

Planiranje kretanja zasnovano na A* algoritmu u OctoMap struktuiranim okruženjima

Damir Bojadžić

Završni rad na
I ciklusu studija



Elektrotehnički fakultet Sarajevo
Odsjek za računarstvo i informatiku
Univerzitet u Sarajevu
Bosna i Hercegovina
Ak. godina 2017/2018.

Planiranje kretanja zasnovano na A* algoritmu u OctoMap struktuiranim okruženjima

Damir Bojadžić

Sažetak

Završni rad "Planiranje kretanja zasnovano na A* algoritmu u OctoMap struktuiranim okruženjima" se bavi obradom i prikazom rezultata dobijenih primjenom A* algoritma za pronalaženje najkratčeg puta u okruženjima struktuiranim kao mrežasta mapa, quadtree i quadtree sa diskretiziranim granicama čvorova. U radu su obrađeni implementacioni detalji ovih struktura podataka, sa proširenjem na octree kao strukturu podataka za modeliranje 3D prostora. Rad poredi Dijkstrin algoritam i popularni A* algoritam te obrađuje detalje implementacije i prilagođavanja A* algoritma da radi sa ovim strukturama podataka uz otvorenu mogućnost modifikacije parametara algoritma. Konačno, na kraju rada, poredi se uspješnost reprezentacije 2D prostora pomoću quadtree strukture podataka te brzine pretrage i dužine dobijenih puteva na dvije mape kroz niz testova sa različitim početnim parametrima.

Motion Planning Based on A* Algorithm in OctoMap Structured Environments

Damir Bojadžić

Abstract

The graduation thesis "Motion Planning Based on A* Algorithm in OctoMap Structured Environments" deals with the processing and presentation of the results obtained by applying the A* algorithm for finding shortest paths on a matrix structured environment as well as quadtree structured environments with the extension to quadtrees with discretized node boundaries. In this paper, the implementation details of these data structures are discussed, with the extension to octrees as a data structure for 3D space modelling. The paper compares Dijkstra's algorithm and the popular A* algorithm and analyses the details of the implementation and adaptation of the A* algorithm to work with these data structures with the open possibility to modify the algorithm's working parameters. Finally, the paper discusses the efficiency of modelling a 2D environment by using quadtrees as well as the speed and distance of the resulting paths obtained on two maps through a series of tests with different starting parameters.

Univerzitet u Sarajevu

Naziv fakulteta/akademije: Elektrotehnički fakultet

Naziv odsjeka i/ili katedre: Odsjek za računarstvo i informatiku

Predmet: Matematička logika i teorija izračunljivosti

Izjava o autentičnosti radova

Seminarski rad, završni (diplomski odnosno magistarski) rad za I i II ciklus studija i integrirani studijski program I i II ciklusa studija, magistarski znanstveni rad i doktorska disertacija¹.

Ime i prezime: Damir Bojadžić

Naslov rada: Planiranje kretanja zasnovano na A* algoritmu u OctoMap strukturiranim okruženjima

Vrsta rada: Završni rad za I(prvi) ciklus studija

Broj stranica: 50

Potvrđujem:

- Da sam pročitao/la dokumente koji se odnose na plagijarizam, kako je to definirano Statutom Univerziteta u Sarajevu, Etičkim kodeksom Univerziteta u Sarajevu i pravilima studiranja koja se odnose na I i II ciklus studija, integrirani studijski program I i II ciklusa i III ciklus studija na Univerzitetu u Sarajevu, kao i uputama o plagijarizmu navedenim na web stranici Univerziteta u Sarajevu;
- Da sam svjestan/na univerzitetskih disciplinskih pravila koja se tiču plagijarizma;
- Da je rad koji predajem potpuno moj, samostalni rad, osim u dijelovima gdje je to naznačeno;
- Da rad nije predat, u cjelini ili djelimično, za stjecanje zvanja na Univerzitetu u Sarajevu ili nekoj drugoj visokoškolskoj ustanovi;
- Da sam jasno naznačio/la prisustvo citiranog ili parafraziranog materijala i da sam se referirao/la na sve izvore;
- Da sam dosljedno naveo/la korištene i citirane izvore ili bibliografiju po nekom od preporučenih stilova citiranja, sa navođenjem potpune reference koja obuhvata potpuni bibliografski opis korištenog i citiranog izvora;
- Da sam odgovarajuće naznačio/la svaku pomoć koju sam dobio/la pored pomoći mentora/ice i akademskih tutora/ica.

Mjesto, datum: _____

Potpis: _____

¹U radu su korišteni sljedeći dokumenti: Izjava autora koju koristi Elektrotehnički fakultet u Sarajevu; Izjava o autentičnosti završnog rada Centra za interdisciplinarnе studije – master studij „Evropske studije”, Izjava o plagijarizmu koju koristi Fakultet političkih nauka u Sarajevu.

Zahvalnice

Veliku zahvalnost, u prvom redu, dugujem svom mentoru doc.dr. Dinku Osmanoviću, bez koga ovaj završni rad ne bi bio moguć. Hvala za sve savjete, sugestije, ukazano povjerenje i saradnju tokom zadnjih nekoliko godina.

Zahvaljujem se svom dugogodišnjem prijatelju, Edvinu Teskeredžiću, na podršci koju mi je pružio tokom izrade ovog rada.

Konačno, posebnu zahvalnost iskazujem svojim roditeljima, koji su uvijek bili tu za mene i bez kojih sve što sam do sada postigao ne bi bilo moguće.

Hvala vam na svemu!

Sadržaj

Sažetak	iii
Abstract	iv
Zahvalnice	vi
Popis slika	ix
Popis tabela	x
1. Uvod	1
1.1. Postavka problema	1
1.2. Zadaci i ciljevi rada	1
1.3. Struktura rada	2
2. Planiranje kretanja u strukturiranim okruženjima	3
2.1. Uvod	3
2.2. A* Algoritam	3
2.2.1. Opis A* algoritma	3
2.2.2. Ilustracija rada algoritma	6
2.2.3. Heuristika	8
2.3. Mrežasta mapa	9
2.3.1. Opis	9
2.3.2. Implementacija	10
2.3.3. A* algoritam i mrežasta mapa	11
2.4. Quadtree	12
2.4.1. Opis quadtree-a	12
2.4.2. Uporedba quadtree-a i mrežaste mape	13
2.4.3. Dubina	14
2.4.4. Apstraktni opis klase	16
2.4.5. A* algoritam i Quadtree	19
2.4.6. Octree	20
2.4.7. Zaključak	21

3. Implementacija	22
3.1. Uvod	22
3.2. Mrežasta mapa	22
3.3. Implementacija quadtree-a	23
3.3.1. Klase	23
3.3.2. Metode	24
3.3.3. Implementacija A* algoritma za QuadTree	25
3.4. Quadtree sa diskretiziranim granicama	26
3.4.1. Proširenje	26
3.4.2. Implementacija A* algoritma za diskretizirani QuadTree	27
3.4.3. Varijacije	27
3.5. Zaključak	28
4. Rezultati	29
4.1. Uvod	29
4.2. Kreiranje QuadTree-a	30
4.3. Rezultati traženja putanje korištenjem A* algoritma na mrežastoj mapi	34
4.4. Rezultati traženja putanje korištenjem A* algoritma na mapi dekomponiranoj u QuadTree-eve	37
4.5. Rezultati traženja putanje korištenjem A* algoritma na mapi dekomponiranoj u QuadTree-eve sa diskretiziranim granicama okvira	45
4.6. Uporedba putanja dobijenih prezentiranim algoritmima	53
4.6.1. Kreiranje QuadTree-a	53
4.6.2. Pretraga	58
4.7. Zaključak	63
5. Zaključak	64
Bibliografija	66

Popis slika

2.1. Dijkstra - početak pretrage [1]	6
2.2. A* - početak pretrage [2]	6
2.3. Dijkstra - prepreka [1]	7
2.4. A* - prepreka [2]	7
2.5. Dijkstra - zaobilazak prepreke [1]	7
2.6. A* - zaobilazak prepreke [2]	7
2.7. Dijkstra - konačni put [1]	8
2.8. A* - konačni put [2]	8
2.9. Primjer okoline sa prerekama predstavljenim regionima crne boje; Bijelom bojom su obojeni regioni koji predstavljaju slobodni prostor	9
2.10. Primjer dekompozicije prostora u quadtree i reprezentacija stabla [3]	12
2.11. Usporedba mrežaste mape i quadtree-a [4]	13
2.12. Prikaz dekompozicije prostora u quadtree-eve različitih dubina	15
2.13. Usporedba A* algoritma na mrežastoj mapi i na Quadtree	20
2.14. Reprezentacija kampusa u Freiburg-u u OctoMap framework-u [5]	21
3.1. Usporedba: obični Quadtree i Quadtree sa diskretiziranim granicama	28
4.1. Mapa A(ljevo) i B(desno); Mapa A predstavlja lakši scenario, a mapa B teži scenario	29
4.2. QuadTree reprezentacija mape A veličine 600x600 na dubinama 4, 5, 6 i 7	53
4.3. QuadTree reprezentacija mape B veličine 600x600 na dubinama 4, 5, 6 i 7	54
4.4. QuadTree reprezentacija mape A veličine 3000x3000 na dubinama 4, 5, 6 i 7	55
4.5. QuadTree reprezentacija mape B veličine 3000x3000 na dubinama 4, 5, 6 i 7	56
4.6. QuadTree reprezentacija mape A veličine 100x100 na dubinama 4, 5, 6 i 7	57
4.7. QuadTree reprezentacija mape B veličine 100x100 na dubinama 4, 5, 6 i 7	58
4.8. Pretraga matrične mreže mapa A i B	59
4.9. Pretraga QuadTree-a dubine 6 mapa A i B	60
4.10. Pretraga QuadTree-a dubine 6 sa d. granicama mapa A i B	61

Popis tabela

2.1.	2D matrica	10
2.2.	Kontinualno alocirana matrica	10
2.3.	Niz nizova	10
2.4.	2D Dvostruko povezana lista	11
4.1.	Vremena kreiranja QuadTree-a dubine 4	30
4.2.	Vremena kreiranja QuadTree-a dubine 5	31
4.3.	Vremena kreiranja QuadTree-a dubine 6	32
4.4.	Vremena kreiranja QuadTree-a dubine 7	33
4.5.	Vremena pretrage - Matrica 100x100	34
4.6.	Vremena pretrage - Matrica 200x200	34
4.7.	Vremena pretrage - Matrica 400x400	35
4.8.	Vremena pretrage - Matrica 500x500	35
4.9.	Vremena pretrage - Matrica 600x600	36
4.10.	Vremena pretrage - Matrica 1000x1000	36
4.11.	Vremena pretrage - Mapa A - QuadTree dubine 4	37
4.12.	Vremena pretrage - Mapa B - QuadTree dubine 4	38
4.13.	Vremena pretrage - Mapa A - QuadTree dubine 5	39
4.14.	Vremena pretrage - Mapa B - QuadTree dubine 5	40
4.15.	Vremena pretrage - Mapa A - QuadTree dubine 6	41
4.16.	Vremena pretrage - Mapa B - QuadTree dubine 6	42
4.17.	Vremena pretrage - Mapa A - QuadTree dubine 7	43
4.18.	Vremena pretrage - Mapa B - QuadTree dubine 7	44
4.19.	Vremena pretrage - Mapa A - QuadTree sa d. granicama dubine 4 . .	45
4.20.	Vremena pretrage - Mapa B - QuadTree sa d. granicama dubine 4 . .	46
4.21.	Vremena pretrage - Mapa A - QuadTree sa d. granicama dubine 5 . .	47
4.22.	Vremena pretrage - Mapa B - QuadTree sa d. granicama dubine 5 . .	48
4.23.	Vremena pretrage - Mapa A - QuadTree sa d. granicama dubine 6 . .	49
4.24.	Vremena pretrage - Mapa B - QuadTree sa d. granicama dubine 6 . .	50
4.25.	Vremena pretrage - Mapa A - QuadTree sa d. granicama dubine 7 . .	51
4.26.	Vremena pretrage - Mapa B - QuadTree sa d. granicama dubine 7 . .	52
4.27.	Vremena pretrage različitih struktura podataka	62

Poglavlje 1.

Uvod

U modernom vremenu kada su autonomna vozila sve zastupljenija, a robotika i druge nauke doživljavaju svoj tehnološki procvat, problem planiranja kretanja postaje sve prisutniji. Bilo da su u pitanju gусте saobraćajne mreže gradova ili navigiranje pustom površinom Marsa, planiranje kretanja je matematičarima još od davnih vremena zadavalo probleme. U ovom radu bit će predstavljen jedan od mogućih načina planiranja kretanja.

1.1. Postavka problema

Planiranje kretanja danas predstavlja jedan od najvećih izazova u robotici. To je proces kojim se određuje putanja do odredišne tačke unutar zadanog okruženja, a da se pri tome izbjegne kolizija se preprekama. Pri računanju se mora voditi računa o faktorima kao što su dužina i vrijeme komputacije.

Kao što se vidi iz priloženog, planiranje kretanja je u tijesnoj vezi sa problemom najkraćeg puta (eng. Shortest Path Problems) koji je značajan zadatak teorije grafova i spada u oblast kombinatorne optimizacije. Po definiciji, neka je dat orijentirani graf $\mathcal{G} = (\mathcal{X}, \mathcal{R})$ i određen polazni čvor $s \in \mathcal{X}$ i neka je svakom luku (i, j) iz \mathcal{R} pridružena određena vrijednost $l_{i,j}$. U problemima najkraćeg puta potrebno je odrediti put od čvora s do nekog drugog zadanog čvora $t \in \mathcal{X}$ tako da suma dužina $l_{i,j}$ koje ulaze u dobijeni put bude minimalna. Dužine $l_{i,j}$ su u konkretnim primjerima nenegativne s obzirom da se radi o euklidskom prostoru. Međutim, u odabiru grana svakako značajnu ulogu igra i način na koji je prostor struktuiran jer od njega ovisi raspored i veličina čvorova.

1.2. Zadaci i ciljevi rada

Završni rad "Planiranje kretanja zasnovano na A* algoritmu u OctoMap strukturiranim okruženjima" se bavi obradom i prikazom rezultata dobijenih primjenom A* algoritma za pronalaženje najkraćeg puta u okruženjima struktuiranim kao mrežasta

mapa, quadtree i quadtree sa diskretiziranim granicama koja će u nastavku rada biti detaljno opisana. Poseban akcenat stavljen je na vrijeme potrebno za pronalaženje najkraćeg puta te dužine najkraćeg puta u ovim okruženjima na mapama različitih veličina.

Na kraju će biti govora o proširenju ovih struktura na octree strukturu podataka.

1.3. Struktura rada

Rad se sastoji iz nekoliko cjelina, gdje svaka predstavlja zaseban dio čije su stavke grupirane na načina da čine kompaktnu i homogenu cjelinu. Rad se sastoji iz 5 glavnih poglavlja, gdje je svako poglavlje podijeljeno u više sekcija.

Prvo poglavlje predstavlja općeniti uvod u problematiku ovog rada te upoznaje čitaoca sa njegovim značajem, sadržajem i ciljevima.

Drugo poglavlje obrađuje A* algoritam i njegov značaj te quadtree i octree strukture podataka. U ovom poglavlju bit će opisan pseudokod za A* algoritam i kroz slike prikazan način na koji pretražuje graf. Zatim će biti predstavljena quadtree struktura podataka kao način za modeliranje 2D okruženja zajedno sa njenom implementacijom i grafičkom reprezentacijom i njena usporedba sa matričnom reprezentacijom 2D okruženja. Konačno će biti predstavljen octree kao struktura podataka za modeliranje 3D okruženja.

Nakon toga će u trećem poglavlju biti opisan način kako je realizovan A* algoritam na quadtree-u sa i bez diskretizacije rubova graničnih okvira čvorova (ili skraćeno - granica čvorova) u C++-u uz priložen odgovarajući programski kod, a onda će biti opisane potrebne modifikacije datog koda za proširenje na octree strukturu podataka.

U četvrtom poglavlju bit će predstavljeni rezultati primjene A* algoritma na ovim strukturama podataka te osvrt na njihove prednosti i nedostatke s obzirom na vrijeme i dužinu i konačno, u petom poglavlju će biti dat završni osvrt na rad.

Poglavlje 2.

Planiranje kretanja u strukturiranim okruženjima

2.1. Uvod

Cilj ovog poglavlja je predstavljanje metoda i alata korištenih za rješavanje postavljenog problema. U nastavku će biti dat pseudokod uz opšti opis funkcionisanja algoritma te osvrt na značajnu ulogu koju igra heuristika u ovom algoritmu. Nakon toga će biti dat opis quadtree-a kao strukture podataka sa osvrtom na octree. Razumijevanje principa na kojima se temelji A* algoritam i načina na koji je struktuirana okolina je od esencijalnog značaja za razumijevanje rada u cijelosti.

2.2. A* Algoritam

2.2.1. Opis A* algoritma

A* algoritam je danas jedan od najpopularnijih i intenzivno korištenih algoritama pretrage. Iako je samo bio zamišljen kao algoritam za pretragu i obilazak grafova stekao je veliku popularnost u industriji kompjuterskih igrica te danas ima mnogo-brojne primjene. Algoritam je objavljen 1968. godine od strane Peter Hart, Nils Nilsson i Bertram Raphael-a, istraživača sa Stanfort instituta. Nakon što su Nilsson i Raphael izvršili niz promjena nad Hart-ovim prvobitnim modifikacijama Dijkstrijevog algoritma nazvanog A1, njih trojica su formirali dokaz da je novi algoritam, koji je nazvan A*, optimalan pod dobro definisanim uslovima, što je matematski dokazano [6].

Kao i većina algoritama pretrage, cilj A* algoritma je da od svih mogućih puteva nađe onaj put koji od starta do cilja posjeduje najmanju cijenu (u ovom slučaju je cijena udaljenost). Jedna od karakteristika A* algoritma jeste da, za razliku od svog prethodnika Dijkstre, ima jednu vrstu "osjećaja" udaljenosti od cilja te je usmjeren

prema cilju. Taj "osjećaj" usmjerenosti ka cilju realizovan je pomoću heurističke funkcije koja se računa za svaki čvor i igra značajnu ulogu u njihovoj evaluaciji. Svaki čvor opisan je funkcijom:

$$f(n) = g(n) + h(n)$$

pri čemu su:

n – čvor koji razmatramo
 $g(n)$ – udaljenost čvora od početka
 $h(n)$ – heuristička funkcija

Funkcija $g(n)$ zapravo predstavlja najkraću udaljenost čvora koji razmatramo od početka po do sada obrađenim putevima. Ukoliko bi $g(n)$ bio zanemaren (odnosno $g(n) = 0$ za svako n), A* se svodi na najobičniji pohlepni (eng. greedy) algoritam, pod uslovom da je heuristička funkcija računa udaljenost od trenutnog čvora do cilja. Postavljanjem heurističke funkcije $h(n) = 0$ za svako n , A* se svodi na Dijkstrin algoritam, tako da se Dijkstrin algoritam zapravo može posmatrati kao modifikacija A* algoritma sa zanemarenom heurstikom [7]. O heurstici i njenom odabiru će biti riječi nešto kasnije.

U konačnici se može reći da A* pripada grupi algoritama poznatoj kao "best-first search", odnosno naredni čvor koji će biti uzet u razmatranje je onaj čvor koji je najvjerojatniji da dovede do cilja.

A* algoritam se može iskazati u formi sljedećeg pseudokoda [2]:

```

1  function A_Star( start , goal )
2      closedSet := {}
3      openSet := { start }
4
5      cameFrom := an empty map
6      gScore := map with default value of Infinity
7      gScore[ start ] := 0
8      fScore := map with default value of Infinity
9
10     fScore[ start ] := heuristic_cost_estimate( start , goal )
11
12     while openSet is not empty
13         current := the node in openSet having the lowest fScore[] value
14         if current = goal
15             return reconstruct_path( cameFrom , current )

```

```

16
17     openSet.Remove( current )
18     closedSet.Add( current )
19
20     for each neighbor of current
21         if neighbor in closedSet
22             continue
23
24         if neighbor not in openSet
25             openSet.Add( neighbor )
26
27         tentative_gScore := gScore[current] + dist_between(current,
28                                         neighbor)
28         if tentative_gScore >= gScore[neighbor]
29             continue
30
31         cameFrom[neighbor] := current
32         gScore[neighbor] := tentative_gScore
33         fScore[neighbor] := gScore[neighbor] +
34                                         heuristic_cost_estimate(neighbor, goal)
35
35     function reconstruct_path(cameFrom, current)
36         total_path := {current}
37         while current in cameFrom.Keys:
38             current := cameFrom[current]
39             total_path.append(current)
40         return total_path

```

Formiraju se dva skupa - *openSet* i *closedSet*. U *openSet* se na početku rada algoritma ubacuje samo početni čvor, te se u *openSet* stavlja svi novootkriveni čvorovi. U *closedSet* se potom prebacuju svi čvorovi čija vrijednost je evaluirana i konačna. Ova izvedba algoritma funkcioniše pod pretpostavkom da je vrijednost čvorova koji se ubacuju u *closedSet* zaista konačna vrijednost te da se čvorovi, nakon što se jednom ubace u *closedSet*, iz njega više ne mogu izbaciti. Postoje izvedbe algoritma gdje to nije slučaj, međutim to nije predmet ovog rada.

Potom se formiraju tri mape koje su imenovane *cameFrom*, *gScore* i *fScore*. Mape *gScore* i *fScore* predstavljaju vrijednosti $g(n)$ i $f(n)$ čvorova respektivno, te svakom otkrivenom čvoru pridružuju odgovarajuću vrijednost. Međutim, kako je na početku rada algoritma poznat samo početni čvor, njegov $g(n)$ je svakako nula, a $f(n)$ uzima čistu heurističku vrijednost $h(n)$. Vrijedi pomenuti da je bilo moguće formirati mapu

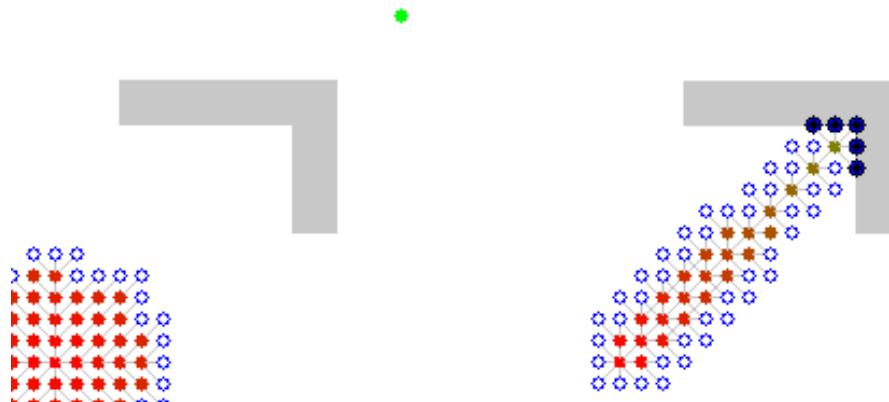
$hScore$, koja bi odgovarala heurističkoj funkciji $h(n)$, ali za tim nije bilo potrebe jer je $h(n)$ konstantno za čvor koji razmatramo, dok se $g(n)$ i $f(n)$ mijenjaju ovisno od puta kojim se dolazi u čvor n . Da bi ustanovili put kojim se može doći do čvora n služi nam mapa *cameFrom* u kojoj se svakom čvoru n bilježi iz kog čvora je moguće doći do njega. Ukoliko je moguće u čvor n doći iz više čvorova, pamti se onaj iz kog je prelazak najisplativiji.

Od linije broj 12 počinje ključni dio algoritma. Iz *openSet*-a se uzima čvor koji ima najmanji *fScore* (u početku se u *openSet*-u nalazi samo početni čvor). Provjerava se da li je taj čvor ujedno i odredišni čvor, i ako nije izbacuje se iz *openSet*-a i ubacuje se u *closedSet*. Na taj način se usput i označava da je do trenutnog čvora dostignut najbolji mogući put. Nakon toga se identificiraju svi čvorovi do kojih se može doći iz trenutnog čvora. Ti čvorovi se nazivaju komšije. Za svakog komšiju trenutnog čvora provjerava se da li je već evaluiran i smještan u *closedSet*. Ukoliko nije, dodaje se u *openSet* i za njega se računa nova vrijednost $g(n)$. Pošto je najčešće u jedan čvor moguće doći na više načina, potrebno je provjeriti da li je pronađeno $g(n)$ koje je manje od trenutnog $g(n)$ za izabranog komšiju. Ukoliko je takvo $g(n)$ pronađeno, ažuriraju se mape *cameFrom*, *gScore* i *fScore*. Proces se ponavlja uzimanjem novog čvora sa najmanjim *fScore*-om iz *openSet*-a dok se ne dođe do odredišnog čvora.

