Dokumentacija

Predmet: Analiza i sinteza algoritama

Projekat: Tema 1

Student: Iman Mahmutović

Funkcije

a) brojMogucihPuteva(int m, int n)

Ova funkcija koristi dinamičko programiranje za pronalaženje svih mogućih puteva od pozicije (0, 0) do pozicije (0, n-1) u zadatoj matrici. Obzirom na dozvoljene korake (uvijek se pomijeramo za jednu kolonu udesno a možemo ići u red iznad, ispod ili ostati u istom redu) i obzirom na to da se u početku nalazimo na poziciji (0, 0), nemoguće je doći do mjesta koje se nalazi ispod glavne dijagonale, tj. mjesto gdje je i > j (i je broj reda a j broj kolone). Isto tako, na pozicije koje se nalaze na glavnoj dijagonali, moguće je doći samo iz smjera (i, j) → (i+1, j+1). U matrici DP ćemo čuvati broj putanja od (0, 0) do trenutne pozicije. Krajnji rezultat će biti DP[0][n-1]. Zbog svega ovoga, matrica će se inicijalno popuniti nulama s tim da će na glavnoj dijagonali biti jedinice. Vrijednosti ispod glavne dijagonale će ostati nule a elemente iznad dijagonale popunjavamo po kolonama na osnovu sljedeće formule:

DP [i][j] = DP [i][j-1] + DP [i-1][j-1] + DP [i+1][j-1]

Ako neki od indeksa izlazi van dometa onda se uzima da je vrijednost na tom mjestu nula.

Vremenska i memorijska složenost ovog algoritma je O(n*m). Ako zanemarimo sami ispis matrice DP, možemo reći da je vremenska složenost kvadratna u odnosu na manju dimenziju matrice jer u ugnježdenoj petlji prolazimo samo kroz elemente iznad glavne dijagonale. Ako je m < n taj broj iznosi $\frac{m*(m-1)}{2}$ a ako je n ≤ m onda je taj broj jednak $\frac{n*(n-1)}{2}$.

b) brojMogucihPutevaSaZabranama(vector<vector<bool>> zabrane)

Ova funkcija je skoro identična gornjoj, radi se ista inicijalizacija matrice DP ali je jedina razlika pri popunjavanju elemenata iznad glavne dijagonale. Prije nego što popunimo polje, moramo provjeriti u matrici zabrane da li je to polje dozvoljeno, ako nije onda ga preskačemo i na njegovom mjestu ostaje nula a ako je dozvoljeno onda ga popunjavamo po istoj formuli kao u prvoj funkciji. Vremenska i memorijska složenost su iste kao i u prethodnoj funkciji.

c) backtrack(int m, int n, int i, int j, vector<int> &put, const vector<int> &indeksi, int &brojac)

Ova funkcije koristi bactracking algoritam za pronalaženje svih putanja od (0, 0) do (0, n-1). Putanju čuvam u vektoru put i čuvam samo red u kojem se nalazi dati element. Na samom početku funkcije provjeravamo da li se indeksi nalaze van dometa i da li je trenutni element ispod glavne dijagonale (jer smo ustanovili da nijedna putanja ne može prolaziti kroz element koji je ispod glavne dijagonale). Ako je bilo šta od toga tačno, vraćamo se u funkciju koja je pozvala trenutnu funkciju. U svakom koraku dodajemo trenutni element u put i provjeravamo da li se nalazimo na poziciji (0, n-1). Ako jesmo na toj poziciji, onda provjeravamo da li trenutni put zadovoljava redoslijed redova koji je dat u vektoru indeksi. Ako zadovoljava, povećavamo brojač. Ako nismo na toj poziciji, onda nastavljamo pretragu sa tri rekurzivna poziva jer iz pozicije (i, j) imamo 3 opcije za idući korak: (i, j+1), (i-1, j+1), (i+1, j+1). Kada završe svi rekurzivni pozivi, skidamo trenutni element sa vektora put.

Vremenska složenost ovog algoritma je poprilično loša i jednaka je O(3^{n*m}). Zapravo, ne prolazimo kroz cijelu matricu već samo kroz elemente iznad ili na glavnoj dijagonali ali taj broj će zavisiti od odnosa brojeva m i n tako da zbog jednostavnosti uzimamo da je složenost O(3^{n*m}). Jedino dobro kod ovog algoritma je memorijska složenost koja je jednaka O(k) pri čemu je k najduža moguća putanja od (0, 0) do bilo koje pozicije (i, j) iznad ili na glavnoj dijagonali.

brojMogucihPutevaSaIndeksima2(int m, int n, vector<int> indeksi)

Cilj ove funkcije je isti kao i cilj prethodne funkcije sa jedinom razlikom što koristi dinamičko programiranje a ne backtracking. Funkcija je slična funkcijama pod a) i b) samo što u matrici DP za svaki element čuvam sve moguće putanje od (0, 0) do trenutnog elementa. Radi lakšeg snalaženja, napravila sam strukturu Putanje koja čuva matricu putanje i vektori u toj matrici čuvaju samo redove elemenata. Znamo da u svaki element sa glavne dijagonale možemo doći samo iz prethodnog elementa sa glavne dijagonale, pa shodno tome prvo popunjavamo elemente glavne dijagonale matrice DP. Ostale elemente iznad glavne dijagonale popunjavam tako što uzmem putanje sa pozicija (i, j-1), (i-1, j-1), (i+1, j-1) ukoliko te pozicije postoje, dodam na sve te putanje trenutni element i spasim u matricu DP [i][j].putanje.

Ovdje je vremenska složenost O(m*n*p) gdje je p maksimalan broj putanja u bilo kojoj matrici. Vidimo da je ovaj pristup puno bolji nego backtracking što se tiče vremenske složenosti ali je memorijska složenost gora u odnosu na prethodni način i ona je jednaka O(m*n*p).

Dakle, ako je prioritet brzina izvršavanja, onda je optimalno rješenje dato pomoću dinamičkog programiranja. Ako je ipak bitnija memorijska složenost, onda je optimalno rješenje dato pomoću backtracking-a.