МІНІСТЕРСТВО ОСВІТИ І НАУКИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Інститут комп'ютерних наук та інформаційних технологій

Кафедра систем штучного інтелекту

Лабораторна робота №5
з курсу “Дискретна математика ”

Виконав:
ст. гр. КН-110
Холод Ігор

Викладач:
Мельникова Н.І.

Львів – 2018

Тема: Знаходження найкоротшого маршруту за алгоритмом Дейкстри. Плоскі планарні графи

Мета роботи: набуття практичних вмінь та навичок з використання алгоритму Дейкстри.

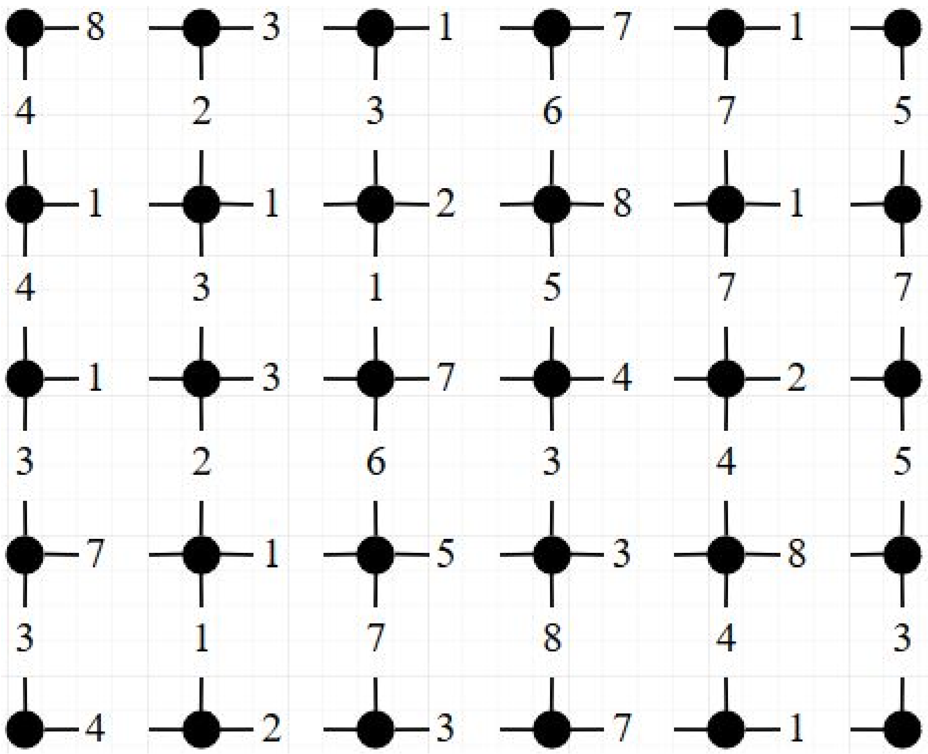
Теоретичні відомості:
Задача знаходження найкоротшого шляху з одним джерелом полягає у знаходженні найкоротших(мається на увазі найоптимальніших за вагою) шляхів від деякої вершини(джерела) до всіх вершин графа G . Для розв'язку цієї задачі використовується «жадібний» алгоритм, який називається алгоритмом Дейкстри.
«Жадібними» називаються алгоритми, які на кожному кроці вибирають оптимальний із можливих варіантів.

Плоскі і планарні графи
Плоским графом називається граф, вершини якого є точками площини, а ребра – безперервними лініями без самоперетинань, що з'єднують відповідні вершини так, що ніякі два ребра не мають спільних точок крім інцидентної їм обом вершини. Граф називається планарним, якщо він є ізоморфним плоскому графу.
Гранню плоского графа називається максимальна по включенню множина точок площини, кожна пара яких може бути з'єднана жордановою кривою, що не перетинає ребра графа. Границею грані будемо вважати множину вершин і ребер, що належать цій грані.

Алгоритм γ укладання графа G являє собою процес послідовного приєднання до деякого укладеного підграфа $\overline{G}$ графа G нового ланцюга, обидва кінці якого належать $\overline{G}$. При цьому в якості початкового плоского графа $\overline{G}$ вибирається будь-який простий цикл графа G . Процес продовжується доти, поки не буде побудовано плоский граф, ізоморфний графові G , або приєднання деякого ланцюга виявиться неможливим. В останньому випадку граф G не є планарным.