Nakon što se pronađe odredišni čvor, potreno je rekonstruisati put od odredišta do početka. To se odvija u funkciji na liniji 35 koja se u ovom pseudokodu naziva *reconstruct_path*. Unutar nje se, pomoću mape *cameFrom* pronalazi put do samog početka, i svaki obrađeni čvor se dodaje u niz *total_path*. Nakon toga se vraća *total_path* i algoritam završava.

2.2.2. Ilustracija rada algoritma

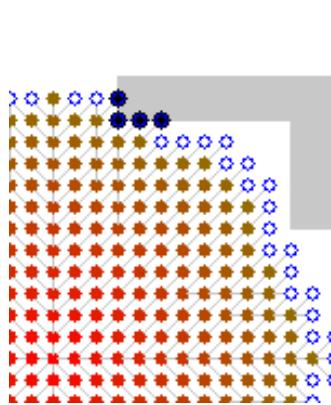
U nastavku će biti analiziran grafički prikaz rada A* algoritma paralelno sa Dijkstrinim algoritmom [6] [16]. Na narednim slikama ispunjeni krugovi predstavljaju posjećene čvorove (čvorove koji su prebačeni u *closedSet*), dok prazni krugovi (plavi) predstavljaju otkrivene čvorove. Tamno plavi krugovi predstavljaju čvorove koji su nedostupni (prepreke), a zeleni krug je ciljni čvor.



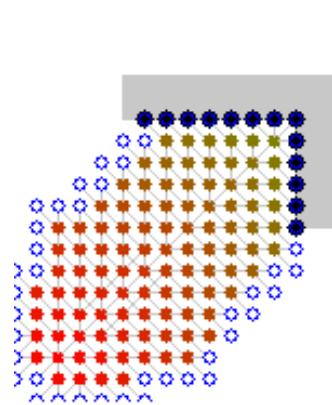
Slika 2.1. Dijkstra - početak pretrage [1]

Slika 2.2. A* - početak pretrage [2]

Na slikama 2.1. i 2.2. može se vidjeti početak rada ova dva algoritma. Kako Dijksttin algoritam ne posjeduje heurističku funkciju koja ga usmjerava prema cilju, on će nastaviti da se širi (kao talas) oko početnog čvora. A* usmjerava heurstika prema cilju, međutim nailaskom na prepreku morat će otvoriti nove čvorove koji su bliži početku i "širiti" se.

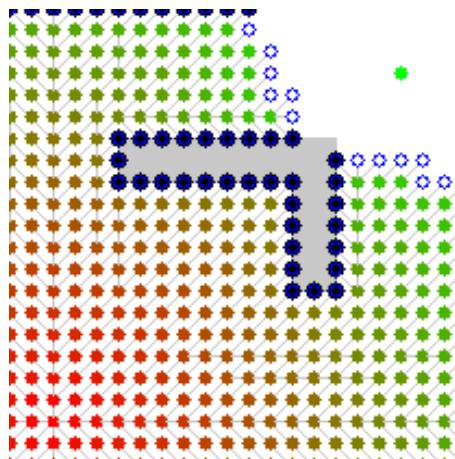


Slika 2.3. Dijkstra - prepreka [1]

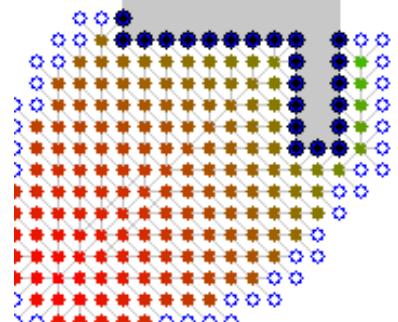


Slika 2.4. A* - prepreka [2]

Na slikama 2.3. i 2.4. se vidi da su oba algoritma naišla na prepreke. Dijkstra nastavlja da otvara čvorove koji su na jednakoj udaljenosti od početkog čvora dok se A* nastavlja širiti u oba smjera od dijagonale dok ne nađe put oko prepreke.

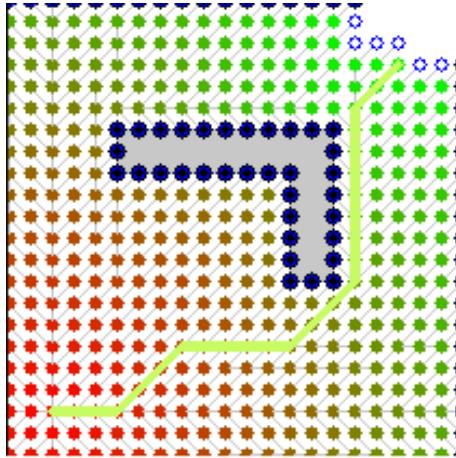


Slika 2.5. Dijkstra - zaobilazak prepreke [1]

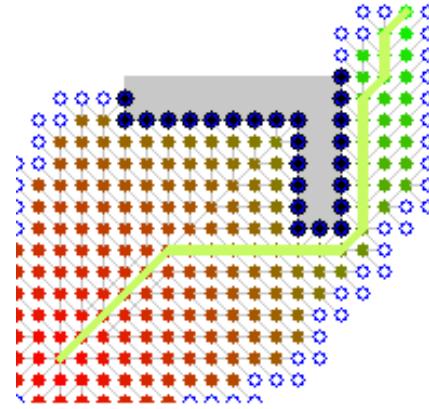


Slika 2.6. A* - zaobilazak prepreke [2]

Dalje se vidi na slici 2.5. da Dijkstra nastavlja otvarati čvorove koji leže na jednakoj udaljenosti od početnog čvora, dok na slici 2.6. vidimo da je A* našao put oko prepreke i nastavlja otvarati čvorove koji ga usmjeravaju prema cilju. U ovoj fazi možda je razlika između ova dva algoritma i najupečatljivija. Dijkstra je analizirao čvorove koji se nalaze na samim uglovima mape dok je A* birao samo one čvorove koji ga vode cilju.



Slika 2.7. Dijkstra - konačni put [1]



Slika 2.8. A* - konačni put [2]

Na kraju, na slikama 2.7. i 2.8. može se vidjeti da su oba algoritma našla put do odredišta. Iako ova dva puta na različiti, oni su iste dužine. Međutim, broj čvorova koji su ova dva algoritma analizirala nije isti. A* algoritam je analizirao značajno manje čvorova od Dijkstrinog algoritma što ujedno znači i kraće vrijeme pretrage. U konačnici, sa adekvatnom heurističkom funkcijom, A* se pokazuje kao znatno bolji algoritam. O heuristici će biti više govora u narednom dijelu.

2.2.3. Heuristika

Odabir adekvatne heuristike je značajan za ispravno funkcionisanje algoritma [8]. Ukoliko heuristička funkcija nikad ne precjenjuje minimalnu udaljenost od trenutnog čvora do cilja onda A* može da izračuna optimalnu vrijednost puta i to eliminiše potrebu za *closedSet*-om. Odnosno, da bi eliminisali *closedSet* heuristička funkcija mora vraćati vrijednost koja nikad neće biti veća od najmanjeg mogućeg puta od trenutnog čvora do cilja. To garantuje da se čvor, nakon što jednom bude posjećen, više neće posjećivati.

Ukoliko ipak koristimo *closedSet* onda mora da vrijedi uslov monotonosi odnosno heuristička funkcija mora biti konzistentna da bi A* bio optimalan. To znači da će procijenjena vrijednost uvijek biti manja ili jednaka pravoj vrijednosti dolaska do cilja plus vrijednost prelaska iz trenutnog čvora u komšiju. Za par čvorova x i y , gdje je $d(x, y)$ udaljenosti između njih, vrijedi:

$$h(x) \leq d(x, y) + h(y)$$

Time se osigurava da će čvor biti samo jednom posjećen. Postoji više filozofija oko odabira heuristike, međutim najčešće se uzimaju Euklidska udaljenost ili Manhattan udaljenost [9]. Euklidska udaljenost predstavlja pravolinijsku udaljenost između trenutnog čvora i odredišta, dok Manhattan udaljenost predstavlja sumu apsolutnih razlika njihovih Kartezijanskih koordinata. Također je moguće dodatno manipulisati algoritmom dodavajući težinske koeficijente na heuristiku, čineći algoritam više ili manje pohlepnim. Postoji niz metoda koji koriste ovu osobinu u pokušaju da čine

algoritam bržim sa ili bez gubitka cijene puta [10] [11]. Međutim te metode neće biti razmatrane u ovom radu.

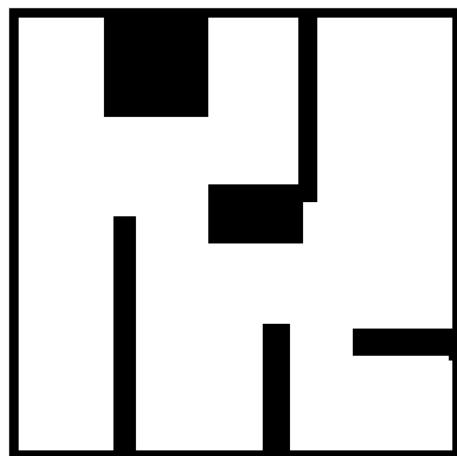
Na kraju treba naglasiti i da ukoliko se implementacija ovog algoritma realizuje preko prioritetnih redova, način na koji se bira između dva čvora sa istim $fScore$ -om može da ima značajne implikacije na performanse algoritma jer su takve situacije dosta česte.

Iako postoje situacije u kojima A* nije najbolji algoritam za rješenje odgovarajućeg problema, iz priloženog vidimo da je za konkretan zadatak značajno bolji od svog prethodnika te vrijedi analizirati rezultate dobijene korištenjem A*-a za pretragu nad quadtree-em, o čemu će biti više riječi u narednom dijelu.

2.3. Mrežasta mapa

2.3.1. Opis

Mrežasta mapa predstavlja najjednostavniji način reprezentacije dvodimenzionalnog prostora. Kao što ime sugerire, ona prostor modelira kao mrežu (eng. grid) te ovisno od sadržaja pojedinih celija, dodjeljuje vrijednosti 0 ili 1. Najčešće 0 označava zauzeto (okupirano, neprohodno) polje dok 1 označava prazno polje. Koristeći razvojno okruženje Matlab i funkciju "*imshow*" urađena je vizualizacija mrežaste mape koja je vidljiva na slici 2.9.. Tako da, ukoliko algoritmom pretrage radi na mrežastoj mapi, on će u razmatranje uzimati samo ona polja koja sadrže vrijednost 1.



Slika 2.9. Primjer okoline sa preprekama predstavljenim regionima crne boje; Bijelom bojom su obojeni regioni koji predstavljaju slobodni prostor

Moguće je praviti varijacije mrežaste mape u kojoj se pored vrijednosti 0 i 1 čuvaju vrijednosti između te se na taj način označavaju teže ili lakše prohodne oblasti ili različiti materijali po kojima se robot kreće. To zahtjeva i prilagođavanje algoritma pretrage jer u takvim okolnostima najkraći put nije nužno i onaj put koji će nas najbrže dovesti do cilja, ali to nije predmet razmatranja ovog rada.

2.3.2. Implementacija

Postoji više načina kako se mrežasta mapa može implementirati. Kako mrežasta mapa nije ništa drugo nego matrica, najčešće se tako i implementira. Unutar strukture se deklariše niz dimenzija m i n i u njega se učitaju vrijednosti. Time kreiramo strukturu kao što je vidljiva u tabeli 2.1.

Tabela 2.1. 2D matrica

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2

$a[red, kolona]$
 $a[red][kolona]$

Ovo naravno nije jedini mogući način implementacije mrežaste mape. Implementacija je moguća i koristeći obični niz, tako da se svi redovi povežu u jedan veliki red kao što je vidljivo u tabeli 2.2.

Tabela 2.2. Kontinualno alocirana matrica

0	1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---	---

$a[row * width + column]$

Kod ovakvih implementacija mora se voditi računa o elementu kom se pristupa zbog specifičnog načina indeksiranja.

Pored ovih načina moguće je formirati mrežu kao niz nizova, čime formiramo strukturu kao u tabeli 2.3.

Tabela 2.3. Niz nizova

0	→	0	1	2
1	→	0	1	2
2	→	0	1	2

$a[red][kolona]$

Kod ovakve implementacije, a predstavlja niz pokazivača koji pokazuju na redove matrice. Ovakva implementacija nije rijetka, pogotovo u situacijama kada je potrebno učitati velike mape. Kako je programski stek ograničene veličine, matricu je potrebno dinamički alocirati.

Pored ovoga, dinamička alokacija igra ulogu u posljednjoj strukturi podataka koju ćemo obraditi u ovom poglavlju, a to su 2D dvostruko povezane liste, ilustrirane u tabeli 2.4.

Tabela 2.4. 2D Dvostruko povezana lista

0	↔	1	↔	2
⇓		⇓		⇓
3	↔	4	↔	5
↑		↓		↑
6	↔	7	↔	8

Kod ovakve strukture podataka potrebno je da svako polje bude alocirano kao čvor koji sadrži 4 pokazivača na svoje komšije. Iako djeluje isuviše komplikovano, ovakva reprezentacija matrice je pogodna zbog specifičnosti i osobina koje povezane liste nude te se koristi za implementaciju *algoritma X*. [12]

2.3.3. A* algoritam i mrežasta mapa

Mrežasta mapa je pogodna za implementaciju A* algoritma, uz odgovarajuće modifikacije. Svako polje mape u A* algoritmu bi predstavljalo čvor sa koordinatima (x, y) . To znači da bi zapravo svaka lokacija u matrici (pod uslovom da se radi sa cijelim brojevima) predstavljala jedan čvor. Ovisno o vrijednosti tog čvora on bi bio prohodan ili neprohodan. Nakon što smo ustanovili da će čvorovi biti zamijenjeni parovima (x, y) , *openSet* i *closedSet* će predstavljati skupove parova, a mape *cameFrom*, *fScore* i *gScore* sada svakom paru dodjeljuju jedinstvenu vrijednost.

Pretraga komšija je izuzetno pogodna u mrežastim mapama, jer ukoliko želimo da analiziramo komšije bilo kojeg čvora, prosto je potrebno posmatrati sve parove koji se nalaze u gornjem i donjem redu iste kolone, i lijevoj i desnoj koloni istog reda. Uslov se može proširiti vrlo jednostavno i za polja dijagonalno od trenutnog para. Pri pristupu ovim poljima, potrebno je samo voditi računa da index polja kojima se pristupa ne ispada iz prihvatljivog opsega.

Jedan od nedostataka ove implementacije i generalno mrežaste mape jeste jako velik broj čvorova koji se moraju pamtit i obraditi [13]. Svako polje ima određenu širinu i dužinu, a kako ove mreže mogu da budu velike rezolucije, širina i dužina mogu biti jako male, što rezultuje u povećanom broju čvorova. Pošto A* ne može preskakati iste čvorove na putu do cilja, već mora ići od komšije do komšije, pretraga ove vrste može biti jako spora upravo zbog velikog broja čvorova koje je potrebno obraditi i to će ujedno biti eksperimentalno pokazano i u ovom radu.

Mrežasta mapa ima brojne prednosti u odnosu na druge načine reprezentacije prostora, prvenstveno zbog detalja koje može da prikaže. Međutim u okruženjima sa kakvim se najčešće susrećemo, dijelovi mape koje posmatramo nemaju toliko rafidne promjene da je potreno modelirati svaki detalj, a ako postoje takvi dijelovi,

onda su oni najčešće izolovani. Mape hodnika i drugih dijelova kroz koje je potrebno navigirati su najčešće monotoni, što znači da ako u mrežastoj mapi nađemo kvadrat koji je prazan, velike su šanse da su svi kvadrati oko njega takođe prazni. Isto vrijedi i za okupirane kvadrate. Iz ove činjenice proizilazi da mrežaste mape jako često bespotrebno zauzimaju puno memorijskog prostora. U nastavku će biti predstavljeno alternativno rješenje, i tema ovog rada, a to je quadtrees i octrees.

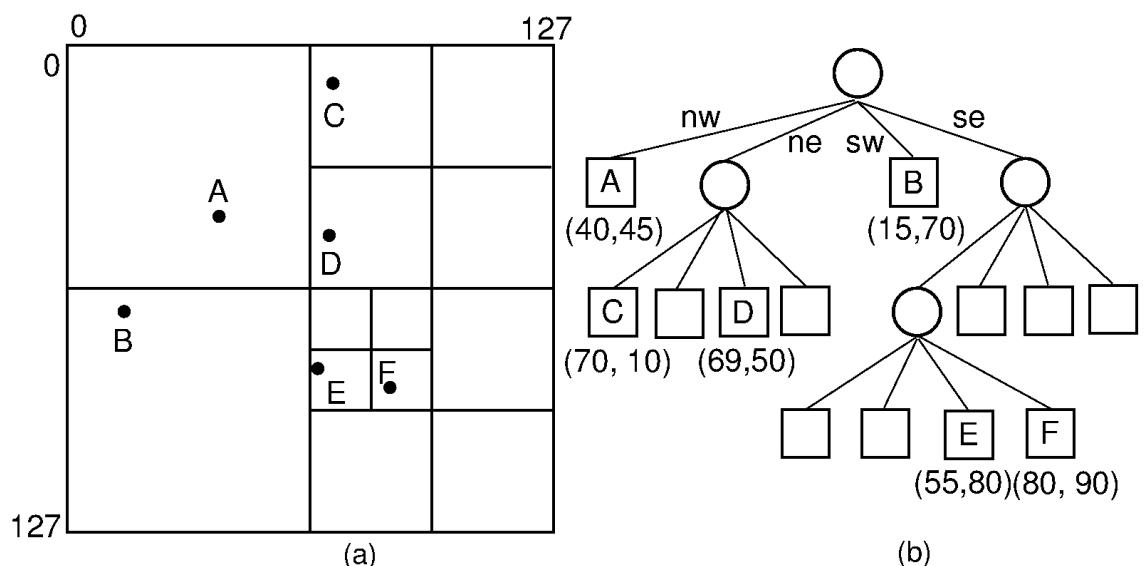
2.4. Quadtree

2.4.1. Opis quadtree-a

Stablo predstavlja apstraktnu strukturu podataka koja je imala značajnu primjenu u računarstvu. Stablo karakteriše hijerarhijski poredak čvorova, gdje svaki čvor sadrži referencu na druge čvorove (djecu). Poseban oblik stabla, koji je takođe često korišten u praksi, jeste binarno stablo, gdje svaki čvor može da posjeduje do dvoje djece. Svaki čvor ima jedan roditeljski čvor, osim korijenskog čvora (eng. root node) od kog počinje grananje stabla.

Quadtree je ime dano strukturi podataka od strane R.A.Finkel i J.L.Bentley-a u njihovom radu "Quad Trees A Data Structure for Retrieval on Composite Keys" objavljenom 1974. godine [14]. U tom radu, Finkel i Bentley predstavljaju novi način reprezentacije 2D prostora koristeći stabla. Kod ovog stabla, svaki čvor ima tačno četiri reference na djecu. Na taj način quadtree dijeli prostor (univerzum) na četiri kvadranta obično imenovani prema stranama svijeta (eng. NW NE SW SE). Daljim dijeljenjem čvorova parčamo svaki od ovih kvadrantata na manje i manje dijelove dok ne dobijemo traženu dubinu. Veća dubina stabla označava i veću rezoluciju.

Da bi bio bolje opisan način na koji je quadtree implementiran, priložena je slika 2.10..



Slika 2.10. Primjer dekompozicije prostora u quadtree i reprezentacija stabla [3]

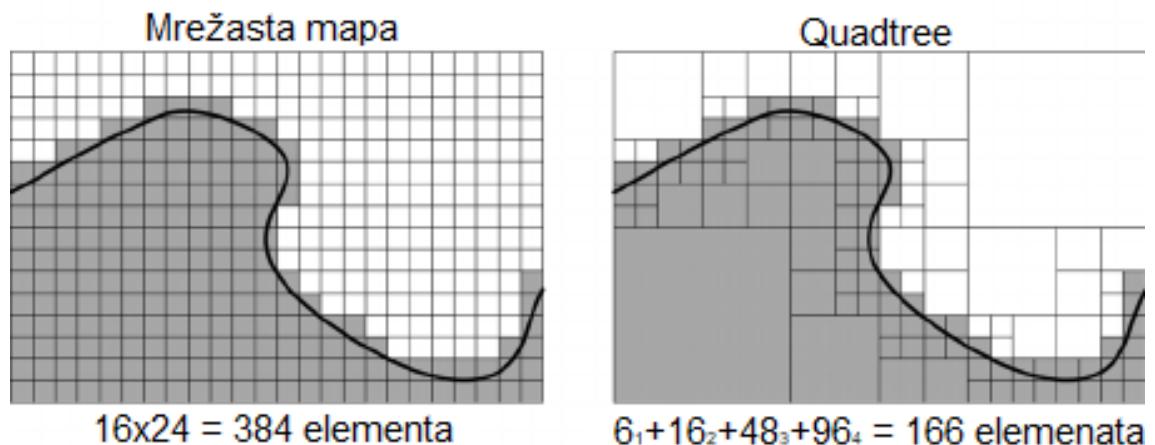
Na slici 2.10. pod (b) jasno je prikazana struktura quadtree-a, dok je na pod (a) označena grafička reprezentacija podjele prostora. Kvadratom su označeni listovi (eng. leaf nodes), odnosno čvorovi koji nemaju djecu, dok su krugovima označeni roditeljski čvorovi. Na slici (a) prikazane su tačke od A do F i njihove odgovarajuće koordinate kom od "listova" stabla oni pripadaju. Neka tačka A ima koordinate (40,45). Nakon što se prostor dimenzija (127, 127) podijeli u četiri jednaka dijela, očigledno je da tačka A pada u nw kvadrant koji obuhvata koordinate od 0 do 64 po širini i dužini. Kako ovaj kvadrant nije dalje dijeljen, što možemo da uočimo i na slici (a) i na slici (b), pronašli smo čvor unutar kog se nalazi tačka A. Da je ovaj čvor zaista imao djecu, rekurzivno bi nastavili parčati prostor dok ne nađemo tačku A.

Ne mora nužno uvijek biti slučaj da se prostor dijeli na četiri jednaka kvadranta. Univerzum je moguće dijeliti i neravnomjerno, tako da na primjer lijeva polovina quadtree-a bude duplo manja od desne, ili koja je god konfiguracija korisniku potrebna, s tim da je u tim slučajevima potrebno čuvati više podataka unutar svakog od čvorova i prilikom dijeljenja čvora potrebno je uložiti više truda u računanje tačnih granica između čvorova, međutim u ovom radu će se obrađivati isključivo ravnomjerno dijeljenje prostora.

Po pitanju granica, potrebno je definisati pravilo pripadnosti za slučaj kada se tačka nađe tačno na granici između dva čvora. Finkel i Bentley predlažu ideju otvorenih i zatvorenih čvorova, konkretno da su čvorovi 1 i 3 zatvoreni a čvorovi 2 i 4 otvoreni, tako da kada tačka padne na granicu, ona zapravo pripada otvorenim čvorovima. U suštini kakva god raspodjela da se izabere, ne bi trebalo da predstavlja problem dok god su sve situacije pokrivene i dobro definisane.

2.4.2. Uporedba quadtree-a i mrežaste mape

Iako je quadtree struktura koja je dosta kompleksnija od mrežaste mape za implementaciju, memorijski je puno manje zahtjevna. To je prikazano na slici 2.11.



Slika 2.11. Usporedba mrežaste mape i quadtree-a [4]

Slika 2.11. crnom linijom ilustrira konturu prepreke, te se površina ispod krive

može posmatrati kao zauzet (neprohodan) prostor, dok je površina iznad krive slobodan prostor. Ova slika izuzetno dobro demonstrira razliku između ove dvije strukture podataka. Kao što je rečeno u prethodnim poglavljima, mrežasta mapa svaki dio terena tretira jednakom i svaki čvor mrežaste mape ima istu površinu. To uzrokuje kreiranje velikog broja redundantnih čvorova.

Quadtree nema probleme tog tipa. Ukoliko veliko područje ima istu vrijednost, ono će biti reprezentirano kao jedan veliki čvor, jer nema potrebe da se dijeli na manje čvorove. Tako nešto se može uočiti u gornjem lijevom i donjem desnom uglu desne slike. Veća preciznost potrebna je na prelazima između zauzetog i slobodnog prostora, odnosno oko crne linije kao što može biti viđeno na slici. Oko granice quadtree se sve više parča da bi što preciznije modelirao graničnu liniju. Ovo parčanje se može vršiti u nedogled, ali obično se zaustavlja kada čvor dostigne neku fiksnu širinu ili dubinu stabla.

