Prirodno-Matematički Fakultet

Univerzitet u Sarajevu

Dokumentacija za projekat : Tema 2

Analiza i Sinteza Algoritama

Student: Hamza Kolašinac

Indeks: 5778/M

Sarajevo, Januar 2024

Sadržaj

Cilj Projekta

Cilj Projekta je da konstruišemo labirint od parametara za broj redova n, broj kolona m i onda u sljedećih $\frac{n-2}{2}$ redova se nalazi lista pozicija na kojima su prolazi u odgovarajućem redu, u rastućem redoslijedu, razdvojene zarezom; u prvom redu su pozicije prolaza u redovima označenim sa B i C, u drugom redu pozicije prolaza u redovima označenimsa D i E itd. Nakon konstrukcije labirinta cilj je za neke dvije koordinate naći najkraći put od jednog čvora do drugog i ispisati taj put.

Implementacija

Projekat se sastoji od klase labirint koja čuva broj redova, broj kolona i matricu labirinta kao i pomoćnih funkcija za Unos, Ispis i udaljenost.

Klasa i Unos

```
class Labirint {
    int broj_redova;
    int broj_kolona;
    vector<vector<int>> labirint;
```

Klasa je standardno implementirana sa vektorom i dva broja tipa int.

Main

```
int main() {
    int broj_redova = 0;
    cin>>broj_redova;
    int broj_kolona = 0;
    cin>>broj_kolona;
    Labirint l(broj_redova,broj_kolona);

    string rupa = "";


 int trenutni_broj = 3;
 for(int i = 0; i < (broj_redova-2)/2; i++){
     string broj = "";
     vector<int> vektor_rupa;
     cin>>rupa;
     for (char karakter : rupa) {
         if (karakter != ',') {
             broj+= karakter;
         }
         else{
             vektor_rupa.push_back(stoi(broj)-1)
             broj = "";
         }
     }
     vektor_rupa.push_back(stoi(broj)-1);
     l.Unesi(vektor_rupa, trenutni_broj);
     trenutni_broj += 4;
 }
```

U main funkciji se unos vrši, uzimamo broj redova i broj kolona pa pravimo novu varijablu tipa Labirint koji stavlja veličinu i pravi vektor labirinta. Nakon toga imamo $\frac{n-2}{2}$ unosa za rupe u zidovima. Za svaki unos rupa u zidovima uzimamo svaki karakter i ako je broj onda stavimo u string a ako je zarez onda taj broj dodamo u vektor rupa i broj ispraznimo. Na kraju još jednom dodamo zadnji broj jer smo dodavali nakon svakog zareza a na kraju nema zareza. Na kraju pozovemo pomoćnu funkciju za unos i dodamo na trenutni broj 4.

Unesi

```
void Labirint::Unesi(vector<int> red, int redni_broj){
    for(int i = 0; i<red.size(); i++){
        labirint[redni_broj][red[i]] = 0;
        labirint[redni_broj+1][red[i]] = 0;
    }
}
```

Unesi je jednostavna funkcija koja za vektor iz maina nađe gdje trebaju biti rupe i u vektoru labirinta ih postavi.

Udaljenost

Glavna funkcija projekta je funkcija udaljenost. Radi na princip Djikstra algoritma gdje relaksiramo svaki čvor i njegove susjede.

Priprema

```
int Labirint::udaljenost(char pocetni_red_char, int pocetna_kolona, char krajnji_red_char, int krajnja_kolona)
    int pocetni_red = int(pocetni_red_char) - int('A');
    int krajnji_red = int(krajnji_red_char) - int('A');
    pocetna_kolona -= 1;
    krajnja_kolona -= 1;
    pocetni_red += 2*ceil(pocetni_red/2.0);
    krajnji_red += 2*ceil(krajnji_red/2.0);
    vector<int> udaljenosti_red(labirint[0].size(), -1);
    vector<vector<int>> DP(labirint.size(), udaljenosti_red);
    vector<pair<int, int>> pomocni;
    pomocni.push_back(make_pair(pocetni_red, pocetna_kolona));
    DP[pocetni_red][pocetna_kolona] = 0;
```

Prvi dio funkcije je pretvaranje koordinata date u engleskom alfabetu u koordinate pogodne za korištenje u funkciji. Sljedeći dio je pravljenje DP vektora koji će spremati rezultate udaljenosti

od početnog čvora do svakog drugog čvora i pravljenje pomoćnog vektora parova gdje ubacujemo početni red i početnu kolonu i stavljamo DP od istog para 0.

