Algoritmul Roy-Floyd

Pentru grafurile care nu au costuri asociate muchiilor (echivalent cu faptul că toate muchiile au același cost) determinarea unui drum minim între două noduri este echivalentă cu a determina un lanț cu număr minim de muchii între cele două noduri. Acest lucru este cunoscut că poate fi realizat folosind un algoritm `BFS`.

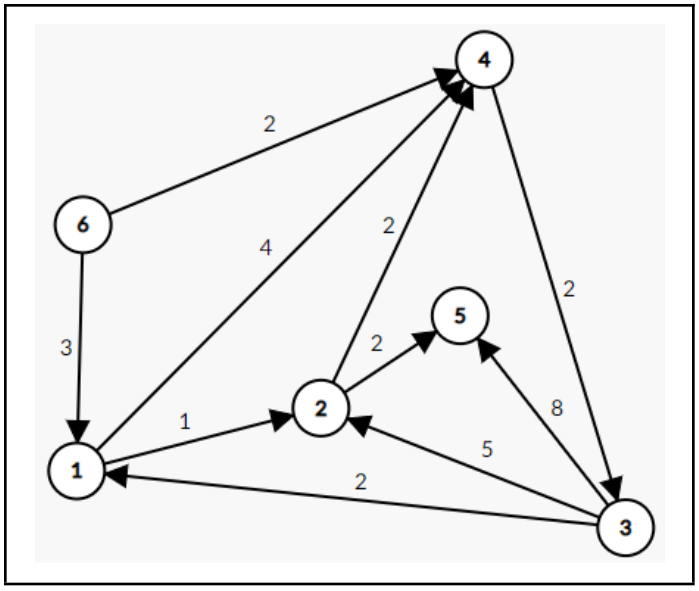
Pentru cazul în care muchiile au asociate costuri ne interesează un lanț în care suma costurilor muchiilor este minimă.

Cei mai populari algoritmi pentru a rezliza acest lucru sunt **Roy-Floyd**, **Dijkstra** și **Bellman-Ford**.

Algoritmii Dijkstra și Bellman-Ford permit găsirea unui drum de cost minim între un nod fixat și toate celelalte.

Algoritmul Roy-Floyd permite găsirea câte unui drum de cost minim între **oricare două noduri**, și din acest motiv el rulează cu un timp de calcul mai mare. Are în schimb avantajul că se scrie mult mai ușor. Vom încerca să explicăm în acest material modul de funcționare pentru acest algoritm.

Pentru prezentarea algoritmului vom folosi drept exemplu următorul graf:



La majoritatea algoritmilor de lucru cu grafuri memorarea folosind liste de vecini este cea care oferă performanțe și din punct de vedere al memoriei și din punct de vedere al timpului de executare.

În cazul determinării drumului minim între oricare două noduri noi avem nevoie oricum de memorie de ordin n^2 (pentru că numărul de perechi de noduri este de ordin n^2). Așadar vom folosi în acest caz memorarea prin matrice de adiacență.
Aici matricea de adiacență este puțin modificată.
`A[i][j]` va reprezenta costul muchiei directe de la nodul `i` la nodul `j`. Acolo unde nu este muchie directă se memorează de obicei o valoare care marchează acest lucru. Această matrice se numește “a costurilor” în loc de “de adiacență”.

Pentru graful din exemplu configurația ei este:

0	**1**	INF	**4**	INF	INF
INF	**0**	INF	**2**	**2**	INF
2	**5**	**0**	INF	**8**	INF
INF	INF	**2**	**0**	INF	INF
INF	INF	INF	INF	**0**	INF
3	INF	INF	**2**	INF	**0**

În contextul algoritmului Roy-Floyd vom da matricei costurilor următoarea interpretare:
`A[i][j]` = costul minim al unui drum de la nodul `i` la nodul `j` folosind doar muchii directe.

În continuare vom arăta că putem face transformări pas cu pas pentru matricea `A` astfel încât la final elementele sale să aibă semnificația:
`A[i][j]` = costul minim al unui drum de la nodul `i` la nodul `j` folosind intermediare orice noduri ale grafului.

Vom face acest lucru în `n` pași.

Pornim așadar de la matricea costurilor.
La pasul `1` vom lua în calcul ca intermediar nodul `1`, adică dacă între două noduri `i` și `j` avem muchie de la `i` la `1` și muchie de la `1` la `j` și suma costurilor celor două muchii este mai mică decât muchia directă de la `i` la `j`, vom actualiza valoarea `A[i][j]`.
Acest efect se obține rulând codul următor:

```
for (i=1;i<=n;i++)
    for (j=1;j<=n;j++)
        if (A[i][j] > A[i][1] + A[1][j])
            A[i][j] = A[i][1] + A[1][j];
```

Pentru a ne concentra pe detaliile de implementare specifice algoritmului vom considera că inițial, acolo unde nu este muchie de la nodul `i` la nodul `j` vom avea `A[i][j] = INF` (o valoare foarte mare, pentru că noi ne dorim ca în elementele matricei să construim niște minime). De asemenea, este în spiritul cerinței să memorăm `A[i][i] = 0` (distanța minimă de la un nod la el însuși este `0`).