Svakim novim parčanjem kreiraju se 4 nova čvora, pri čemu se ne smiju zanemariti roditeljski čvorovi. Na desnoj ilustraciji slike 2.11. može se uočiti 6 čvorova prvog nivoa (dubina 1, ako je korijenski čvor 0), 16 čvorova dubine 2 (koji se nalaze unutar čvorova prvog nivoa), 48 čvorova trećeg nivoa i 96 čvorova četvrtog nivoa. To rezultuje konačnim brojem od 166 čvorova, što je više nego duplo manje u odnosu na običnu mrežastu mapu.

Međutim postoje situacije kada se mreža ponaša bolje od quadtree-a. To su obično situacije kada se zauzeće čvorova rapidno mijenja te je potrebna velika rezolucija za modeliranje takve situacije. Uzmimo najgori mogući slučaj i zamislimo da je ilustracija sa prethodnog primjera bila šahovnica dimenzija 16x24. Kako u takvoj situaciji nije moguće ujediniti susjedne čvorove, quadtree bi morao imati 6 čvorova prvog nivoa, $6 \cdot 4$ čvorova drugog nivoa, $6 \cdot 4 \cdot 4$ čvorova trećeg nivoa i $6 \cdot 4 \cdot 4 \cdot 4$ čvorova četvrtog nivoa. To je ukupno 510 čvorova. To nije toliko strašno u poređenju sa 384 čvora koju bi imala mreža, ali jeste situacija u kojoj se mreža ponaša bolje od stabla.

Postoji još jedan nedostatak quadtree-a u odnosu na mrežastu mapu, a to je pretraga komšija. Kako svaki čvor pojedinačno ne zna kojih dimenzija su mu komšije ili svoju poziciju u odnosu na druge, jako je teško navigirati kroz stablo. Da bi se pronašle sve komšije nekog čvora, potrebno je krenuti od korijenskog čvora i spuštati se dublje kroz stablo vršeći rekurzivno pretragu da li čvorovi sadrže neke od graničnih tačaka trenutnog čvora. Ovo je značajno komplikovaniji proces, u odnosu na prosto posmatranje susjednih redova i kolona mrežaste mape.

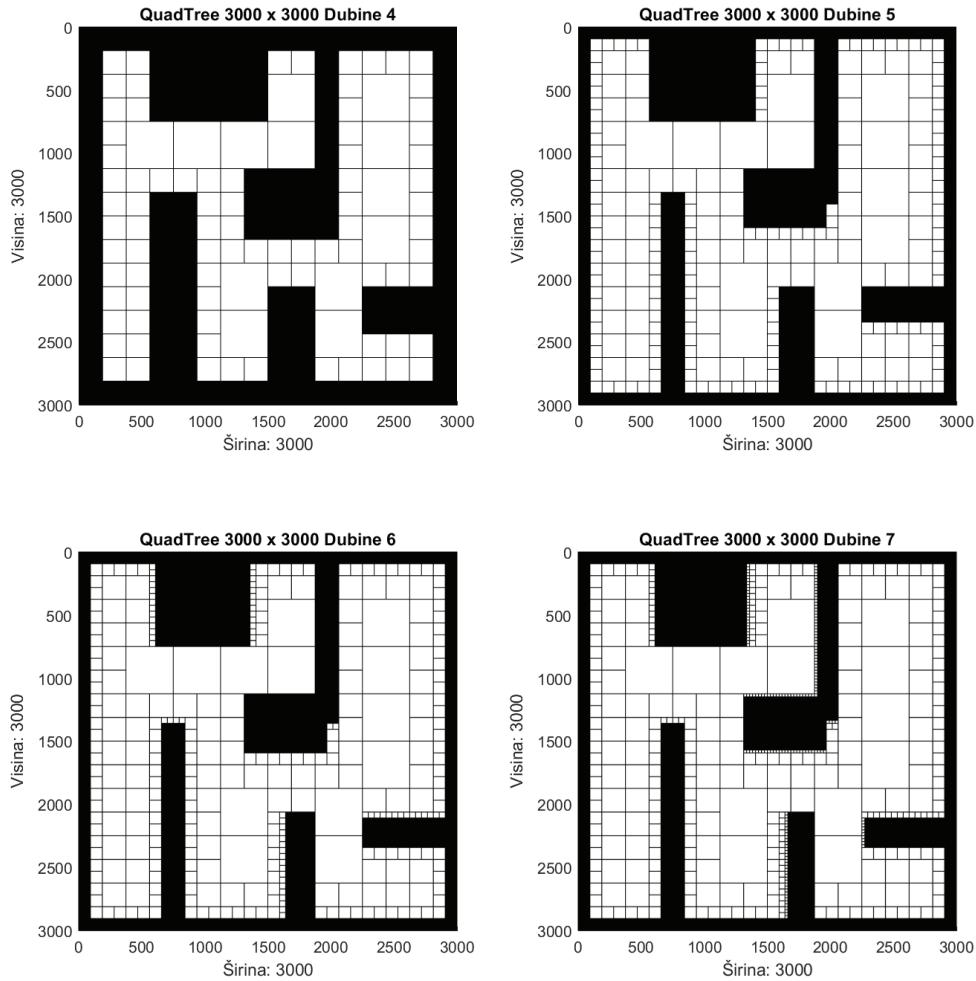
2.4.3. Dubina

Već je u prethodnom dijelu naglašen značaj koji dubina ima u quadtree-u, međutim, to se može i matematski izraziti formulom:

$$\text{širina_čvora} = \frac{\text{širina_mape}}{2^{\text{dubina}}}$$

Ukoliko je širina mape 100, a dubina 4, širina čvora će iznositi 6.25. Na dubini 7 širina jednog čvora će biti nešto veća od 0.78. Ako ponovo uzmemo u razmatranje sliku 2.9., koju smo koristili za demonstraciju mrežaste mape, i iz nje generišemo quadtree, kao što je ilustrirano na slici 2.12., uočljive su razlike između dubina.

Respektivno, ilustracije na slici 2.12. predstavljaju dubine 4, 5, 6 i 7. Imajući na umu da je svaki čvor na originalnoj mapi imao širinu 1x1, a da na dubini 4 čvorovi imaju širinu 6.25, uočljivi su značajni gubici preciznosti. Na dubini 5 i 6 gubici su zanemarivo mali, dok na dubini 7 gubici su skoro pa nepostojeći (kako je 0.78 manje od 1, da bi se predstavila linija širine 1 potrebno je koristiti dva ova čvora, međutim do takvih nepreciznosti dolazi i kod mrežaste mape i nisu vrijedni razmatranja).



Slika 2.12. Prikaz dekompozicije prostora u quadtree-eve različitim dubinama

Iako većom dubinom mape dobijamo na preciznosti i rezoluciji, dubina ne mora nužno biti dobra stvar. Ukoliko je mapa previše duboka, a prostor se modelira tako što se unosi tačka po tačka, zbog diskretne vrijednosti tačaka može se desiti da se između dvije susjedne tačke koje predstavljaju okupiran prostor formiraju čvorovi-djeca koji nisu okupirani. To se najčešće desi ako je širina čvora stabla puno manja nego širina čvora mreže iz koje se stablo modelira. Međutim to je van opsega ovog rada. Kako dubina 6 daje širinu čvora 1.56, ona je sasvim zadovoljavajuća za primjenu A* algoritma.

2.4.4. Apstraktni opis klase

Dati su naredni elementi potrebni za implementaciju quadtree-a [15]:

```

1 struct XY
2 {
3     float x;
4     float y;
5
6     function __construct( float _x, float _y) { ... }
7 }
```

```

1 struct AABB
2 {
3     XY center;
4     float halfDimension;
5
6     function __construct(XY center , float halfDimension ) { ... }
7     function containsPoint(XY point) { ... }
8     function intersectsAABB(AABB other) { ... }
9 }
```

Struktura *XY* predstavlja obični par Kartezijanskih koordinata, dok struktura *AABB* predstavlja čvor sa centrom u tački *XY* i polovinom njegove širine. Iz centra i poluprečnika moguće je rekonstruisati sve ostale tačke. Pomoćne metode *containsPoint* i *intersectsAABB* pomažu prilikom određivanja da li se neka tačka nalazi u konkretnom čvoru te da li čvor ima dodirnih površina sa drugim čvorovima.

Nakon što su definisane pomoćne klase, moguće je preći na interface samog quadtree-a:

```

1 class QuadTree
2 {
3     constant int QT_NODE_CAPACITY = 4;
4     AABB boundary;
5     Array of XY [ size = QT_NODE_CAPACITY] points ;
6
7     QuadTree* northWest ;
8     QuadTree* northEast ;
9     QuadTree* southWest ;
```

```

10    QuadTree* southEast;
11
12    function __construct(AABB _boundary) { ... }
13    function insert(XY p) { ... }
14    function subdivide() { ... }
15    function queryRange(AABB range) { ... }
16 }
```

Odmah su uočljiva četiri pokazivača na *QuadTree* koji predstavljaju djecu ovog quadtree-a. Imenovani su adekvatno po stranama svijeta. Pored djece, quadtree je opisan jednim elementom tipa *AABB*, koji smo analizirali u prošlom isječku koda. On predstavlja granice trenutnog quadtree-a. Dva dodatna parametra koji su vidljivi u ovoj implementaciji su *QT_NODE_CAPACITY* i *points*. Ova dva atributa nisu toliko relevantna za ovaj rad, ali vrijedi ih pomenuti. U opisu, na slici 2.10. bilo je govora o tome da se tačke A, B, C i td. nalaze unutar listova quadtree-a, što omogućava ubrzanu pretragu prostora. Čuvanje tačaka je upravo realizovano pomoću niza *points* koji je zadane konstantne veličine *QT_NODE_CAPACITY*. Taj detalj nije od velikog značaja za ovaj rad, te će u trećem poglavljju biti više govora o implementaciji.

Od metoda *QuadTree*-a, upečatljive su metode *insert(XY p)*, *querryRange(AABB range)* i *subdivide()*. Metoda *subdivide()* ima prostu ulogu da, ukoliko pokazivači nw, ne, sw i se nisu inicijalizirani (odnosno ne pokazuju na objekte tipa *QuadTree*), podijeli postojeći quadtree na 4 nova i svakom pokazivaču dodijeli jedan novi čvor. Metode *insert* i *querryRange* vrijedi uzeti dodatno pod lupu. U nastavku, dat je pseudokod metode *insert*:

```

1 class QuadTree
2 {
3     ...
4     function insert(XY p)
5     {
6         if (!boundary.containsPoint(p))
7             return false;
8
9         if (points.size < QT_NODE_CAPACITY)
10        {
11            points.append(p);
12            return true;
13        }
14    }
```

```

15     if (northWest == null)
16         subdivide();
17
18     if (northWest->insert(p)) return true;
19     if (northEast->insert(p)) return true;
20     if (southWest->insert(p)) return true;
21     if (southEast->insert(p)) return true;
22
23     return false;
24 }
25 }
```

Metoda je vrlo jednostavna, ali je njeno razumijevanje od velikog značaja. Prvo se vrši provjera da li se tačka p , koja se prima kao argument funkcije, nalazi unutar granica ovog čvora. Ukoliko je to slučaj, provjerava se da li čvor ima kapacitet da primi tačku. Ukoliko je čvor već popunjén, vrši se podjela čvora i redom se pokušava u svaki od čvorova ubaciti tačku p . Ukoliko dođe do greške, vraća se vrijednost false.

U nastavku dat je pseudokod metode *querryRange*:

```

1 class QuadTree
2 {
3     ...
4     function queryRange(AABB range)
5     {
6         Array of XY pointsInRange;
7
8         if (!boundary.intersectsAABB(range))
9             return pointsInRange; // empty list
10
11        for (int p = 0; p < points.size; p++)
12        {
13            if (range.containsPoint(points[p]))
14                pointsInRange.append(points[p]);
15        }
16
17        if (northWest == null)
18            return pointsInRange;
19    }
```

```

20     pointsInRange.appendArray( northWest->queryRange( range ) );
21     pointsInRange.appendArray( northEast->queryRange( range ) );
22     pointsInRange.appendArray( southWest->queryRange( range ) );
23     pointsInRange.appendArray( southEast->queryRange( range ) );
24
25     return pointsInRange;
26 }
27 }
```

I ova metoda je jednostavna s tim da njeno razumijevanje nije od velikog značaja za ovaj rad, ali služi kao dobra ilustracija rekurzivne prirode stabala. Funkcija kao argument prima objekat range koji je tipa *AABB*. Iz prethodih primjera kod poznato je da objekat *AABB* predstavlja određenu površinu. Uloga ove funkcije je da vrati sve tačke koje su zahvaćene tom površinom.

Prvo se provjerava da li ova površina uopšte ima dodira sa površinom koju obuhvata trenutni čvor. Nakon toga se provjeravaju sve tačke iz niza points. Ukoliko tačka iz niza pada na površinu *range*, potrebno ju je označiti. Označavanje se vrši tako što se tačke smještaju u niz tačaka imenovan *pointsInRange*. Nakon što se provjere sve tačke ovog čvora, potrebno je provjeriti njegovu djecu - pod uslovom da ima djecu. Ukoliko ima, rekurzivno se nad svakim djetetom poziva metoda *querryRange* prosljeđujući joj, ponovo, range kao parametar i njen se rezultat nadovezuje na niz *pointsInRange*. Nakon toga se niz vraća iz funkcije.

U ovoj implementaciji, quadtree se može više posmatrati kao objekat za čuvanje i pretragu određenih tačaka nekog prostora, te nije pogodan za modeliranje prepreka, o čemu je bilo govora na početku. Potrebno je izvršiti niz modifikacija nad ovim pseudokodom da bi on bio pogodan za primjenu A* algoritma, ali o tome će biti govora u narednim poglavljima.

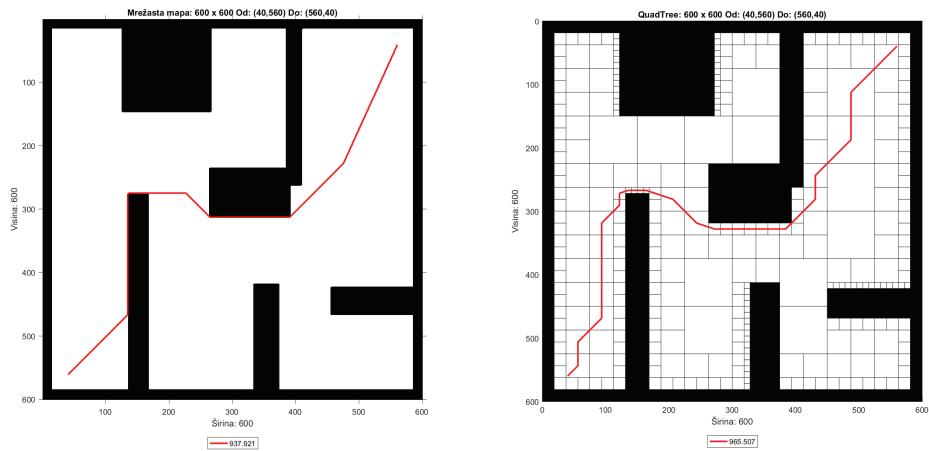
2.4.5. A* algoritam i Quadtree

Da bi bilo moguće koristiti quadtree za pretragu, potrebno je izvršiti niz promjena kako na pseudokodu algoritma tako i na pseudokodu samog stabla. *QuadTree* je potrebno prilagoditi tako da umjesto niza tačaka koje čuva, čuva svoje stanje koje može biti zauzeto i slobodno, analogno kao kod mrežaste mape. Sve tačke zahvaćene površinom čvora zapravo pripadaju tom čvoru i nema potrebe da pojedinačno tačke budu izdvojene, jer prilikom modeliranja prostora to nije više potrebno.

Algoritam isto tako može ostati manje-više isti, s tim da *openSet* i *closedSet* moraju da rade sa čvorovima iz *QuadTree*-a, a *cameFrom*, *gScore* i *fScore* pridružuju čvorovima *QuadTree*-a odgovarajuću vrijednost. Kako više nisu u pitanju tačke, nego površine, potrebno je dodefinisati funkcije *g(n)* i *h(n)* tako da rade sa čvorovima. Najlakše je odabrati centar čvora kao referentnu tačku za računanje *g(n)* i *h(n)*. Politika oko računanja udaljenosti od početka i heuristike se mogu razlikovati od implementacije do implementacije. U ovom radu bit će predstavljena još jedna

metoda koja se oslanja na diskretizaciju ruba čvorova umjesto kretanja od centra do centra, ali o tome će biti govora u implementacionom dijelu ovog rada i rezultatima istog.

Na slici 2.13. uočljivo je da upravo ovo kretanje od centra do centra kod QuadTree-a značajno produžuje put između dva čvora, pogotovo ako su u pitanju veliki čvorovi. A* nad mrežastom mapom nema tih problema jer se može kretati vrlo precizno od tačke do tačke. Usljed toga, ukupna udaljenost za mrežu na slici 2.13. iznosi 938 dok za quadtree iznosi 965.



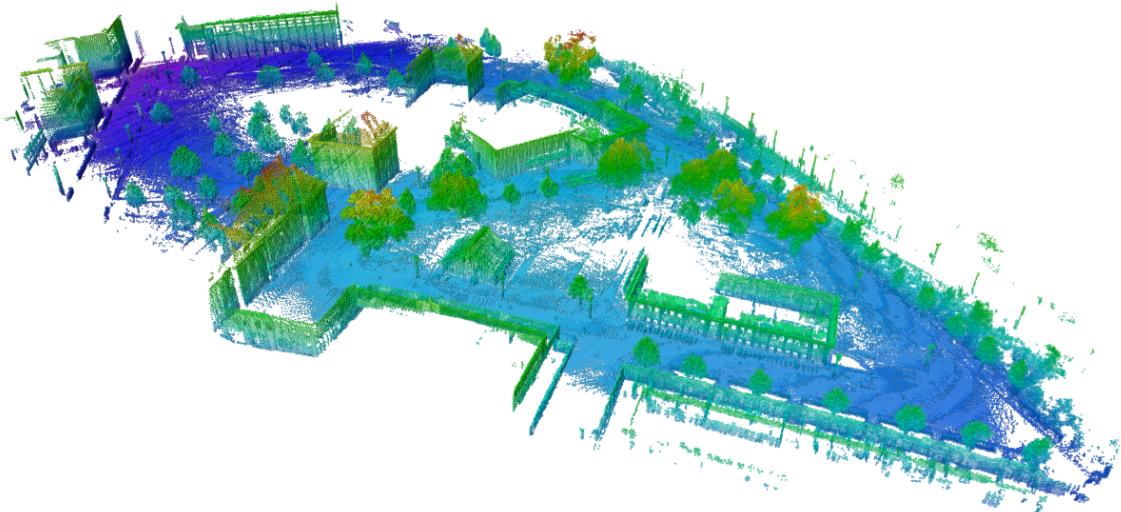
Slika 2.13. Usporedba A* algoritma na mrežastoj mapi i na Quadtree

Treba naglasiti da pored ovih razlika, postoje određeni problemi sa pronašlaskom komšija kod quadtree-a, što je već ranije bilo pomenuto. Da bi ispravno funkcionišao algoritam, potrebno je promijeniti kod koji traži komšije. Pošto se stabla pretražuju rekurzivno, potrebno je proširiti pseudokod stabla dodatnom funkcijom za pretragu komšija trenutnog čvora i njen rezultat vratiti algoritmu.

Još jedna izmjena koja se može napraviti jeste da se ukinu mape *cameFrom*, *fScore* i *gScore*, te da se te vrijednosti čuvaju u svakom čvoru kao atributi koji se ažuriraju po potrebi, ali o tim detaljima će biti više govora u implementaciji.

2.4.6. Octree

U uvodnom dijelu je već pomenuto da se filozofija koja se krije iza quadtree-a može proširiti za n-dimenzionalan prostor. Octree predstavlja posebnu strukturu podataka, takođe realizovanu kao stablo, koja je 3D analog quadtree-u. Kao što ime sugeriše, octree dijeli prostor u oktante, te može poslužiti za modeliranje 3D prostora. Jedan od najkorištenijih framework-ova za mapiranje 3D prostora baziranih na octree-ma jeste OctoMap [5]. Na slici 2.14. se vidi upravo jedna reprezentacija okoline bazirana na octree-ma. U pitanju je kampus u Freiburg-u modeliran koristeći rezoluciju od 0.2m.



Slika 2.14. Reprezentacija kampusa u Freibrug-u u OctoMap framework-u [5]

Implementacija octree-a je vrlo slična quadtree-u, s tim da će svaki čvor umjesto četiri imati osam pokazivača na osam čvorova. Umjesto površine koja definiše quadtree, potrebno je definisati određenu zapreminu, s tim da i za to može poslužiti ista klasa *AABB* kao i kod *QuadTree*-a, jer ona definiše centar i radijus svakog čvora. Ostala pozadinska logika ostaje ista.

2.4.7. Zaključak

Quadtree predstavlja učinkovitu strukturu podataka za smještanje dvodimenzionalnih podataka. Najjednostavniji algoritam umetanja ima kompleksnost $n \log(n)$ sa nasumično izabranim koordinatama. Pretraga regiona je takođe jako efikasna jer se svodi na analizu roditeljskog čvora i njegovih roditelja.

Nažalost brisanje iz stabla i stapanje dva ili više stabala predstavlja problem i nije jednostavno za implementaciju.

Kao što smo vidjeli u prethodnom poglavljju, koncept quadtree-ja može biti generaliziran na proizvoljan broj dimenzija, tako što za m-dimenzionalan prostor svakom roditeljskom čvoru pridružimo 2^m čvorova (djece). [14]

Poglavlje 3.

Implementacija

3.1. Uvod

Treće poglavlje obrađuje detalje implementacije mrežaste mape, quadtree-ja, quadtree-ja sa diskretiziranim granicama. U ovom poglavlju bit će predstavljena rješenja za probleme opisane u drugom poglavlju, zajedno sa implementacijom. Čitav kod je dostupan na linku: <https://github.com/dbojadzic/QuadTree>. U nastavku slijede mrežasta mapa, quadtree i quadtree sa diskretiziranim granicama. Četvrto poglavlje će obrađivati rezultate postignute ovim implementacijama.

3.2. Mrežasta mapa

Mrežasta mapa opisana je koristeći kontinualnu alokaciju. To je rađeno radi olakšanja komunikacije sa RoS-om (Robot Operating System). Klasa *AstarMatrix* u svom konstruktoru prima pokazivač na niz te širinu i visinu koje su neophodne za njeno funkcionisanje. Niz mora biti pripremljen i alociran jer klasa *AstarMatrix* o tome ne vodi računa. U main.cpp-u su pripremljene pomoćne funkcije *loadMatrix* i *generateMatrix* koje mogu da olakšaju u tom poslu.

Funkcija *loadMatrix* kao argument prima put i naziv fajla gdje je smještena mapa, a kao drugi argument prima pokazivač na alocirani niz. Nakon toga funkcija u niz piše vrijednosti iz fajla i time učitava mapu. Druga funkcija, *generateMatrix*, također prima pokazivač na početak niza, ali ona nasumice popunjava polja matrice dok ne bude postignuto zauzeće od 10%. Ovaj procenat može biti mijenjan manipulacijom atributa *PERCENTAGE*.

Pretraga je realizovana unutar metode *FindPath* koja prima kao svoje argumente dvije tačke, početak i cilj, a vraća vektor tačaka - put od početka do cilja. U ovoj metodi implementiran je A* algoritam na način kako je opisano u drugom poglavlju bez puno modifikacija. Za čuvanje tačaka se koriste closedSet i openSet koje su realizovane koristeći *std::set*, a *cameFrom*, *fScore* i *gScore* su realizirani koristeći *std::map*.

Za čuvanje vrijednosti koordinata x i y kreirana je posebna struktura nazvana *Point*. Unutar nje, pored atributa za x i y bilo je potrebno preklopiti *operator ()* i iskoristiti ga za kriterij sortiranja *std::set*-a, da se isti elementi ne bi više puta ponavljali. Takođe je preklopljen i *operator ==* za lakše poređenje tačaka.

