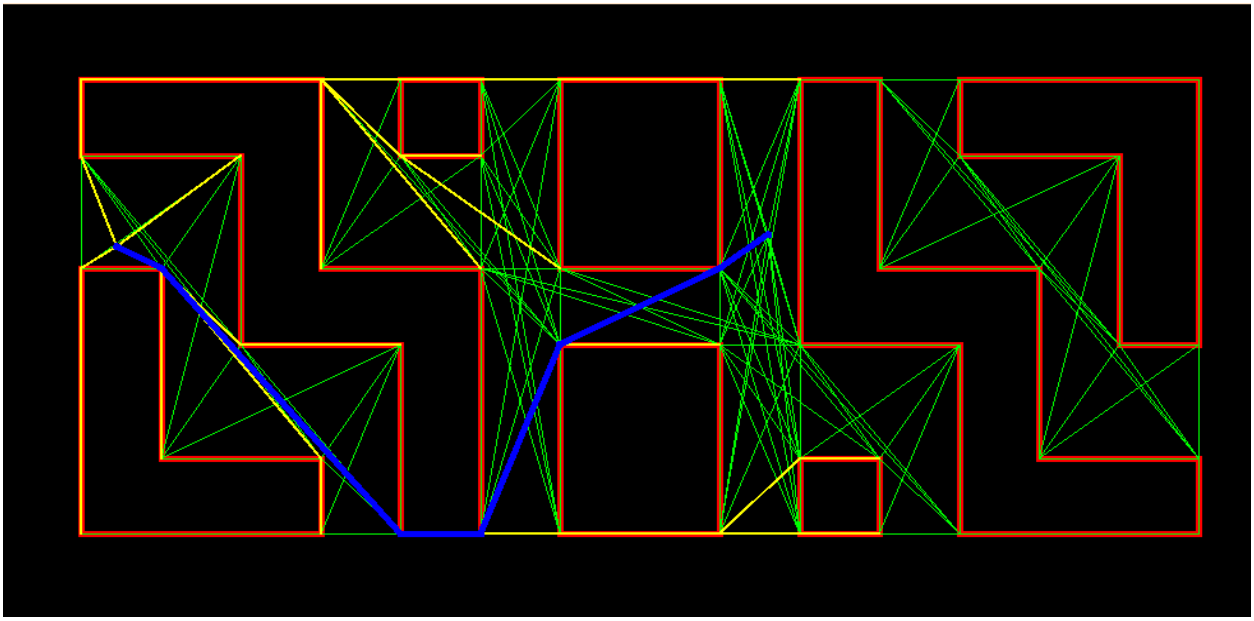
Geometrijski algoritmi @ MATF

Najkraći put između dve tačke sa obilaženjem prepreka za tačkastog robota.

Matei Jon Stanču



Opis problema

Dat je skup prepreka u ravni i dve tačke koje se nalaze u slobodnom prostoru. Potrebno je planirati kretanje za robota koji se kreće u posmatranoj ravni koje predstavlja najkraći put između datih tačaka tako da se na tom putu robot ne sudari ni sa jednom preprekom. Domen ovog problema se ograničava na robote tačkastog tipa dok prepreke predstavljaju otvorene skupove, što znaci da robot može da dodiruje ivice i da se kreće po ivicama prepreka. Ovo ograničenje bi predstavljalo problem u praksi, ali sa druge strane posmatrane prepreke bi mogle

pretprocesiranjem da se prošire za određeni parametar i time bi se izbeglo sudaranje robota sa preprekama.

Ulaz: *skup poligona koji predstavlja skup prepreka i dve tačke koje predstavljaju početni položaj i odredište robota.*

Izlaz: *najkraći put između početnog položaja i odredišta.*

Rešenje problema

Graf vidljivosti skupa poligona predstavlja graf čiji su čvorovi skup temena svih poligona a grane predstavljaju duži koje spajaju temena koja su vidljiva, pri čemu dve tačke su vidljive ako duž koju ih spaja ne preseca ni jedna stranica nekog poligona iz posmatranog skupa poligona. U slučaju da duž koja spaja dva temena u potpunosti pripada unutrašnjosti poligona tada ta dva temena nisu vidljiva.

Računanje najkraćeg puta u ravni sa obilaženjem prepreka sastoji se od dva potproblema. Prvi je računanje grafa vidljivosti za posmatrani skup prepreka pri čemu u slučaju problema najkraćeg puta, grafu vidljivosti su pridružena dodatna dva čvora koja predstavljaju početnu i krajnju tačku puta koji se računa. Drugi potproblem je računanje najkraćeg puta između početnog i završnog čvora u izračunatom grafu vidljivosti.

