Generator slučajnog lavirinta

Matija Lojović, 1017/2023

13. april 2024.

Sažetak

U ovom radu biće opisan problem generisanja slučajnog lavirinta, njegove primene, kao i osam različitih algoritama koji ovaj problem rešavaju. Za svaki od algoritama biće navedene njegove karakteristike, složenost, težina rešavanja generisanog lavirinta, kao i detalji implementacije. Konačno, biće predstavljena uporedna analiza rezultata dobijenih merenjem vremena izvršavanja razmatranih algoritama.

Sadržaj

1 Uvod

Lavirint predstavlja mrežu puteva u kojoj postoje početna i krajnja tačka i makar jedan put između njih. Kako bi se lavirint rešio, potrebno je, krenuvši iz početne tačke i krećući se putevima lavirinta, stići do krajnje tačke. Težina ovog zadatka zavisi od načina konstrukcije lavirinta. U tu svrhu, osmišljen je veliki broj algoritama različitih karakteristika. Njihova primena može se naći u oblastima poput video igara, obrazovanja, robotike, arhitekture, psihologije i slično.

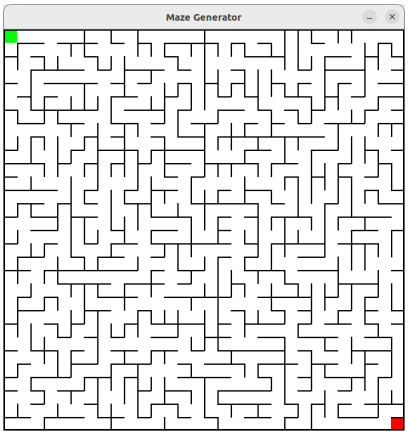
U mnogim video igrama nivoi su u osnovi lavirinti kroz koje igrač prolazi i traži izlaz. Kako bi igračima ovaj aspekt bio interesantan, potrebno je svaki put generisati novi lavirint i samim tim novi izazov. U obrazovanju se lavirinti često koriste za vežbanje veštine rešavanja problema, još od obdaništa gde deca rešavaju jenostavne lavirinte, pa sve do visokih stepena obrazovanja gde mogu koristiti da ilustruju koncepte u programiranju. U robotici lavirinti se često koriste za obuku i testiranje performansi algoritama za traženje puta koji se implementiraju u robotima. Lavirinti takođe mogu biti i ukrasi u vrtovima, pa je u tom slučaju posao arhitekte da projektuje izazovan lavirint za goste. Još jedna primena je u psihologiji, gde se lavirinti koriste za istraživanje učenja i procesa memorije, pogotovu kod životinja.

Ovaj rad ima za cilj da predstavi neke od najpoznatijih algoritama za generisanje nasumičnog lavirinta, njihove prednosti i mane, kao i izgled i težinu rešavanja lavirinta generisanog njima. Takođe, u radu će biti analizirana vremenska složenost pomenutih algoritama za različite dimenzije lavirinta.

2 Algoritmi za konstrukciju slučajnog lavirinta

Problem genrisanja slučajnog lavirinta podrazumeva zadovoljavanje sledećih uslova:

- Lavirint treba da ima jedinstven put od početnog do krajnjeg polja
- Lavirint ne sme da sadrži cikluse
- Ne sme biti delova lavirinta do kojih se ne može doći
- Algoritam koji rešava ovaj problem ne sme uvek generisati isti lavirint



Slika 1: Primer lavirinta

Na slici iznad dat je primer jednog lavirinta koji zadovoljava gorepomenute uslove. Uočavamo da je sačinjen od pravougaone mreže *polja*, a da između svaka dva polja mogu, a ne moraju postojati *zidovi*. Ukoliko između dva polja ne postoji zid, kažemo da između njih postoji *prolaz*. Početno polje generisanog lavirinta će u svakom algoritmu biti gornje levo, a krajnje polje donje desno.

Algoritmi iz ovog rada pristupaju problemu generisanja slučajnog lavirinta na različite načine, ali svi zadovoljavaju gorenavedene uslove.

