, ***A**** algoritam, i ***HPA**** algoritam **bez** keširanja puteva između klastera, i ***A**** algoritam **s** keširanjem puteva između klastera.

6.2. Implementacije algoritama

Svi algoritmi implementirani su u *C++* programerskom jeziku, te su za svaki algoritam korišteni isti tipovi podataka kako bi implementacije bile što sličnije i usporedive.

6.2.1. Dijkstra

Dijkstrin algoritam je implementiran je, kao i ostali algoritmi, korištenjem prioritetnog reda. Jedna od optimizacijskih metoda koja je uzeta, je ta da se ne pretražuje cijeli graf, nego se algoritam smatra završenim jednom kada se stigne do ciljnog čvora. Uzimajući u obzir korištenje prioritetnog reda, može se sa sigurnošću reci kako bi svi ostali čvorovi koji dolaze iza ciljnog čvora u prioritetnom redu bili uvijek dulji putem od početnog čvora, te se stoga mogu sa sigurnošću prikočiti.

6.2.2. Rezultati mjerenja Dijkstrinog algoritma

U grafu veličine ***10x10***, u 100 iteracija s početnim čvorom *(0, 0)* i ciljnim čvorom *(99, 99)*, prosječno vrijeme izračuna algoritma iznosilo je **0.75758** milisekundi (ms).

Za graf veličine ***20x20***, u 100 iteracija prosječno vrijeme s istim početnim i ciljnim čvorom, iznosilo je **2.6002 ms**.

Za graf veličine ***40x40***, prosječno vrijeme je iznosilo je **11.5659 ms**.

Za graf veličine ***80x80***, prosječno vrijeme je iznosilo je **52.8685 ms**.

Za graf veličine ***160x160***, prosječno vrijeme je iznosilo je **241.919 ms**.

Za graf veličine ***320x320***, prosječno vrijeme je iznosilo je **1067.5 ms**.

Tablica 18 Rezultati vremenskog mjerenja Dijkstrinog algoritma

Veličina grafa	*Vrijeme izračuna (u milisekundama)*
10 x 10	**0.75758**
20 x 20	**2.6002**
40 x 40	**11.5659**
80 x 80	**52.8685**
160 x 160	**241.919**
320 x 320	**1067.5**

Po tablici je vidljivo kako *Dijkstrin* algoritam eksponencijalno postaje sve skuplji i sporiji kako se povećava veličina grafa u kojemu se koristi. Time je vidljivo kako je *Dijkstrin* algoritam, iako vrlo točan i lagano razumljiv, vrlo ograničim u kompleksnijim sustavima u kojima se koriste velike količine podataka.

6.2.3. A*

A* algoritam implementiran je s prioritetnim redom, te je za heurističku funkciju uzeta zračna duljina između čvora i ciljnog čvora. Ovom heurističkom funkcijom se dolazi do vrlo optimalne, no ne i najoptimalnije brzine kao što će biti vidljivo na rezultatima mjerenja.

6.2.4. Rezultati mjerenja A* algoritma

U grafu veličine ***10x10***, u 100 iteracija s početnim čvorom *(0, 0)* i ciljnim čvorom *(99, 99)*, prosječno vrijeme izračuna algoritma iznosilo je **0,0562** milisekundi (ms).
Za graf veličine ***20x20***, u 100 iteracija prosječno vrijeme s istim početnim i ciljnim čvorom, iznosilo je **0,1211 ms**.
Za graf veličine ***40x40***, prosječno vrijeme je iznosilo je **0,24646 ms**.
Za graf veličine ***80x80***, prosječno vrijeme je iznosilo je **0,5431 ms**.
Za graf veličine ***160x160***, prosječno vrijeme je iznosilo je **1,1655 ms**.
Za graf veličine ***320x320***, prosječno vrijeme je iznosilo je **2,4575 ms**.