3.3. Implementacija quadtree-a

3.3.1. Klase

Implementacija quadtree-a je bila dosta izazovnija od mrežaste mape. Kao preduslov funkcionisanja quadtree-a opet je potrebno kreirati strukturu *Point* koja čuva koordinate x i y . U ovoj implementaciji nije bilo potrebno preklopiti *operator ()* jer se pojedine tačke neće sortirati, ali jeste bilo potrebno preklopiti *operator ==* da bi se mogle poređiti.

Sam quadtree realizovan je dosta drugačije od pseudokoda. Unutar klase *QuadTree* ne nalaze se 4 pokazivača na objekat tipa *QuadTree* već samo jedan pokazivač na korijenski čvor tipa *Node* i 2 objekta tipa *Point* imenovani *topLeft* i *botRight*. Na ovaj način klasa *QuadTree* predstavlja jednu vrstu omotača oko stabla te pruža određenu vrstu pristupnog interface-a pravom stablu.

Postoji više razloza zašto se autor odlučio na ovakvu vrstu realizacije quadtree-a. Jedan od razloga je razdvajanje logike. Pošto je prilikom implementacije bilo potrebo da se specifični dijelovi koda testiraju, i jer je trebalo da se struktura čitavog stabla na neki način vizualizira, najlogičniji pristup je bilo napraviti klasu koja je omotač stablu i koja bi se brinula o njemu. Puno je logičnije da se svaki čvor brine o sebi i poziva metode nad djecom, a da jedino klasa *QuadTree* ima privilegije da poziva metode nad korijenskim čvorom.

Na ovaj način je takođe puno lakše upravljanje validacijom parametara koji se prosljeđuju metodama i upravljanje izuzecima. Pored toga je upravljanje memorijom olakšano jer klasa *QuadTree* preuzima svu odgovornost za alokaciju i dealokaciju pojedinih čvorova. Kroz globalni parametar *DEPTH* moguće je kontrolisati dubinu ovog stabla, o kojoj se takođe brine klasa *QuadTree*.

Najznačajnija prednost ove implementacije jeste njena fleksibilnost. Uz pamćenje centra i poluprečnika kao što je demonstrirano u pseudokodu moguće je pamtiti i rekonstruisati svaku tačku kvadrata. Isto je moguće ako se pamte gornja lijeva i donja desna tačka, ali pamćenje ove dvije tačke daje još jednu veliku prednost. Objekat koji se pamti ne mora biti samo kvadrat, već može biti i pravougaonik, a proširivo je i na 3D objekte kao što su kocka i kvadar uz dodavanje z koordinate klasi *Point* - analogno kao i u pseudokodu, s tim da nije moguće modelirati kvadar na način opisan pseudokodom. Pored navedenog, pamćenjem dvije tačke je puno jednostavnije praviti nebalansirani quadtree upravo iz razloga što je moguće modelirati pravougaonik.

Unutar pomenute klase *Node*, čuvaju se ponovo dvije tačke - gornja lijeva i donja desna da bi se mogle lakše odrediti granice datog čvora i čuva se pet pokazivača na objekte tipa *Node*. Četiri pokazivača na djecu i jedan pokazivač na roditelja.

Roditeljski pokazivač je tu da bi se mogla pozvati metoda za ujedinjenje djece nad roditeljem.

Ono što je različito od interface-a datog u prethodnom poglavlju jesu atribut tipa `bool` koji označava stanje čvora - zauzet ili slobodan, te integer za dubinu čvora. Pored ovih atributa implementacija čvora koristi jednu olakšicu za A*, a to je interno pohranjivanje vrijednosti `fScore`, `gScore` i `cameFrom`. Atributi `fScore` i `gScore` su po defaultu postavljeni na vrijednost $+\infty$, a `cameFrom` je postavljen na `nullptr`. Ovo značajno olakšava izvršenje A* algoritma jer eliminiše potrebu za kreiranjem istoimenih mapa, s tim da prilikom pokretanja svake nove pretrage, ovi atributi moraju biti restartovani.

Pored ovih klasa implementirana je i jedna pomoćna klasa `Comparator` koja unutar sebe samo ima preklopljen `operator ()` tako da prima dva objekta tipa node i poredi ih. Koliko god ova klasa djelovala naivno, od velikog je značaja za funkcionišanje algoritma te će njen značaj biti otkriven u kasnijim poglavljima.

3.3.2. Metode

QuadTree sadrži dva konstruktora, jedan koji prima samo dvije tačke i na osnovu njih kreira prazan univerzum, i drugi koji pored širine i dužine univerzuma prima pokazivač na niz. Ovaj konstruktor je analogan konstruktoru iz implementacije mrežaste mape.

Drugi konstruktor se oslanja na rad metode `InsertPoint` koja kao argument prima tačku sa koordinatama x i y . Ta tačka zapravo predstavlja jednu prepreku u mapi. Metoda nakon toga rekurzivno pretražuje stablo u onim čvorovima gdje se nalaze te koordinate. Ukoliko pretraga dođe do maksimalne dozvoljene dubine, a razmatrani čvor nije zauzet, bit će označen kao zauzet. Ako metoda dođe do čvora koji nema djecu, a nije na maksimalnoj dubini, on će biti podijeljen, i proces pretrage i dijeljenja će se rekurzivno nastaviti na njegovoj djeci.

Naredne metode korisniku nisu toliko značajne i koriste se u A* algoritmu. Metoda `Reset` svim čvorovima vraća početna stanja atributa `fScore`, `gScore` i `cameFrom`, da bi novi ciklus pretrage mogao biti započet. Metoda `FindAdjacentNoOccupied` vraća sve komšije čvora koji joj je prosljeden kao argument, s tim da ti čvorovi ne smiju biti zauzeti. Ova metoda se oslanja na činjenicu da svi komšijski čvorovi moraju dijeliti barem jednu koordinatu sa čvorom koji se posmatra. Na primjer ako se želi pronaći desni komšija posmatranog čvora, x koordinata njegove gornje lijeve tačke mora odgovarati x koordinati donje desne tačke posmatranog čvora. Ukoliko je to ispunjeno posmatraju se y koordinate svih tačaka da bi se ustanovalo da li je ovaj čvor iznad, ispod, unutar ili oko granica posmatranog čvora. Implementacija ove metode je dosta komplikovana i zasnovana je na top-down pristupu, tako što se prosljedenom čvoru odrede 4 tjemena i onda se rekurzivno od korijena posmatra da li neko od ta 4 tjemena padaju u površ koju djeca obuhvataju. Ovo se većinom odvija u klasi Node uz upotrebu pomoćnih metoda kao što je metoda `Overlap`. Autor se odlučio na ovaj pristup pretrage komšija jer je pregledan i jednostavan za implementaciju. Postoje i druga rješenja koja kreću iz posmatranog čvora umjesto iz korijenskog čvora, ali ta rješenja zahtjevaju dosta kompleksniju implementaciju.

Takođe prednost ove implementacije je što je lako proširiva na octree strukturu podataka.

Pored ove metode implementirano je mnoštvo pomoćnih metoda koje nisu pretjedno interesantne za opisivanje ali će biti pomenute. Metode *GetWidth* i *GetHeight* vraćaju širinu i dužinu respektivno, *FindNode* i *IsOccupied* za proslijedenu tačku vraćaju čvor i njegovo stanje respektivno dok *CountNodes* koristi post-order obilazak stabla da vrati ukupni broj čvorova. Ostale pomoćne metode su korištene za testiranje ili generisanje izvještaja u Matlab razvojnom okruženju.

3.3.3. Implementacija A* algoritma za QuadTree

A* algoritam implementiran je u klasi *Astar*. Klasa *Astar* interno čuva pokazivač na *QuadTree* koji se pretražuje i poziva njegove metode. Autor se odlučio na ovaj način implementacije ponovo radi razdvajanja logike i odgovornosti klasa. Pored broja pomoćnih metoda, klasa *Astar* posjeduje metode *FindPath* i *FindDistance*, koje primaju početnu i krajnju tačku - analogno kao kod implementacije mrežaste mape.

Iako se suštinski implementacija A* algoritma ne razlikuje mnogo od pseudokoda, može se naglasiti da su izbačene mape *cameFrom*, *gScore* i *fScore* te su prebačene na odgovornost pojedinih čvorova što značajno ubrzava algoritam i smanjuje njegovu memorijsku kompleksnost. Iako se inače *openSet* implementira kao prioritetski red, autor ovog rada se odlučio da ga implementira kao skup sa pravilom poređenja.

Ova implementacija ima brojne prednosti. Jedna od prednosti jeste da obični skup ne može sadržavati više istih objekata u sebi. Kako *openSet* i *closedSet* u ovom slučaju rade sa pokazivačima na čvorove, oni neće moći čuvati iste pokazivače te samim time neće moći čuvati isti čvor. To je sasvim dovoljno za *closedSet*, dok je prilikom kreiranja *openSet*-a, kao argument za sortiranje proslijedena klasa *Comparator*. Klasa *Comparator* poredi čvorove na osnovu njihovog *fScore*-a te će u rastućem redu biti postavljeni u ovom setu.

Pored navedenog, *Comparator* razdvaja logiku sortiranja elemenata od samog algoritma, te u slučaju da dođe do promjene heuristike, ili korisnik naprsto ima želju da dodatno definiše šta se dešava u slučaju da dva čvora imaju isti *fScore*, može to vrlo jednostavno da uradi u ovoj klasi. Još neke značajne karakteristike klase *Comparator* bit će obrađene u narednom dijelu.

Skupovi su između ostalog korišteni jer su transparentni. Skupu se može pristupiti svakom elementu dok se redovima može pristupiti samo prednjem elementu. To značajno očekšava provjeru da li se neki od komšijskih čvorova nalazi u *closedSet*-u.

Kad je riječ o komšijskim čvorovima, kod koji je pretraživao komšije je razdvojen od algoritma korištenjem metode *FindAdjacentNoOccupied*, za razliku od mrežaste mape gdje je pretraga komšija implementirana u algoritmu, najviše zbog njene jednostavnosti.

3.4. Quadtree sa diskretiziranim granicama

3.4.1. Proširenje

Do sada je bilo samo govora o kretanju od centra do centra čvora. Međutim da bi uopšte robot došao do centra čvora on mora prije toga preći preko njegove granice. Prebacivanjem težišta sa centra čvora na njegove granice, odnosno tačno određene tačke na njegovoj granici, je poenta ovog poglavlja.

Prije prelaska na sami algoritam potrebno je uvesti nove termine. Kako se prelazak iz čvora u čvor odvija preko njegove granice, ta dva čvora imaju jednu zajedničku tačku. Za prvi čvor to je izlazna tačka, a za drugi čvor je to ulazna tačka. Zbog toga bi imalo smisla proširiti klasu Node sa dodatnim atributom koji predstavlja tačku ulaza u čvor, odnosno kako će u nastavku teksta biti pominjana - *accessPoint*. Kako su ulazna i izlazna tačka zapravo jedna te ista tačka, nema potrebe da se obje pamte.

Čvor će, osim sa ulaznom tačkom, biti proširen i jednom tačkom nazvanom *bestPoint*. Ova tačka neće nužno biti tačka kroz koju će se kretati algoritam, ali igra značaju ulogu u navigiranju algoritma jer će se pomoću nje za svaki komšijski čvor određivati ulazna tačka. Značaj tačke *bestPoint* bit će jasniji u idućem poglavlju.

Pored ove dvije tačke klase je proširena sa tačkama *possibleAccessPoint* i *possibleBestPoint*, koje se koriste kao pomoćne variabile iz razloga što je moguće na više načina doći u jedan čvor. Ukoliko se pronađe bolji put tek se onda ažuriraju vrijednosti *accessPoint* i *bestPoint*. U suštini kad god se ažurira *cameFrom* ažuriraju se i ove dvije tačke.

Osim ova četiri atributa tipa *Point*, konstruktor klase je proširen još jednim argumentom koji određuje vrijednost atributa *borderPoint*. Atribut *borderPoint* govori koliko se tačaka nalazi na rubu između dva tjemena. Postojanje ovog atributa je bilo neophodno za implementaciju dvije nove metode - *GetBorderPoints* i *GetBorder*.

Metoda *GetBorderPoints*, koristeći tačke *topLeft* i *botRight*, diskretizira granicu čvora koji posmatramo i vraća sve tačke na njegovoj granici, uključujući i tačke *topLeft* i *botRight*. Metoda ima jednostavnu implementaciju. Prvo se kalkulišu sva tjemena posmatranog čvora, a nakon toga se interpolira granica između njih na onoliko tačaka koliko je određeno atributom *borderPoints*.

Metoda *GetBorder* je nešto komplikovnija u smislu da je njen zadatak da diskretizira dodirnu liniju između dva čvora. Ova metoda funkcioniše jako slično metodi koja pronalazi komšije datog čvora. Međutim olakšavajuća je okolnost činjenica da je u pitanju quadtree tako da mogu nastupiti samo tri moguće situacije - Dva čvora ili nemaju dodirnih tačaka ili imaju samo jednu dodirnu tačku (tjeme) ili se diraju čitavom linijom manjeg čvora. Kako je poznato da će funkcija biti pozivana samo nad čvorovima za koje znamo sigurno da su komšije, prvi uslov se nikada neće dogoditi.

3.4.2. Implementacija A* algoritma za diskretizirani Quad-Tree

Iako suština algoritma ostaje ista, funkcija kriterija po kom se bira najbolji čvor je značajno promijenjena.

Uvodni dio algoritma ostaje isti, s tim da se početnom čvoru kao *accessPoint* dodijeljuje tačka od koje algoritam počinje. Ovo je jedini *accessPoint* koji se ne nalazi na rubu između dva čvora.

Nakon što se identifikuju komšije posmatranog čvora, za svakog komšiju se određuju sve njegove rubne tačke i određuju se one tačke koje se nalaze na granici između trenutnog i komšijskog čvora. Ovo se radi pozivajući funkcije *GetBorderPoints* i *GetBorder* respektivno.

Nakon toga se od svih tačaka na rubovima komšijskog čvora traži *possibleBestPoint*. Uzima se pojedinačno svaka od tačaka ruba komšijskog čvora i njena euklidska udaljenost od *accessPointa* trenutnog čvora se sabira sa njenom euklidskom udaljenošću od cilja. Ona tačka koja ima najmanju vrijednost proglašava se za *possibleBestPoint*.

Zatim se traži *possibleAccessPoint* komšijskog čvora. Za svaku tačku na granici između trenutnog čvora i komšijskog čvora se sabira udaljenost do *accessPoint-a* trenutnog čvora i *possibleBestPoint-a* komšijskog čvora. Tačka koja ima najmanju vrijednost proglašava se za *possibleAccessPoint*. Kao što je rečeno u uvodu, *bestPoint* zapravo služi samo da se odredi *accessPoint* čvoru.

Međutim, potrebno je sada odrediti novi mogući *gScore*. Pošto se *gScore* do sada računao kao udaljenost između dva centra, sada će se računati kao udaljenost između dva *accessPointa*. Ukoliko smo pronašli bolji *gScore*, ažuriraju se *cameFrom*, *accessPoint*, *bestPoint*, *gScore* i *fScore*. Nova heuristika se računa kao vrijednost od *accessPoint-a* do cilja.

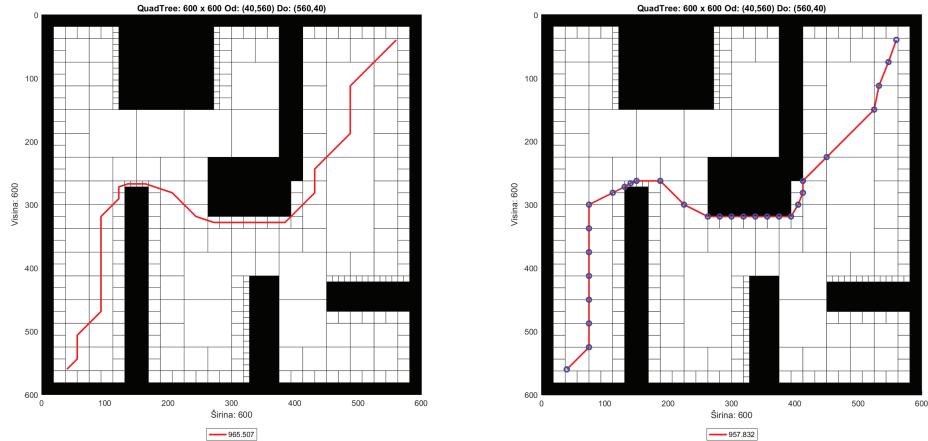
3.4.3. Varijacije

U prethodnom dijelu obrazložena je implementacija A* algoritma i novi način računanja *gScore* i *fScore* vrijednosti. Međutim ovo nije jedini način kako se ove vrijednosti mogu računati. Jedan od alternativnih načina jeste da se računa heuristika od centra do cilja ili čak od *bestPoint-a* do cilja. Time bi čvorovi koji su bliže cilju trebali da imaju prioritet prilikom izbora u odnosu na druge čvorove. Prilikom testiranja, na nekim primjerima su ove varijacije dale bolje rezultate, dok su na drugim davale lošije rezultate tako da se autor rada ipak odlučio za najjednostavniju implementaciju koja zadovoljava potrebe ovog rada.

Još jedna od mogućih varijacija jeste da se $g(n)$ ili $h(n)$ množe sa nekim koeficijentom k , te se na taj način daje prioritet čvorovima koji su bliže startu ili cilju. Ovaj princip je bio opisan kad je bilo govora o A* algoritmu. Autor je isprobao implementaciju algoritma tako što je otežao heurstiku sa 20% i dobijeni rezultati su u većini slučajeva bili bolji, ali je ipak za potrebe ovog rada zadržan normalan odnos.

Diskretizacijom granica čvorova algoritmu se daje na fleksibilnosti jer se u jednu

ruku povećava rezolucija mape dok se ipak zadržava jednostavnost quadtree strukture podataka. Modifikacije potrebne da se A* algoritam prilagodi novokreiranoj strukturi podataka su neznatne. Na slici 3.1. vidljiva je razlika između običnog quadtree-a i quadtree-a sa diskretiziranim granicama.



Slika 3.1. Usporedba: obični Quadtree i Quadtree sa diskretiziranim granicama

Na desnoj ilustraciji, plavim krugovima označene su *accessPoint* tačke svakog čvora. Diskretiziranjem granica se dužina puta smanjila sa 967 na 958, međutim značaj diskretizacije će biti puno uočljiviji u narednim poglavljima.

3.5. Zaključak

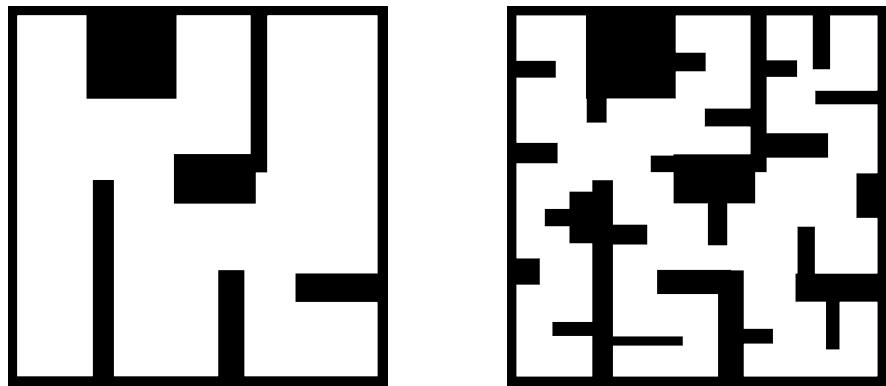
Ova implementacija nije nužno optimalna te je kod moguće poboljšati. U radu je implementiran quadtree i algoritam pretrage na ovaj način jer ne odstupa puno od pseudokoda i od obične implementacije, a dovoljno je jednostavan da se može razumjeti i modifikovati. U narednom poglavljju bit će obrađeni rezultati postignuti ovom implementacijom.

Poglavlje 4.

Rezultati

4.1. Uvod

Ovo poglavlje obrađivat će rezultate dobijene implementacijom opisanom u prethodnom poglavlju. Za testiranje su korištene dvije mape koje se mogu vidjeti na slici 4.1. Sa mapom A je već bilo rađeno u prethodnim poglavljima. Mape su skalirane na veličine 100x100, 200x200, 400x400, 500x500, 600x600, 1000x1000, 1500x1500, 3000x3000 te je na svakoj od veličina vršeno testiranje.



Slika 4.1. Mapa A(lijevo) i B(desno); Mapa A predstavlja lakši scenario, a mapa B teži scenario

Ovo poglavlje podijeljeno je u više dijelova. Prvi dio se bavi vremenom kreiranja QuadTree-a, a naredna tri dijela se bave pretragom različitih struktura podataka. QuadTree je kreiran sa dubinama 4, 5, 6 i 7 radi boljeg uvida u ponašanje stabla sa

različitom rezolucijom. Pretraga se vrši preko tačaka koje su za 1/15 širine ili visine udaljene od ruba mape. Na ovaj način su održane proporcije pretrage.

Sva vremena su izražena u sekundama, dok su sve simulacije i rezultati dobiveni na računaru sa konfiguracijom: Intel Core i7-6700HQ; NVIDIA GeForce GTX 690M; 8GB DDR4 Memory; 128GB SSD + 1TB HDD.

4.2. Kreiranje QuadTree-a

U narednim tabelama data su vremena kreiranja QuadTree-a. U tabelama su po kolonama redom zapisane dubina, mapa, širina, visina, deset vremena za deset testova, prosječno vrijeme kreiranja *QuadTree-a* i broj čvorova koji su kreirani.

Dubina	Mapa	Širina	Visina	Vrijeme(s)	Broj čvorova
4	A	100	100	0.0001992	269
4	A	200	200	0.0007976	257
4	A	400	400	0.0031917	257
4	A	500	500	0.0040889	257
4	A	600	600	0.0056847	257
4	A	1000	1000	0.0205451	257
4	A	1500	1500	0.0480712	257
4	A	3000	3000	0.19069	257
4	B	100	100	0.0002992	281
4	B	200	200	0.0009966	253
4	B	400	400	0.0035908	241
4	B	500	500	0.0053843	241
4	B	600	600	0.0072806	241
4	B	1000	1000	0.0224396	241
4	B	1500	1500	0.0594369	241
4	B	3000	3000	0.250929	237

Tabela 4.1. Vremena kreiranja QuadTree-a dubine 4

Dubina	Mapa	Širina	Visina	Vrijeme(s)	Broj čvorova
5	A	100	100	0.0003985	601
5	A	200	200	0.0008978	613
5	A	400	400	0.0029909	613
5	A	500	500	0.0044923	613
5	A	600	600	0.0064859	613
5	A	1000	1000	0.0187536	613
5	A	1500	1500	0.042686	613
5	A	3000	3000	0.201561	613
5	B	100	100	0.0004985	749
5	B	200	200	0.0010966	773
5	B	400	400	0.0040895	793
5	B	500	500	0.0062827	797
5	B	600	600	0.0091709	797
5	B	1000	1000	0.0282261	797
5	B	1500	1500	0.0725105	793
5	B	3000	3000	0.262598	797

Tabela 4.2. Vremena kreiranja QuadTree-a dubine 5

Dubina	Mapa	Širina	Visina	Vrijeme(s)	Broj čvorova
6	A	100	100	0.0005985	1137
6	A	200	200	0.0011968	801
6	A	400	400	0.0032911	773
6	A	500	500	0.0046875	773
6	A	600	600	0.0071828	773
6	A	1000	1000	0.0187506	773
6	A	1500	1500	0.0444776	773
6	A	3000	3000	0.221506	777
6	B	100	100	0.0007039	1737
6	B	200	200	0.0017955	1449
6	B	400	400	0.005086	1485
6	B	500	500	0.0089713	1497
6	B	600	600	0.0127649	1453
6	B	1000	1000	0.0306182	1569
6	B	1500	1500	0.0863708	1553
6	B	3000	3000	0.352158	1569

Tabela 4.3. Vremena kreiranja QuadTree-a dubine 6

Dubina	Mapa	Širina	Visina	Vrijeme(s)	Broj čvorova
7	A	100	100	0.0009996	5357
7	A	200	200	0.0024936	2337
7	A	400	400	0.0064867	1893
7	A	500	500	0.0077827	2017
7	A	600	600	0.0103644	1893
7	A	1000	1000	0.0248368	1269
7	A	1500	1500	0.0644312	1297
7	A	3000	3000	0.273868	1281
7	B	100	100	0.0011951	7913
7	B	200	200	0.0029967	3637
7	B	400	400	0.0079837	3253
7	B	500	500	0.0103723	2745
7	B	600	600	0.0130651	2681
7	B	1000	1000	0.0377083	2637
7	B	1500	1500	0.071409	2533
7	B	3000	3000	0.337896	2565

Tabela 4.4. Vremena kreiranja QuadTree-a dubine 7

Iz ovih tabela proizilazi interesantna činjenica da iako veličina mape raste, vrijeme kreiranja mape i broj čvorova ostaje konstantan. Jedina stvar koja utiče na vrijeme kreiranja mape i broj čvorova jeste dubina mape i struktura same mape. Jedini test koji odstupa od ovog pravila jeste mapa veličine 100x100 zbog činjenice da dolazi do brzih promjena u redovima i kolonama po pitanju zauzeća, te se mali broj čvorova uspije ujediniti u veliki čvor. Međutim to nije jedini razlog zašto se pojavljuju ovakve anomalije u rezultatima. Kako je stablo visoke rezolucije, a mapa sitna, desi se da se barijere unutar mape ne preslikaju dobro, te se formiraju rupe u zidovima. O ovome je bilo riječi prilikom opisa *QuadTree-a* kao strukture podataka. U nastavku, u poglavljju vizuelizacije rezultata će biti prikazano šta se dešava.