Naivno rešenje problema

Naivni pristup rešavanju ovog problema bi bio da se za svaki par čvorova **v** i **w** računa vidljivost tako što se proverava presek duži **vw** sa svakom ivicom iz skupa prepreka. Ukoliko presek ne postoji onda se u grafu vidljivosti dodaje grana izmedju cvorova v i w. Posto postoje n*(n-1)/2 = O(n^2) mogucih parova cvorova i n ivica vremenska slozenost ovog pristupa bila bi O(n^3).

Napredno rešenje problema

Napredni pristup rešavanju problema je zasnovan na brišućoj pravoj, ali ne na klasičnom pristupu brišuće prave. Zapravo radi se o brišućoj polupravoj koja se ne pomera vertikalno ili horizontalno već se rotira u smeru kazaljke na satu oko svog temena. Ideja je da se ne računa vidljivost za svaki par temena posebno već da se informacija o vidljivosti prethodnih čvorova iskoristi za računanje vidljivosti za tekući čvor.

Napredni algoritam za računanje grafa vidljivosti sastoji se od glavne petlje koja prolazi kroz niz čvorova grafa. U svakom koraku inicijalizuju se brišuća poluprava, struktura koja čuva tačke događaja i struktura koja čuva status preseka ivica prepreka sa brišućom polupravom koja se zove status. Brišuća poluprava se u svakom koraku inicializuje sa temenom u tekućem čvoru koji se obrađuje i horizontalnim položajem, zatim se inicijalizuju strukture podataka. Struktura tačaka događaja predstavlja balansirano stablo koje čuva događaje u sortiranom poretku u odnosu na ugao koji formiraju sa inicijalnim položajem brišuće poluprave, što znači da će brišuća poluprava obilaziti tačke događaja u smeru kazaljke na satu. Na kraju, incijalizacija statusa se vrši tako što brišuća prava obiđe jedan krug oko svog temena i ubacuje u status ili izbacuje iz statusa, u zavisnosti od tipa dogadjaja na koji je naišla, sve dok se ne vrati u svoj inicijalni položaj. Ovaj korak je neophodan jer u opštem slučaju ne može jednostavno da se odredi položaj brišuće prave takav da u tom položaju ne seče ni jednu ivicu prepreke.

Nakon inicijalizacije sledi obilazak tačaka događaja po redosledu definisanim redom događaja. U svakom događaju vrši se ispitivanje vidljivosti sa tačkom koja definiše tekući događaj i ažuriranje statusa. Ako je tekući čvor “vidi” tačku događaja onda se u grafu vidljivosti dodaje grana između ta dva čvora u suprotnom se ne dodaje. Status se ažurira tako što se ispituje da li je tačka koja se nalazi na drugom kraju ivice koja pripada tački događaja u smeru kazaljke na satu ili u suprotnom smeru kazaljke na satu. U slučaju da je drugi kraj ivice u smeru kazaljke na satu onda se ta ivica dodaje u status, a inače se briše iz statusa. Na kraju se tekići događaj briše iz reda događaja i postupak se ponavlja za tekući čvor grafa vidljivosti sve dok ima događaja u redu događaja. Glavna petlja se završava kada su obrađeni svi čvorovi grafa vidljivosti. Algoritam za računanje grafa vidljivosti prikazan pseudokodom:

Algoritam: GrafVidljivosti(**S**)
Ulaz: skup prepreka **S**
Izlaz: graf vidljivosti **G(S)**

```
1.  inicijalizuj  G(V,E) tako da je V skup svih tmena iz S a E je prazan skup
2.  forall v iz V do
3.      inicijalizuj brišuću polupravu i strukture podataka Q i T
4.      forall w iz Q do
5.          if Vidljiv(v, w) then
6.              dodaj granu između v i w u E
7.          ažuriraj T
```

Funkcija Vidljivi() računa vidljivost između dva čvora tako što proverava da li prva ivica iz statusa seče duž koju čine dva posmatrana čvora. U slučaju da seče onda ta dva čvora nisu vidljiva. Sledi pseudokod funkcije Vidljivi():