2.1 Algoritmi zasnovani na teoriji grafova

Problem generisanja slučajnog lavirinta tesno je povezan sa teorijom grafova. Naime, ukoliko posmatramo polja lavirinta kao čvorove, a prolaze u lavirintu kao grane, problem rešavanja lavirinta se svodi na problem traženja puta od čvora koji odgovara početnoj tački lavirinta do čvora koji odgovara krajnjoj. Algoritmi za generisanje lavirinta iz ovog odeljka se upravo koriste poznatim algoritmima iz teorije grafova kako bi osigurali da takav put postoji i da je jedinstven.

2.1.1 Pretraga u dubinu

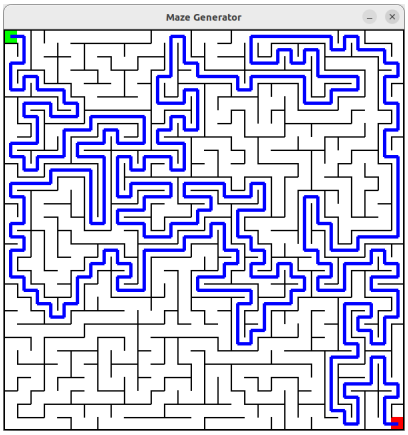
Algoritam pretrage u dubinu (eng. DFS) obilazi sve čvorove grafa i pritom redosledom obilaska implicitno generiše podgraf polaznog grafa koji je sastavljen od grana kojima se stiže u do tog trenutka neposećene čvorove (tzv. DFS drvo). Kako u drvetu nema ciklusa, dovoljno je po-

krenuti DFS iz početnog čvora lavirinta i implicitno će se generisati jedinstven put od početnog do krajnjeg čvora lavirinta. Bitno je napomenuti da čvorove početnog grafa čine polja, a grane svi mogući zidovi lavirinta datih dimenzija.

Implementacija DFS-a u ovom radu je rekurzivna - funkcija će za prosleđeni čvor najpre označiti da je posećen, a zatim i pozvati sebe rekurzivno za svakog od neposećenih suseda datog čvora, u nasumičnom poretku. Kad god se funkcija pozove za nekog neposećenog suseda, "rušimo" zid preko kog se prelazi i time gradimo DFS stablo, odnosno lavirint.

Složenost obilaska grafa je $O(|V| + |E|)$, a kako u ovom slučaju imamo $M \cdot N$ ćelija i $2 \cdot M \cdot N + M + N$ grana, ukupna složenost je $O(M \cdot N)$, gde su M i N dimenzije lavirinta.

Lavirinti generisani na ovaj način često imaju dugačke puteve bez mnogo grananja, pa im težina rešavanja nije velika.



Slika 2: DFS lavirint

2.1.2 Primov algoritam

Primov algoritam se koristi za traženje *minimalnog razapinjućeg stabla* (eng. MST) neusmerenog težinskog grafa, tj. acikličnog podgrafa minimalne ukupne težine koji povezuje sve čvorove polaznog grafa. Ukoliko polja lavirinta predstavimo kao čvorove, a zidove kao grane polaznog grafa, nalazeći njegov MST dobijamo stablo koje povezuje sve čvorove grafa, pa samim tim i početni i krajnji.

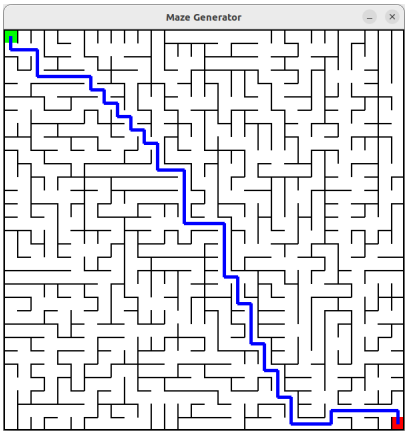
Koraci u implementaciji su sledeći:

1. Početi sa lavirintom u kom postoji zid između svaka dva polja.
2. Izabrati polje i obeležiti ga kao posećeno. Dodati zidove susedne tom polju u listu zidova.
3. Dok postoje zidovi u listi:
   (a) Izabrati nasumično zid iz liste.
   (b) Ako je tačno jedno njemu susedno polje posećeno:
      i. Napraviti prolaz na mestu tog zida i postaviti i drugo susedno polje na posećeno.
      ii. Dodati zidove neposećenog polja u listu zidova.

