Documentatie implementare secventiala Algoritm Dijkstra

1. Detalii despre IDE-ul si sistemul pe care a fost rulat algoritmul

Algoritmul a fost implementat si testat in C++ folosind IDE-ul Visual Studio 2019 pe laptopul personal. Laptopul are urmatoarea configuratie:

```
Operating System: Windows 10 Pro 64-bit (10.0, Build 18363)
        Language: English (Regional Setting: English)
System Manufacturer: Dell Inc.
    System Model: Latitude 5500
            BIOS: 1.6.5
       Processor: Intel(R) Core(TM) i7-8665U CPU @ 1.90GHz (8 CPUs), ~2.1GHz
          Memory: 16384MB RAM
       Page file: 9841MB used, 8785MB available
  DirectX Version: DirectX 12
```

2. Reprezentarea datelor:

A) de intrare:

- Se citesc din fisiere .txt
- Structura fisierelor .txt este urmatoarea:
  - Pe prima line se afla numarul de noduri si numarul de arce (n si m), 0<n<=297, 0<m<=2148
  - Pe a doua linie se afla nodul sursa si destinatie (source si destination), 0<destination, source<=n
  - Pe urmatoarele m linii se afla muchia si costul acesteia. Muchia este reprezentata prin perechea de noduri (x,y), iar costul prin cost (0<x,y<=n si cost <= 9999)
  - Datele de pe fiecare linie sunt separate prin spatiu

B) de iesire:

- Se afiseaza atat in fisiere .txt cat si in fisier .csv fiecare continand outputul corespunzator
- Structura fisierelor .txt este urmatoarea:
  - Daca exista drum intre nodul sursa si destinatie atunci:
    - Prima linie contine costul total al drumului. Prin cost total se intelege suma tuturor costurilor de pe fiecare muchie de pe cel mai scurt drum identificat.

        - A doua linie, separat prin spatiu, contine drumul de la nodul sursa la destinatie. Prin drum se intelege multimea nodurilor care il reprezinta.
    - Daca nu exista drum intre nodul sursa si destinatie atunci fisierul .txt va contine o singura linie cu un mesaj de atentionare.
- Rolul fisierului .csv este de a salva timpii de executie precum si dimensiunea grafului. Structura fisierului este urmatoarea:
    - Prima coloana reprezinta numarul de noduri n
    - A doua coloana, numarul de arce
    - A treia, nodul sursa
    - A patra, nodul destinatie
    - A cincea, timpul de executie reprezentat prin microsecunde

C) structuri de date folosite:

- Vectori normali din C/C++ (p[MAX], dist[MAX] unde MAX este o constanta definita de catre mine)
- Un dictionar care are ca si cheie primara un nod x, iar ca si valoare un vector de structuri de noduri, unde prin structuri de noduri se intelege nodul adiacent nodului reprezentat de cheie si costul aferent acestei muchii. Acest dictionar este simulat printr-un vector din stl astfel: vector<Node>G[MAX], unde Node este o structura care are urmatoarea reprezentare:

```
/*the struct of the node*/
struct Node {
    int y;
    int cost;
    Node(const int& y, const int& cost) {
        this->y = y;
        this->cost = cost;
    }
    /*overriding the < operator*/
    bool operator < (const Node& other) const {
        return this->cost > other.cost;
    }
};
```

- Coada cu prioritati, unde prioritatea este definita in felul urmator: are prioritate muchia cu costul cel mai mic. Aceasta este definita: priority_queue<Node>Q
- Un iterator peste dictionar definit vector<Node>::iterator it
- Stiva din pentru a afisa drumul de la sursa la destinatie si nu invers. Stiva este definita astfel: stack<int>s

3. Proiectarea implementarii

Pentru citirea datelor din fisier, am implementat o metoda readData(filename) unde filename este denumirea fisierului de intrare. In timpul citirii se salveaza corespunzator in dictionar adiacenta nodurilor.

Referitor la partea de algoritm, am implementat o metoda dijkstra() unde am avut nevoie in plus un vector de distante unde dist[i] inseamna distanta de la nodul sursa la nodul i, un vector de “parinti", unde prin parinti se intelege nodul anterior nodului curent. La inceput a trebuit sa fac o initializare a vectorului de parinti, precum si a vectorului de distante. In aceasta etapa am definit si coada cu prioritati care contine la inceput nodul sursa cu distanta 0. Algoritmul consta in adaugarea si scoaterea repetata din coada cu prioritati pana cand nu mai sunt noduri de parcurs (in algoritm cat coada nu este vida). Din coada cu prioritati se scoate nodul care contine muchia cu cel mai mic cost, se parcurg vecinii acestui nod, iar vecinul care se afla la cea mai scurta distanta se adauga in coada si se face un update la vectorul de parinti si la vectorul de distante.

Pentru afisarea solutiei am folosit vectorul de parinti, pe care l-am parcurs de la destinatie la sursa adaugand intr-o stiva fiecare nod pentru a afisa mai usor rezultatul conform cerintei.