Algoritam: Vidljiv(**v**, **w**)
Ulaz: dva čvora **v** i **w**
Izlaz: tačno ako je **w** vidljiv iz **v**, netačno inače

```
1.  if T je prazan
2.      return tačno
3.  if duž vw seče unutrašnjost u okolini v then
4.      return netačno
5.  if u i w nisu kolinearni then                (u je prethodnik od w)
6.      if prva ivica iz statusa seče duž vw then
7.          return netačno
8.      else return tačno
9.  else if u nije vidljiv then
10.     return netačno
11. else if duž uv ne seče neka ivica iz T then
12.     return netačno
13. else return tačno
```

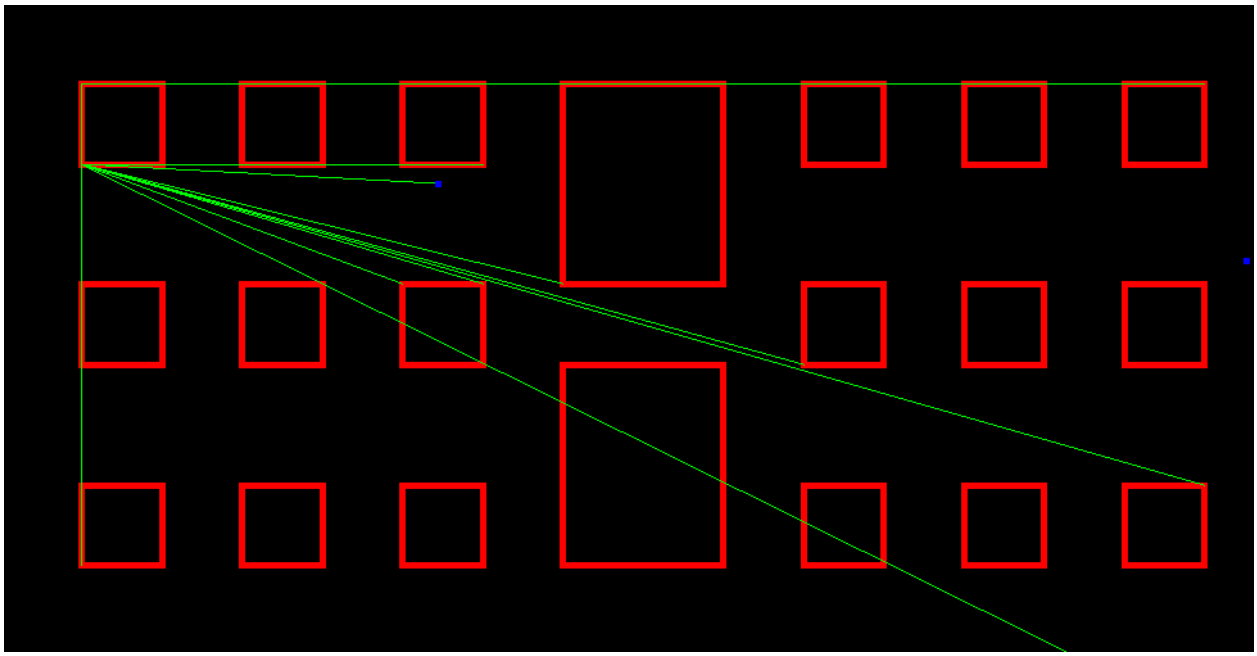
Nakon računanja grafa vidljivosti sledi pretraga najkraćeg puta između početne i krajnje tačke. Za pretragu najkraćeg puta u grafu implementirana su dva algoritma, prvi je A* pri čemu je za heuristiku izabrano Euklidsko rastojanje do odredišta, a drugi algoritam je specijalan slučaj algoritma A* kada je heuristika 0, poznatiji kao Dijkstrin algoritam. Oba algoritma se koriste za računanje najkraćeg puta između zadatog čvora i svih ostalih čvorova u grafu, pa su iz tog razloga algoritmi podešeni

da zaustave pretragu kada se naiđe na čvor koji predstavlja odredište. Nakon što se završi pretraga najkraći put može jednostavno da se konstruiše tako što se iz svakog čvora, počevši od čvora koji predstavlja odredište, pozove prethodnik sve dok se ne naiđe na početni čvor u grafu vidljivosti. Ovo je moguće jer svaki čvor sadrži informaciju o svom prethodniku (može da ima i više prethodnika ako najkraći put nije jedinstven).

