Laboratorul 12

Algoritmul Dijkstra. Algoritmul Prim.

Fiind dat un graf G = (V, E), se consideră funcția w: E → W, numită funcție de cost, care asociază fiecărei muchii o valoare numerică. Domeniul funcției poate fi extins, pentru a include şi perechile de noduri între care nu există muchie directă, caz în care valoarea este +∞ .

Costul minim al drumului dintre două noduri este minimul dintre costurile drumurilor existente între cele doua noduri.

Relaxarea unei muchii (v1, v2) constă în a testa dacă se poate reduce costul ei, trecând printr-un nod intermediar u. Fie w_{12} costul inițial al muchiei (v1, v2), w_{1u} costul muchiei (v1, u) şi w_{u2} costul muchiei (u, v2). Dacă w > w_{1u} + w_{u2}, muchia directă este înlocuită cu succesiunea de muchii (v1, u), (u, v2).

ALGORITMUL DIJKSTRA

Algoritmul Dijkstra poate fi folosit doar în grafuri care au toate muchiile nenegative.

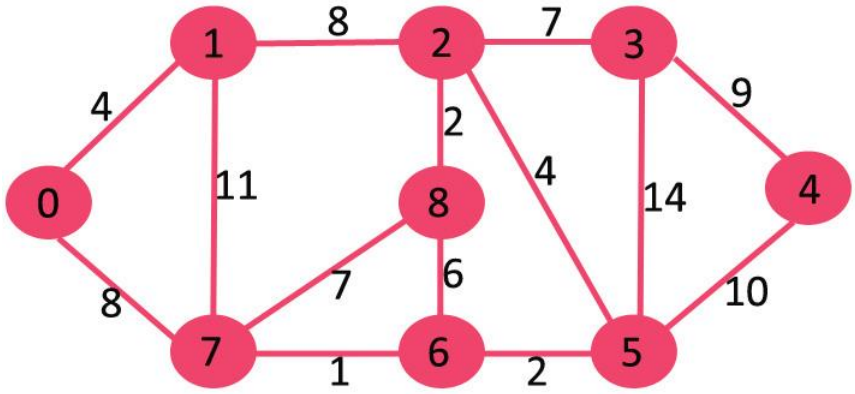
Optimul local căutat este reprezentat de costul drumului dintre nodul sursă s şi un nod v. Pentru fiecare nod se reține un cost estimat d[v], inițializat la început cu costul muchiei (s,v), sau cu +∞, daca nu există muchie. Algoritmul selectează, în mod repetat, nodul u care are, la momentul respectiv, costul estimat minim (față de nodul sursă). În continuare se încearcă să se relaxeze restul costurilor d[v]. Daca d[v] >= d[u] + w_{uv} , atunci d[v] ia valoarea d[u] + w_{uv}.

Pentru a ține evidența muchiilor care trebuiesc relaxate, se folosesc două structuri: una care conține lista de vârfuri deja vizitate şi o coada cu priorități – Min Heap (în care se află noduri nevizitate încă sau pentru care nu s-a determinat drumul minim). Nodurile se află ordonate după distanța față de sursă. Din Min Heap este mereu extras nodul aflat la distanța minimă față de nodul sursă. Complexitatea algoritmului pentru implementarea cu Min Heap este O(log V), pentru implementarea cu matrice de adiacență este $O(V^2)$, iar pentru cea cu liste de adiacență, O(ElogV).

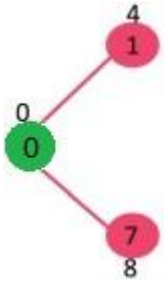
Paşii algoritmului:

1. Se crează un Min Heap de dimensiune V, unde V reprezintă numărul de vârfuri din graf. Fiecare nod al Min Heap-ului trebuie să conțina indexul vârfului şi distanța față de nodul sursă.
2. Se inițializează Min Heap-ul cu nodul sursă în rădăcină (distanța asociată nodului sursă este 0). Distanța asignată tuturor celorlalte vârfuri este INF.
3. Cât timp în Min Heap mai există elemente, se execută:
    a. Se extrage u, primul nod din Min Heap şi cel pentru care distanța față de nodul sursă este minimă.
    b. Pentru fiecare nod v adiacent cu u, se verifică dacă v este în Min Heap. Daca v este în Min Heap şi d[v]>w_{uv}+d[u], atunci se actualizează d[v]=w_{uv}+d[u].

