Univerzitet u Sarajevu
Prirodno-matematički fakultet
ODSJEK ZA MATEMATIKU

A Zmaja od Bosne 33-35, 71 000 Sarajevo, BiH
T +387 33 279 874 F +387 33 649 342
W www.pmf.unsa.ba/matematika
E matematika@pmf.unsa.ba

Strukture podataka i algoritmi

Projekat 3 - dokumentacija

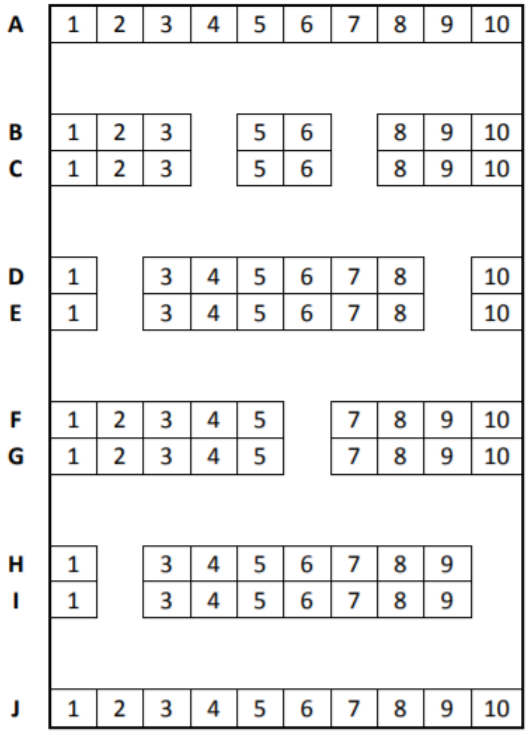
Autor: Said Salihefendić
E-mail: saidsalihefendic@gmail.com
Datum: 15.02.2018

1 Uvod

Ovaj dokument sadržava materijal koji daje diskusiju o problemu pronalaženja najkraćeg puta unutar labirinta za bilo koja dva polja koja su data kao argumenti za pronalazak najkraći put i da vrati njenu dužinu. Odabir puta nije bitan, jer nije jedinstven, nego nas interesuje koja je najkraća udaljenost. Ovaj problem se, uz date uslove, može riješiti na optimalnije načine nego na klasičan način da provjerimo sve putanje od jednog do drugog čvora i uzimamo onu putanju koja ima najkraću udaljenost. Vidjet ćemo da se ovaj način može riješiti na više načina, ali ćemo na ovom problemu detaljno razmatrati dva načina, to jeste, dva algoritma koja daje najkraću udaljenost.
Što želimo razmatrati ovaj problem? Motivacija bi mogla biti da želimo što povoljnije da nađemo optimalnu udaljenost između dvije tačke sa preprekama koje se mogu naći između njih. Dobar dio toga može imati realnu primjenu u robotici i pronalaženja puta od jednog grada do drugog u GPS sistemima.

Format labirinta koje razmatramo u ovom projektu je sljedeći - u jednom fajlu se nalazi jedan format labirinta koji ima dva broja na početku, pri čemu je jedan broj paran i ne može biti veći od 26 i drugi broj koji je pozitivan i može biti bilo koji. Prvi broj označava broj horizontalnih zidova, a drugi broj označava širinu labirinta. Prvi i zadnji zid su uvijek puni, tako da imamo jos m - 2 zidova za razmatrati, pri čemu su zidovi grupisanu u paru, a između njih se nalaze hodnik veličine 2 zida, tako da u fajlu nakon prva dva broja se nalazi (m - 2)/2 specifikacija gdje se nalaze prolazi.

Tipičan primjer labirinta bi bio i koji ćemo uzeti u razmatranje za kasnije algoritme, specifikacija je ((10), (10), (4, 7), (2, 9), (6), (2, 10)):



Napomenimo još da u labirintu se možemo kretati horizontalno i vertikalno i da pristupamo poljima iznad ili ispod, zavisno od datih polja do kojih trebamo doći.

2 Parsiranje labirinta

Prvi problem koji se treba riješiti jeste na osnovu čega ćemo rješavati ovaj problem, tako da ćemo posmatrati kako možemo ovaj labirint parsirati u jednu matricu koja će predstavljati naš labirint. Pošto je poprilično detaljan format podataka unutar jednog tekstualnog fajla, relativno jednostavno možemo i parsirati u matricu cijelih brojeva. Pošto želimo razmatrati više opcija i više funkcija bismo željeli imati uz ovu matricu, onda ćemo probati da ovo rješavamo samo jednom i omogućiti da imamo očuvanje kompleksnosti algoritama žrtvovanjem memorije.
Inicijaliziramo matricu cijelih brojeva sa 0, a onda popunimo prvi zid sa brojevima od 1 do n, pri čemu je n širina labirinta. Potom, krećemo od i-

tog reda, pri čemu i kreće od 3 i povećava se za 4 (3, 7, 11, ..., 2m - 1) i popunimo i i $i + 1$ sa brojevima od 1 do n, a $i + 2$ i $i + 3$ ostavimo redove 0, pošto one predstavljaju hodnik. Sada, čitamo trenutnu liniju u fajlu i parsiramo. Usput dok čitamo liniju, na tom mjestu stavimo u i i $i + 1$ redu na tim pozicijama stavimo 0, da shvatimo ovo kao prolaz. Za kasnije algoritme, također pamtimo prolaze po njihovim ivicama, tj. ako imamo veliki prolaz, onda pamtimo samo pozicije početka i kraja prolaza po pozicijama u listu pozicija, koju ćemo koristiti za algoritam koji ćemo kasnije analizirati.