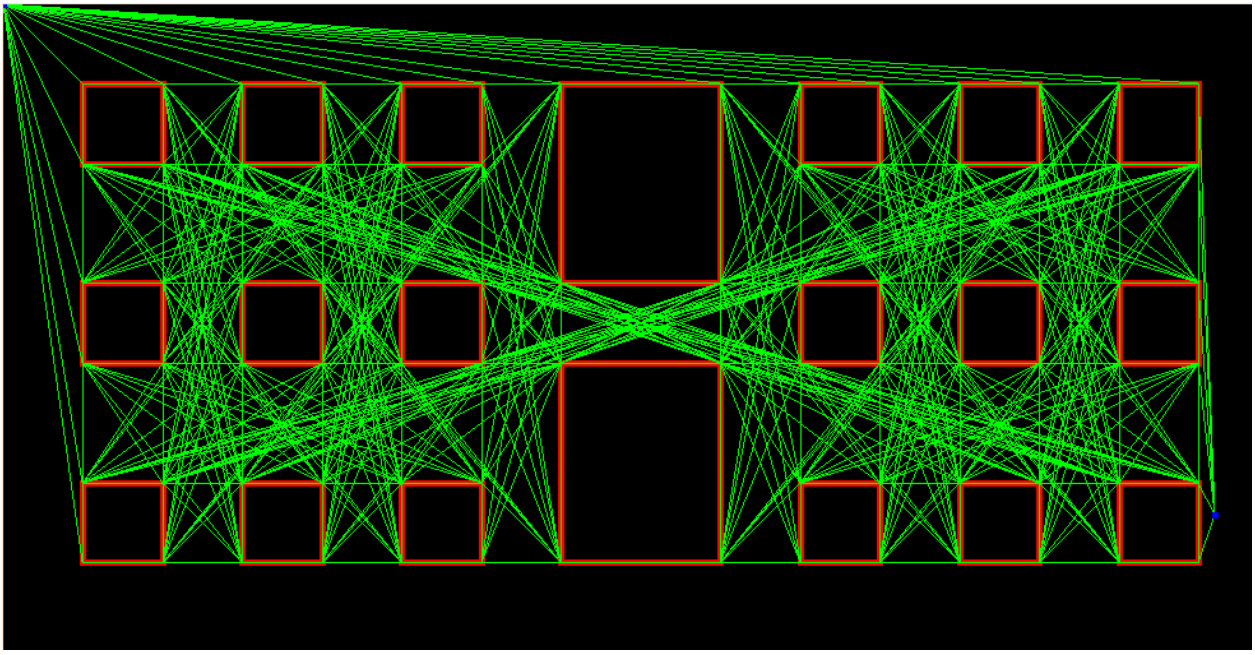
Vizuelizacija algoritma

U ovom delu biće reprezentovana vizuelizacija naprednog algoritma. Izdvojeno je nekoliko slika koje će predstaviti algoritam u ključnim trenucima. Na svakoj slici prepreke su predstavljene crvenom bojom, početna i krajnja tačka plavom, brišuća poluprava i grane grafa vidljivosti zelenom, pretraga je predstavlejna žutom bojom i na kraju najkraći put između početka i odredišta je predstavljen plavom bojom.

