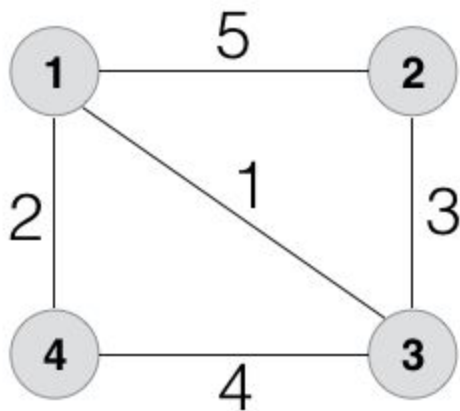


1. Алгоритъмът на Floyd Warshall се използва за намиране на най-преките пътища от връх до връх в претеглен граф. Той е със сложност  $n^3$  и при него както при Ford могат да се намерят отрицателни цикли и съответно да може да се каже дали алгоритъмът може да се използва за даден претеглен граф. Сравнявайки го с Дейкстра откриваме, че използвайки Дейкстра от всеки до всеки връх получаваме сложност  $O(V^2 + V^2 \lg(V))$ , която съответно показва, че Floyd Warshall е по-бързият в конкретната ситуация. Други + -ове на алгоритъмът са работата с отрицателни пътища, сравнително лесна имплементация.
2. Имплементация:

Нека имаме дадения граф



Използваме следните наименования:

We will use the following notations:

$k$  = Iteration number

$D_k$  = Distance table in  $k^{\text{th}}$  iteration

$S_k$  = Sequence table in  $k^{\text{th}}$  iteration

$d_{ij}$  = The distance between vertex  $i$  and  $j$

there are 4 vertices in the graph so our tables will have 4 rows and 4 columns

iteration = 0     $k = 0$

$D_0$	1	2	3	4
1				
2				
3				
4				

Distance table

$S_0$	1	2	3	4
1				
2				
3				
4				

Sequence table

Напълваме таблица D със стойностите на колоната

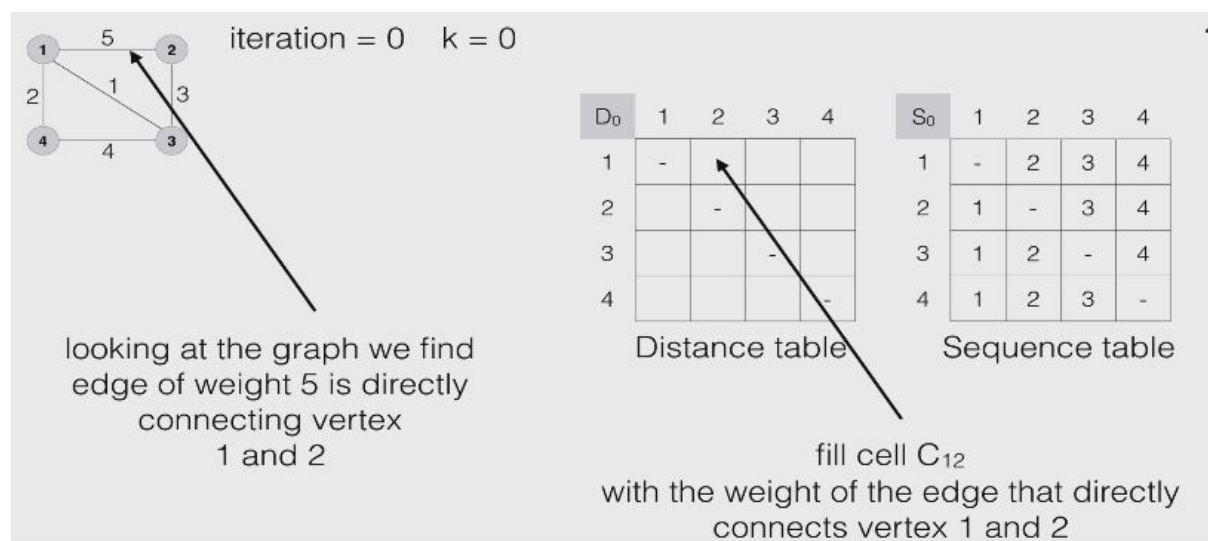
$D_0$	1	2	3	4
1				
2				
3				
4				

Distance table

$S_0$	1	2	3	4
1	-	2	3	4
2	1	-	3	4
3	1	2	-	4
4	1	2	3	-

Sequence table

Сега е ред на напълним временните таблици



Преди първата итерация където имаме връзка се попълва с дължината на съответния път където не слагаме никаква максимална стойност (MAX INTEGER) нека я приемем на таблицата за безкрайност!