Djikstra algoritam

```
while(pomocni.size()){
    pair<int, int> trenutni = pomocni[pomocni.size() - 1];
    pomocni.pop_back();
    if(trenutni.first > 0){
        if(labirint[trenutni.first - 1][trenutni.second] != -1){
            if(DP[trenutni.first - 1][trenutni.second] == -1){
                DP[trenutni.first - 1][trenutni.second] = DP[trenutni.first][trenutni.second] + 1;
                pomocni.push_back(make_pair(trenutni.first - 1, trenutni.second));
            }
            else{
                if(DP[trenutni.first][trenutni.second] + 1 < DP[trenutni.first - 1][trenutni.second]){
                    DP[trenutni.first - 1][trenutni.second] = DP[trenutni.first][trenutni.second] + 1;
                    pomocni.push_back(make_pair(trenutni.first - 1, trenutni.second));
                }
            }
        }
    }


if(trenutni.second > 0){
    if(labirint[trenutni.first][trenutni.second - 1] != -1){
        if(DP[trenutni.first][trenutni.second - 1] == -1){
            DP[trenutni.first][trenutni.second - 1] = DP[trenutni.first][trenutni.second] + 1;
            pomocni.push_back(make_pair(trenutni.first, trenutni.second - 1));
        }
        else{
            if(DP[trenutni.first][trenutni.second] + 1 < DP[trenutni.first][trenutni.second - 1]){
                DP[trenutni.first][trenutni.second - 1] = DP[trenutni.first][trenutni.second] + 1;
                pomocni.push_back(make_pair(trenutni.first, trenutni.second - 1));
            }
        }
    }
}
```

```
if(trenutni.first < labirint.size()-1){
    if(labirint[trenutni.first + 1][trenutni.second] != -1){
        if(DP[trenutni.first + 1][trenutni.second] == -1){
            DP[trenutni.first + 1][trenutni.second] = DP[trenutni.first][trenutni.second] + 1;
            pomocni.push_back(make_pair(trenutni.first + 1, trenutni.second));
        }
        else{
            if(DP[trenutni.first][trenutni.second] + 1 < DP[trenutni.first + 1][trenutni.second]){
                DP[trenutni.first + 1][trenutni.second] = DP[trenutni.first][trenutni.second] + 1;
                pomocni.push_back(make_pair(trenutni.first + 1, trenutni.second));
            }
        }
    }
}


    if(trenutni.second < labirint[0].size()-1){
        if(labirint[trenutni.first][trenutni.second + 1] != -1){
            if(DP[trenutni.first][trenutni.second + 1] == -1){
                DP[trenutni.first][trenutni.second + 1] = DP[trenutni.first][trenutni.second] + 1;
                pomocni.push_back(make_pair(trenutni.first, trenutni.second + 1));
            }
            else{
                if(DP[trenutni.first][trenutni.second] + 1 < DP[trenutni.first][trenutni.second + 1]){
                    DP[trenutni.first][trenutni.second + 1] = DP[trenutni.first][trenutni.second] + 1;
                    pomocni.push_back(make_pair(trenutni.first, trenutni.second + 1));
                }
            }
        }
    }
}
```

While petlja je tok djikstra algoritma. Na početku uzmemo zadnji element pomoćnog vektora i poppamo iz pomoćnog vektora taj element. Taj element je trenutni koji razmatramo. Uzmemo njegove susjede i ukoliko je udaljenost susjeda -1 to znači da ga prvi put vidimo i onda mu je udaljenost jednaka udaljenosti trenutnog + 1. Ukoliko nije -1 onda provjerimo da li je udaljenost od trenutnog + 1 manji od udaljenosti susjeda, ako jeste onda mu postavljamo to kao novu udaljenost. I u jednoj i u drugoj situaciji ako promijenimo udaljenost nekog čvora onda ga dodajemo u pomoćni vektor. While traje dok vektor nije prazan i na kraju imamo vektor DP u kojem su udaljenosti od početnog čvora.