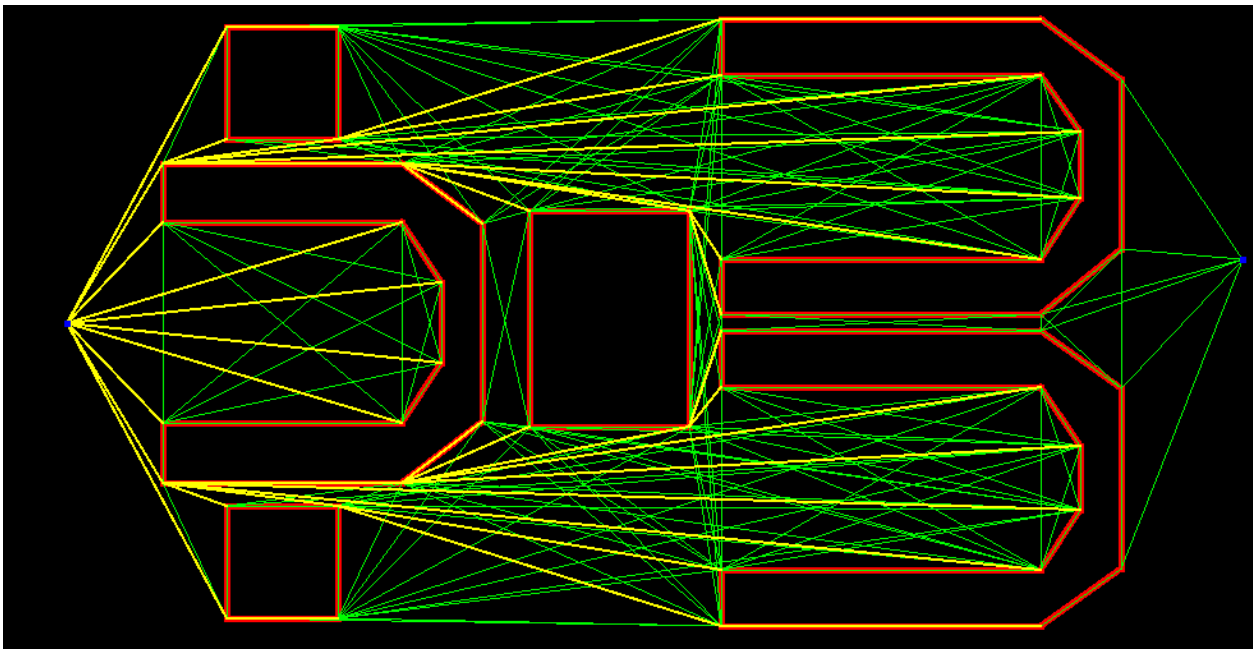
Na prvoj slici algoritam računa vidljivost iz donjeg levog temena prve prepreke do svih ostalih temena. Takođe se može videti da su u tom trenutku već izračunati neki vidlijivi čvorovi među kojima je i početna tačka, to su svi čvorovi horizontalno desno od čvora iz kog se računa vidlivost do trenutnog položaja brišuće prave čiji se beskonačni kraj nalazi u donjem desnom uglu slike.



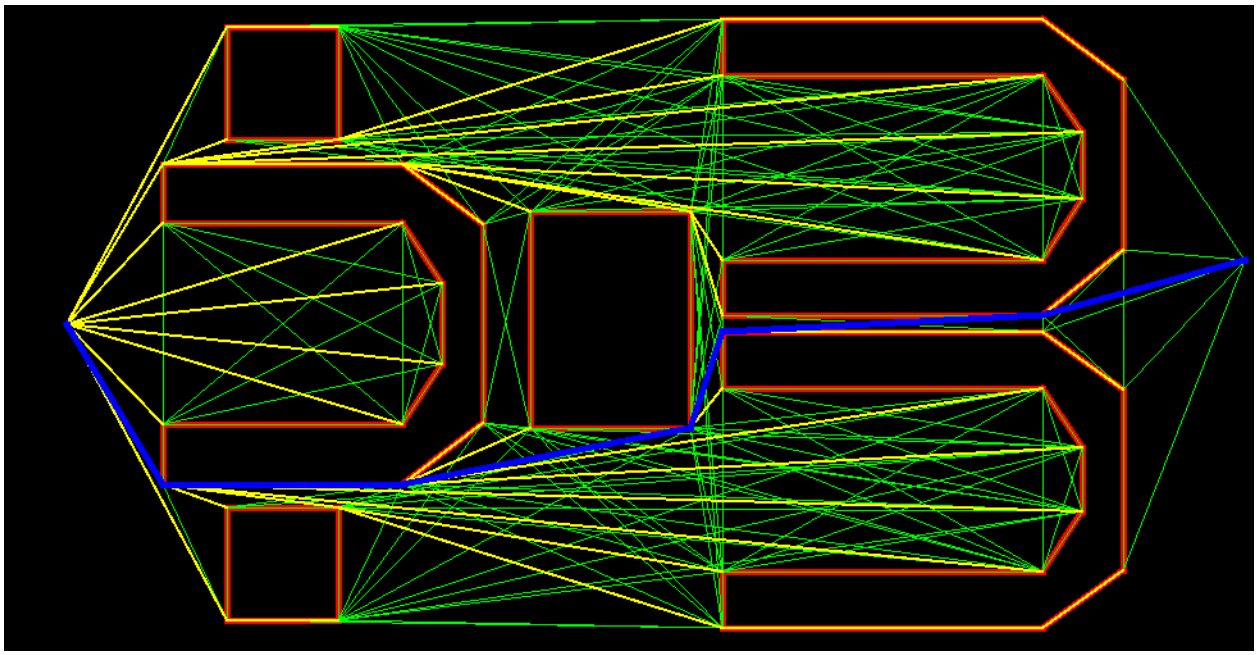
Na drugoj slici je završeno računanje grafa vidljivosti. Početni čvor je u gornjem levom uglu a završni u donjem desnom. Nakon ovog trenutka sledi pretraga najkraceg puta iz početnog čvora.