4.3. Rezultati traženja putanje korištenjem A* algoritam na mrežastoj mapi

Narednih šest tabele prikazuju pretragu matrične mreže. Tabele prikazuju mapu koja se pretražuje, njenu širinu i visinu, početnu i krajnju tačku, deset vremena za deset testova, prosječno vrijeme i distancu puta između tačaka A i B. Tačke su birane tako da budu za 1/15 udaljene od rubova mape i pretraživanje je vršeno iz gornjeg lijevog u gornji desni ugao, iz donjeg lijevog u gornji desni ugao, iz donjeg lijevog u donji desni ugao i iz gornjeg lijevog u donji desni ugao.

Mapa	Širina	Visina	Početak	Cilj	Vrijeme(s)	Udaljenost
A	100	100	(6 6)	(93 6)	0.239606	139.752
A	100	100	(6 93)	(93 6)	0.284144	156.953
A	100	100	(6 93)	(93 93)	0.130597	144.924
A	100	100	(6 6)	(93 93)	0.135246	127.723
B	100	100	(6 6)	(93 6)	0.180992	157.451
B	100	100	(6 93)	(93 6)	0.247036	176.409
B	100	100	(6 93)	(93 93)	0.0892688	147.853
B	100	100	(6 6)	(93 93)	0.10392	128.894

Tabela 4.5. Vremena pretrage - Matrica 100x100

Mapa	Širina	Visina	Početak	Cilj	Vrijeme(s)	Udaljenost
A	200	200	(13 13)	(186 13)	1.63239	277.676
A	200	200	(13 186)	(186 13)	2.4293	312.25
A	200	200	(13 186)	(186 186)	1.1282	288.605
A	200	200	(13 13)	(186 186)	1.29034	254.032
B	200	200	(13 13)	(186 13)	1.63337	313.073
B	200	200	(13 186)	(186 13)	2.2112	351.161
B	200	200	(13 186)	(186 186)	0.799822	294.463
B	200	200	(13 13)	(186 186)	0.922114	256.375

Tabela 4.6. Vremena pretrage - Matrica 200x200

Mapa	Širina	Visina	Početak	Cilj	Vrijeme(s)	Udaljenost
A	400	400	(26 26)	(373 26)	16.012	556.938
A	400	400	(26 373)	(373 26)	22.7182	625.671
A	400	400	(26 373)	(373 373)	8.82917	578.21
A	400	400	(26 26)	(373 373)	10.2294	509.477
B	400	400	(26 26)	(373 26)	13.9778	627.489
B	400	400	(26 373)	(373 26)	18.9704	703.252
B	400	400	(26 373)	(373 373)	6.63725	589.926
B	400	400	(26 26)	(373 373)	7.3294	514.164

Tabela 4.7. Vremena pretrage - Matrica 400x400

Mapa	Širina	Visina	Početak	Cilj	Vrijeme(s)	Udaljenost
A	500	500	(33 33)	(466 33)	26.7532	694.862
A	500	500	(33 466)	(466 33)	38.2378	780.382
A	500	500	(33 466)	(466 466)	16.7347	721.306
A	500	500	(33 33)	(466 466)	19.2359	635.786
B	500	500	(33 33)	(466 33)	23.8005	782.283
B	500	500	(33 466)	(466 33)	32.3423	878.004
B	500	500	(33 466)	(466 466)	11.6241	737.365
B	500	500	(33 33)	(466 466)	14.9338	641.644

Tabela 4.8. Vremena pretrage - Matrica 500x500

Mapa	Širina	Visina	Početak	Cilj	Vrijeme(s)	Udaljenost
A	600	600	(40 40)	(560 40)	45.7463	833.2
A	600	600	(40 560)	(560 40)	67.2347	937.921
A	600	600	(40 560)	(560 560)	30.0166	868.23
A	600	600	(40 40)	(560 560)	34.4431	763.509
B	600	600	(40 40)	(560 40)	42.4944	939.492
B	600	600	(40 560)	(560 40)	58.4391	1054.76
B	600	600	(40 560)	(560 560)	20.7406	886.389
B	600	600	(40 40)	(560 560)	26.0118	771.124

Tabela 4.9. Vremena pretrage - Matrica 600x600

Mapa	Širina	Visina	Početak	Cilj	Vrijeme(s)	Udaljenost
A	1000	1000	(66 66)	(933 66)	237.811	1391.9
A	1000	1000	(66 933)	(933 66)	347.598	1563.94
A	1000	1000	(66 933)	(933 933)	152.348	1445.61
A	1000	1000	(66 66)	(933 933)	176.574	1273.57
B	1000	1000	(66 66)	(933 66)	226.033	1566.15
B	1000	1000	(66 933)	(933 66)	306.716	1755.77
B	1000	1000	(66 933)	(933 933)	110.094	1474.9
B	1000	1000	(66 66)	(933 933)	139.192	1285.29

Tabela 4.10. Vremena pretrage - Matrica 1000x1000

Iz priloženih tabela se vidi da vrijeme pretrage raste sa veličinom mape. Na mapama veličine 1000x1000 vrijeme pretrage može čak trajati i po 5 minuta. Zbog ovako dugih vremena testiranja, testovi su obustavljeni nakon veličine 1000x1000 te 1500x1500 nije testirano.

4.4. Rezultati traženja putanje korištenjem A* algoritma na mapi dekomponiranoj u QuadTree-eve

QuadTree je pretražen analogno kao i mrežasta mapa, s tim su se testovi dodatno pokretali za dubine 4, 5, 6 i 7. Tabele prikazuju mapu koja se pretražuje, njenu širinu i visinu, početnu i krajnju tačku, deset vremena za deset testova, prosječno vrijeme i distancu puta između tačaka A i B i broj posjećenih čvorova (listova).

Dubina	Mapa	Širina	Visina	Početak	Cilj	Vrijeme(s)	Udaljenost	Broj posjećenih čvorova
4	A	100	100	(6 6)	(93 6)	0	0	-1
4	A	100	100	(6 93)	(93 6)	0	0	-1
4	A	100	100	(6 93)	(93 93)	0	0	-1
4	A	100	100	(6 6)	(93 93)	0	0	-1
4	A	200	200	(13 13)	(186 13)	0.0004989	303.292	116
4	A	200	200	(13 186)	(186 13)	0.0006011	349.015	117
4	A	200	200	(13 186)	(186 186)	0.0004979	314.948	100
4	A	200	200	(13 13)	(186 186)	0.0004963	269.224	97
4	A	400	400	(26 26)	(373 26)	0.0005983	607.24	116
4	A	400	400	(26 373)	(373 26)	0.0005991	699.343	117
4	A	400	400	(26 373)	(373 373)	0.0004986	631.966	100
4	A	400	400	(26 26)	(373 373)	0.0004987	539.863	97
4	A	500	500	(33 33)	(466 33)	0.0005986	757.806	116
4	A	500	500	(33 466)	(466 33)	0.0004989	873.104	117
4	A	500	500	(33 466)	(466 466)	0.0004987	789.067	100
4	A	500	500	(33 33)	(466 466)	0.0004986	673.768	97
4	A	600	600	(40 40)	(560 40)	0.0006981	909.068	116
4	A	600	600	(40 560)	(560 40)	0.0007977	1048.26	117
4	A	600	600	(40 560)	(560 560)	0.0010971	948.278	100
4	A	600	600	(40 40)	(560 560)	0.0006972	809.088	98
4	A	1000	1000	(66 66)	(933 66)	0.0005984	1516.3	116
4	A	1000	1000	(66 933)	(933 66)	0.0013963	1747.58	117
4	A	1000	1000	(66 933)	(933 933)	0.0005392	1580.23	100
4	A	1000	1000	(66 66)	(933 933)	0.0005986	1348.95	97
4	A	1500	1500	(100 100)	(1400 100)	0.0003988	2272.67	116
4	A	1500	1500	(100 1400)	(1400 100)	0.0007989	2620.65	117
4	A	1500	1500	(100 1400)	(1400 1400)	0.0005926	2370.7	100
4	A	1500	1500	(100 100)	(1400 1400)	0.0005984	2022.72	98
4	A	3000	3000	(200 200)	(2800 200)	0.0006976	4545.34	116
4	A	3000	3000	(200 2800)	(2800 200)	0.0007979	5241.3	117
4	A	3000	3000	(200 2800)	(2800 2800)	0.0004987	4741.39	100
4	A	3000	3000	(200 200)	(2800 2800)	0.0008973	4045.44	98

Tabela 4.11. Vremena pretrage - Mapa A - QuadTree dubine 4

Dubina	Mapa	Širina	Visina	Početak	Cilj	Vrijeme(s)	Udaljenost	Broj posjećenih čvorova
4	B	100	100	(6 6)	(93 6)	0	0	-1
4	B	100	100	(6 93)	(93 6)	0	0	-1
4	B	100	100	(6 93)	(93 93)	0	0	-1
4	B	100	100	(6 6)	(93 93)	0	0	-1
4	B	200	200	(13 13)	(186 13)	0	0	-1
4	B	200	200	(13 186)	(186 13)	0	0	-1
4	B	200	200	(13 186)	(186 186)	0	0	-1
4	B	200	200	(13 13)	(186 186)	0	0	-1
4	B	400	400	(26 26)	(373 26)	0	0	-1
4	B	400	400	(26 373)	(373 26)	0	0	-1
4	B	400	400	(26 373)	(373 373)	0	0	-1
4	B	400	400	(26 26)	(373 373)	0	0	-1
4	B	500	500	(33 33)	(466 33)	0	0	-1
4	B	500	500	(33 466)	(466 33)	0	0	-1
4	B	500	500	(33 466)	(466 466)	0	0	-1
4	B	500	500	(33 33)	(466 466)	9.98E-05	0	-1
4	B	600	600	(40 40)	(560 40)	9.98E-05	0	-1
4	B	600	600	(40 560)	(560 40)	0	0	-1
4	B	600	600	(40 560)	(560 560)	0	0	-1
4	B	600	600	(40 40)	(560 560)	0	0	-1
4	B	1000	1000	(66 66)	(933 66)	0	0	-1
4	B	1000	1000	(66 933)	(933 66)	0	0	-1
4	B	1000	1000	(66 933)	(933 933)	9.98E-05	0	-1
4	B	1000	1000	(66 66)	(933 933)	0.0001	0	-1
4	B	1500	1500	(100 100)	(1400 100)	9.97E-05	0	-1
4	B	1500	1500	(100 1400)	(1400 100)	0	0	-1
4	B	1500	1500	(100 1400)	(1400 1400)	0	0	-1
4	B	1500	1500	(100 100)	(1400 1400)	0	0	-1
4	B	3000	3000	(200 200)	(2800 200)	0.000101	0	-1
4	B	3000	3000	(200 2800)	(2800 200)	0	0	-1
4	B	3000	3000	(200 2800)	(2800 2800)	0	0	-1
4	B	3000	3000	(200 200)	(2800 2800)	0	0	-1

Tabela 4.12. Vremena pretrage - Mapa B - QuadTree dubine 4

Dubina	Mapa	Širina	Visina	Početak	Cilj	Vrijeme(s)	Udaljenost	Broj posjećenih čvorova
5	A	100	100	(6 6)	(93 6)	0.0011969	150.883	224
5	A	100	100	(6 93)	(93 6)	0.0018942	170.25	254
5	A	100	100	(6 93)	(93 93)	0.0013963	158.005	225
5	A	100	100	(6 6)	(93 93)	0.0013344	138.638	275
5	A	200	200	(13 13)	(186 13)	0.0008976	287.416	206
5	A	200	200	(13 186)	(186 13)	0.0010972	332.352	224
5	A	200	200	(13 186)	(186 186)	0.0011967	313.373	238
5	A	200	200	(13 13)	(186 186)	0.0021942	269.224	191
5	A	400	400	(26 26)	(373 26)	0.0010368	575.488	206
5	A	400	400	(26 373)	(373 26)	0.0009973	666.016	224
5	A	400	400	(26 373)	(373 373)	0.0011964	628.817	238
5	A	400	400	(26 26)	(373 373)	0.0009971	539.863	191
5	A	500	500	(33 33)	(466 33)	0.0012965	718.115	206
5	A	500	500	(33 466)	(466 33)	0.0015957	831.446	223
5	A	500	500	(33 466)	(466 466)	0.000997	785.13	238
5	A	500	500	(33 33)	(466 466)	0.0011967	673.768	191
5	A	600	600	(40 40)	(560 40)	0.0012966	861.44	206
5	A	600	600	(40 560)	(560 40)	0.0014959	998.269	223
5	A	600	600	(40 560)	(560 560)	0.0024932	943.555	238
5	A	600	600	(40 40)	(560 560)	0.001097	809.088	191
5	A	1000	1000	(66 66)	(933 66)	0.0011967	1436.92	206
5	A	1000	1000	(66 933)	(933 66)	0.0014312	1664.27	223
5	A	1000	1000	(66 933)	(933 933)	0.0014955	1572.36	238
5	A	1000	1000	(66 66)	(933 933)	0.0008976	1348.95	191
5	A	1500	1500	(100 100)	(1400 100)	0.0009973	2153.6	207
5	A	1500	1500	(100 1400)	(1400 100)	0.0012959	2495.67	223
5	A	1500	1500	(100 1400)	(1400 1400)	0.0014958	2358.89	238
5	A	1500	1500	(100 100)	(1400 1400)	0.0007977	2022.72	191
5	A	3000	3000	(200 200)	(2800 200)	0.0010972	4307.2	205
5	A	3000	3000	(200 2800)	(2800 200)	0.0012964	4991.34	222
5	A	3000	3000	(200 2800)	(2800 2800)	0.0013963	4717.77	238
5	A	3000	3000	(200 200)	(2800 2800)	0.001097	4045.44	191

Tabela 4.13. Vremena pretrage - Mapa A - QuadTree dubine 5

Dubina	Mapa	Širina	Visina	Početak	Cilj	Vrijeme(s)	Udaljenost	Broj posjećenih čvorova
5	B	100	100	(6 6)	(93 6)	0.0018949	173.596	262
5	B	100	100	(6 93)	(93 6)	0.001795	198.254	272
5	B	100	100	(6 93)	(93 93)	0.0011968	164.591	239
5	B	100	100	(6 6)	(93 93)	0.0013965	139.932	197
5	B	200	200	(13 13)	(186 13)	0.0027925	358.412	291
5	B	200	200	(13 186)	(186 13)	0.0035904	399.628	276
5	B	200	200	(13 186)	(186 186)	0.0017946	326.917	262
5	B	200	200	(13 13)	(186 186)	0.0017951	286.487	299
5	B	400	400	(26 26)	(373 26)	0.0029919	717.479	295
5	B	400	400	(26 373)	(373 26)	0.0028922	800.569	280
5	B	400	400	(26 373)	(373 373)	0.0014959	655.904	280
5	B	400	400	(26 26)	(373 373)	0.0015957	574.389	301
5	B	500	500	(33 33)	(466 33)	0.0026934	895.605	286
5	B	500	500	(33 466)	(466 33)	0.0027925	999.637	281
5	B	500	500	(33 466)	(466 466)	0.0019946	818.989	266
5	B	500	500	(33 33)	(466 466)	0.0020904	716.925	288
5	B	600	600	(40 40)	(560 40)	0.0032912	1074.43	286
5	B	600	600	(40 560)	(560 40)	0.0026928	1200.1	281
5	B	600	600	(40 560)	(560 560)	0.0017951	984.185	266
5	B	600	600	(40 40)	(560 560)	0.0020942	860.876	288
5	B	1000	1000	(66 66)	(933 66)	0.0021942	1881.71	279
5	B	1000	1000	(66 933)	(933 66)	0.002593	2090.46	279
5	B	1000	1000	(66 933)	(933 933)	0.0012962	1640.08	266
5	B	1000	1000	(66 66)	(933 933)	0.0016959	1435.26	287
5	B	1500	1500	(100 100)	(1400 100)	0.0027929	2820.79	278
5	B	1500	1500	(100 1400)	(1400 100)	0.0026017	3134.97	278
5	B	1500	1500	(100 1400)	(1400 1400)	0.0014419	2460.46	266
5	B	1500	1500	(100 100)	(1400 1400)	0.0019945	2152.19	287
5	B	3000	3000	(200 200)	(2800 200)	0.0020942	5641.58	276
5	B	3000	3000	(200 2800)	(2800 200)	0.0019941	6269.93	277
5	B	3000	3000	(200 2800)	(2800 2800)	0.0016959	4920.93	280
5	B	3000	3000	(200 200)	(2800 2800)	0.0021947	4292.57	310

Tabela 4.14. Vremena pretrage - Mapa B - QuadTree dubine 5

Dubina	Mapa	Širina	Visina	Početak	Cilj	Vrijeme(s)	Udaljenost	Broj posjećenih čvorova
6	A	100	100	(6 6)	(93 6)	0.0020904	147.905	353
6	A	100	100	(6 93)	(93 6)	0.0030957	161.812	435
6	A	100	100	(6 93)	(93 93)	0.002398	152.545	341
6	A	100	100	(6 6)	(93 93)	0.0023024	138.638	368
6	A	200	200	(13 13)	(186 13)	0.0016234	284.728	234
6	A	200	200	(13 186)	(186 13)	0.0017909	318.744	288
6	A	200	200	(13 186)	(186 186)	0.0015957	302.453	309
6	A	200	200	(13 13)	(186 186)	0.0012967	268.437	289
6	A	400	400	(26 26)	(373 26)	0.001097	575.488	228
6	A	400	400	(26 373)	(373 26)	0.0014959	644.175	271
6	A	400	400	(26 373)	(373 373)	0.0011968	608.55	303
6	A	400	400	(26 26)	(373 373)	0.0016955	539.863	360
6	A	500	500	(33 33)	(466 33)	0.0010978	718.115	228
6	A	500	500	(33 466)	(466 33)	0.0014957	804.144	271
6	A	500	500	(33 466)	(466 466)	0.0012956	759.797	303
6	A	500	500	(33 33)	(466 466)	0.0016955	673.768	360
6	A	600	600	(40 40)	(560 40)	0.001496	861.44	228
6	A	600	600	(40 560)	(560 40)	0.0014941	965.507	271
6	A	600	600	(40 560)	(560 560)	0.0012965	913.155	303
6	A	600	600	(40 40)	(560 560)	0.0017951	809.088	360
6	A	1000	1000	(66 66)	(933 66)	0.0010966	1436.92	228
6	A	1000	1000	(66 933)	(933 66)	0.0016956	1609.66	271
6	A	1000	1000	(66 933)	(933 933)	0.0012958	1521.7	303
6	A	1000	1000	(66 66)	(933 933)	0.0016956	1348.95	360
6	A	1500	1500	(100 100)	(1400 100)	0.0012539	2153.6	229
6	A	1500	1500	(100 1400)	(1400 100)	0.001496	2413.77	271
6	A	1500	1500	(100 1400)	(1400 1400)	0.0012965	2282.89	303
6	A	1500	1500	(100 100)	(1400 1400)	0.0016953	2022.72	360
6	A	3000	3000	(200 200)	(2800 200)	0.001595	4307.2	231
6	A	3000	3000	(200 2800)	(2800 200)	0.0016992	4827.54	273
6	A	3000	3000	(200 2800)	(2800 2800)	0.0013983	4565.77	303
6	A	3000	3000	(200 200)	(2800 2800)	0.0022939	4045.44	362

Tabela 4.15. Vremena pretrage - Mapa A - QuadTree dubine 6

Dubina	Mapa	Širina	Visina	Početak	Cilj	Vrijeme(s)	Udaljenost	Broj posjećenih čvorova
6	B	100	100	(6 6)	(93 6)	0.0037899	171.93	580
6	B	100	100	(6 93)	(93 6)	0.0047837	192.83	598
6	B	100	100	(6 93)	(93 93)	0.0019946	151.115	424
6	B	100	100	(6 6)	(93 93)	0.0023935	135.59	412
6	B	200	200	(13 13)	(186 13)	0.0031874	326.922	444
6	B	200	200	(13 186)	(186 13)	0.0034906	373.643	478
6	B	200	200	(13 186)	(186 186)	0.0016954	309.544	321
6	B	200	200	(13 13)	(186 186)	0.0017952	270.937	304
6	B	400	400	(26 26)	(373 26)	0.0041883	655.485	468
6	B	400	400	(26 373)	(373 26)	0.006418	748.599	494
6	B	400	400	(26 373)	(373 373)	0.0020958	621.158	356
6	B	400	400	(26 26)	(373 373)	0.0016954	538.399	343
6	B	500	500	(33 33)	(466 33)	0.0042775	818.111	511
6	B	500	500	(33 466)	(466 33)	0.004887	934.674	503
6	B	500	500	(33 466)	(466 466)	0.0023936	775.556	356
6	B	500	500	(33 33)	(466 466)	0.0020944	671.937	343
6	B	600	600	(40 40)	(560 40)	0.0034913	982.616	457
6	B	600	600	(40 560)	(560 40)	0.0044889	1107.79	480
6	B	600	600	(40 560)	(560 560)	0.0014961	932.066	312
6	B	600	600	(40 40)	(560 560)	0.0020944	806.891	321
6	B	1000	1000	(66 66)	(933 66)	0.0048868	1664.22	545
6	B	1000	1000	(66 933)	(933 66)	0.0037898	1872.15	513
6	B	1000	1000	(66 933)	(933 933)	0.0017959	1553.21	269
6	B	1000	1000	(66 66)	(933 933)	0.0020953	1345.29	380
6	B	1500	1500	(100 100)	(1400 100)	0.0038901	2490.75	520
6	B	1500	1500	(100 1400)	(1400 100)	0.0046867	2826.33	578
6	B	1500	1500	(100 1400)	(1400 1400)	0.001496	2330.16	323
6	B	1500	1500	(100 100)	(1400 1400)	0.001496	2017.23	337
6	B	3000	3000	(200 200)	(2800 200)	0.0041888	4981.51	513
6	B	3000	3000	(200 2800)	(2800 200)	0.0039285	5607.38	529
6	B	3000	3000	(200 2800)	(2800 2800)	0.0014971	4660.33	295
6	B	3000	3000	(200 200)	(2800 2800)	0.0017952	4034.45	334

Tabela 4.16. Vremena pretrage - Mapa B - QuadTree dubine 6

Dubina	Mapa	Širina	Visina	Početak	Cilj	Vrijeme(s)	Udaljenost	Broj posjećenih čvorova
7	A	100	100	(6 6)	(93 6)	0.0047794	114.753	931
7	A	100	100	(6 93)	(93 6)	0.0085826	135.918	1540
7	A	100	100	(6 93)	(93 93)	0.0072894	92.3118	1154
7	A	100	100	(6 6)	(93 93)	0.0061022	132.92	1200
7	A	200	200	(13 13)	(186 13)	0.0041859	283.094	566
7	A	200	200	(13 186)	(186 13)	0.0043987	317.11	601
7	A	200	200	(13 186)	(186 186)	0.002293	296.316	493
7	A	200	200	(13 13)	(186 186)	0.0022925	257.198	592
7	A	400	400	(26 26)	(373 26)	0.0041888	570.681	552
7	A	400	400	(26 373)	(373 26)	0.0044892	639.368	553
7	A	400	400	(26 373)	(373 373)	0.004092	606.976	509
7	A	400	400	(26 26)	(373 373)	0.0024164	538.289	557
7	A	500	500	(33 33)	(466 33)	0.0039898	717.311	535
7	A	500	500	(33 466)	(466 33)	0.0045867	798.135	575
7	A	500	500	(33 466)	(466 466)	0.0024923	757.829	495
7	A	500	500	(33 33)	(466 466)	0.0022938	671.8	547
7	A	600	600	(40 40)	(560 40)	0.0041347	854.229	551
7	A	600	600	(40 560)	(560 40)	0.0044996	958.297	553
7	A	600	600	(40 560)	(560 560)	0.0030918	910.793	509
7	A	600	600	(40 40)	(560 560)	0.002294	806.726	557
7	A	1000	1000	(66 66)	(933 66)	0.0025458	1435.31	365
7	A	1000	1000	(66 933)	(933 66)	0.0027965	1608.05	390
7	A	1000	1000	(66 933)	(933 933)	0.0013963	1469.11	392
7	A	1000	1000	(66 66)	(933 933)	0.0028924	1345.01	597
7	A	1500	1500	(100 100)	(1400 100)	0.0022937	2151.19	376
7	A	1500	1500	(100 1400)	(1400 100)	0.0031293	2411.36	392
7	A	1500	1500	(100 1400)	(1400 1400)	0.0018945	2204	400
7	A	1500	1500	(100 100)	(1400 1400)	0.0022939	1943.84	488
7	A	3000	3000	(200 200)	(2800 200)	0.0024932	4302.37	377
7	A	3000	3000	(200 2800)	(2800 200)	0.0027926	4822.71	403
7	A	3000	3000	(200 2800)	(2800 2800)	0.0014991	4408.01	391
7	A	3000	3000	(200 200)	(2800 2800)	0.0028922	4033.63	599