D <sub>0</sub>	1	2	3	4
1	-	5		
2	5	-		
3			-	
4				-

Distance table

S <sub>0</sub>	1	2	3	4
1	-	2	3	4
2	1	-	3	4
3	1	2	-	4
4	1	2	3	-

Sequence table

iteration = 0   k = 0

looking at the graph we find that there is no edge that directly connects vertex 2 and 4  
so we will write infinity ( $\infty$ ) in cell C<sub>24</sub> and C<sub>42</sub>

D <sub>0</sub>	1	2	3	4
1	-	5	1	2
2	5	-	3	
3	1	3	-	
4	2			-

Distance table

S <sub>0</sub>	1	2	3	4
1	-	2	3	4
2	1	-	3	4
3	1	2	-	4
4	1	2	3	-

Sequence table

fill cell C<sub>24</sub>  
with the weight of the edge that directly connects vertex 2 and 4

След което влизаме в три цикъла - първия върти итерациите (ще ги погледна по късно по-подробно) , вторият колоните и третият редовете и вече сме напълнили постоянните таблици.

Така сега следващия етап е презапишем съответния ред и колона отговаряща на съответната итерация

D <sub>0</sub>	1	2	3	4
1	-	5	1	2
2	5	-	3	$\infty$
3	1	3	-	4
4	2	$\infty$	4	-

Distance table

S <sub>0</sub>	1	2	3	4
1	-	2	3	4
2	1	-	3	4
3	1	2	-	4
4	1	2	3	-

Sequence table

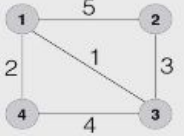
D <sub>1</sub>	1	2	3	4
1	-	5	1	2
2	5	-		
3	1		-	
4	2			-

S <sub>1</sub>	1	2	3	4
1	-	2	3	4
2	1	-		
3	1		-	
4	1			-

$$d_{ij} > d_{ik} + d_{kj}$$

Ако съответното условие е изпълнено съответната стойност на разстоянието  $S$  се заменя със сбора на горепосочените 2 и стойността на предшественика се слага тази на съответната итерация.

iteration = 1    k = 1



Fill the cell  $C_{23}$  as per the following condition:

we have  
 $i = 2, j = 3$  and  $k = 1$

therefore,  
 Is  $d_{ij} > d_{ik} + d_{kj}$  [in  $D_0$ ]  
 Is  $d_{23} > d_{21} + d_{13}$   
 Is  $3 > 5 + 1$   
 NO

So,  $C_{23} = d_{23} = 3$

**Distance table  $D_0$**

	1	2	3	4
1	-	5	1	2
2	5	-	3	$\infty$
3	1	3	-	4
4	2	$\infty$	4	-

**Sequence table  $S_0$**

	1	2	3	4
1	-		3	4
2	1	-		
3	1	2	-	
4	1	2	3	-

**Distance table  $D_1$**

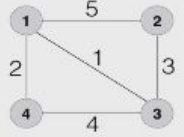
	1	2	3	4
1	-	5	1	2
2	5	-		
3	1		-	
4	2			-

**Sequence table  $S_1$**

	1	2	3	4
1	-		3	4
2	1	-		
3	1		-	
4	1			-

Тук нямаме нужда от замяна на стойността и преписваме старата от горната таблица

iteration = 1    k = 1



Fill the cell  $C_{24}$  as per the following condition:

we have  
 $i = 2, j = 4$  and  $k = 1$

therefore,  
 Is  $d_{ij} > d_{ik} + d_{kj}$  [in  $D_0$ ]  
 Is  $d_{24} > d_{21} + d_{14}$   
 Is  $\infty > 5 + 2$   
 YES

**Distance table  $D_0$**

	1	2	3	4
1	-	5	1	2
2	5	-	3	$\infty$
3	1	3	-	4
4	2	$\infty$	4	-

**Sequence table  $S_0$**

	1	2	3	4
1	-		3	4
2	1	-		
3	1	2	-	
4	1	2	3	-

**Distance table  $D_1$**

	1	2	3	4
1	-	5	1	2
2	5	-	3	
3	1	3	-	
4	2			-

**Sequence table  $S_1$**

	1	2	3	4
1	-		3	4
2	1	-	3	
3	1	2	-	
4	1			-

Тук вече имаме нужда от замяна и тъй като сме в първата итерация поставяме 1 в таблицата с предшествениците

D <sub>0</sub>	1	2	3	4
1	-	5	1	2
2	5	-	3	∞
3	1	3	-	4
4	2	∞	4	-

Distance table

S <sub>0</sub>	1	2	3	4
1	-	2	3	4
2	1	-	3	4
3	1	2	-	4
4	1	2	3	-

Sequence table

След което двете временни таблици се презаписват с постоянните и започваме итерация 2

D <sub>1</sub>	1	2	3	4
1	-	5	1	2
2	5	-	3	7
3	1	3	-	3
4	2	7	3	-

Distance table

S <sub>1</sub>	1	2	3	4
1	-	2	3	4
2	1	-	3	1
3	1	2	-	1
4	1	1	1	-

Sequence table

Вече избираме колона 2 (1 в програмата :))

И процедурата се повтаря докато не стигнем последната итерация (последна ред и колона)