Na trećoj slici je predstavljena pretraga, početni čvor je na levom kraju slike završni je na desnom. Zuti deo grafa vidljivosti pokazuje dokle je stigla pretraga, preciznije to su parcijalni najkraći putevi do početnog čvora.



Na četvrtoj slici, pretraga se završava jer je problem najkraćeg puta između početne i krajnje tačke rešen, ostali putevi su za ovaj problem irelevantni.

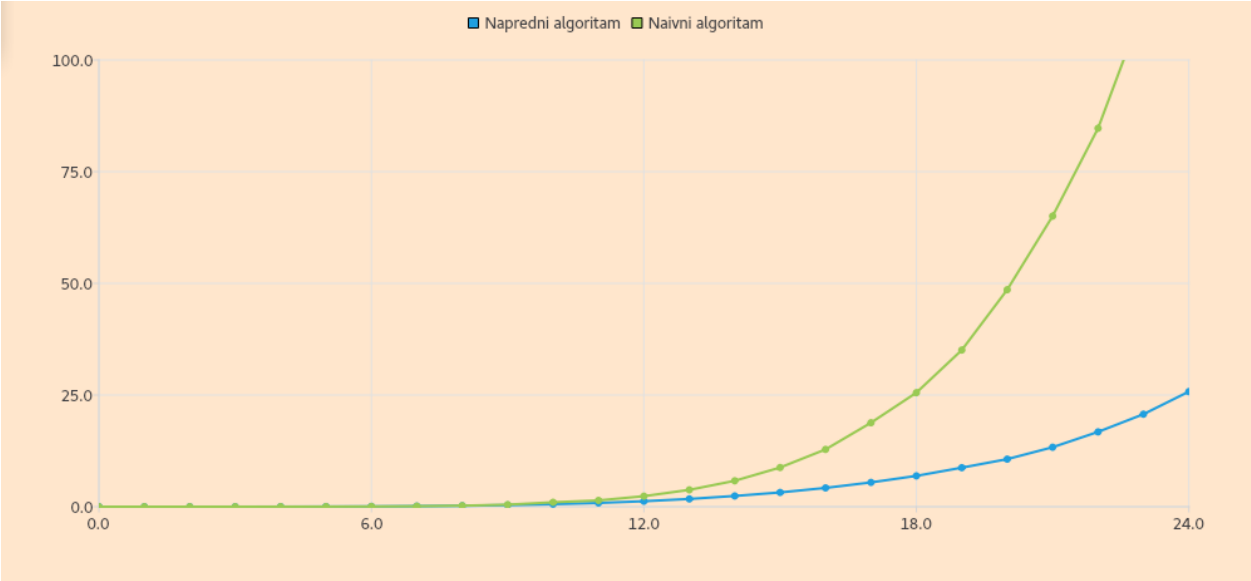


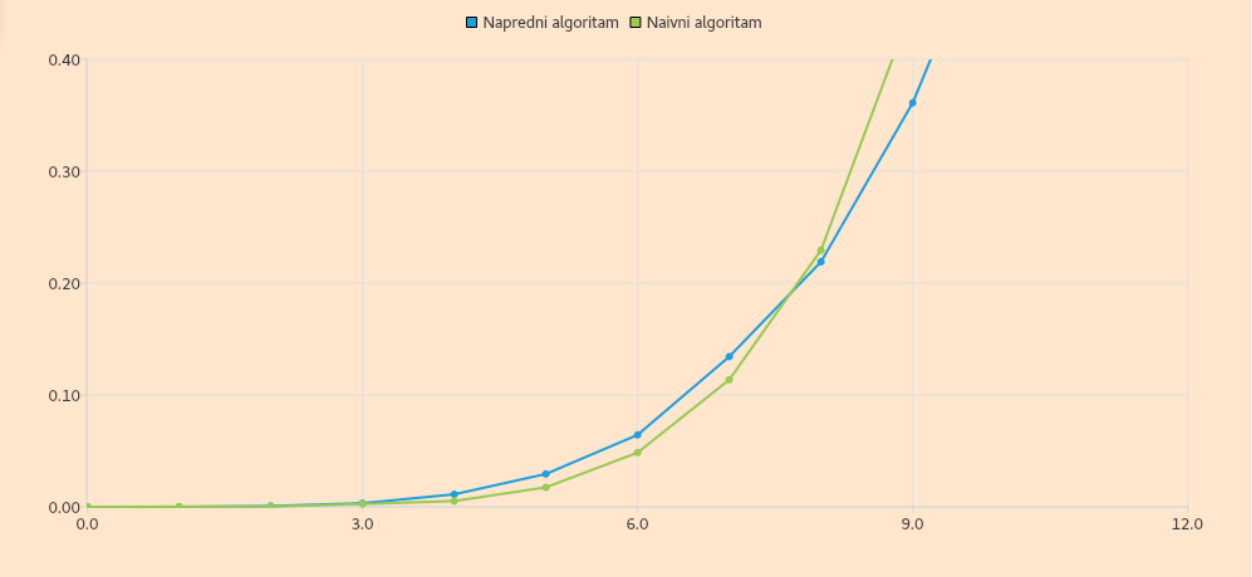
Poređenje efikasnosti naivnog i naprednog algoritma

