Generator slučajnog lavirinta

Matija Lojović, 1017/2023

9. 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. Na kraju, biće predstavljena uporedna analiza rezultata dobijenih merenjem vremena konstrukcije za svaki od 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 lavirnita, stići do kranje tačke. Težina ovog zadatka zavisi od načina konstrukcije lavirinta. U tu svrhu, smišljen je veliki broj algoritama različitih karakteristika. Njihova primena može se naći u oblastima poput video igara, edukacije, robotike, arhitekture, psihologije i slično.

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

2 Algoritmi

2.1 Algoritmi zasnovani na teoriji grafova

Problem generisanja slučajnih lavirinata tesno je povezan sa teorijom grafova. Naime, ukoliko posmatramo ćelije lavirinta kao čvorove, a prolaze u lavirintu kao grane, problem rešavanja lavirinta se svodi na problem traženja puta od čvora koji odgovara početnoj tački do čvora koji odgovara krajnjoj. Algoritmi za generisanje lavirinata 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 novi graf koji je zapravo drvo (tzv. DFS drvo). Kako u drvetu nema ciklusa, dovoljno je pokrenuti 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 prosledjeni 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 $M * N$ ćelija i $2 * M * N + M + N$ grana, ukupna složenost je $O(M * N)$, 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.

2.1.2 Primov algoritam

Primov algoritam se koristi za traženje minimalnog razapinjućeg stabla (eng. MST) grafa, tj. acikličnog podgrafa 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 punim zidova.
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. 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.

Složenost algoritma leži u tome što se svi zidovi u nekom trenutku dodaju u listu što je $O(M * N)$, a takođe svaki zid treba i obrisati iz liste, što zahteva linearno vreme u najgorem slučaju. Ukupna složenost je, dakle, $O((M * N)^2)$, 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.

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. Iz ovog razloga se može efikasno implementirati pomoću strukture union-find.

Koraci implementacije su sledeći:

1. Početi sa listom svih mogućih zidova i sa skupovima takvim da svaki skup sadrži po jedno polje lavirinta.
2. Za svaki od zidova, u nasumičnom redosledu:
   (a) Ako polja susedna tom zidu pripadaju različitim skupovima:
      i. Napraviti prolaz na mestu tog zida.
      ii. Spojiti skupove u kojima se nalaze susedna polja.

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 $O(M * N)$, gde su M i N dimenzije lavirinta.

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

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 - svaki zid koji dodamo ima 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 povlačenjem linije. Ukoliko bilo koji od delova ima neku od dimenzija manju od 2, izaći iz rekurzije.

Složenost najgoreg slučaja je $O(M*N)$ 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 $O(log(M*N))$, gde su M i N dimenzije lavirinta.

Izgledom dobijenih lavirinata dominiraju dugi hodnici, pa težina rešavanja nije visoka - često je dovoljno pratiti hodinke do prvog izlaza.

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 od njega levo ili gore kako bi se dobio validan lavirint. Počinjemo sa lavirintom punim zidova. Prolazimo redom kroz polja lavirinta i za svako 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 $O(M*N)$, 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.

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, bilo zato što smo došli do desnog kraja lavirinta ili izborom slučajne vrednosti tačno ili netačno. Kada zatvorimo hodnik, dodajemo prolaze gore i dole na nasumična mesta u hodinku.

Složenost algoritma je takođe $O(M * N)$, 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 binatrnog stabla, ali su takođe trivijalni za rešavanje polaskom iz krajnjeg polja.

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 $O((M * N)^2)$, 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.

2.4.2 Wilsonov algoritam

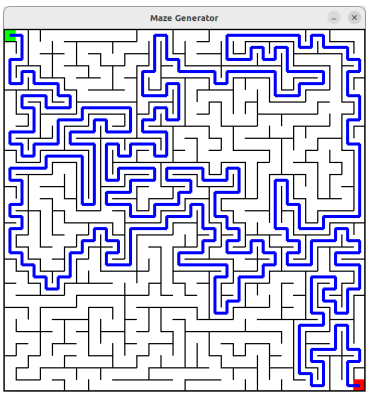
Ovo je algoritam za koji se smatra da ima najbolji balans efikasnosti i težine rešavanja lavirinta. Takođe je zasnovan na slučajnim šetnjama, ali ima značajne modifikacije u odnosu na Aldous-Broder algoritam.

Algoritam teče na sledeći način: Biramo početnu ćeliju u lavirintu nasumično. Nakon toga biramo sledeću 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 novu šetnju iz novog nasumičnog polja koje nije već u lavirintu.

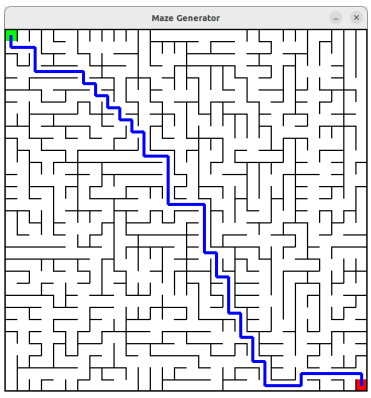
Složenost može da varira značajno od pokretanja do pokretanja, ali može se aproksimirati sa $O((M * N) * log(M * N))$, gde su M i N dimenzije lavirinta, što je nešto efikasnije od Aldous-Broder algoritma.