Functia dijkstra() se poate vedea in figura de mai jos:

```
/*the algorithm*/
void dijkstra() {
    //init the vector of "fathers" and the vector of distances
    for (int i = 1; i <= n; i++) {
        dist[i] = MAX;
        p[i] = -1;
    }
    dist[source] = 0;
    priority_queue<Node>Q;
    Q.push(Node(source, dist[source]));
    while (!Q.empty()) {
        int node = Q.top().y;
        Q.pop();
        vector<Node>::iterator it;
        //foreach neighbour of the node from the top of priority_queue
        for (it = G[node].begin(); it != G[node].end(); it++) {
            if (dist[it->y] > dist[node] + it->cost) { //if there is a shorter path, update it and push that node to the priority queue
                dist[it->y] = dist[node] + it->cost;
                p[it->y] = node;
                Q.push(Node(it->y, dist[it->y]));
            }
        }
    }
    G->clear();
}
```

4. Analiza performanta

- Verificare corectitudine:

Input	Output
5 9 1 3 1 2 10 1 5 5	9 1 5 2 3

2 5 2 5 2 3 2 3 1 5 3 9 3 4 4 4 3 6 5 4 2	
5 9 3 1 1 2 10 1 5 5 2 5 2 5 2 3 2 3 1 5 3 9 3 4 4 4 3 6 5 4 2	There is no path between source: 3 and destination: 1

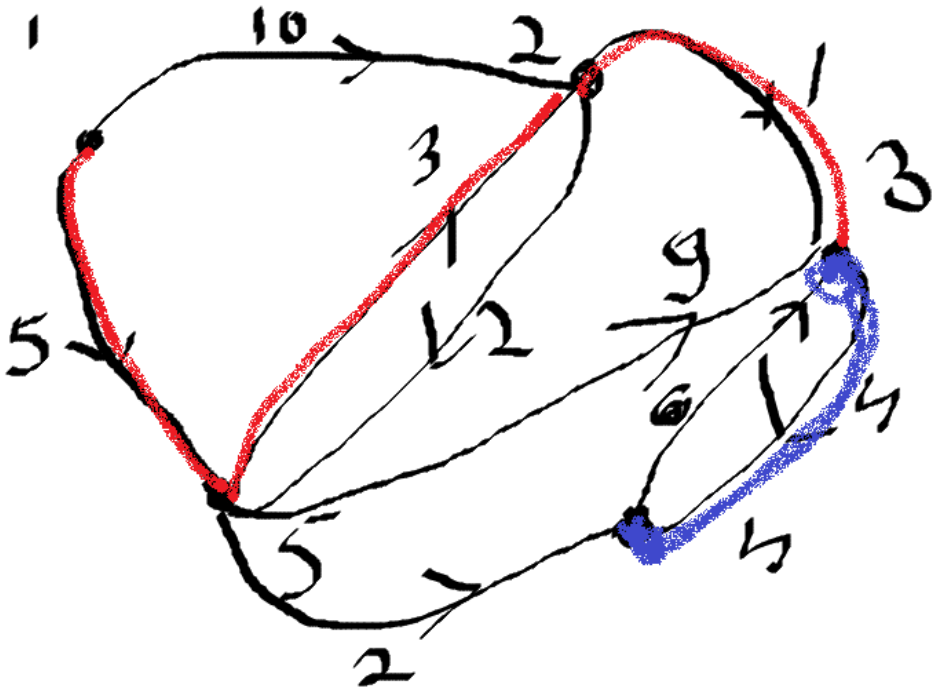
- Timpii de executie pentru aceste doua inputuri:

5	9	3	1	43
5	9	1	3	59

Unde fiecare coloana este descrisa la Reprezentarea datelor punctul B.

Algoritmul a fost testat si pe grafuri de dimensiuni mai mari, iar analiza performantei se poate verifica daca se intra pe github, in folderul Lab3/Dijkstra/Dijkstra in fisierul time.csv. Inputul a fost preluat din fisierele de input pentru laboratoarele propuse de Mihai Suciu la materia Algoritmica Grafurilor An 1 semestrul 2.

- Desen graf:



- Drumul cu rosu reprezinta solutia de la primul input
- Drumul cu albastru ar fi solutia de la al doilea input. Putem observa ca algoritmul se opreste la nodul 4, deoarece nu are al nod adiacent decat cel din care a plecat, fapt ce determina afisarea unui mesaj corespunzator.

5. Analiza teoretica a complexitatii

Avand in vedere ca am folosit structura de data, coada cu prioritati, acest lucru scade complexitatea destul de mult comparativ daca nu as fi folosit o coada cu prioritati, ci o coada normala.

Fie |E| numarul de varfuri si |V| numarul de arce din graf. Pentru orice structura de data folosita pentru reprezentarea lui Q, complexitatea algoritmului este O(|E| * Tdk + |V| * Tem) unde Tdk este complexitatea operatiei de key-decreasing, iar Tem este complexitatea operatiei de scoatere a minimului din Q. Deoarece am folosit o coada cu prioritati, Tem aici este O(1) si Tdk este O(1), dar operatia de adaugare in coada poate fi facuta in O(|V|). Prin urmare complexitatea teoretica este O(|E| + |V|^2). Cum in grafurile mai dense, |E| este mult mai mic decat |V|, putem spune ca valoarea complexitatii este O(|V|^2).