Poređenje efikasnosti algoritama je vršena na instancama različitih dimenzija. Instance su kreirane pomoćnom funkcijom generateObstacleGrid() koja kao argument prima dimenziju instance a izlaz generiše u datoteci u OFF formatu. Izlaz predstavlja kvadratnu matricu prepreka pri čemu dimenzija matrice je zadata kao argument funkcije, a prepreke su sve u obliku kvadrata. Dakle, generisana matrica dimenzije n sadrži n^2 prepreka u obliku kvadrata što znaci da će dimenzija problema biti 4* n^2 za instancu generisanom argumentom koji ima vrednos n.

Upoređivanje preformansi se vršilo na instancama dimenzija između 1 i 24. Na grafiku se može videti superiornost naprednog algoritma u odnosu na naivni kada je u pitanju vremenska složenost, međutim napredni algoritam nije dominirao na celom domenu ispitivanja brzine izvršavanja, naime preuzeo je vođstvo tek nakon instance dimenzije 7.

U nastavku su prikazani grafici koji prikazuju asimptitsko ponašanje oba algoritma kao i tačku preokreta:





Testiranje ispravnosti algoritama

Prilikom testiranja algoritma ispitivano je više aspekata funkcionalnosti algoritama. Prvi aspekt testiranja predstavlja testiranje ispravnosti funkcije pointInsideObstacle() koja obezbeđuje robusnost prilikom učitavanja podataka. Funkcija kao argument prima tačku i u odnosu na prethodno učitan skup prepreka ispituje da li se ta tačka nalazi u unutrašnjosti neke od prepreka, u slučaju da jeste funkcija vraća tačno što prosleđuje poruku glavnom algoritmu koji na osnovu toga ispisuje da se početna ili krajnja tačka nalazi u zabranjenom prostoru i prekida izvršavanje, inače vraća netačno što glavnom algoritmu omogućava da nastavi da se izvršava.

U nastavku je data tabela koja sadrži detalje o svim testovima iz skupa testova ga15_pointInsideObstacleTests:

Naziv testa	Opis testa	Ulaz	Očekivani izlaz
insideObstacele	Funkciji se prosleđuje tačka koja se nalazi u unutrašnjosti neke prepreke.	Tačka u unutrašnjosti proizvoljne prepreke	Tačno
outsideObstacle	Funkciji se prosleđuje tačka koja se nalazi u slobodnom prostoru.	Tačka u slobodnom prostoru	Netačno
onObstacleEdge	Funkciji se prosleđuje tačka koja se nalazi na ivici prerpreke.	Tačka na ivici prepreke	Netačno
onObstacleVertex	Funkciji se prosleđuje tačka koja se nalazi na temenu prepreke.	Tačka na temenu prepreke	Netačno

Drugi skup testova se odnosi na ispitivanje postojanja njakraćeg puta. U ovom slučaju testirana je glavna komponeneta algoritma pri čemu testovi se odnose na pojedinačne instance ulaza. Svaka instanca predstavlja datoteku u OFF formatu koja sadrži definisane položaje prepreka, pri čemu za svaku instancu početna i krajnja tačka su inicijalizovane tako da postoji put između njih osim za instancu ga15_obstacles03.off za koju su tačke definisane tako da je prepreka svuda oko jedne od zadatih tačaka.