Primećujemo da, za razliku od klasičnog Primovog algoritma, u svakom koraku uzimamo nasumičnu granu, a ne najkraću, pa samim tim nema potrebe za korišćenjem reda sa prioritetom, već je dovoljna obična lista.

Složenost algoritma leži u tome što se svi zidovi u nekom trenutku dodaju u listu što je vremenske složenosti $O(M \cdot N)$, a takođe svaki zid treba i obrisati iz liste, što zahteva linearno vreme po broju zidova u listi u najgorem slučaju. Ukupna složenost je, dakle, $O((M \cdot N)^2)$, gde su M i N dimenzije lavirinta.

Što se tiče težine rešavanja, dobijeni lavirinti imaju dosta račvanja, ali su često pogrešni putevi kratki, pa se može reći da je težina rešavanja srednja.



Slika 3: Primov lavirint

2.1.3 Kruskalov algoritam

Kao i Primov algoritam, Kruskalov algoritam služi za nalaženje minimalnog razapinjućeg stabla grafa. Ideja za generisanje lavirinta ostaje ista kao u Primovom algoritmu - generisanjem MST-a generišemo i put od početnog do krajnjeg čvora. Za razliku od Primovog algoritma, Kruskalov algoritam osim liste zidova koristi i skupove.

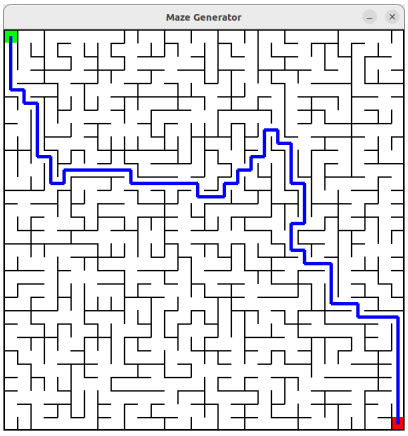
Koraci implementacije su sledeći:

1. Početi sa listom svih mogućih zidova i sa skupovima takvim da svaki skup sadrži po jedno polje lavirinta.
2. Iterirati kroz niz zidova u nasumičnom redosledu i pritom pamtiti broj dodatih grana:
   (a) Ukoliko je broj dodatih grana manji za jedan od ukupnog broja polja, završiti sa izvršavanjem
   (b) Ako polja susedna tekućem zidu pripadaju različitim skupovima:
      i. Napraviti prolaz na mestu tog zida.
      ii. Spojiti skupove u kojima se nalaze susedna polja.

Na osnovu implementacije vidimo da su operacije koje izvršavamo nad skupovima provera kom skupu pripada dato polje i spajanje skupova, odnosno unija. Iz tog razloga, struktura koja se nameće za korišćenje u ovom slučaju je *union-find*.

Složenost algoritma zavisi od načina implementacije union-find strukture, pošto se u svakoj iteraciji izvršava neka od operacija nad tom strukturom. Verzija koja je implementirana u radu ima amortizovanu vremensku složenost $O(\alpha(V))$, pri čemu je vrednost $\alpha(V) < 5$ za svaku praktičnu veličinu lavirinta, pa složenost operacija možemo smatrati konstantnom. Dakle, ukupna složenost se svodi na iteraciju kroz niz zidova i iznosi $O(M \cdot N)$, gde su M i N dimenzije lavirinta.

Što se tiče težine rešavanja i izgleda lavirinta, oni su slični kao u Primovom algoritmu.



Slika 4: Kruskalov lavirint

2.2 Algoritam rekurzivne podele

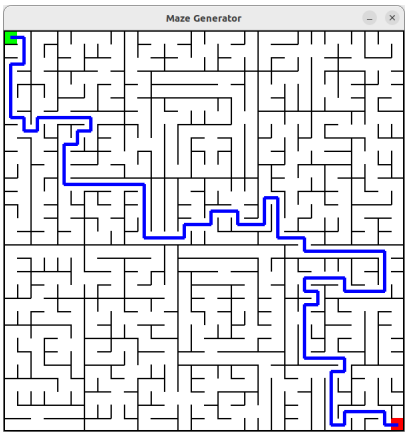
Ideja ovog algoritma je sledeća - ako prazan lavirint (bez zidova) podelimo proizvoljnom vertikalnom linijom i dodamo jedan prolaz na proizvoljnom mestu u njoj, svešćemo početni problem na dva problema manje dimenzije. Bitno je napomenuti da nakon svake podele menjamo orijentaciju zida koji dodajemo, iz vertikalne u horizontalnu i obrnuto, kako rešavanje ne bi bilo trivijalno prolaženje kroz sve prolaze redom. Primetimo da ovakvim dodavanjem zidova ceo lavirint ostaje povezan i acikličan - za svaki zid koji dodamo postoji tačno jedan način da dođemo sa jedne njegove strane na drugu.