Tabela 4.17. Vremena pretrage - Mapa A - QuadTree dubine 7

Dubina	Mapa	Širina	Visina	Početak	Cilj	Vrijeme(s)	Udaljenost	Broj posjećenih čvorova
7	B	100	100	(6 6)	(93 6)	0.0108826	122.913	1803
7	B	100	100	(6 93)	(93 6)	0.0275853	141.883	3063
7	B	100	100	(6 93)	(93 93)	0.0084863	92.1109	1342
7	B	100	100	(6 6)	(93 93)	0.0100765	135.262	1677
7	B	200	200	(13 13)	(186 13)	0.0079197	319.815	1077
7	B	200	200	(13 186)	(186 13)	0.0095116	361.87	1126
7	B	200	200	(13 186)	(186 186)	0.0039873	303.551	649
7	B	200	200	(13 13)	(186 186)	0.0034907	261.364	753
7	B	400	400	(26 26)	(373 26)	0.0109634	645.878	996
7	B	400	400	(26 373)	(373 26)	0.008181	729.205	1107
7	B	400	400	(26 373)	(373 373)	0.0040888	615.541	565
7	B	400	400	(26 26)	(373 373)	0.0036929	541.608	688
7	B	500	500	(33 33)	(466 33)	0.0071812	806.595	890
7	B	500	500	(33 466)	(466 33)	0.0075796	929.528	954
7	B	500	500	(33 466)	(466 466)	0.0027926	773.948	515
7	B	500	500	(33 33)	(466 466)	0.0031915	675.949	590
7	B	600	600	(40 40)	(560 40)	0.0068817	968.206	854
7	B	600	600	(40 560)	(560 40)	0.0089759	1116.56	872
7	B	600	600	(40 560)	(560 560)	0.0033909	930.136	546
7	B	600	600	(40 40)	(560 560)	0.0045878	811.705	553
7	B	1000	1000	(66 66)	(933 66)	0.0074798	1612.93	838
7	B	1000	1000	(66 933)	(933 66)	0.0077788	1859.48	914
7	B	1000	1000	(66 933)	(933 933)	0.0034906	1550	550
7	B	1000	1000	(66 66)	(933 933)	0.0033908	1341.09	594
7	B	1500	1500	(100 100)	(1400 100)	0.0066784	2444.65	824
7	B	1500	1500	(100 1400)	(1400 100)	0.0073795	2759.72	854
7	B	1500	1500	(100 1400)	(1400 1400)	0.0030917	2325.34	483
7	B	1500	1500	(100 100)	(1400 1400)	0.0046873	2029.26	602
7	B	3000	3000	(200 200)	(2800 200)	0.0060837	4889.3	822
7	B	3000	3000	(200 2800)	(2800 200)	0.0075796	5519.44	859
7	B	3000	3000	(200 2800)	(2800 2800)	0.002693	4715.25	502
7	B	3000	3000	(200 200)	(2800 2800)	0.0037898	4058.52	623

Tabela 4.18. Vremena pretrage - Mapa B - QuadTree dubine 7

U tabeli 4.12. mogu se uočiti anomalije u vremenima pretraga, a broj posjećenih čvorova -1. Ovo znači da pretraga nije mogla biti pokrenuta ili ne postoji put između date dvije tačke. Do toga je došlo zbog niske rezolucije stabla (dubina 4), a velikih promjena na mapi, što je zahtjevalo puno veću preciznost. U dijelu o vizuelizaciji rezultata će biti prikazana ova mapa.

Što je izuzetno interesantno i značajno ja ovaj rad jeste da pretraga između dvije tačke u QuadTree-u zahtijeva rijetko više od 10 ms dok je kod matrice čak najbrža pretraga na najmanjoj mapi trajala 100 ms. Međutim brzina pretrage dolazi po cijenu puta. Dužina puta je na nekim mjestima čak i do 10% duža nego nad matricom, što bi trebalo da se popravi diskretizacijom granica.

4.5. Rezultati traženja putanje korištenjem A* algoritma na mapi dekomponiranoj u QuadTree-eve sa diskretiziranim granicama okvira

Testiranje je vršeno analogno kao u prethodnom poglavlju, s tim da je stepen diskretizacije bio fiksiran na vrijednost 5. Što znači da se između dva tjemena nalazi 5 tačaka. Veći stepen diskretizacije znači i duže vrijeme pretrage.

Dubina	Mapa	Širina	Visina	Početak	Cilj	Vrijeme(s)	Udaljenost	Broj posjećenih čvorova
4	A	100	100	(6 6)	(93 6)	0	0	-1
4	A	100	100	(6 93)	(93 6)	0	0	-1
4	A	100	100	(6 93)	(93 93)	0	0	-1
4	A	100	100	(6 6)	(93 93)	0	0	-1
4	A	200	200	(13 13)	(186 13)	0.0030907	292.392	73
4	A	200	200	(13 186)	(186 13)	0.0050863	335.499	79
4	A	200	200	(13 186)	(186 186)	0.0022939	302.019	60
4	A	200	200	(13 13)	(186 186)	0.0042886	263.066	62
4	A	400	400	(26 26)	(373 26)	0.0042885	585.467	69
4	A	400	400	(26 373)	(373 26)	0.0109703	672.367	78
4	A	400	400	(26 373)	(373 373)	0.0019946	606.136	59
4	A	400	400	(26 26)	(373 373)	0.0023938	527.547	62
4	A	500	500	(33 33)	(466 33)	0.0030916	730.593	69
4	A	500	500	(33 466)	(466 33)	0.0037898	839.391	78
4	A	500	500	(33 466)	(466 466)	0.0066821	756.783	59
4	A	500	500	(33 33)	(466 466)	0.0025921	658.373	62
4	A	600	600	(40 40)	(560 40)	0.0026926	876.422	69
4	A	600	600	(40 560)	(560 40)	0.0048866	1007.82	78
4	A	600	600	(40 560)	(560 560)	0.002593	909.547	59
4	A	600	600	(40 40)	(560 560)	0.0030912	790.613	62
4	A	1000	1000	(66 66)	(933 66)	0.0028935	1461.88	69
4	A	1000	1000	(66 933)	(933 66)	0.0030927	1680.18	78
4	A	1000	1000	(66 933)	(933 933)	0.0037896	1515.68	59
4	A	1000	1000	(66 66)	(933 933)	0.0152593	1318.16	62
4	A	1500	1500	(100 100)	(1400 100)	0.0032905	2191.05	69
4	A	1500	1500	(100 1400)	(1400 100)	0.0080798	2519.55	78
4	A	1500	1500	(100 1400)	(1400 1400)	0.0029918	2273.87	59
4	A	1500	1500	(100 100)	(1400 1400)	0.002194	1976.53	62
4	A	3000	3000	(200 200)	(2800 200)	0.0024933	4382.11	69
4	A	3000	3000	(200 2800)	(2800 200)	0.0026931	5039.1	78
4	A	3000	3000	(200 2800)	(2800 2800)	0.0022938	4547.73	59
4	A	3000	3000	(200 200)	(2800 2800)	0.0020943	3953.06	62

Tabela 4.19. Vremena pretrage - Mapa A - QuadTree sa d. granicama dubine 4

**Rezultati traženja putanje korištenjem A* algoritma na mapi
dekomponiranoj u QuadTree-eve sa diskretiziranim granicama okvira 46**

Dubina	Mapa	Širina	Visina	Početak	Cilj	Vrijeme(s)	Udaljenost	Broj posjećenih čvorova
4	B	100	100	(6 6)	(93 6)	0	0	-1
4	B	100	100	(6 93)	(93 6)	0	0	-1
4	B	100	100	(6 93)	(93 93)	0	0	-1
4	B	100	100	(6 6)	(93 93)	0	0	-1
4	B	200	200	(13 13)	(186 13)	0	0	-1
4	B	200	200	(13 186)	(186 13)	0	0	-1
4	B	200	200	(13 186)	(186 186)	0.0001007	0	-1
4	B	200	200	(13 13)	(186 186)	0	0	-1
4	B	400	400	(26 26)	(373 26)	0	0	-1
4	B	400	400	(26 373)	(373 26)	0	0	-1
4	B	400	400	(26 373)	(373 373)	9.96E-05	0	-1
4	B	400	400	(26 26)	(373 373)	0	0	-1
4	B	500	500	(33 33)	(466 33)	0	0	-1
4	B	500	500	(33 466)	(466 33)	0	0	-1
4	B	500	500	(33 466)	(466 466)	0	0	-1
4	B	500	500	(33 33)	(466 466)	0	0	-1
4	B	600	600	(40 40)	(560 40)	0	0	-1
4	B	600	600	(40 560)	(560 40)	9.94E-05	0	-1
4	B	600	600	(40 560)	(560 560)	9.98E-05	0	-1
4	B	600	600	(40 40)	(560 560)	0	0	-1
4	B	1000	1000	(66 66)	(933 66)	0	0	-1
4	B	1000	1000	(66 933)	(933 66)	0	0	-1
4	B	1000	1000	(66 933)	(933 933)	0	0	-1
4	B	1000	1000	(66 66)	(933 933)	0	0	-1
4	B	1500	1500	(100 100)	(1400 100)	9.91E-05	0	-1
4	B	1500	1500	(100 1400)	(1400 100)	9.99E-05	0	-1
4	B	1500	1500	(100 1400)	(1400 1400)	0	0	-1
4	B	1500	1500	(100 100)	(1400 1400)	0	0	-1
4	B	3000	3000	(200 200)	(2800 200)	0	0	-1
4	B	3000	3000	(200 2800)	(2800 200)	0	0	-1
4	B	3000	3000	(200 2800)	(2800 2800)	0	0	-1
4	B	3000	3000	(200 200)	(2800 2800)	0	0	-1

Tabela 4.20. Vremena pretrage - Mapa B - QuadTree sa d. granicama dubine 4

**Rezultati traženja putanje korištenjem A* algoritma na mapi
dekomponiranoj u QuadTree-eve sa diskretiziranim granicama okvira 47**

Dubina	Mapa	Širina	Visina	Početak	Cilj	Vrijeme(s)	Udaljenost	Broj posjećenih čvorova
5	A	100	100	(6 6)	(93 6)	0.0047872	142.472	128
5	A	100	100	(6 93)	(93 6)	0.006383	162.933	160
5	A	100	100	(6 93)	(93 93)	0.0045887	149.855	136
5	A	100	100	(6 6)	(93 93)	0.003092	131.123	109
5	A	200	200	(13 13)	(186 13)	0.0054854	282.42	130
5	A	200	200	(13 186)	(186 13)	0.0082778	325.528	165
5	A	200	200	(13 186)	(186 186)	0.0100732	299.694	136
5	A	200	200	(13 13)	(186 186)	0.0046874	260.831	110
5	A	400	400	(26 26)	(373 26)	0.0050863	565.139	128
5	A	400	400	(26 373)	(373 26)	0.010871	652.038	165
5	A	400	400	(26 373)	(373 373)	0.0057844	601.802	136
5	A	400	400	(26 26)	(373 373)	0.0029926	523.077	125
5	A	500	500	(33 33)	(466 33)	0.0054854	705.163	128
5	A	500	500	(33 466)	(466 33)	0.0098733	813.961	164
5	A	500	500	(33 466)	(466 466)	0.0055849	751.676	136
5	A	500	500	(33 33)	(466 466)	0.0042887	652.785	125
5	A	600	600	(40 40)	(560 40)	0.0084778	847.801	132
5	A	600	600	(40 560)	(560 40)	0.007081	979.199	171
5	A	600	600	(40 560)	(560 560)	0.0040891	903.967	136
5	A	600	600	(40 40)	(560 560)	0.0034905	783.908	125
5	A	1000	1000	(66 66)	(933 66)	0.0045877	1410.65	128
5	A	1000	1000	(66 933)	(933 66)	0.0061835	1628.94	164
5	A	1000	1000	(66 933)	(933 933)	0.0052856	1505.76	136
5	A	1000	1000	(66 66)	(933 933)	0.0027951	1306.98	125
5	A	1500	1500	(100 100)	(1400 100)	0.0045879	2119.5	132
5	A	1500	1500	(100 1400)	(1400 100)	0.005787	2448	174
5	A	1500	1500	(100 1400)	(1400 1400)	0.0045879	2259.92	136
5	A	1500	1500	(100 100)	(1400 1400)	0.0037884	1959.77	125
5	A	3000	3000	(200 200)	(2800 200)	0.0047872	4190.91	132
5	A	3000	3000	(200 2800)	(2800 200)	0.0058842	4847.9	168
5	A	3000	3000	(200 2800)	(2800 2800)	0.004488	4519.83	136
5	A	3000	3000	(200 200)	(2800 2800)	0.0028922	3919.54	125

Tabela 4.21. Vremena pretrage - Mapa A - QuadTree sa d. granicama dubine 5

**Rezultati traženja putanje korištenjem A* algoritma na mapi
dekomponiranoj u QuadTree-eve sa diskretiziranim granicama okvira 48**

Dubina	Mapa	Širina	Visina	Početak	Cilj	Vrijeme(s)	Udaljenost	Broj posjećenih čvorova
5	B	100	100	(6 6)	(93 6)	0.0053857	162.982	169
5	B	100	100	(6 93)	(93 6)	0.0071806	185.471	200
5	B	100	100	(6 93)	(93 93)	0.0028907	152.23	108
5	B	100	100	(6 6)	(93 93)	0.0019978	130.909	81
5	B	200	200	(13 13)	(186 13)	0.0053817	349.121	189
5	B	200	200	(13 186)	(186 13)	0.006682	387.475	208
5	B	200	200	(13 186)	(186 186)	0.0028923	311.251	110
5	B	200	200	(13 13)	(186 186)	0.0023977	268.08	105
5	B	400	400	(26 26)	(373 26)	0.0052821	698.704	182
5	B	400	400	(26 373)	(373 26)	0.0062849	776.303	207
5	B	400	400	(26 373)	(373 373)	0.0039883	624.606	110
5	B	400	400	(26 26)	(373 373)	0.0028898	537.263	97
5	B	500	500	(33 33)	(466 33)	0.0075795	872.347	183
5	B	500	500	(33 466)	(466 33)	0.0084773	969.374	208
5	B	500	500	(33 466)	(466 466)	0.0028921	780.254	110
5	B	500	500	(33 33)	(466 466)	0.0022938	670.593	97
5	B	600	600	(40 40)	(560 40)	0.0057844	1046.48	189
5	B	600	600	(40 560)	(560 40)	0.0087769	1163.84	208
5	B	600	600	(40 560)	(560 560)	0.0032912	938.028	110
5	B	600	600	(40 40)	(560 560)	0.0033895	805.044	97
5	B	1000	1000	(66 66)	(933 66)	0.0064827	1824.81	197
5	B	1000	1000	(66 933)	(933 66)	0.0071781	2019.77	217
5	B	1000	1000	(66 933)	(933 933)	0.0038898	1562.63	110
5	B	1000	1000	(66 66)	(933 933)	0.0031914	1342.31	97
5	B	1500	1500	(100 100)	(1400 100)	0.0070812	2735.66	196
5	B	1500	1500	(100 1400)	(1400 100)	0.0073803	3029.05	216
5	B	1500	1500	(100 1400)	(1400 1400)	0.0033908	2345.07	110
5	B	1500	1500	(100 100)	(1400 1400)	0.0026927	2012.61	97
5	B	3000	3000	(200 200)	(2800 200)	0.0066803	5471.32	196
5	B	3000	3000	(200 2800)	(2800 200)	0.0118682	6058.11	216
5	B	3000	3000	(200 2800)	(2800 2800)	0.0048869	4690.14	112
5	B	3000	3000	(200 200)	(2800 2800)	0.002992	4025.22	99

Tabela 4.22. Vremena pretrage - Mapa B - QuadTree sa d. granicama dubine 5

**Rezultati traženja putanje korištenjem A* algoritma na mapi
dekomponiranoj u QuadTree-eve sa diskretiziranim granicama okvira 49**

Dubina	Mapa	Širina	Visina	Početak	Cilj	Vrijeme(s)	Udaljenost	Broj posjećenih čvorova
6	A	100	100	(6 6)	(93 6)	0.0099743	138.405	246
6	A	100	100	(6 93)	(93 6)	0.0129653	155.686	284
6	A	100	100	(6 93)	(93 93)	0.0073831	146.33	247
6	A	100	100	(6 6)	(93 93)	0.0065809	129.471	268
6	A	200	200	(13 13)	(186 13)	0.0076712	278.6	179
6	A	200	200	(13 186)	(186 13)	0.0096742	315.349	208
6	A	200	200	(13 186)	(186 186)	0.0078788	293.765	180
6	A	200	200	(13 13)	(186 186)	0.0048869	260.426	151
6	A	400	400	(26 26)	(373 26)	0.0054853	565.139	160
6	A	400	400	(26 373)	(373 26)	0.0076794	639.32	192
6	A	400	400	(26 373)	(373 373)	0.0057845	589.944	157
6	A	400	400	(26 26)	(373 373)	0.0048868	522.267	140
6	A	500	500	(33 33)	(466 33)	0.0060838	705.163	160
6	A	500	500	(33 466)	(466 33)	0.0082761	798.063	191
6	A	500	500	(33 466)	(466 466)	0.005286	736.853	157
6	A	500	500	(33 33)	(466 466)	0.0039898	651.773	140
6	A	600	600	(40 40)	(560 40)	0.0091756	847.801	164
6	A	600	600	(40 560)	(560 40)	0.0083777	957.832	186
6	A	600	600	(40 560)	(560 560)	0.0054857	886.18	157
6	A	600	600	(40 40)	(560 560)	0.0045843	782.693	140
6	A	1000	1000	(66 66)	(933 66)	0.0068816	1410.65	160
6	A	1000	1000	(66 933)	(933 66)	0.0111701	1597.14	187
6	A	1000	1000	(66 933)	(933 933)	0.0056848	1476.12	157
6	A	1000	1000	(66 66)	(933 933)	0.0037899	1304.96	140
6	A	1500	1500	(100 100)	(1400 100)	0.0069814	2119.5	164
6	A	1500	1500	(100 1400)	(1400 100)	0.0083777	2394.58	186
6	A	1500	1500	(100 1400)	(1400 1400)	0.004687	2215.45	157
6	A	1500	1500	(100 100)	(1400 1400)	0.0037905	1956.73	140
6	A	3000	3000	(200 200)	(2800 200)	0.0061835	4190.91	166
6	A	3000	3000	(200 2800)	(2800 200)	0.007878	4752.51	190
6	A	3000	3000	(200 2800)	(2800 2800)	0.005983	4430.9	157
6	A	3000	3000	(200 200)	(2800 2800)	0.0051862	3913.47	140

Tabela 4.23. Vremena pretrage - Mapa A - QuadTree sa d. granicama dubine 6

**Rezultati traženja putanje korištenjem A* algoritma na mapi
dekomponiranoj u QuadTree-eve sa diskretiziranim granicama okvira 50**

Dubina	Mapa	Širina	Visina	Početak	Cilj	Vrijeme(s)	Udaljenost	Broj posjećenih čvorova
6	B	100	100	(6 6)	(93 6)	0.0145612	161.28	408
6	B	100	100	(6 93)	(93 6)	0.0192488	180.322	476
6	B	100	100	(6 93)	(93 93)	0.0067818	144.357	222
6	B	100	100	(6 6)	(93 93)	0.0061829	130.521	234
6	B	200	200	(13 13)	(186 13)	0.0116687	323.535	333
6	B	200	200	(13 186)	(186 13)	0.0143637	364.205	388
6	B	200	200	(13 186)	(186 186)	0.0053898	299.783	189
6	B	200	200	(13 13)	(186 186)	0.0044916	261.115	179
6	B	400	400	(26 26)	(373 26)	0.013759	648.111	356
6	B	400	400	(26 373)	(373 26)	0.0174559	730.343	397
6	B	400	400	(26 373)	(373 373)	0.0051862	601.67	184
6	B	400	400	(26 26)	(373 373)	0.0057846	523.334	192
6	B	500	500	(33 33)	(466 33)	0.0148602	808.898	360
6	B	500	500	(33 466)	(466 33)	0.0162565	911.949	401
6	B	500	500	(33 466)	(466 466)	0.0051859	751.584	184
6	B	500	500	(33 33)	(466 466)	0.0051901	653.181	192
6	B	600	600	(40 40)	(560 40)	0.0141621	975.528	348
6	B	600	600	(40 560)	(560 40)	0.0164557	1100.31	391
6	B	600	600	(40 560)	(560 560)	0.0058843	903.624	177
6	B	600	600	(40 40)	(560 560)	0.005186	784.149	188
6	B	1000	1000	(66 66)	(933 66)	0.0164563	1651.14	392
6	B	1000	1000	(66 933)	(933 66)	0.018255	1858.14	431
6	B	1000	1000	(66 933)	(933 933)	0.0049824	1505.29	181
6	B	1000	1000	(66 66)	(933 933)	0.0050888	1307.48	192
6	B	1500	1500	(100 100)	(1400 100)	0.0143663	2474.93	382
6	B	1500	1500	(100 1400)	(1400 100)	0.016057	2786.88	423
6	B	1500	1500	(100 1400)	(1400 1400)	0.0045877	2259.06	177
6	B	1500	1500	(100 100)	(1400 1400)	0.0046874	1960.37	192
6	B	3000	3000	(200 200)	(2800 200)	0.0173536	4949.86	390
6	B	3000	3000	(200 2800)	(2800 200)	0.0163604	5573.76	432
6	B	3000	3000	(200 2800)	(2800 2800)	0.0049863	4518.12	181
6	B	3000	3000	(200 200)	(2800 2800)	0.0069813	3920.75	246

Tabela 4.24. Vremena pretrage - Mapa B - QuadTree sa d. granicama dubine 6

**Rezultati traženja putanje korištenjem A* algoritma na mapi
dekomponiranoj u QuadTree-eve sa diskretiziranim granicama okvira 51**

Dubina	Mapa	Širina	Visina	Početak	Cilj	Vrijeme(s)	Udaljenost	Broj posjećenih čvorova
7	A	100	100	(6 6)	(93 6)	0.0034907	87.4373	155
7	A	100	100	(6 93)	(93 6)	0.0211434	129.09	773
7	A	100	100	(6 93)	(93 93)	0.0031903	87.0834	150
7	A	100	100	(6 6)	(93 93)	0.0152605	128.597	627
7	A	200	200	(13 13)	(186 13)	0.0168576	277.022	467
7	A	200	200	(13 186)	(186 13)	0.021046	313.771	510
7	A	200	200	(13 186)	(186 186)	0.0097771	292.468	312
7	A	200	200	(13 13)	(186 186)	0.007683	259.129	313
7	A	400	400	(26 26)	(373 26)	0.0144588	561.05	394
7	A	400	400	(26 373)	(373 26)	0.0256314	635.231	441
7	A	400	400	(26 373)	(373 373)	0.0125667	589.944	293
7	A	400	400	(26 26)	(373 373)	0.008577	522.267	272
7	A	500	500	(33 33)	(466 33)	0.018055	700.052	424
7	A	500	500	(33 466)	(466 33)	0.0223443	783.398	470
7	A	500	500	(33 466)	(466 466)	0.0105708	736.853	308
7	A	500	500	(33 33)	(466 466)	0.0091767	651.773	304
7	A	600	600	(40 40)	(560 40)	0.0146608	839.379	396
7	A	600	600	(40 560)	(560 40)	0.0213428	940.234	444
7	A	600	600	(40 560)	(560 560)	0.0133653	886.18	293
7	A	600	600	(40 40)	(560 560)	0.0090758	782.693	264
7	A	1000	1000	(66 66)	(933 66)	0.011569	1400.43	282
7	A	1000	1000	(66 933)	(933 66)	0.0134667	1586.92	320
7	A	1000	1000	(66 933)	(933 933)	0.006682	1476.12	226
7	A	1000	1000	(66 66)	(933 933)	0.0061835	1304.96	236
7	A	1500	1500	(100 100)	(1400 100)	0.0099774	2098.45	282
7	A	1500	1500	(100 1400)	(1400 100)	0.0125662	2379.25	318
7	A	1500	1500	(100 1400)	(1400 1400)	0.0065818	2215.45	222
7	A	1500	1500	(100 100)	(1400 1400)	0.0074801	1956.73	243
7	A	3000	3000	(200 200)	(2800 200)	0.0100741	4166.14	286
7	A	3000	3000	(200 2800)	(2800 200)	0.0129692	4727.74	329
7	A	3000	3000	(200 2800)	(2800 2800)	0.0068855	4430.9	228
7	A	3000	3000	(200 200)	(2800 2800)	0.0064785	3913.47	234