U nastanvku je prikazana tabela koja sadrži detalje o svim testovima iz skupa testova ga15_pathExistanceTests:

Naziv testa	Opis testa	Ulaz	Očekivani izlaz
shortestPathExistence00	Testiranje postojanja puta pri zadatom ulazu	(90, 330) , (550, 200) ga15_obstacles00.off	Dužina najkraćeg puta je veća od nule
shortestPathExistence01	Testiranje postojanja puta pri zadatom ulazu	(0, 0) , (950, 400) ga15_obstacles01.off	Dužina najkraćeg puta je veća od nule
shortestPathExistence02	Testiranje postojanja puta pri zadatom ulazu	(900, 200) , (600, 190) ga15_obstacles02.off	Dužina najkraćeg puta je veća od nule
ShortestPathExistence03	Testiranje postojanja puta pri zadatom ulazu	(400, 200) , (600, 200) ga15_obstacles03.off	Dužina najkraćeg puta je jednaka nuli
ShortestPathExistence04	Testiranje postojanja puta pri zadatom ulazu	(50, 250) , (970, 200) ga15_obstacles04.off	Dužina najkraćeg puta je veća od nule

Poslednji skup testova odnosi se na proveru poklapanja rezultata naprednog i naivnog algoritma za oba tipa pretrage, Dijkstra i A*, pri različitim instancama problema. Provera poklapanja rezultata je vršena samo pri instancama problema kod kojih postoji put između početnog položaja i odredišta.

U nastavku je prikazana tabela sa detaljima testova iz skupa testova ga15_pathsCompareTests:

Naziv testa	Opis testa	Ulaz	Očekivani izlaz
shortestPathLengthsCompare00	Provera da li su duzine izracunatih puteva jednake	(90, 330) , (550, 200) ga15_obstacles00.off	Poklapanje rezultata naivnog i naprednog algoritma
shortestPathLengthsCompare01	Provera da li su duzine izracunatih puteva jednake	(0, 0) , (950, 400) ga15_obstacles01.off	Poklapanje rezultata naivnog i naprednog algoritma
shortestPathLengthsCompare02	Provera da li su duzine izracunatih puteva jednake	(900, 200) , (600, 190) ga15_obstacles02.off	Poklapanje rezultata naivnog i naprednog algoritma
shortestPathLengthsCompare04	Provera da li su duzine izracunatih puteva jednake	(50, 250) , (970, 200) ga15_obstacles04.off	Poklapanje rezultata naivnog i naprednog algoritma

Zaključak

Predloženi napredni algoritam je vremenske složenosti O(n^2*logn) i otkrio ga je Lee. Složenost ovog algoritma potiče od računanja grafa vidljivosti, jer računanje najkraćeg puta u grafu je u najgorem slučaju O(n^2). Međutim, u ovoj implementaciji napredni i naivni algoritmi su iste vremenske složenosti, tj. O(n^3). To je zato što je u ovoj implementaciji upotrebljeno linearno brisanje iz statusa umesto logaritamskog. U svakom slučaju perforamnse naprednog algoritma su daleko bolje nego performanse naivnog, što se i jasno može videti u sekciji upoređivanja performansi.

Postoje i napredniji algoritmi za računanje grafa vidljivosti, neki od njih su Welzl i Overmars-ov algoritam složenosti O(n^2), kao i najefikasniji Gosh i Mount-ov izlazno zavisni algoritam složenosti O(nlogn + k), gde je k broj grana u grafu vidljivosti, ovaj algoritam predstavlja optimalan algoritam za računanje grafa vidljivosti.

Da bi se spustila teorijska složenost celom algoritmu do O(nlogn + k) potrebno je u implementaciji pretrage koristiti Fibonačijeva stabla jer jedino tako može da se spusti složenost pretrage do O(nlogn + k). Kombinacija Fibonačijevih stabala sa navedenim optimalnim algoritmom za računanje grafa vidljivosti je dobra kada se slobodan prostor sastoji od uskih hodnika i kada graf vidljivosti nije gust.

Optimalan algoritam za računanje najkraćeg puta sa obilaženjem prepreka zasnovan je na triangulaciji slobodnog prostora i predložili su ga Hershberger i Suri i on je optimalne O(nlogn) vremenske složenosti.

Što se tiče trodimenzionalne verzije ovog problema, za nju je dokazano da je NP-teška. Problem nastaje kada je potrebno konstruisati graf vidljivosti, jer grane grafa se mogu nalaziti bilo gde na ivici prepreke, pa prema tome ne postoji jednostavan način da se problem diskretizuje. Sa druge strane, postoje algoritmi koji nalaze aproksimaciju najkraćeg puta u trodimenzionalnom prostoru u polinomijalnom vremenu.