3 Naivna ideja

Nakon što smo uspjeli parsirati, lako možemo vidjeti da se ovaj problem može riješiti jednostavnom rekurzijom koja, zavisno od toga da li se prvo polje nalazi iznad ili ispod drugog polja. Uzmimo za primjer da je prvo polje iznad drugog polja, tj. da krecemo odozgo prema dole. Onda je rekurzija

$$T(z1, p1, z2, p2) = min(rek) + 1$$

$$rek = T(z1 + 1, p1, z2, p2), T(z1, p1 - 1, z2, p2), T(z1, p1 + 1, z2, p2)$$

Baza rekurzije je da, ako je otišlo van granica labirinta ili je zid u pitanju, onda bi trebao vratiti INFINITY vrijednost, a ako je to traženo polje, vraća 1.
Primijetimo da na ovaj način računamo udaljenost svakog mogućeg polja do ciljanog polja, i to više puta, što ovo baš i nije efikasno. Radi se o eksponencijalnom vremenu kompleksnosti i imamo višak računanja ovdje. Razmotrimo drugu, ljepšu ideju.

4 Ideja - korištenje BFS-a

Prethodna ideja je koristila rekurziju i možemo vidjeti da je poprilično neefikasna, koja u međuvremenu računa ponovo neke stvari, a u pitanju je eksponencijalna kompleksnost. Pošto se radi o preklapanju podproblema, možemo ovo riješiti dinamičkim programiranjem. Ideja je da iskoristimo iz teorije grafova algoritam Breadth-First Search koja garantuje najkraću udaljenost od jednog čvora do drugog čvora u grafu. Možemo na vrlo sličan način ovaj labirint posmatrati kao jedan graf, pri čemu su čvorovi, zapravo, prazna mjesta. Krenuvši od početnog polja, računamo udaljenost među njegovim susjedima i spasimo u tabelu udaljenosti (matrica istih dimenzija kao labirint) i onda ubacujemo u red njegove susjede. Zatim, uzimamo iz

reda sljedeći čvor, tj. poziciju koju želimo razraditi, pri čemu spašavamo u tabeli udaljenost koja je do tada izračunata i povećamo je za 1, a zatim guramo u red njegove susjede koji nisu posjećeni i ponavljamo ovo sve dok red nije prazan. Ponovo, ovo je algoritam koji radi odozgo prema dole, a ako je prvo polje ispod drugog polja, onda samo zamijenimo njihove uloge i pozivamo se na ovaj algoritam, pošto je svejedno odakle krenemo, nas interesuje samo udaljenost. Onda vratimo rezultat koji se nalazi na poziciji drugog polja, a na toj poziciji se nalazi i udaljenost u tabeli koju smo izračunali.

Optimalna podstruktura je da susjedi drugog polja do kojeg želimo doći imaju optimalnu udaljenost do prvog polja. Kompleksnost je O(mn), pri čemu je $m \leq 2 * 26$, a n širina labirinta. Problem je što dosta prostora zauzima ovaj algoritam, a dosta računa neke stvari koji se mogu računati u konstantnom vremenu. Možemo primijetiti da prepreke su, zapravo, samo zidovi na određenim pozicijama. Mi poznajemo prolaze na svakom zidu (osim prvog i zadnjeg zida), pa možemo tu informaciju iskoristiti za modificirani algoritam, koji zapravo ubrzava računanje najkraće udaljenosti i još je memorijski efikasniji od BFS algoritma.

5 Konačna ideja - modifikacija

Ideja je slična prethodnoj, s tim da je memorijski efikasnija i da je manje kompleksnosti nego prethodna ideja. Prethodna ideja koja je koristila BFS je imala potrebu da ide korak po korak računanjem do drugog polja, što nije najoptimalniji način za računanje udaljenosti. Samo računanje u istom redu zahtijeva O(n) kompleksnost, zbog korak po korak računanja. Možemo iskoristiti strukturu labirinta da maksimiziramo efikasnost i da iskoristimo Manhattanovu udaljenost da imamo O(c) računanje ne samo u redu, nego u komplet hodniku. Jedini problem je što moramo obraćati pažnju na prolaze kroz zidove. Prisjetimo se u parsiranju labirinta da smo usput u listu spašavali pozicije rubove prolaza. Ovo ćemo sada da iskoristimo.
Krećemo od polja koji se nalazi iznad drugog polja, pošto algoritam funkcioniše odozgo prema dole. Nevažno odakle god da krenemo, uzimamo u razmatranje samo dva ruba prolaza, ukoliko je to moguće, a to je najbliži rub s lijeve i rub s desne strane u odnosu na polje koje posmatramo. Može se desiti da imamo i direktan prolaz ispod polja, a da nije rub, tako da i to testiramo da li takvu situaciju imamo. Razlog što uzimamo ovakve rubove je to što, ma koji god uzimali rub lijevo od polja koje posmatramo, dobit