D <sub>3</sub>	1	2	3	4
1	-	4	1	2
2	4	-	3	6
3	1	3	-	3
4	2	6	3	-

Distance table

S <sub>3</sub>	1	2	3	4
1	-	3	3	4
2	3	-	3	3
3	1	2	-	1
4	1	3	1	-

Sequence table

D <sub>4</sub>	1	2	3	4
1	-			
2		-		
3			-	
4				-

S <sub>4</sub>	1	2	3	4
1	-			
2		-		
3			-	
4				-

След което създаваме резултатна таблица и следваме пътищата до като дестинацията което сме задали не съвпадне с тази в проследяваната таблица(тази с дестинциите)

Ако нямаме промяна в таблицата след дадена итерация тоест ако резултатната и временна таблици съвпадат може да прекъснем програмата тъй като промяна няма да настъпи и съответно сме открили най-преките пътища!



Важно е да вмъкна ,че ако направи проба отново за итерация 1 и след като сме изпълнили програмата и има разлики между временната и постоянната таблици значи най-прекият път не може да бъде открит!

Код:

```
#include <stdio.h>

#define INF 99999
#define MAX 10

void display(int arr[][MAX], int size);
void floyds(int D[][MAX], int S[][MAX], int size);

int main(void) {

    //distance array
    /*
        we have created a distance array of size 10x10 (MAX x
MAX)
        but we will be using only 4x4 as the graph has 4
vertices
    */
```



```

int D[MAX][MAX] = {
    {INF, 5, 1, 2},
    {5, INF, 3, INF},
    {1, 3, INF, 4},
    {2, INF, 4, INF}
};

//sequence array
/*
    we have created a sequence array of size 10x10 (MAX x
MAX)
    but we will be using only 4x4 as the graph has 4
vertices
*/
int S[MAX][MAX] = {
    {INF, 2, 3, 4},
    {1, INF, 3, 4},
    {1, 2, INF, 4},
    {1, 2, 3, INF}
};

int size = 4;          //total number of vertices in the graph

floyds(D, S, size);

return 0;
}

```

```

void floyds(int D[][MAX], int S[][MAX], int size) {
    int i, j, k, l;

    //arrays to hold data for current iteration
    /*
        we have created arrays of size 10x10 (MAX x MAX)
        but we will be using only 4x4 as the graph has 4
vertices
*/

```

```
int Dk[MAX][MAX], Sk[MAX][MAX];
```

```
//set Dk and Sk to 0
```

```
for(i = 0; i < size; i++) {  
    for(j = 0; j < size; j++) {  
        if(i == j) {  
            Dk[i][j] = INF;  
            Sk[i][j] = INF;  
        } else {  
            Dk[i][j] = 0;  
            Sk[i][j] = 0;  
        }  
    }  
}
```

```
//iteration
```

```
/*
```

```
    since array indexing start from 0 in C programming  
    so, we are setting i,j,k,l to 0
```

```
*/
```

```
for(k = 0; k < size; k++) {
```

```
    //step 1:
```

```
    //for each iteration we copy the kth row and kth column
```

to

```
    //the current array
```

```
    for(l = 0; l < size; l++) {
```

```
        //copy row
```

```
        Dk[k][l] = D[k][l];
```

```
        Sk[k][l] = S[k][l];
```

```
        //copy column
```

```
        Dk[l][k] = D[l][k];
```

```
        Sk[l][k] = S[l][k];
```

```
    }
```

```

//step 2:
//compute the distance and sequence value for
//current iteration
for(i = 0; i < size; i++) {

    //for kth iteration we skip the kth row
    if(i == k) {
        continue;
    }

    for(j = 0; j < size; j++) {

        //for kth iteration we skip the kth column
        if(j == k) {
            continue;
        }
        //if i and j are same i.e., referring to same
vertex we skip it
        if(i == j) {
            continue;
        }

        //checking
        if(D[i][j] > D[i][k] + D[k][j]) {
            Dk[i][j] = D[i][k] + D[k][j];
            Sk[i][j] = (k+1);    //kth iteration,
as indexing starts from 0 so, we add 1
        } else {
            Dk[i][j] = D[i][j];
            Sk[i][j] = S[i][j];
        }
    }
}

//step 3:
//copy content of Dk and Sk to D and S
for(i = 0; i < size; i++) {

```

```

        for(j = 0; j < size; j++) {
            D[i][j] = Dk[i][j];
            S[i][j] = Sk[i][j];
        }
    }

}

//print the distance array and sequence array result
printf("Distance array: \n");
display(D, size);
printf("Sequence array: \n");
display(S, size);
}

void display(int arr[][MAX], int size) {
    int i, j;
    for(i = 0; i < size; i++) {
        for(j = 0; j < size; j++) {
            if(arr[i][j] == INF) {
                printf("%10s ", "INF");
            } else {
                printf("%10d ", arr[i][j]);
            }
        }
        printf("\n");
    }
}

```