În urma rulării acestui cod asupra matricei `A` se vor modifica valorile `A[3][2]` și `A[6][2]`, `A[3][4]`.

Să explicăm:
`A[3][2]` avea înainte valoarea `5` (costul muchiei directe de la `3` la `2`). Prin codul de mai sus se detectează că `A[3][1] + A[1][2] = 2 + 1 = 3` face să se obțină între `3` și `2` un drum mai scurt.
De asemenea, între `6` și `2`, înainte era `INF` iar acum, prin intermediul lui `1` avem `A[6][1] + A[1][2] = 3 + 1 = 4`. Deci distanța de la `6` la `2` se îmbunătățește, devenind `4`.
`A[3][4]` devine `6`, în loc de `INF` (`A[3][1] + A[1][4]`)

De exemplu la acest pas `A[6][4]` nu se schimbă (Înainte `A[6][4]` era `2` și încercarea `A[6][1] + A[1][4] = 3+4` dă `7` deci nu este o valoare mai bună decât cea anterioară).

Matricea costurilor devine:

0	**1**	INF	**4**	INF	INF
INF	**0**	INF	**2**	**2**	INF
2	**3**	**0**	**6**	**8**	INF
INF	INF	**2**	**0**	INF	INF
INF	INF	INF	INF	**0**	INF
3	**4**	INF	**2**	INF	**0**

Pasul 2:
Avem deja matricea `A` transformată, deci pentru fiecare pereche de noduri `A[i][j]` = costul minim al unui drum de la `i` la `j` folosind muchia directă și eventual intermediar pe `1`.
Aplicăm asupra ei un cod similar dar având acum intermediar pe `2`.

```
for (i=1;i<=n;i++)
    for (j=1;j<=n;j++)
        if (A[i][j] > A[i][2] + A[2][j])
            A[i][j] = A[i][2] + A[2][j];
```

Cu același raționament ca anterior deducem că acum vom obține drumuri minime luând ca intermediar și pe `2`. Atenție, nu sunt drumuri minime luând ca interemediar doar pe `2` ci luând ca intermediari pe `1` și pe `2` (întrucât aplicăm acest cod pe matricea deja transformată prin intermediul lui `1`).

Se obține:

0	**1**	INF	**3**	**3**	INF
INF	**0**	INF	**2**	**2**	INF
2	**3**	**0**	**5**	**5**	INF
INF	INF	**2**	**0**	INF	INF
INF	INF	INF	INF	**0**	INF
3	**4**	INF	**2**	**6**	**0**

Elementele evidențiate sunt cele care s-au modificat.

Vom descrie în detaliu:

- `A[1][4]` era anterior `4` (costul muchiei `1,4`) dar acum luăm ca intermediar și pe `2` și `A[1][2] + A[2][4] = 3`, deci obținem un cost mai bun.
- `A[1][5]` era anterior `INF`, dar acum avem muchie și de la `1` la `2` și de la `2` la `5`, deci îmbunătățim drumul de la `1` la `5` prin intermediul lui `2`.
- `A[3][4]` era anterior `6`. Această valoare s-a actualizat și când s-a utilizat intermediar `1`. Acum `A[3][2] = 3`, `A[2][4] = 2`, deci `A[3][4]` devine `5`. Observăm că această valoare s-a obținut chiar luând în calcul și pe `1` și pe `2` ca intermediari (de exemplu dacă se lua în calcul doar `2` am fi obținut drumul cu muchiile `3,2` și `2,4` care avea costul `7`, deci mai mare).

Acum este evident că nu ne rămâne decât să introducem nodurile ca intermediar unul câte unul, fiecare aplicându-se asupra matricei modificată de cele anterioare.

Secvențele de cod prezentate anterior pentru `1` și `2` au doar scop demonstrativ. Noi vom itera cu un `k` intermediar de la `1` la `n` și vom aplica acest cod pentru fiecare `k`. Astfel, în afară de cele două foruri prin care punem în evidență toate perechile de noduri, vom mai avea unul care pune în evidență intermediarul.