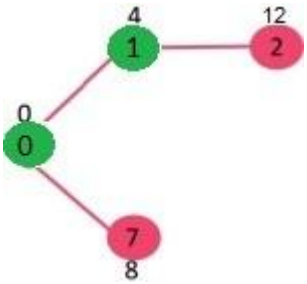
Fie următorul graf:



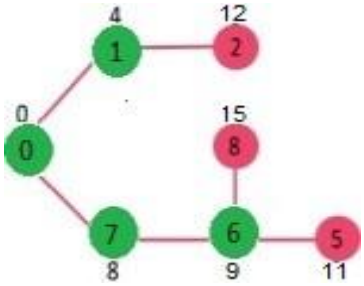
Nodul sursă este 0. Dorim să aflăm costurile drumurilor de lungime minimă de la nodul 0 la toate celelalte noduri din graf. Iniţial, d[0]=0, iar d[v]=INF, pentru toate celelalte vârfuri din graf. Nodul sursă (0) este extras din Min Heap şi se actualizează distanţele pentru vârfurile adiacente (1 şi 7). Astfel, d[1]=4 şi d[7]=8. Acum, Min Heap-ul conţine toate vârfurile grafului, mai puţin nodul 0 care a fost eliminat.



Se extrage nodul 1 din Min Heap si se actualizează distanţa pentru nodurile adiacente cu 1 (distanţa este actualizată dacă nodul se găseşte în Min Heap şi distanţa prin vârful 1 este mai mică decât distanţa anterioară).

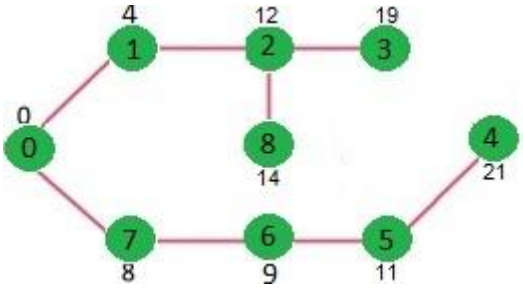


Se extrage nodul 7 din Min Heap şi se actualizează distanţele nodurilor adiacente cu el.



Se repetă paşii de mai sus cât timp se mai găsesc elemente în Min Heap.

La final, costurile drumurilor minime de la nodul sursă 0 la fiecare dintre nodurile grafului sunt urmatoarele:



ALGORITMUL PRIM

Dându-se un graf conex neorientat G =(V, E), se numeşte arbore de acoperire al lui G un subgraf G'=(V, E') care conţine toate vârfurile grafului G şi o submulţime minimă de muchii E'⊆ E cu proprietatea că uneşte toate vârfurile şi nu conţine cicluri. Cum G' este conex şi aciclic, el este arbore. Pentru un graf oarecare, există mai mulţi arbori de acoperire.

Dacă asociem o matrice de costuri, w, pentru muchiile din G, fiecare arbore de acoperire va avea asociat un cost egal cu suma costurilor muchiilor conţinute. Un arbore care are costul asociat mai mic sau egal cu costul oricărui alt arbore de acoperire se numeşte **arbore minim de acoperire** (minimum spanning tree) al grafului G. Un graf poate avea mai mulţi arbori minimi de acoperire. Dacă toate costurile muchiilor sunt diferite, există un singur AMA.