Pronalazak Čvora i konstrukcija puta

```
vector<vector<int>> put(labirint);
pair<int, int> trenutni;
pair<int, int> kraj;
if((krajnji_red > 0) && (labirint[krajnji_red - 1][krajnja_kolona] != -1) &&
   (DP[krajnji_red - 1][krajnja_kolona] > 0)){
    trenutni = make_pair(krajnji_red - 1, krajnja_kolona);
}
else if((krajnja_kolona != 0) && (labirint[krajnji_red][krajnja_kolona-1] != -1)){
    trenutni = make_pair(krajnji_red, krajnja_kolona - 1);
}
else if((krajnja_kolona < broj_kolona) && (labirint[krajnji_red][krajnja_kolona + 1] != -1) &&
        (DP[krajnji_red][krajnja_kolona + 1] > 0)){
    trenutni = make_pair(krajnji_red, krajnja_kolona + 1);
    if((krajnja_kolona != 0) && (labirint[krajnji_red][krajnja_kolona-1] != -1) &&
       (DP[krajnji_red][krajnja_kolona - 1] < DP[trenutni.first][trenutni.second])){
        trenutni = make_pair(krajnji_red, krajnja_kolona - 1);
    }
}
else{
    trenutni = make_pair(krajnji_red + 1, krajnja_kolona);
}
```

```
if(DP[trenutni.first][trenutni.second] == -1){
    cout<<"Nije pronadjen put!"<<endl;
    return -1;
}
else{
    kraj = trenutni;
    put[trenutni.first][trenutni.second] = DP[trenutni.first][trenutni.second];
}
```

Ispis labirinta se vrši na način da prvo nađemo najbliži čvor do završnog čvora. Taj čvor kada odredimo ubacujemo ga u while petlju.

While Petlja Ispisa

```
while(true){
    if(DP[trenutni.first][trenutni.second] == 1){
        break;
    }
    if(DP[trenutni.first - 1][trenutni.second] != -1){
        if(trenutni.first > 0){
            if(DP[trenutni.first - 1][trenutni.second] < DP[trenutni.first][trenutni.second]){
                put[trenutni.first - 1][trenutni.second] = DP[trenutni.first - 1][trenutni.second];
                trenutni.first = trenutni.first - 1;
                continue;
            }
        }
    }
    if(trenutni.second > 0){
        if(DP[trenutni.first][trenutni.second - 1] != -1){
            if(DP[trenutni.first][trenutni.second - 1] < DP[trenutni.first][trenutni.second]){
                put[trenutni.first][trenutni.second - 1] = DP[trenutni.first][trenutni.second - 1];
                trenutni.second = trenutni.second - 1;
                continue;
            }
        }
    }


    if(trenutni.first < labirint.size() - 1){
        if(DP[trenutni.first + 1][trenutni.second] != -1){
            if(DP[trenutni.first + 1][trenutni.second] < DP[trenutni.first][trenutni.second]){
                put[trenutni.first + 1][trenutni.second] = DP[trenutni.first + 1][trenutni.second];
                trenutni.first = trenutni.first + 1;
                continue;
            }
        }
    }
    if(trenutni.second < labirint[0].size()){
        if(DP[trenutni.first][trenutni.second - 1] != -1){
            if(DP[trenutni.first][trenutni.second + 1] < DP[trenutni.first][trenutni.second]){
                put[trenutni.first][trenutni.second + 1] = DP[trenutni.first][trenutni.second + 1];
                trenutni.second = trenutni.second + 1;
                continue;
            }
        }
    }
}
```

While petlja radi na način da pogledamo kroz DP vektor koji nam drži udaljenosti i od krajnjeg čvora dodajemo u vektor put gdje držimo samo jedan optimalni put od početnog do krajnjeg čvora.

Ispis puta

```
    int duzina = to_string(int(put.size()/4) * put[0].size()).length();
    for(int i = 0; i < put.size(); i++){
        for(int j = 0; j < put[0].size(); j++){
            if(put[i][j] == -1){
                cout<<setw(duzina)<<to_string(j+1)<<" ";
            }
            else if(put[i][j] == 0){
                cout<<setw(duzina)<<""<<" ";
            }
            else{
                cout<<setw(duzina)<<to_string(put[i][j])<<" ";
            }
        }
        cout<<endl;
    }

    return DP[kraj.first][kraj.second];
}
```

Na kraju imamo ispis vektora put. Korišten je setw da ljepše izgleda ispis i nakon ispisa vratimo krajnji čvor.

Pokretanje

Nakon pokretanja na primjeru iz postavke projekta dobijemo ovakav ispis.

Brzina Algoritma

Algoritam je baziran na Djikstra algoritmu koji je brzine O(E * log(V)), pošto E ima otprilike 4 * V jer svaki čvor ima u najgorem slučaju 4 susjeda to je brzina O(V * log(V)). V je skup svih čvorova što je u najgorem slučaju 26 * n pa je brzina algoritma O(n * log(n)). Brzina ispisa je n pa je brzina cijelog projekta O(n * log(n)).