Obținem:

```
for (k=1;k<=n;k++)
    for (i=1;i<=n;i++)
        for (j=1;j<=n;j++)
```

```
            if (A[i][j] > A[i][k] + A[k][j])
                A[i][j] = A[i][k] + A[k][j];
```

Configurația finală a matricei `A` este:

0	**1**	**5**	**3**	**3**	INF
6	**0**	**4**	**2**	**2**	INF
2	**3**	**0**	**5**	**5**	INF
4	**5**	**2**	**0**	**7**	INF
INF	INF	INF	INF	**0**	INF
3	**4**	**4**	**2**	**6**	**0**

Astfel, aceasta s-a transformat dintr-o matrice a costurilor (ce era inițial) într-o matrice a distanțelor minime (`A[i][j]` = costul minim al unui drum de la nodul `i` la nodul `j`).

Se observă că pe linia `5` avem doar valori `INF` întrucât din `5` nu se poate ajunge în niciun nod (în `5` doar se intră), iar pe coloana `6` avem doar `INF` întrucât în `6` nu se poate ajunge din niciun nod (din `6` doar se iese).

Determinarea unui drum de cost minim între două noduri oarecare
Anterior am determinat doar costul minim al drumurilor iar acum arătăm o modalitate prin care găsim și nodurile componente ale unui drum de cost minim.

Vom folosi o matrice suplimentară `T` în care `T[i][j]` semnifică: nodul aflat imediat înaintea lui `j` pe drumul minim de la `i` la `j`.

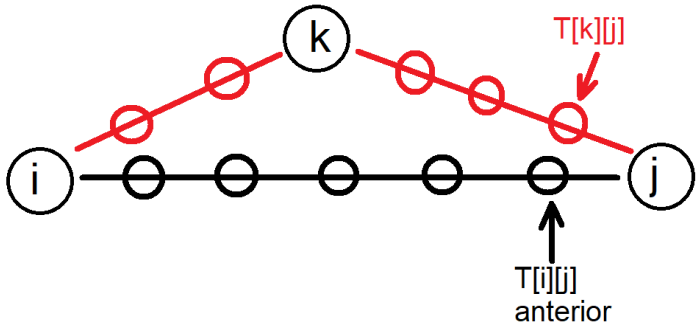
Ținând cont că la început `A` reprezintă matricea costurilor (echivalent: matricea drmurilor minime folosind doar muchii directe),

vom inițializa:

`T[i][j]` =	`i`, dacă există muchia `i,j` (deci fiind vorba doar de drumul format din `i` și `j`, nodul aflat imediat înaintea lui `j` pe drumul minim de la `i` la `j` este chiar `i`)
	Valoare nedefinită dacă nu avem muchie de la `i` la `j`

Faptul că la un moment dat găsim un intermediar `k` pentru care `A[i][k] + A[k][j] < A[i][j]` are semnificația: este mai scurt un drum de la `i` la `j` care trece prin `k`. Deci, acest

drum se compune din drumul minim de la `i` la `k` care se concatenează cu drumul minim de la `k` la `j`. Așadar, drumul minim de la `k` la `j` devine ultima porțiune a drumului minim de la `i` la `j`. Deci ultimul nod înaintea lui `j` pe drumul minim de la `i` la `j` va deveni chiar ultimul nod de dinaintea lui `j` pe drumul minim de la `k` la `j`. Așadar, `T[i][j] = T[k][j]`.



Algoritmul devine:

```
fin>>n>>m;
for (i=1;i<=n;i++)
    for (j=1;j<=n;j++) {
        A[i][j] = 100;
        if (i == j)
            A[i][j] = 0;
    }

for (i=1;i<=m;i++) {
    fin>>x>>y>>cost;
    A[x][y] = cost;
    T[x][y] = x;
}

for (k=1;k<=n;k++)
    for (i=1;i<=n;i++)
        for (j=1;j<=n;j++)
            if (A[i][j] > A[i][k] + A[k][j]) {
                A[i][j] = A[i][k] + A[k][j];
                T[i][j] = T[k][j];
            }
```

În codul de mai sus am ales să facem și preluarea datelor de intrare, urmată de inițializarea celor două tablouri (cel de costuri și cel de “tați”).

Odată construit `T`, pentru a determina drumul minim între nodurile `i` și `j` putem folosi următoarea secvență:

```
while (j!=0) {
    cout<<j<<” ”;
    j = T[i][j];
}
```

Ne-am bazat pe faptul că `T[i][i]` rămâne cu valoarea `0`.
Secvența de mai sus permite obținerea nodurilor de pe drum în ordinea de la `j` la `i`. Pentru a le obține în ordinea dorită avem varianta să le stocăm într-un tablou și apoi să le afișăm invers sau varianta de a folosi o funcție recursivă și să afișăm la revenire, ca mai jos:

```
void drum(int i, int j) {
    if (j!=0) {
        drum(i, T[i][j]);
        cout<<j<<” ”;
    }
}
```