ćemo istu udaljenost, tako da, zbog implementacije, tražimo najbliži lijevi, jer nam je odmah lahko odrediti i najbliži desni. Može se desiti da imamo samo prolaze s lijeve strane i samo prolaze s desne strane, ali ti slučajevi se lahko rješavaju, onda uzimamo samo najbliži prolaz sa strane gdje imamo te prolaze. Nakon što imamo ove rubove, onda, jednostavno, izračunamo Manhattan distance od polja koje razmatramo do tih prolaza i guramo u red u vidu para (pozicija, potencijal), pri čemu je pozicija sama za sebe par cijelih brojeva koji označavaju poziciju polja unutar labirinta, a potencijal je do tada izračunata udaljenost. Ove operacije su u konstantom vremenu računanja. Zatim, sve dok ne dođemo do nivoa gdje se nalazi naše drugo polje, uzimamo sa reda par koji možemo smatrati kao polje, ponovo tražiti najbliže rubove na isti način kao i inicijalno, računamo Manhattan udaljenost od polja koje razmatramo do tih rubova i guramo isto u red te rubove, s tim da spašavamo zbir dotadašnjeg potencijala i Manhattan udaljenost.
U nivou gdje se nalazi drugo polje, u redu ćemo imati samo prolaze koji se nalaze u zidu iznad drugog polja, gdje imamo sve moguće potencijale. Procesiramo čitav red, tražeći minimalan zbir potencijala i udaljenost od tog prolaza sa takvim potencijalom do drugog polja.
Kompleksnost ovog algoritma je O(k), pri čemu je k broj prolaza u labirintu. Ovo je neobična modifikacija grafova i dinamičkog programiranja za rješavanje ovog problema, ali definitivno je i vremenski i memorijski efikasniji od BFS algoritma u prethodnoj sesiji.

6 Tura polja

Do sada smo razmatrali računanje najkraće udaljenosti od jednog polja do drugog unutar labirinta, ali šta ako imamo potrebu posjećivati više polja i tražimo najbolju moguću turu u odnosu na ukupnu udaljenost? Možemo uzeti inicijalnu turu koja se sastoji od takvih polja unutar nekog vektora i primijeniti heuristiku nad njima, tj. 2-opt i 3-opt algoritme koje smo razmatrali u prošlom projektu. Jednostavnom modifikacijom možemo izračunati dovoljno dobre ture za naš problem.
Međutim, moramo pogledati kako ovo možemo ubrzati i na koji način. Trenutna implementacija u projektu je vrlo neefikasna, pri čemu se poziva funkcija da se izračuna udaljenost od jednog polja do drugog polja tokom heuristike, pri čemu zahtijeva dodatno vremena, povećavajući kompleksnost algoritma. Jedan od načina da ne pozivamo se uvijek na ovu funkciju je da žrtvujemo prostor i da napravimo matricu udaljenosti inicijalne ture, koja računa najkraću udaljenost između svakog para polja unutar inici-

jalne ture (uzimamo u obzir labirint nad kojom je tura osnovana), a onda primijenimo 3-opt heuristiku, pri čemu zadržavamo njegovu kompleksnost. Uglavnom, veliki je problem ako imamo više tura koje moramo procesirati, ali također nam se ne isplati uvesti jednu veliku matricu udaljenosti koja sadržava najkraću udaljenost između svakog polja mogućeg u labirintu, tako da ta ideja otpada.
Tako da, krajnja implementacija bi sadržavala inicijalnu turu koja bi povezala polja sa ključevima koji odgovara indeksima u inicijalnoj turi i onda bismo to iskoristili da izračunamo matricu, a onda 3-opt primijenimo nad turom cijelih brojeva koji su, zapravo, indeksi u inicijalnoj turi. Kada završi se 3-opt, onda računamo ukupnu dužinu ture pomoću matrice, ponovo, a vratimo turu sa poljima onako kako su poredani indeksi u turi indeksa.

7 Zaključak

Na ovom projektu smo mogli primijetiti kako razne kombinacije struktura podataka, algoritama i dinamičkog programiranja možemo doći do korektnog rješenja vrlo efikasno, što dokazuje činjenicu da moramo uzeti u razmatranje svo naše znanje koje imamo da rješimo razne probleme kao što je ovaj. U kompjuterskim nauka je jako važno razmišljati kako što efikasnije naći što bolje, po mogućnosti optimalno, rješenje. U tom cilju važno je da razumijemo suštinu problema i šablon koji se nalazi u datim podacima i iskoristiti to u kombinaciji sa našim znanjem da dođemo do boljeg rješenja.