Tablica 19 Rezultati vremenskog mjerenja A algoritma*

Veličina grafa	*Vrijeme izračuna (u milisekundama)*
10 x 10	**0,0562**
20 x 20	**0,1211**
40 x 40	**0,24646**
80 x 80	**0,5431**
160 x 160	**1,1655**
320 x 320	**2,4575**

Uspoređujući rezultate mjerenja s rezultatima *Dijkstrinog* algoritma, vidljivo je višestruko ubrzanje, pogotovo kako se povećava veličina grafa. Kako se graf povećava za ***N***, vrijeme izračuna se multiplicira brojem skoro duplo manjim od navedenog, stoga možemo biti sigurni i povjerljivi u vrijeme izračuna za graf koji još nismo pokrenuli.

6.2.5. HPA* algoritam

HPA* algoritam implementiran je s prioritetnim redom, te je ograničen na 4 klastera jednako raspoređenih po sredini grafa. Ova metoda izrade klastera je vrlo neefektivna, što će i biti vidljivo na rezultatima izračuna rute.

6.2.6. Rezultati mjerenja HPA* algoritma

U grafu veličine ***10x10***, u 100 iteracija s početnim čvorom *(0, 0)* i ciljnim čvorom *(99, 99)*, prosječno vrijeme izračuna algoritma iznosilo je **0,2755** milisekundi (ms).

Za graf veličine ***20x20***, u 100 iteracija prosječno vrijeme s istim početnim i ciljnim čvorom, iznosilo je **0,4762 ms**.

Za graf veličine ***40x40***, prosječno vrijeme je iznosilo je **0,8574 ms**.

Za graf veličine ***80x80***, prosječno vrijeme je iznosilo je **1,7531 ms**.

Za graf veličine ***160x160***, prosječno vrijeme je iznosilo je **3,6024 ms**.

Za graf veličine ***320x320***, prosječno vrijeme je iznosilo je **7,4506 ms**.

Tablica 20 Rezultati vremenskog mjerenja HPA algoritma bez keširanja puteva*

Veličina grafa	*Vrijeme izračuna (u milisekundama)*
10 x 10	**0,2755**
20 x 20	**0,4762**
40 x 40	**0,8574**
80 x 80	**1,7531**
160 x 160	**3,6024**
320 x 320	**7,4506**

Iz ovih rezultata vidljivo je kako je *HPA** , u ovoj implementaciji, tri puta sporiji od *A** algoritma. Problem leži u dvije stvari.

Broj klastera je neoptimalan, kako bi se optimalno ubrzao algoritam, potrebno je što bolje organizirati i isjeći klastere tako da je prijelaz između njih što kraći.

Drugi problem leži u činjenici kako je nakon svakog izračuna apstraktne putanje potrebno izračunati svaku putanju između apstraktnih čvorova.

Drugi problem možemo riješiti tako da keširamo putanje između klastera, i da izračunavamo samo putanje između startnog čvora i prijelaznih čvorova startnog klastera, i putanje između ciljnog čvora i prijelaznih točaka njegovog klastera.

Kada se keširaju putanje između klastera, dobiva se poboljšani rezultati.