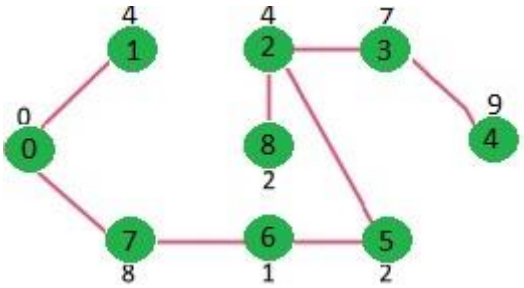
Algoritmul PRIM consideră ca există un arbore principal, iar la fiecare pas se adaugă acestuia muchia cu cel mai mic cost care uneşte un nod din arbore cu un nod din afara sa. Nodul rădăcină al arborelui principal se alege arbitrar. Când s-au adăugat muchii care ajung în toate nodurile grafului, s-a obţinut arborele minim de acoperire (sau arborele parţial de cost minim) dorit. Abordarea seamănă cu algoritmul Dijkstra de găsire a drumului minim între două noduri ale unui graf.

Pentru o implementare eficientă, următoarea muchie de adăugat la arbore trebuie să fie uşor de selectat. Vârfurile care nu sunt în arbore trebuiesc sortate în funcţie de distanţa până la acesta (de fapt costul minim al unei muchii care leagă nodul dat de un nod din interiorul arborelui). Se poate folosi pentru aceasta o structură de heap. Presupunând că (u, v) este muchia de cost minim care uneşte nodul u cu un nod v din arbore, se vor reţine două informaţii:

d[u] = w[u,v] - distanţa de la u la arbore
p[u] = v - predecesorul lui u în drumul minim de la arbore la u.

La fiecare pas se va selecta nodul u cel mai apropiat de arborele principal, reunind apoi arborele principal cu subarborele corespunzător nodului selectat. Se verifică apoi dacă există noduri mai apropiate de u decât de nodurile care erau anterior în arbore, caz în care trebuie modificate distanţele dar şi predecesorul. Modificarea unei distanţe impune şi refacerea structurii de heap. Complexitatea este O(log V).

Pentru exemplul anterior, arborele minim de acoperire este:



CERINŢE

1). Pornind de la Min Heap-ul studiat în Laboratorul 9 şi a cărui implementare vă este prezentată în fişierul MinHeap.h, completaţi funcţiile echivalente din heap.h, pentru lucrul cu heap-uri de noduri. Aveţi următoarele definiţii:

```
typedef struct MinHeapNode
{
   int  v;
   int d;
} MinHeapNode;


typedef struct MinHeap
{
   int size;
   int capacity;
   int *pos;
   MinHeapNode **elem;
} MinHeap;
```

MinHeapNode newNode(int v, int d)* – iniţializează un nou nod din Min Heap

(0.2 p)

MinHeap newQueue(int capacity)* – iniţializează Min Heap-ul (coada cu priorităţi)

(0.2 p)

*void swap(MinHeapNode** a, MinHeapNode** b)* – interschimbă doua noduri din Min Heap

(0.2 p)

void SiftDown(MinHeap h, int idx)* – actualizează Min Heap-ul şi poziţiile nodurilor care sunt interschimbate

(0.75 p)

int isEmpty(MinHeap h)* – întoarce 1 dacă Min Heap-ul este vid, 1 altfel

(0.2 p)

MinHeapNode removeMin(MinHeap* h)* – elimină şi întoarce elementul minim din Min Heap

(0.5 p)

void SiftUp(MinHeap h, int v, int d)*

(0.75 p)

*int isInMinHeap(MinHeap *h, int v)* – întoarce 1 dacă nodul v se găseşte în Min Heap şi 0 altfel

(0.2 p)

2). În fişierul graph.c, definiţi funcţia *void dijkstra(TGraphL G, int s)*, care afişează costurile tuturor drumurilor minime de la nodul de start s la toate celelalte noduri din graf. Pentru exemplul anterior, unde nodul de start este 0, funcţia va afişa:

Varf	Drum minim
0	0
1	4
2	12
3	19
4	21
5	11
6	9
7	8
8	14

(3 p)

3). În fişierul graph.c, definiţi funcţia *void Prim(TGraphL G)*, care determină arborele minim de acoperire al grafului G. Funcţia va afişa muchiile care fac parte din arborele minim de acoperire. Pentru exemplul anterior, se va afişa:

Parinte	Varf
0	1
5	2
2	3
3	4
6	5
7	6
0	7
2	8

(3 p)