Tabela 4.25. Vremena pretrage - Mapa A - QuadTree sa d. granicama dubine 7

**Rezultati traženja putanje korištenjem A* algoritma na mapi
dekomponiranoj u QuadTree-eve sa diskretiziranim granicama okvira 52**

Dubina	Mapa	Širina	Visina	Početak	Cilj	Vrijeme(s)	Udaljenost	Broj posjećenih čvorova
7	B	100	100	(6 6)	(93 6)	0.0039891	87.5508	180
7	B	100	100	(6 93)	(93 6)	0.0399923	131.01	1285
7	B	100	100	(6 93)	(93 93)	0.0031893	87.0834	143
7	B	100	100	(6 6)	(93 93)	0.0209413	129.795	838
7	B	200	200	(13 13)	(186 13)	0.028523	316.676	790
7	B	200	200	(13 186)	(186 13)	0.0357046	354.47	900
7	B	200	200	(13 186)	(186 186)	0.0125696	289.786	417
7	B	200	200	(13 13)	(186 186)	0.0082778	256.295	327
7	B	400	400	(26 26)	(373 26)	0.0309174	636.212	766
7	B	400	400	(26 373)	(373 26)	0.036604	719.557	893
7	B	400	400	(26 373)	(373 373)	0.0121675	589.087	382
7	B	400	400	(26 26)	(373 373)	0.0107712	517.766	350
7	B	500	500	(33 33)	(466 33)	0.0279254	794.72	686
7	B	500	500	(33 466)	(466 33)	0.0313185	901.25	749
7	B	500	500	(33 466)	(466 466)	0.0096747	745.655	319
7	B	500	500	(33 33)	(466 466)	0.0097738	646.221	332
7	B	600	600	(40 40)	(560 40)	0.0242352	954.067	643
7	B	600	600	(40 560)	(560 40)	0.0275239	1083.02	716
7	B	600	600	(40 560)	(560 560)	0.0115688	896.086	323
7	B	600	600	(40 40)	(560 560)	0.0101728	775.798	323
7	B	1000	1000	(66 66)	(933 66)	0.0294212	1597.74	653
7	B	1000	1000	(66 933)	(933 66)	0.0298245	1811.7	720
7	B	1000	1000	(66 933)	(933 933)	0.0098735	1493.23	324
7	B	1000	1000	(66 66)	(933 933)	0.010073	1293.56	333
7	B	1500	1500	(100 100)	(1400 100)	0.028424	2402.29	625
7	B	1500	1500	(100 1400)	(1400 100)	0.0282245	2724.67	691
7	B	1500	1500	(100 1400)	(1400 1400)	0.0093781	2240.22	314
7	B	1500	1500	(100 100)	(1400 1400)	0.0100729	1939.49	323
7	B	3000	3000	(200 200)	(2800 200)	0.0248338	4804.58	640
7	B	3000	3000	(200 2800)	(2800 200)	0.0308833	5449.34	701
7	B	3000	3000	(200 2800)	(2800 2800)	0.0101761	4480.43	322
7	B	3000	3000	(200 200)	(2800 2800)	0.0092725	3878.99	332

Tabela 4.26. Vremena pretrage - Mapa B - QuadTree sa d. granicama dubine 7

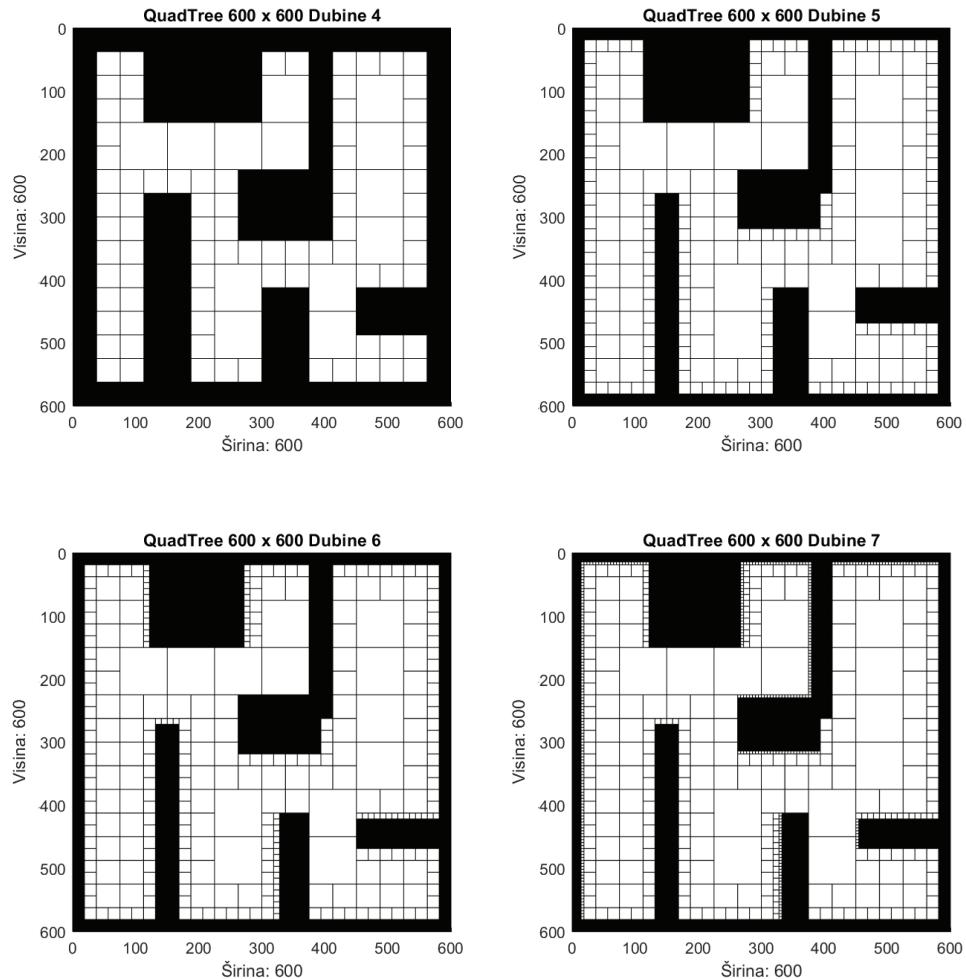
Iako je algoritam složeniji, vrijeme pretrage na rijetkim mjestima prelazi 30 ms što predstavlja neznatno usporenenje u odnosu na implementaciju bez diskretizacije. Iz tabele se može očitati da je diskretizacijom skraćena dužina puta na nekim pretragama za 2-3%.

4.6. Uporedba putanja dobijenih prezentiranim algoritmima

4.6.1. Kreiranje QuadTree-a

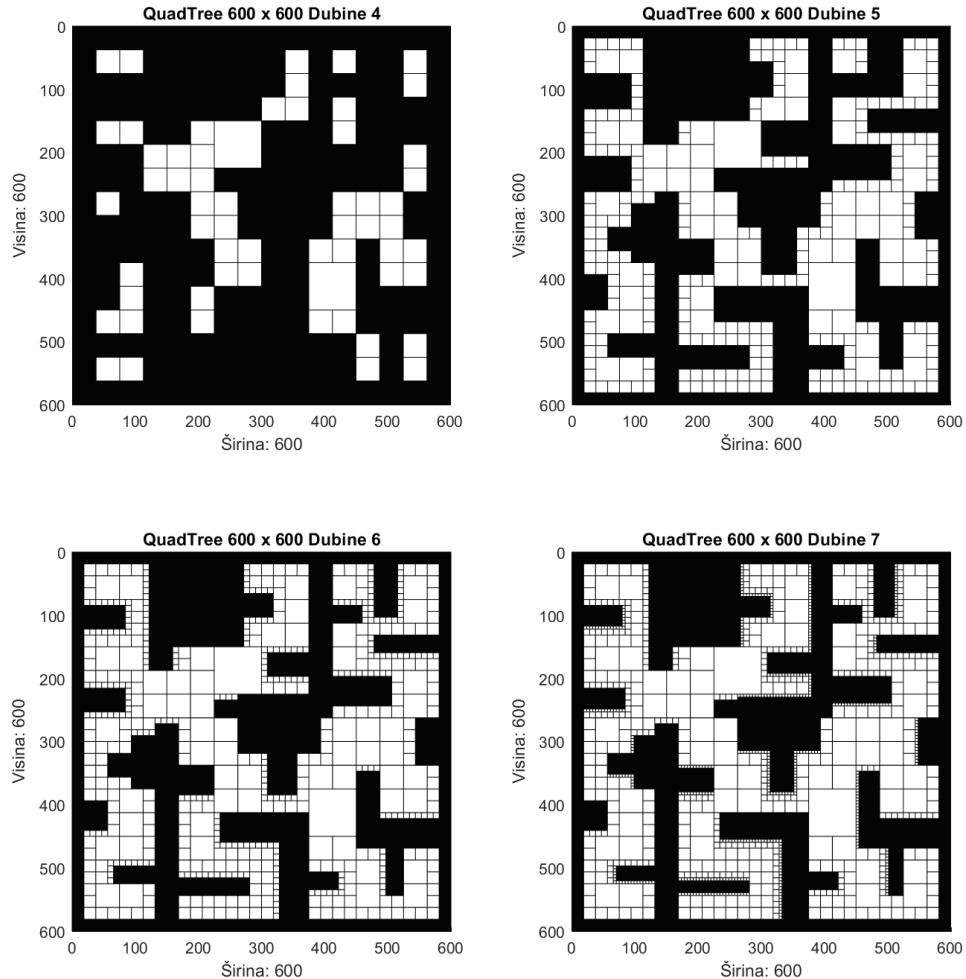
U ovom poglavlju će biti ilustrirani rezultati dobijeni iz tabele radi sticanja detaljnijeg uvida u dešavanja i bolje usporedbe između različitih izvedbi algoritma.

Prije svega će biti analizirane QuadTree reprezentacije mape A i B sa slike 4.1. na različitim dubinama.



Slika 4.2. QuadTree reprezentacija mape A veličine 600x600 na dubinama 4, 5, 6 i 7

Kako je mapa A značajno jednostavnija od mape B, ona se bez problema može predstaviti i sa QuadTree-em dubine 4, s tim da su zidovi mnogo deblji u odnosu na ostale dubine, što je jako dobro prikazano prvom ilustracijom sa slike 4.2.

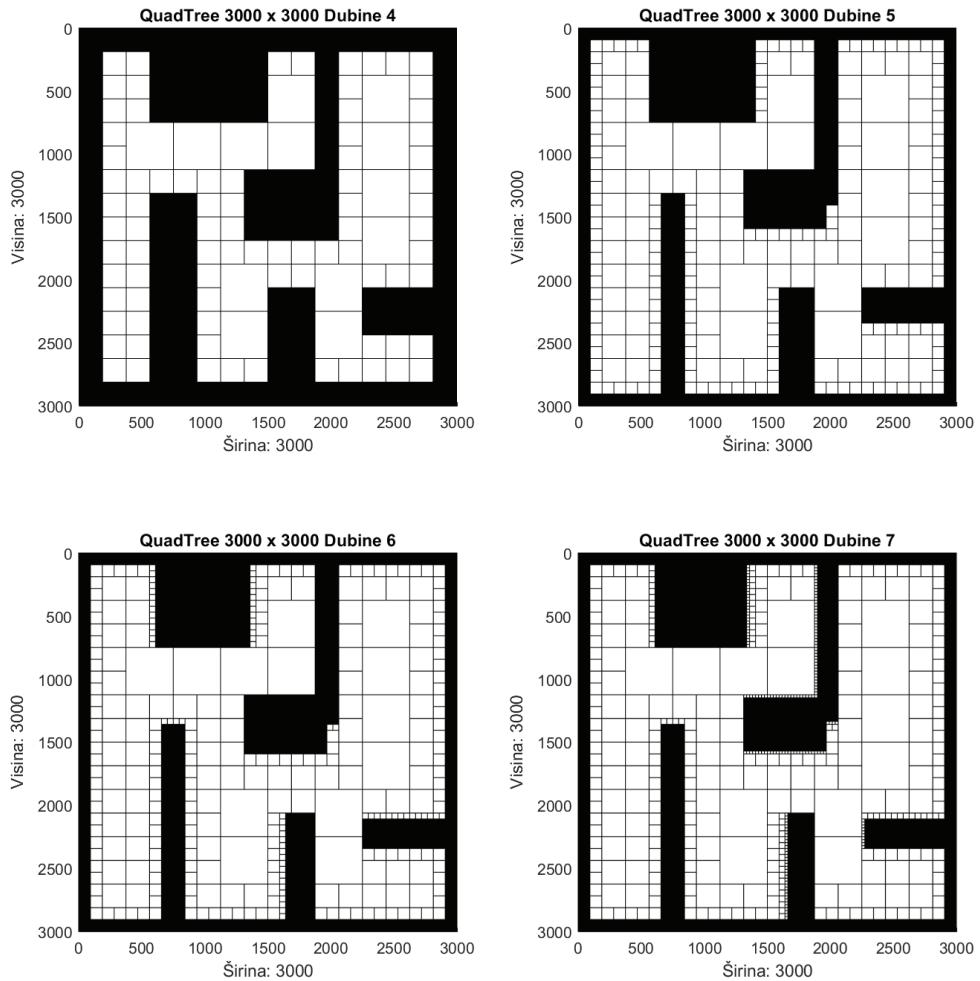


Slika 4.3. QuadTree reprezentacija mape B veličine 600x600 na dubinama 4, 5, 6 i 7

Kako je mapa B značajno detaljnija i komplikovanija od mape A, nju nije bilo moguće reprezentirati ni sa QuadTree-em dubine 4, ni dubine 5. Iako na slici 4.3. na prvi pogled djeluje kao da je moguće predstaviti mapu B sa QuadTree-em dubine 5, moguće je uočiti ograđeni prostor oko koordinata (450, 50). Kako su zidovi hodnika bili previše blizu, oni su se spojili. Tek na dubini 6 je mapa B dovoljno dobro prikazana.

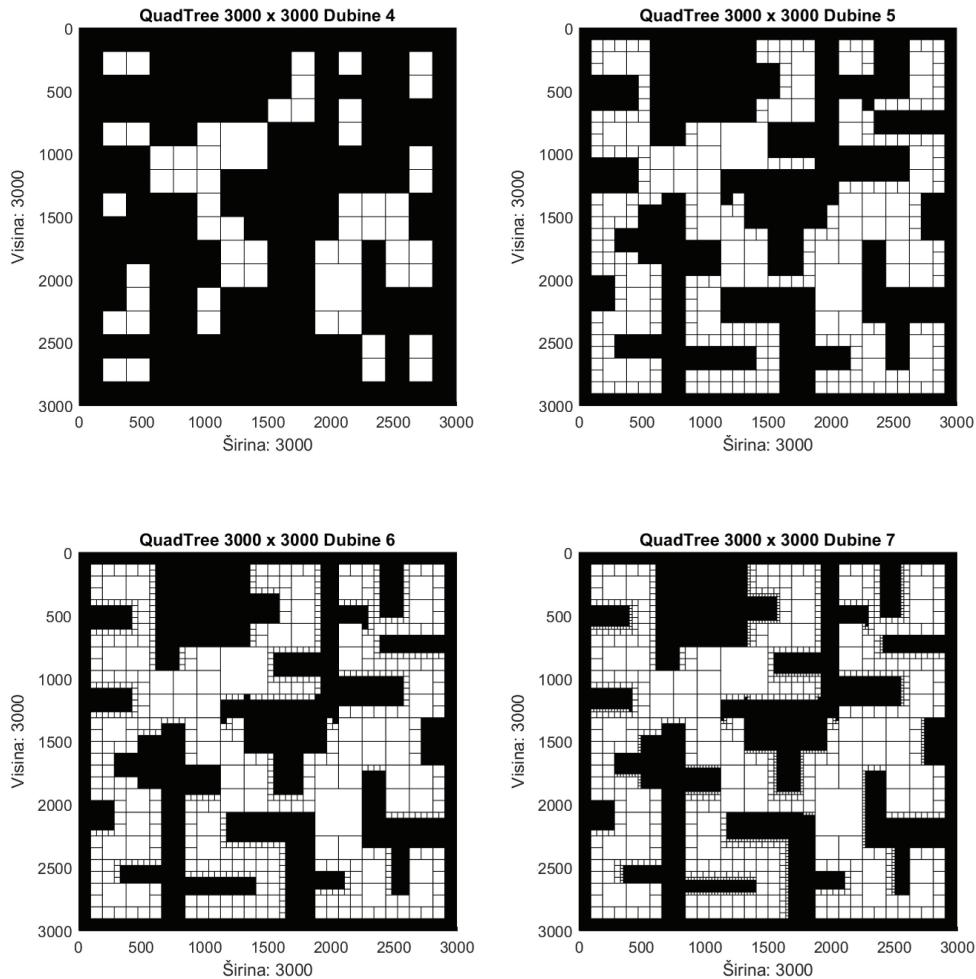
Međutim, činjenica da će zidovi nekad biti deblji u odnosu na originalnu mapu može imati svoje prednosti. Ukoliko se robot navigira kroz ovu mapu, to može osigurati određenu distancu između robota i zidova mape.

U nastavku će biti prikazane mape A i B veličine 3000x3000 kao QuadTree.



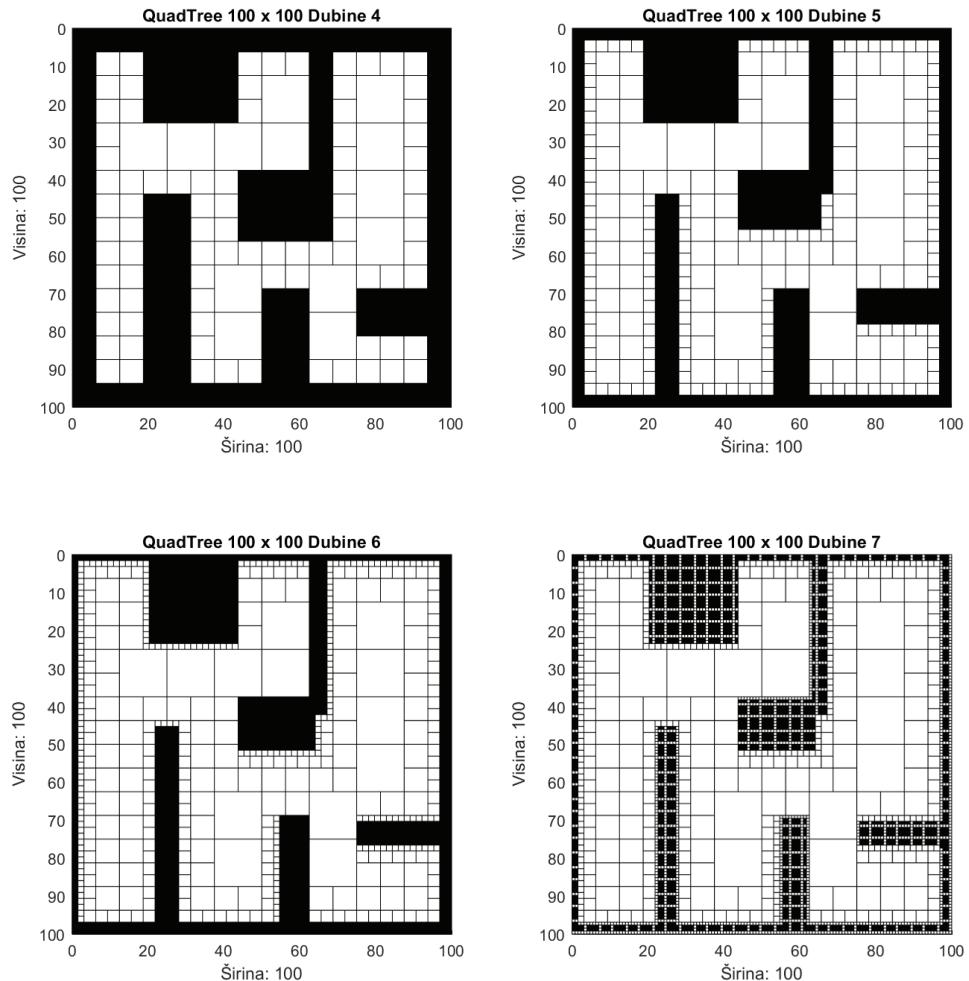
Slika 4.4. QuadTree reprezentacija mape A veličine 3000x3000 na dubinama 4, 5, 6 i 7

Poređenjem slike 4.2. i 4.4. primjećuje se da iako je mapa 5 puta veća, broj čvorova ostaje skoro isti. To potvrđuju i tabele 4.1., 4.2., 4.3. i 4.4. U posljednjoj koloni ovih tabela vidljivo je da iako veličina mape raste, broj čvorova se skoro pa ne mijenja. Isto je vidljivo i na slici 4.5.



Slika 4.5. QuadTree reprezentacija mape B veličine 3000x3000 na dubinama 4, 5, 6 i 7

Međutim u tabelama 4.1., 4.2., 4.3. i 4.4. postoji odstupanje od pravila da ista mapa različitih veličina u stablu ima sličan broj čvorova. Mape veličine 100x100 imaju značajno veći broj čvorova od svih ostalih mapa. Ovaj fenomen je već jednom objašnjen, ali na slikama 4.6. i 4.7. se može i vidjeti.