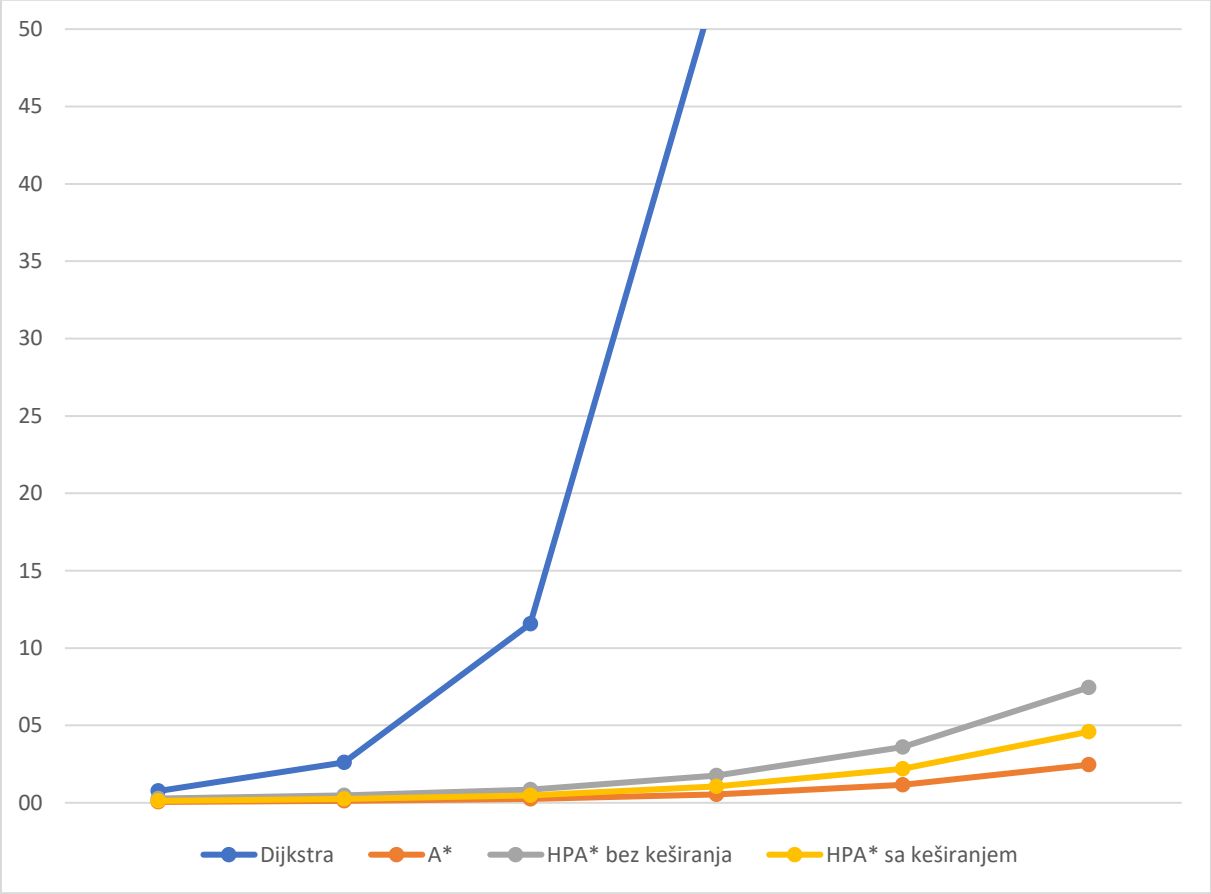
Tablica 21 Rezultati vremenskog mjerenja HPA algoritma s keširanjem puteva*

Veličina grafa	*Vrijeme izračuna (u milisekundama)*
10 x 10	**0,1234**
20 x 20	**0,24825**
40 x 40	**0,48563**
80 x 80	**1,0481**
160 x 160	**2,1953**
320 x 320	**4,59**

Rezultati su sada gotovo dvostruko brzi, jer je uvelike smanjen broj potrebnih izračuna među-ruta u klasterima. No, ako se usporede rezultati s mjerenjima pretrage A* algoritmom, vidimo kako su ova mjerenja svejedno dvostruko sporija.

Problem je još uvijek u broju potrebnih izračuna među-ruta, konkretnije izračun ruta između početnog i ciljnog čvora s prijelaznim čvorovima njihovih klastera. Ako se optimalnije odrede klasteri, vrijeme izračuna bi se uvelike približio, i ovisno o grafu, postao brzi od *A** pretrage.

6.3. Krajnja usporedba rezultata



Slika 23 Usporedni graf brzina uspoređenih algoritama

6.3.1. Dijkstra

Uspoređujući vizualno rezultate, vidljivo je kako je *Dijkstra* vrlo neoptimalan algoritam kako se povećava broj čvorova u grafu. U malom broju čvorova algoritam je relativno brz i iskoristiv, no kako se povećava broj čvorova, vidljivo je kako je potreban drugi algoritam.

U istraživanju AbuSalim, Samah WG, et al. (AbuSalim, Samah WG, et al., 2020) iako je vrijeme izračuna Dijkstre drugačije u usporedbi s vremenom izračunatim u ovome radu, vidljiv je isti uzorak povećanja vremena potrebnog za izračun kako se povećava veličina grafa.

Tablica 22 Rezultati mjerenja Dijkstrinog algoritma (AbuSalim, Samah WG, et al.)

Veličina grafa	*Vrijeme izračuna (u milisekundama)*
5	**1351**
10	**3724**
50	**2072**

Dijkstrin se algoritam, dakle, ne preporučuje koristiti u svojemu generičkom obliku jer nije skalabilan, te za velike grafove neće uopće biti upotrebljiv.