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

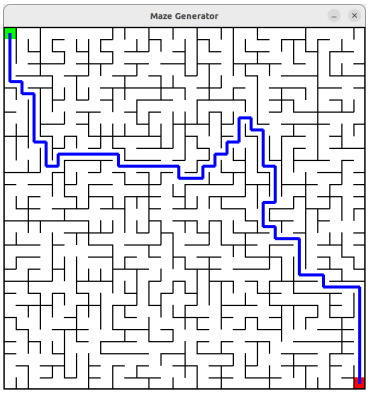
3 Slike algoritama



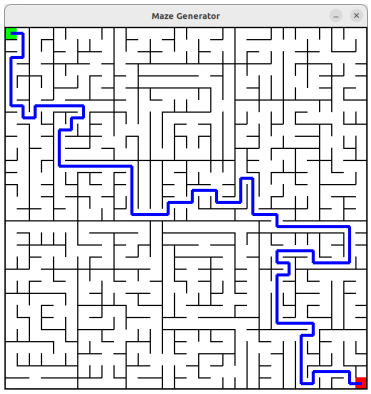
(a) DFS



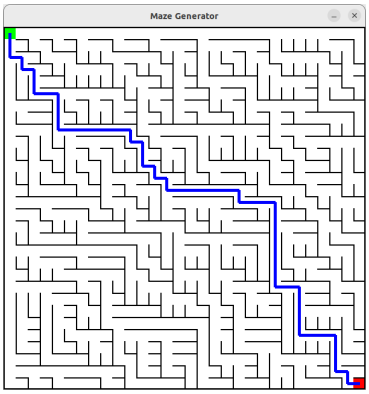
(b) Primov algoritam



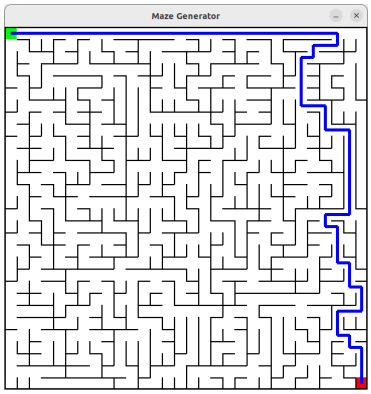
(a) Kruskalov agoritam



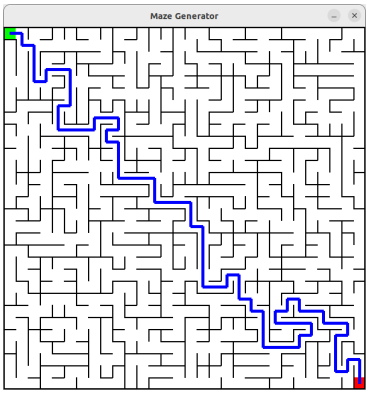
(b) Rekurzivna podela



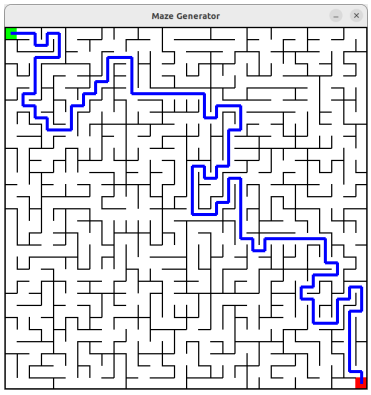
(a) Algoritam binarnog drveta



(b) Algoritam zvečarke



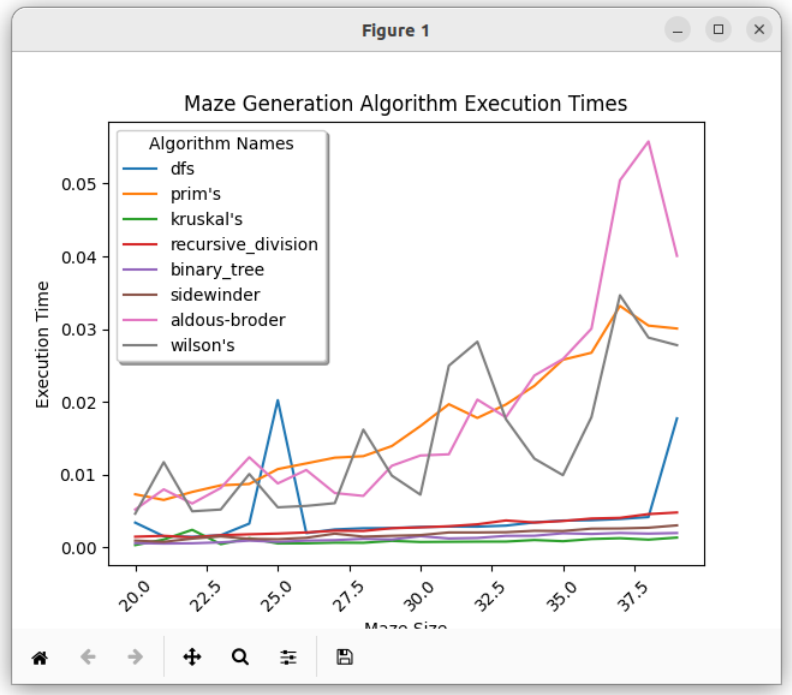
(a) Aldous-Broder algoritam



(b) Wilsonov algoritam

4 Poređenje vremena izvršavanja

Na sledećoj slici dat je grafik koji prikazuje vremena izvršavanja različitih algoritama za različite ulazne veličine lavirinta.



Slika 5: 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 da ova dva algoritma generišu najkomplikovanije lavirinte, ukoliko je to cilj, bolje je koristiti Wilsonov algoritam što se slaže sa našom pretpostavkom iz analize.

5 Zaključak

U ovom izveštaju videli smo šta je problem generisanja slučajnog lavirinta, koje su njegove primene, kao i koji su neki od najpopularnijih algoritama korišćenih za njegovo rešavanje. Prošli smo kroz opise algoritma 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 najpogodniji algoritam po balansu vremena izvršavanja i težine rešavanja lavirinta Wilsonov algoritam. Kako je ovaj problem značajan za razna polja nauke, ostaje prostor i potreba za daljim unapređenjima i novim algoritmima.