Koraci u implementaciji su sledeći:

1. Početi sa praznim lavirintom.
2. Podeliti na proizvoljnom mestu lavirint horizontalnom ili vertikalnom linijom i dodati na nasumičnom mestu na liniji prolaz.
3. Ponoviti korak 2 rekurzivno za oba dela dobijena podelom. Ukoliko bilo koji od dobijenih potproblema ima neku od dimenzija manju od 2, izaći iz rekurzije.

Složenost najgoreg slučaja je $O(M \cdot N)$ jer se može desiti da podelu uvek vršimo tako da nam sa neke strane ostane samo jedan red ili kolona. Ipak složenost je u proseku $O(\log(M \cdot N))$, gde su M i N dimenzije lavirinta.

Izgledom dobijenog lavirinta dominiraju dugi hodnici, pa težina rešavanja nije visoka - često je dovoljno pratiti hodinke do prvog izlaza. Za ovo su zaslužni prvih par koraka u izvršavanju algoritma koji prave dugačke podele, a one često rezultuju dugačkim hodnicima u krajnjem lavirintu.



Slika 5: Lavirint dobijen algoritmom rekurzivne podele

2.3 Jednostavni algoritmi

Algoritmi iz ovog odeljka zasnivaju se na jednostavnim strategijama pravljenja putanja. Jednostavni su za implementaciju i efikasni, ali mogu dati lavirinte koji su laki za rešavanje.

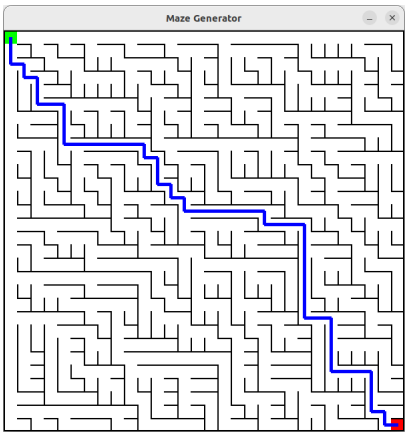
2.3.1 Algoritam binarnog drveta

Ovaj algoritam se služi činjenicom da je dovoljno za svako polje napraviti prolaz ka polju levo ili iznad njega kako bi se dobio validan lavirint. Počinjemo sa lavirintom punim zidova. Prolazimo redom kroz polja lavirinta i za svako polje, osim polja prve vrste i kolone, na nasumičan način biramo da li ćemo dodati prolaz levo ili gore. Ukoliko smo u prvoj koloni, dodajemo prolaze isključivo gore, a ukoliko smo u prvom redu, dodajemo ih isključivo levo. Početno polje preskačemo.

Složenost algoritma je $O(M \cdot N)$, gde su M i N dimenzije lavirinta, jer svako polje prolazimo tačno jednom.

Ime algoritma potiče od toga što lavirinti izgledom podsećaju na binarno stablo sa korenom u početnom čvoru. Rešavanje je trivijalno

ako imamo pregled celog lavirinta - ako krenemo od krajnjeg polja i uvek idemo levo ili gore gde god možemo, pratimo upravo jedinstven put do početnog polja.



Slika 6: Lavirint binarnog drveta

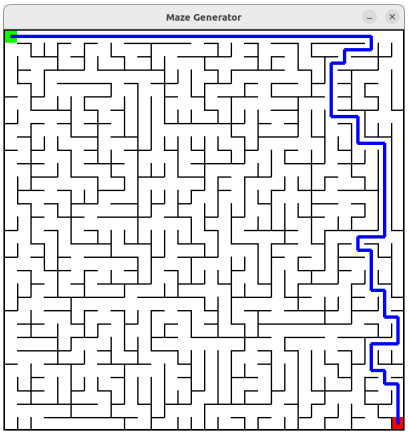
2.3.2 Algoritam zvečarke

Ovaj algoritam takođe kreće od lavirinta punog zidova, a funkcioniše tako što u prvom koraku "prokrči" ceo prvi red, a zatim u svakom sledećem redu pravi horizontalne hodnike nasumične dužine koji imaju tačno jedan prolaz ka gore i ka dole.