6.3.2. A*

A* Algoritam pokazao se kao najbrži algoritam od izmjerenih. Velika prednost A* algoritma je njegova linearnost, koja je i potvrđena i u drugim istraživanjima (Maddison, J., Tarlow, & Minka, 2014).

Velika mana algoritma leži u implementaciji njegove heurističke funkcije, stoga je direktna usporedba različitih implementacija kompliciranа.

Velikim razvojem umjetne inteligencije zadnjih godina, počeli su se razvijati i modeli za heurističku funkciju za A* algoritam. Nakon velikog broja iteracija, istraživanje (Numeroso, Bacciu, & Veličković, 2022) je pokazalo kako se konstantnim treniranjem umjetne inteligencije može doci do heuristike koja će uvijek imati gotovo konstantno vrijeme izračuna najkraćeg puta u rastućem grafu.

6.3.3. HPA*

HPA* algoritam, bas poput A* algoritma, uvelike ovisi o njegovoj implementaciji. Način na koji su klasteri napravljeni, ovisno o pozicijama neprohodnih čvorova, može dati uvelike ubrzano vrijeme uspoređujući s A* algoritmom. Mjerenje u ovome radu izvršeno je na tipu grafa, i s klasterima koji su napravljeni na vrlo rudimentaran način, što pogoduje boljem vremenu izračuna za A* algoritam. U situaciji koja pogoduje HPA* algoritmu, on može dati puno brzi izračun od A* algoritma, što je i vidljivo u istraživanju (Agcaoili, Jacob, Bernabe, & Agustin, 2023). U spomenutom istraživanju vidljivo je kako je jedina prednost A* algoritma, čak i u za njega nepovoljnim uvjetima to što će vrlo često A* dati kraći put.

7. Zaključak

Problem pronalaska najkraćeg puta pokazao se kao vrlo važan problem čije rješenje nije niti jednostavno niti identično u svim situacijama. Kroz povijest mnoga rješenja pokazala su se kao parcijalno rješenje koje se fokusira na specifične situacije i pod probleme.

Razni algoritmi za pronalazak najkraćeg puta stvoreni su i optimalni za razne vrste problema. Njihova je uporaba široka, te se za svaki problem može naći specijalizirani algoritam.

Algoritam poput Dijkstrinog jedan je od algoritama koji pokušavaju univerzalno riješiti problem pronalaska najkraćeg puta. No, mana ovoga algoritma je ta što je vrlo spor i skup, te se povećanjem grafa povećava i kompleksnost algoritma te je vrlo neefikasan na velikim količinama podataka.

A* algoritam vrlo je efikasan te se vrlo dobro skalira s grafom. U situaciji u kojoj se radi sa statičkim grafom, ovaj algoritam pokazao se kao idealno rješenje kada je heuristička funkcija optimalno definirana.

U situacijama u kojima je graf definiran kvadratna mreža, HPA* algoritam dopusta keširanje puteva između ranije definiranih klastera što uvelike smanjuje vrijeme izračuna rute, onda kada su klasteri efikasno raspoređeni.

D* Lite algoritam u dinamičkim grafovima daje mogućnost točnog izračuna bez obzira na frekvenciju promjene u grafu.

Primjena algoritama uvelike ovisi o tipu grafa kroz koji se kreće, kao što je i mjerenje u ovome radu pokazalo. Statički graf bez ikakvih prepreka najbrže će biti prijeđen algoritmom koji se fokusira na smjer, dok će za grafove puni rupa, tj. neprohodnih čvorova, efikasniji biti algoritmi koji će moći razdijeliti graf u dijelova tako da se što više izbjegnu neprohodni dijelovi. Vrlo je važno stoga biti svjestan koji se problem rješava, jer univerzalni algoritam ne postoji, te se mora jasno definirati problem kako bi se našao optimalan algoritam za rješenje.

8. Literatura

AbuSalim, Samah WG, et al. (2020). Comparative analysis between dijkstra and bellman-ford algorithms in shortest path optimization. *IOP Conference Series: Materials Science and Engineering. Vol. 917. No. 1*.