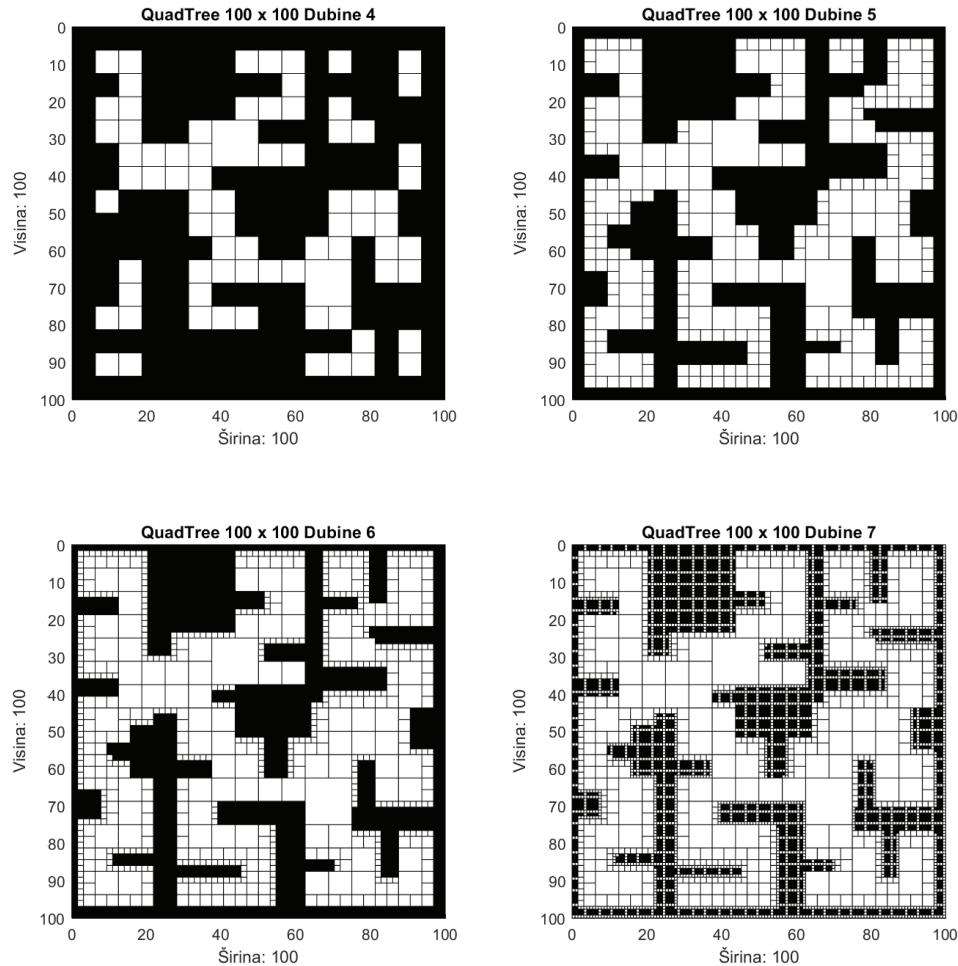


Slika 4.6. QuadTree reprezentacija mape A veličine 100x100 na dubinama 4, 5, 6 i 7

Rupe u zidovima mape su posljedica formule:

$$\text{širina_čvora} = \frac{\text{širina_mape}}{2^{\text{dubina}}}$$

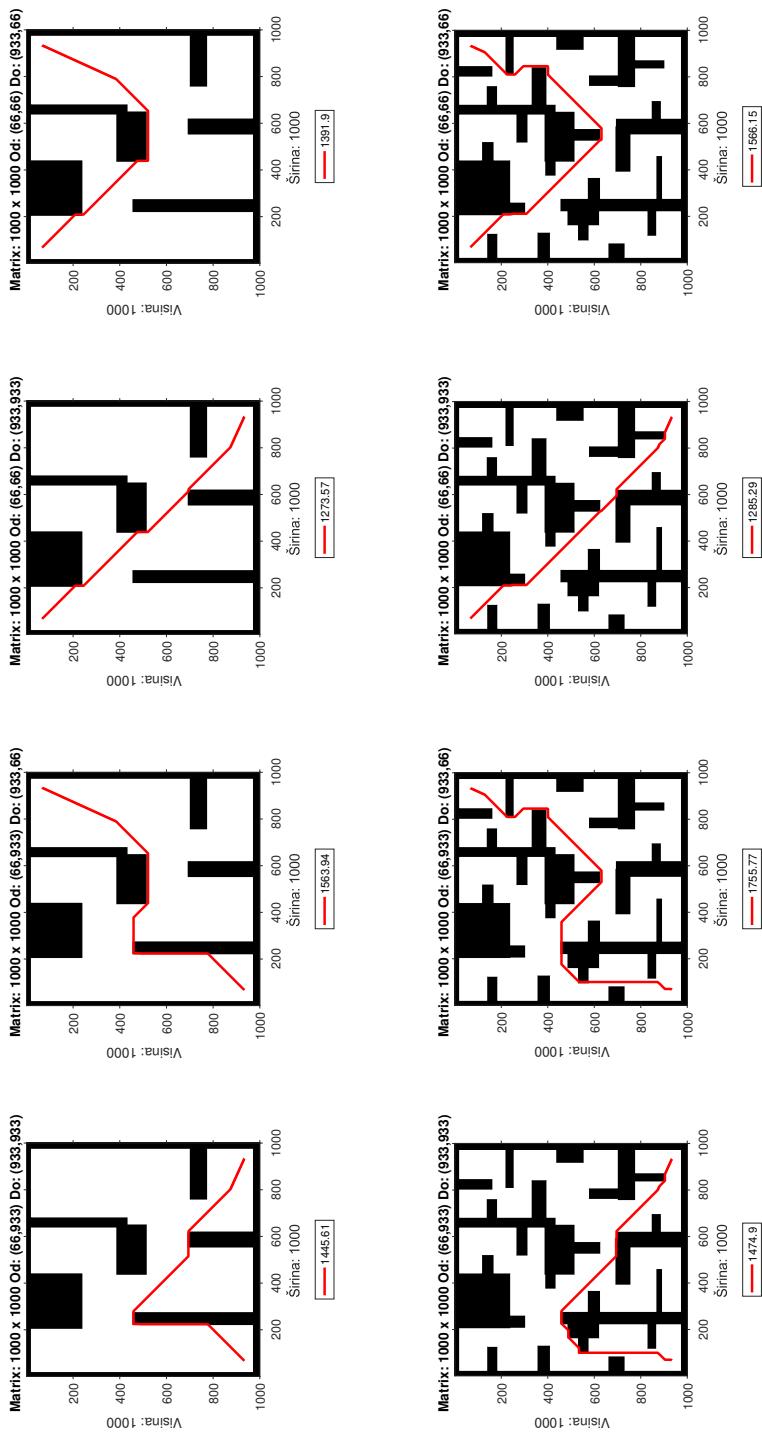
Kako je širina mape 100, a dubina 7, to znači da će najmanja čelija u mapi biti širine oko 0.78. Zbog toga će se desiti da neke čelije ostanu prazne. Međutim na slici 4.7. se vidi da je sada samo dubina 6 adekvatna za ovu mapu. Na dubini 7 se pojavljuju šupljine u zidovima, a na dubini 5 su zidovi previše debeli i spajaju se.



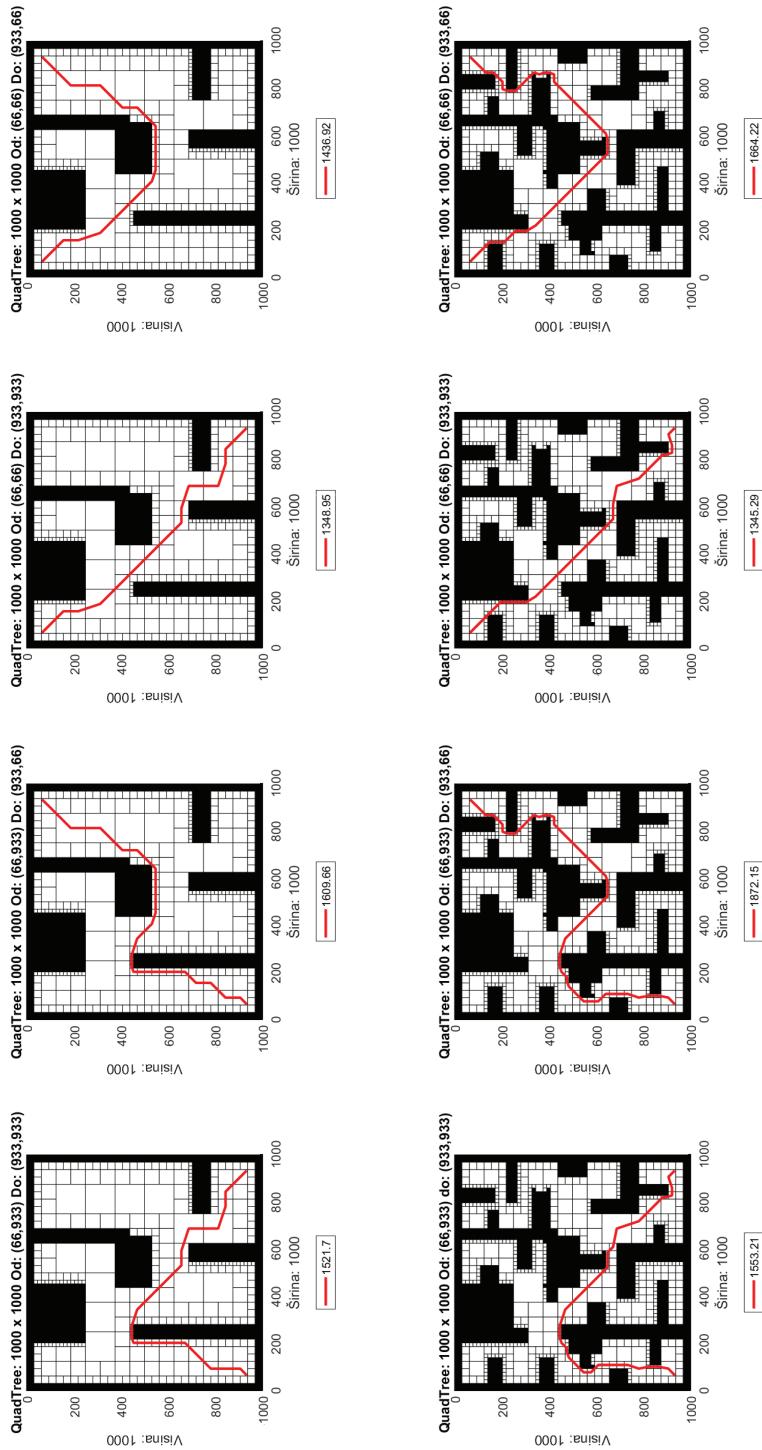
Slika 4.7. QuadTree reprezentacija mape B veličine 100x100 na dubinama 4, 5, 6 i 7

4.6.2. Pretraga

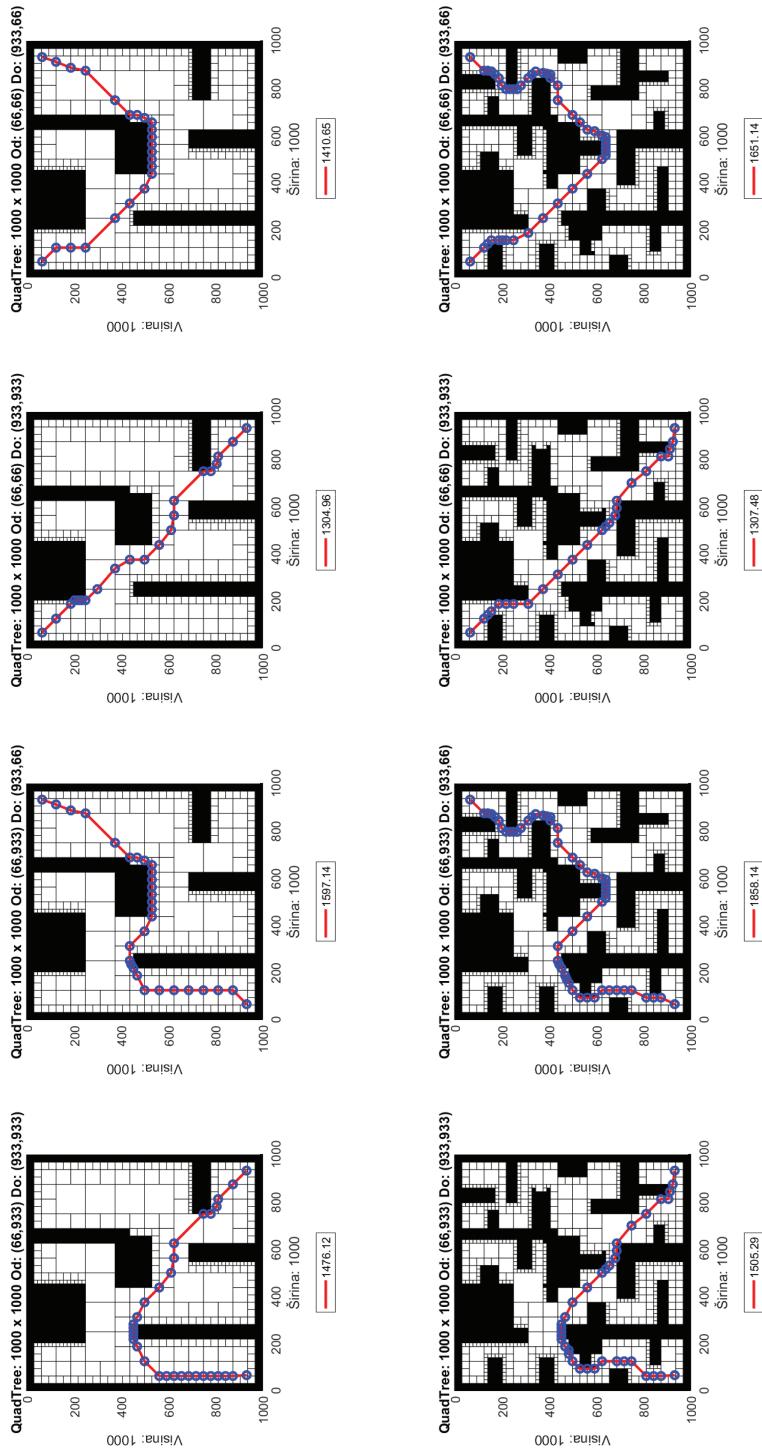
Na slikama 4.8., 4.9. i 4.10. se nalazi po osam ilustracija pretragre mape A i B veličine 1000x1000. U prvom redu pretraga je vršena na mapi A, dok je u drugom redu vršena na mapi B. Svaka kolona predstavlja različitu startnu poziciju pretrage. U prvoj koloni početna pozicija je u donjem lijevom uglu, a krajnja u donjem desnom uglu, u drugoj koloni pretraga je vršena iz donjeg lijevo u gornji desni ugao, u trećoj koloni iz gornjeg lijevog u donji desni i konačno iz gornjeg lijevog u gornji desni. Radi očuvanja konzistentnosti tačke su udaljene za 1/15 od ukupne širine i visine mape od rubova mape. Iznad ilustracije nalaze se koordinate početne i krajnje tačke, a ispod mape prikazana je ukupna dužina puta. Na slici 4.10. su plavim krugovima naznačene ulazne tačke (*accessPoint*) pojedinih čvorova.



Slika 4.8. Pretraga matrične mreže mapa A i B



Slika 4.9. Pretraga QuadTree-a dubine 6 mapa A i B



Slika 4.10. Pretraga QuadTree-a dubine 6 sa d. granicama mapa A i B

Mapa	Širina	Visina	Početak	Cilj	Vrijeme(s)	Udaljenost
Mrežasta mapa						
A	1000	1000	(66 66)	(933 66)	237.811	1391.9
A	1000	1000	(66 933)	(933 66)	347.598	1563.94
A	1000	1000	(66 933)	(933 933)	152.348	1445.61
A	1000	1000	(66 66)	(933 933)	176.574	1273.57
B	1000	1000	(66 66)	(933 66)	226.033	1566.15
B	1000	1000	(66 933)	(933 66)	306.716	1755.77
B	1000	1000	(66 933)	(933 933)	110.094	1474.9
B	1000	1000	(66 66)	(933 933)	139.192	1285.29
QuadTree						
A	1000	1000	(66 66)	(933 66)	0.0010966	1436.92
A	1000	1000	(66 933)	(933 66)	0.0016956	1609.66
A	1000	1000	(66 933)	(933 933)	0.0012958	1521.7
A	1000	1000	(66 66)	(933 933)	0.0016956	1348.95
B	1000	1000	(66 66)	(933 66)	0.0048868	1664.22
B	1000	1000	(66 933)	(933 66)	0.0037898	1872.15
B	1000	1000	(66 933)	(933 933)	0.0017959	1553.21
B	1000	1000	(66 66)	(933 933)	0.0020953	1345.29
QuadTree sa diskretiziranim granicama						
A	1000	1000	(66 66)	(933 66)	0.0068816	1410.65
A	1000	1000	(66 933)	(933 66)	0.0111701	1597.14
A	1000	1000	(66 933)	(933 933)	0.0056848	1476.12
A	1000	1000	(66 66)	(933 933)	0.0037899	1304.96
B	1000	1000	(66 66)	(933 66)	0.0164563	1651.14
B	1000	1000	(66 933)	(933 66)	0.018255	1858.14
B	1000	1000	(66 933)	(933 933)	0.0049824	1505.29
B	1000	1000	(66 66)	(933 933)	0.0050888	1307.48

Tabela 4.27. Vremena pretrage različitih struktura podataka

U tabeli 4.27. radi lakšeg poređenja rezultata dat je tabelarni prikaz vremena

pretrage i udaljenosti vizualiziranih na slikama 4.8., 4.9. i 4.10. Vrijeme pretrage je u nekim slučajevima (Mapa A (66,933) do (933,66)) čak i 200000 puta kraće. To ide na račun ukupne udaljenosti koja je kod *QuadTree-a* za 3% duža.

Kada je u pitanju *QuadTree* sa diskretiziranim granicama, rezultati su nešto bolji. Na istom primjeru iz prethodnog paragrafa je vrijeme pretrage za *QuadTree* sa diskretiziranim granicama čak 10 puta gore, međutim ukupna dužina puta je kraća. U svim primjerima je dužina puta sa diskretiziranim granicama manja. U principu se dužina puta kod *QuadTree-a* sa diskretiziranim granicama nalazi negdje na sredini između vremena pretrage matrične mreže i vremena pretrage običnog *QuadTree-a*.

Mrežasta mapa ima veliku manu, a to je da se putanja može kretati samo horizontalno, vertikalno ili dijagonalno pod uglom od $\pm 45^\circ$ u oba smjera. *QuadTree* sa diskretiziranim granicama nema ovu manu, što bi čak moglo značiti da je uz odabir dobre heurističke funkcije moguće dobiti put koji je kraći čak i od puta dobijenog pretraživanjem matrične mreže.

4.7. Zaključak

Sve u svemu, *QuadTree* kao struktura podataka pokazuje značajne rezultate. Kao struktura podataka, dovoljno dobro modelira prostor, a moguće ga je čak modelirati i bez greške ukoliko su širina i dužina stepen broja 2, a dubina dovoljna da širina celije bude 1. Pored toga što dobro i precizno modelira prostor zauzima značajno manje memorije od mrežaste mape.

QuadTree se kreira velikom brzinom, te je limitirajući faktor veličina mape, ali kao što je bilo vidljivo na tabelama 4.1. do 4.4., čak na mapama veličine 3000x3000 kreiranje *QuadTree-a* traje oko 250ms.

Pretraga *QuadTree-a* je za nekoliko redova veličine brža od pretrage matrične mreže, a dužina puta je veća za oko 3%, što je prihvatljiva cijena koju je dodatno moguće smanjiti na oko 1.5% diskretizacijom rubova čvora.

Autor vjeruje da je moguće da u specijalnim uslovima i prilagođavanjem heuristike uz pomoć diskretizacije granice put pronađen *QuadTree-om* bude čak i kraći od puta pronađenog matričnom mrežom što bi značilo da je jedini nedostatak ove metode nešto komplikovanija implementacija.

Poglavlje 5.

Zaključak

Završni rad "Planiranje kretanja zasnovano na A* algoritmu u OctoMap struktuiranim okruženjima" se bavi obradom i prikazom rezultata dobijenih primjenom A* algoritma za pronalaženje najkraćeg puta u okruženjima struktuiranim kao mrežasta mapa, quadtree i quadtree sa diskretiziranim granicama.

U uvodnom dijelu rada predstavljene su mrežasta mapa i quadtree kao strukture podataka te je skrenuta pažnja na njihove prednosti i nedostatke. Iako mrežasta mapa predstavlja prostor jako precizno, za to koristi mnogo resursa kao što je memorijski prostor. Quadtree predstavlja jako pogodnu strukturu podataka baziranu na stablima koja pruža niz pogodnosti, od vremenskih do prostornih na račun preciznosti (koja se u specijalnim uslovima čak ni ne gubi)

Obrađeni su i Dijkstrin i A* algoritam zajedno sa njihovim prednostima i nedostacima. Posebna je pažnja skrenuta na značaj heuristike kod A* algoritma te prilagođavanje ovog algoritma na takav način da funkcioniše sa quadtree-em i kasnije sa klasom *QuadTree*.

Rad kroz niz tabele prezentuje rezultate pretrage na dvije mape skalirane na različite veličine sa različitim početnim parametrima. Sa *QuadTree*-om je manipulirano mijenjanjem njegove dubine da bi se stekao bolji uvid u osobine ove strukture podataka.

Na kraju je predstavljen metod diskretizacije granica čvorova *QuadTree*-a i rezultati dobijeni na taj način. Moguće je da bi se diskretizacijom granica čvorova i odabirom različite heuristike mogli postići bolji rezultati nego oni dobijeni pretragom matrične mreže, što će biti predmet budućeg istraživanja.

Nažalost, zbog obima i vremenske ograničenosti, ovaj rad se nije bavio implementacijom i obradom rezultata octree strukture podataka i primjene A* algoritma na nju. To se ostavlja za proširenje i nastavak ovog rada.

Pored implementacije octomap strukture podataka, rad otvara put i drugim istraživanjima kao što su isprobavanje i implementacija različitih algoritama pretrage komšija quadtree i octree struktura podataka, modifikacija heuristike korištene za pretragu ili mijenjanje kompletног algoritma pretrage drugim.

Od velike koristi i interesa za ovaj rad bilo bi i njegovo proširenje praktičnim rezultatima dobijenim primjenom metoda pomenutih u ovom radu na pravim terenima i robotima te njihovom analizom sa osvrtom na upotrebljivost.

Sve u svemu, ukoliko se sagledaju svi rezultati, moguće je reći da je rad jako uspješan i da je doprinio gradnji hovih znanja i unapređenju već postojećih. A* algoritam se pokazao jako dobrim u traženju puta za kretanje robota kroz mape modeliranje pomoću quadtree strukture podataka.

Dodatak A: Preuzimanje koda, kompajliranje i pokretanje

Čitav kod opisan u ovom radu se može preuzeti sa Github-a sa narednog linka: <https://github.com/dbojadzic/QuadTree>. U repozitoriju se nalaze tri projekta - matrica, quadtree i quadtree sa diskretiziranim granicama. Da bi pokrenuo neki od projekata korisnik mora imati instaliran Codeblocks, Clion ili neki drugi IDE koji može da pokrene C++. Nakon toga se u main.cpp datoteci treba napisati main funkcija u kojoj korisnik zadaje visinu, širinu, mapu koja se čita i druge parametre. Primjer pokretanja programa dat je u nastavku.

```
1 int main() {
2     int width = 3000;
3     int height = 3000;
4     int* matrix = new int [width*height] {};
5     loadMatrix( "maps/A3000.txt" , matrix , width , height );
6     QuadTree dumir( matrix , width , height , 5 );
7     Astar astar(&dumir);
8     std :: vector<Point> tacke = astar.FindPath( Point(300, 300) , Point
9         (300, 2800));
10    delete [] matrix;
11 }
```

U ovom primjeru se čita iz fajla A3000.txt koji je dimenzija 3000x3000. Učitani niz se pretvara u *QuadTree* i onda se traži put od tačke (300,300) do (300,2800).

Bibliografija

- [1] “Dijkstras algorithm.” https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm. Accessed: 01-07-2018.
- [2] “Astar algorithm.” https://en.wikipedia.org/wiki/A*_search_algorithm. Accessed: 01-07-2018.
- [3] “Prexamp.png.” <https://opendsa-server.cs.vt.edu/ODSA/Books/CS3/html/PRquadtree.html>. Accessed: 02-07-2018.
- [4] M. Sortino, G. Totis, K. , and G. Cukor, “Simulation of cutting forces and cutting conditions in complex turning operations,” pp. 203–207, 01 2009.
- [5] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, “Octomap: An efficient probabilistic 3d mapping framework based on octrees,” *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 2013.
- [6] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.
- [7] M. J. De Smith, M. F. Goodchild, and P. Longley, *Geospatial analysis: a comprehensive guide to principles, techniques and software tools*. Troubador Publishing Ltd, 2007.
- [8] R. Dechter and J. Pearl, “Generalized best-first search strategies and the optimality of a,” *Journal of the ACM (JACM)*, vol. 32, no. 3, pp. 505–536, 1985.
- [9] X. Liu and D. Gong, “A comparative study of a-star algorithms for search and rescue in perfect maze,” in *Electric Information and Control Engineering (ICEICE), 2011 International Conference on*, pp. 24–27, IEEE, 2011.
- [10] I. Pohl, “First results on the effect of error in heuristic search,” *Machine Intelligence*, vol. 5, pp. 219–236, 1970.
- [11] I. Pohl, “The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving,” in *Proceedings of the 3rd international joint conference on Artificial intelligence*, pp. 12–17, Morgan Kaufmann Publishers Inc., 1973.
- [12] D. E. Knuth, “Dancing links,” *arXiv preprint cs/0011047*, 2000.

- [13] T. Balch, “Grid-based navigation for mobile robots,” *The Robotics Practitioner*, vol. 2, no. 1, pp. 6–11, 1996.
- [14] R. A. Finkel and J. L. Bentley, “Quad trees a data structure for retrieval on composite keys,” *Acta informatica*, vol. 4, no. 1, pp. 1–9, 1974.
- [15] “Quadtree data structure.” https://en.wikipedia.org/wiki/Quadtree#Pseudo_code. Accessed: 02-07-2018.
- [16] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.