Ime je dobio po tome što kroz polja prolazimo red po red, nalik na kretanje zvečarke. U svakom koraku proveravamo da li treba da završimo horizontalni hodnik, što se svodi najpre na proveru da li smo došli do desnog kraja lavirinta, a zatim na biranje slučajne vrednosti tipa *bool*. Ukoliko je u pitanju vrednost *tačno*, zatvaramo hodnik, a u suprotnom nastavljamo da ga gradimo. Kada zatvorimo hodnik, dodajemo prolaze gore i dole na nasumična mesta u njemu.

Složenost algoritma je takođe $O(M \cdot N)$, gde su M i N dimenzije lavirinta, jer svako polje prolazimo tačno jednom.

Lavirinti dobijeni ovim algoritmom imaju dosta heterogeniji izgled od onih dobijenih algoritmom binarnog stabla, ali su takođe trivijalni za rešavanje polaskom iz krajnjeg polja.



Slika 7: Lavirint dobijen algoritmom zvečarke

2.4 Algoritmi zasnovani na slučajnoj šetnji

Naredna dva algoritma zasnovani su na konceptu *slučajne šetnje*. Ideja je da se na nasumičan način krećemo kroz ćelije, sve dok ne posetimo sve ćelije barem jedanput. Zbog potpuno nasumičnog načina generisanja, lavirinti dobijeni na ovaj način umeju da budu teški za rešavanje.

2.4.1 Aldous-Broder algoritam

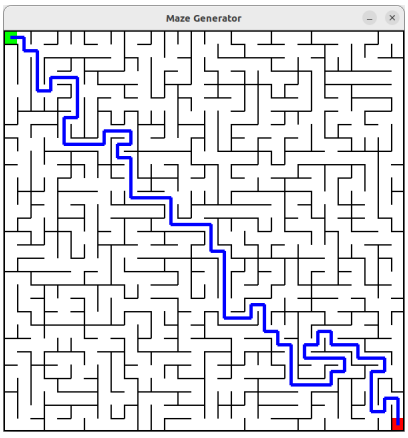
Ovaj algoritam poznat je po svojoj neefikasnosti, ali i lavirintima lepog izgleda i velike težine rešavanja.

Koraci algoritma su sledeći:

1. Početi sa lavirintom punim zidova i izabrati nasumičnu početnu ćeliju. Označiti je kao posećenu.
2. Dok ima neposećenih ćelija u lavirintu:
   (a) Izabrati nasumičnog suseda trenutne ćelije.
   (b) Ako taj sused nije posećen:
      i. Dodati prolaz ka datom susedu
      ii. Označiti ga kao posećenog
   (c) Postaviti trenutnu ćeliju na tog suseda.

Složenost se može aproksimirati sa $O((M \cdot N)^2)$, gde su M i N dimenzije lavirinta, ali može da varira značajno između dva pokretanja zbog prirode slučajnih šetnji.

Dobijeni lavirinti su vrlo heterogeni i prilično teški za rešavanje, pošto ne postoji jasno pravilo po kome se dati lavirint konstruiše.



Slika 8: Aldous-Broder lavirint

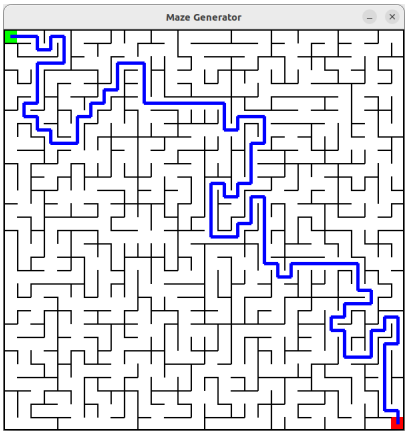
2.4.2 Wilsonov algoritam

Ovo je algoritam za koji se smatra da ima najbolji balans efikasnosti i težine rešavanja lavirinta. Takođe je zasnovan na slučajnim šetnjama, ali ima značajne modifikacije u odnosu na Aldous-Broder algoritam.