Agcaoili, Jacob, C., Bernabe, X. Y., & Agustin, V. A. (2023). Hybridization of A* Pathfinding and Hierarchical Pathfinding A* Algorithm for Pathfinding in Grid Based Games. *United International Journal for Research & Technology*.

Ali, D. M. (2021). Recent advances in graph theory and its applications: Exploring techniques and real-world implementations. *IJFANS International Journal of Food and Nutritional Sciences*.

Badwaik, J. S. (2020). Recent Advances in Graph theory and its applications. *International Res. Journal of Science & Engineering*.

Deo, N. (2017). Graph theory with applications to engineering and computer science. Courier Dover Publications.

Dijkstra, E. (1959). *A note on two problems in connexion with graphs.* Nerische mathematik.

Harabor, D., & Adi, B. (2008). *Hierarchical path planning for multi-size agents in heterogeneous environments.* 008 ieee symposium on computational intelligence and games.

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). *A Formal Basis for the Heuristic Determination of Minimum Cost Paths.* IEEE Trans. Syst. Sci. Cybern.

Maddison, J., C., Tarlow, D., & Minka, T. (2014). A* sampling. *Advances in neural information processing systems 27*.

Numeroso, D., Bacciu, D., & Veličković, P. (2022). Learning heuristics for A* .

Russell, S. J., & Norvig, P. (2016). *Artificial Intelligence : A Modern Approach.* Pearson.

Sven, K., Maxim, L., & David, F. (2004). *Lifelong Planning A*,.* Artificial Intelligence.

Wilson, R. J. (1979). Introduction to graph theory (4th Edition). Pearson Education India.

9. Popis slika

10. Popis tablica

Vizualizacija i usporedba algoritama za pronalazak najkraćeg puta

Sažetak

U radu se uspoređuju najpopularniji algoritmi za pronalaženje najkraćeg puta u dvodimenzionalnoj kvadratnoj mreži. U uvodnom dijelu se definiraju osnovni pojmovi iz teorije grafova te se daje važnost i mogućnosti primjene navedenih algoritama. Opisan je rad najpopularnijih algoritama koji uključuju Dijkstrin algoritam, A*, D* Lite i HPA*. Osim usporedbe načina rada i kompleksnosti, u radu se opisuju prednosti, mane i moguća unaprjeđenja, odnosno optimizacije algoritama. Vizualno se prikazuju koraci rada algoritama te se primjerima dolazi do rješenja izračuna puta u adekvatnim grafovima. Uz vizualno, prikazuje se i empirički mjereno vrijeme pronalaska put za svaki od algoritama koji su implementirani u C++ jeziku sa identičnim tipovima podataka kako bi rezultati bili što točniji. Objašnjavaju se točni problemi koji dovode to mjerenih rezultata te se daju primjeri rješenja koji bi mogli poboljšati rezultate. U sklopu rada izrađena je i opisana aplikacija koja vizualizira rad algoritama, odnosno koja za zadanu veličinu grafa, te početak i kraj putanje prikazuje pojedine korake rada algoritma i pretraženih puteva.

Ključne riječi: završni rad, Dijkstra, a*, d* lite, hps*, algoritmi, pronalazak puta, teorija grafa

Visualization and comparison of algorithms for finding the shortest path

Summary

The paper compares the most popular pathfinding algorithms in a two-dimensional square grid. In the introductory part, basic terms from graph theory are defined and the importance and application possibilities of the mentioned algorithms are given. The operation of the most popular algorithms such as Dijkstra's, A*, D* Lite and HPA* is described. In addition to the comparison of working methods and complexity, the paper describes the advantages, disadvantages, and possible improvements, that is, optimizations of the algorithms. Every step of the algorithm is visually shown, and the run time is empirically measured for each of the algorithms, each implemented in C++ programming language, using the same data types so that the results can be as accurate as possible. Problems that can and do increase the run time are mentioned and described, and the possible solutions are given so that the results can be improved. As part of the work, a program was created that visualizes the algorithms, which for a given graph size, and the beginning and end of the path, shows each step of the algorithm's operation and the searched paths.

Key words: BA thesis, Dijkstra, a*, d* lite, hpa*, algorithms, pathfinding, graph theory