Prirodno matematički fakultet
Aldin Ahmethodžić (Teorijska kompjuterska nauka)
5532/M
aldinahmethozdic@gmail.com

Problem trgovačkog putnika (dinamičko programiranje)

29. Januar 2021

Struktura grafa

Graf sam čuvao kao matricu susjedstva, gdje sam matricu pravio kao vektor vektora tipa double. Kroz konstruktore sam omogućio da se ili proslijedi matrica i napravi graf, ili da se unese broj čvorova a potom dodaju grane.

Atributi klase

```
class Graf{
    int n; // broj čvorova
    vector<vector<double>> matrica_susjedstva;
    int sviPosjeceni; // Za bazni slučaj rekurzivnog DP algoritma
    vector<vector<int>>dp; // DP 2^n * n matrica za spašavanje rezultata u algoritmu
```

Metode klase

```
public:
    Graf(int broj_cvorova); // Kreira graf sa n čvorova
    Graf(vector<vector<double>>); // Kreira graf sa listom susjedstva jednakoj proslijeđenoj matrici
    void dodajGranu(int u, int v, double tezina); // Dodaje granu
    void ispisi(); // Ispis matrice susjedstva

    int PTP_Sporo(int); // O(n!)
    int PTP_Dinamicko(int,int); // O(2^n * n^2)
};
```

Brute force pristup - $O\ (n!)$

Prvobitno sam uradio spori algoritam (funkcija **PTP_Sporo(int pocetni))** gdje sam napravio vektor vrhova grafa, te generisao sve permutacije i redom računao ukupnu težinu prolaska kroz čvorove svake permutacije. Za ovu svrhu sam koristio ugrađenu C++ funkciju unutar biblioteke <algorithm>, **next_permutation()**.

Pristup dinamičkim programiranjem - $O\ (2^n \cdot n^2)$

Unutar konstruktora deklarišemo potrebne varijable i DP matricu za algoritam:

```
sviPosjeceni = (1 << n) - 1;
dp.resize((int)pow(2,n), vector<int>(n));

for(int i = 0; i < (1<<n); i++){
    for(int j = 0; j < n; j++){
        dp[i][j] = -1;
    }
}
```

sviPosjećeni - n bita postavljenih na 1, npr. za n = 4, varijabla sviPosjećeni će biti jednaka **1111**, što označava da su sva četiri grada posjećena. Prvi bit sa desna označava prvi grad.

vector<vector<int>>dp - u dp matrici spašavam rezultate, odnosno već izračunate težine između gradova. Vrijednost -1 označava da težinu na toj relaciji nismo prethodno izračunali. Veličina je $2^n \times n$ jer postoji 2^nkombinacija bita (posjećenih gradova) i n gradova.

```
Graf::PTP_Dinamicko(int biti, int trenutni){

// Bazni slučaj rekurzije, ako su svi gradovi
if(biti == sviPosjeceni){
    return matrica_susjedstva[trenutni][0];
}

// Provjera rezultata u dp
if(dp[biti][trenutni] != -1){
    return dp[biti][trenutni];
}
```

biti - predstavlja posjećene gradove, gdje bit predstavljen sa 1 označava posjećen grad. Kada funkciju pozivamo, proslijedit ćemo 1 (0001) kao vrijednost varijable **biti** kao znak da je prvi grad posjećen.

Bazni slučaj jeste onaj kada su svi biti postavljeni na 1 tj. svi gradovi su posjećeni. Tada vraćamo težinu povratka od trenutnog do početnog grada.

dp predstavlja matricu već izračunatih vrijednosti težine između gradova. Ukoliko je izračunata odnosno != -1, vraćamo težinu udaljenosti od trenutnog do preostalih gradova.

```
int rezultat = INT_MAX;

// Od trenutnog zelimo doci do svih neposjecenih gradova
for(int i = 0; i < n; i++){
    if((biti&(1 << i)) == 0){ // Ako grad nije posjecen
        // Rezultat je udaljenost od trenutnog do grada i + rekurzivno udaljenost grada i od
        int noviRezultat = matrica_susjedstva[trenutni][i] + PTP_Dinamicko(biti|(1 << i), i);
        rezultat = min(rezultat,noviRezultat);
    }
}

return dp[biti][trenutni] = rezultat;
```

biti & (1 << i) predstavlja informaciju o posjećenosti i-tog grada. Npr. za bite jednake 0110 i za i = 2,

1 << i će biti jednak bitima 0100, odnosno samo bit koji označava posjećenost drugog grada će biti postavljen na jedinicu.

Operacijom **&** nad **0110** i **0100**, dobićemo **rezultat 0100** što je različito od nule, pa je grad posjećen.

noviRezultat predstavlja rekurzivno računanje udaljenosti od i-tog grada do preostalih neposjećenih gradova.

biti | **(1 << i)** unutar rekurzivnog poziva označavaju da je i naredni grad posjećen, pa takve bite šaljemo dalje u pozive u funkcije. Naprimjer, ako smo se nalazili u prvom gradu, biti su 0001, prelaskom na drugi grad zbog **OR** operacije nad bitima, bite mijenjamo na 0011 čime označavamo da je i drugi grad sada posjećen.

rezultat = **min(rezultat, noviRezultat)** određuje putanju minimalne težine.

return dp[biti][trenutni] = **rezultat** vraća rezultat i u isto vrijeme pamti u dp matrici koja je težina za trenutni položaj bita odnosno posjećene gradove krenuvši od trenutnog grada.

KRAJ