Algoritam teče na sledeći način: Biramo početnu ćeliju u lavirintu nasumično. Nakon toga biramo još jednu ćeliju, takođe nasumično, i iz nje pokrećemo slučajnu šetnju kroz lavirint. Ukoliko u bilo kom trenutku tokom šetnje naiđemo na polje koje se već nalazi u trenutnoj šetnji, odnosno zatvara ciklus, brišemo ceo ciklus iz šetnje. Ukoliko tokom šetnje naiđemo na polje koje je već u lavirintu, dodajemo celu šetnju u lavirint i pokrećemo ponovo šetnju iz novog nasumičnog polja koje nije već u lavirintu.

Složenost može da varira značajno od pokretanja do pokretanja, ali može se aproksimirati sa $O((M \cdot N) \cdot log(M \cdot N))$, gde su M i N dimenzije lavirinta, što je nešto efikasnije od Aldous-Broder algoritma.

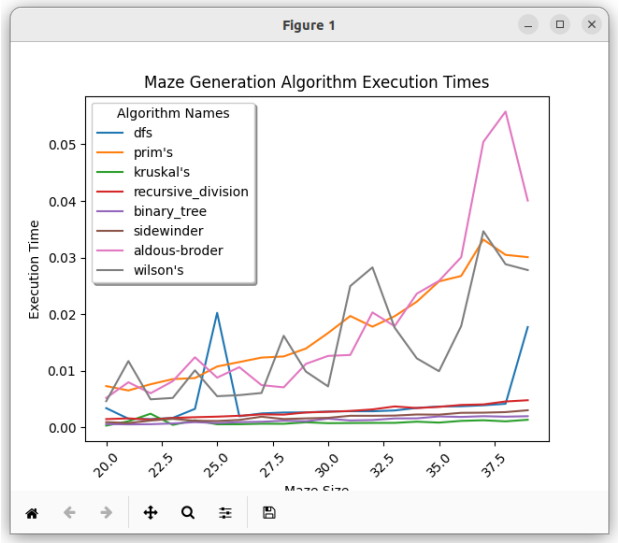
Dobijeni lavirinti su, kao i kod Aldous-Broder algoritma, vrlo heterogeni i teški za rešavanje.



Slika 9: Wilsonov lavirint

3 Poređenje vremena izvršavanja

Na sledećoj slici dat je grafik koji prikazuje vremena izvršavanja različitih algoritama za kvadratne lavirinte čije se dužine stranica kreću od 20 do 40.



Slika 10: Uporedna analiza vremena izvršavanja algoritama

Možemo primetiti da se izdavajaju dve grupe, jedna koju čine manje efikasni algoritmi (Aldous-Broder, Prim, Wilson) i drugu koji čine

ostali efikasniji algoritmi. Ovo odgovara analizi asimptotske složenosti koju smo sproveli - svi efikasni algoritmi imaju linearnu složenost po veličini lavirinta, dok kod neefikasnih ona ide sve do kvadratne.

Vidimo i da je Aldous-Broder očekivano neefikasniji od Wilsonovog algoritma, pogotovu za veće ulaze. S obzirom na to da ova dva algoritma generišu najteže lavirinte, ukoliko je to cilj, bolje je koristiti Wilsonov algoritam što se slaže sa našom pretpostavkom iz analize.

4 Zaključak

U ovom izveštaju formulisali smo problem generisanja slučajnog lavirinta, koje su njegove primene, kao i koji su neki od najpopularnijih algoritama korišćenih za njegovo rešavanje. Razmotrili smo osnovne algoritme za rešavanje ovog problema i analizirali prednosti i mane svakog od algoritama. Na osnovu analize složenosti, kao i simulacije izvršavanja, zaključili smo da postoje dve grupacije algoritama po složenosti, a da je u slučaju da je potrebno generisati težak lavirint najbolje koristiti Wilsonov algoritam, dok inače ima smisla koristiti neki od efikasnijih algoritama. Kako je ovaj problem značajan za razna polja nauke, ostaje prostor i potreba za daljim unapređenjima i novim algoritmima.