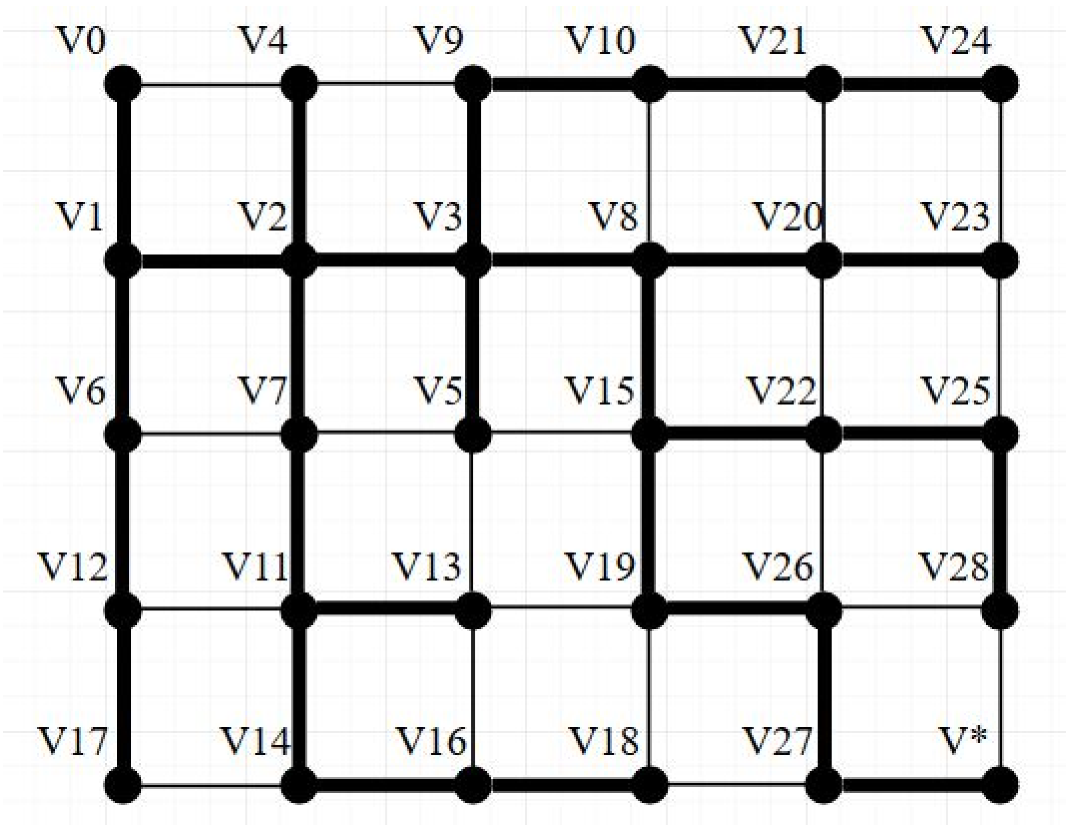
Завдання:

1)За допомогою алгоритму Дейкстра знайти найкоротший шлях у графі поміж парою вершин V_0 і V^* .



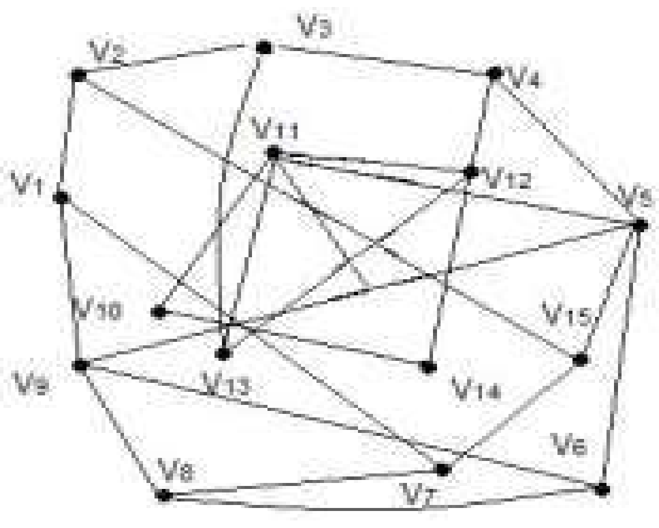
Будемо позначати найближчі вершини v_1, v_2, v_3, ... у порядку їхньої появи.

$l(v_1) = 4$, $l(v_2) = 5$, $l(v_3) = 6$, $l(v_4) = 7$, $l(v_5) = 7$, $l(v_6) = 8$, $l(v_7) = 8$, $l(v_8) = 8$, $l(v_9) = 9$, $l(v_{10}) = 10$, $l(v_{11}) = 10$, $l(v_{12}) = 11$, $l(v_{13}) = 11$, $l(v_{14}) = 11$, $l(v_{15}) = 13$, $l(v_{16}) = 13$, $l(v_{17}) = 14$, $l(v_{18}) = 16$, $l(v_{19}) = 16$, $l(v_{20}) = 16$, $l(v_{21}) = 17$, $l(v_{22}) = 17$, $l(v_{23}) = 17$, $l(v_{24}) = 18$, $l(v_{25}) = 19$, $l(v_{26}) = 19$, $l(v_{27}) = 23$, $l(v_{28}) = 24$, $l(v^*) = 24$.

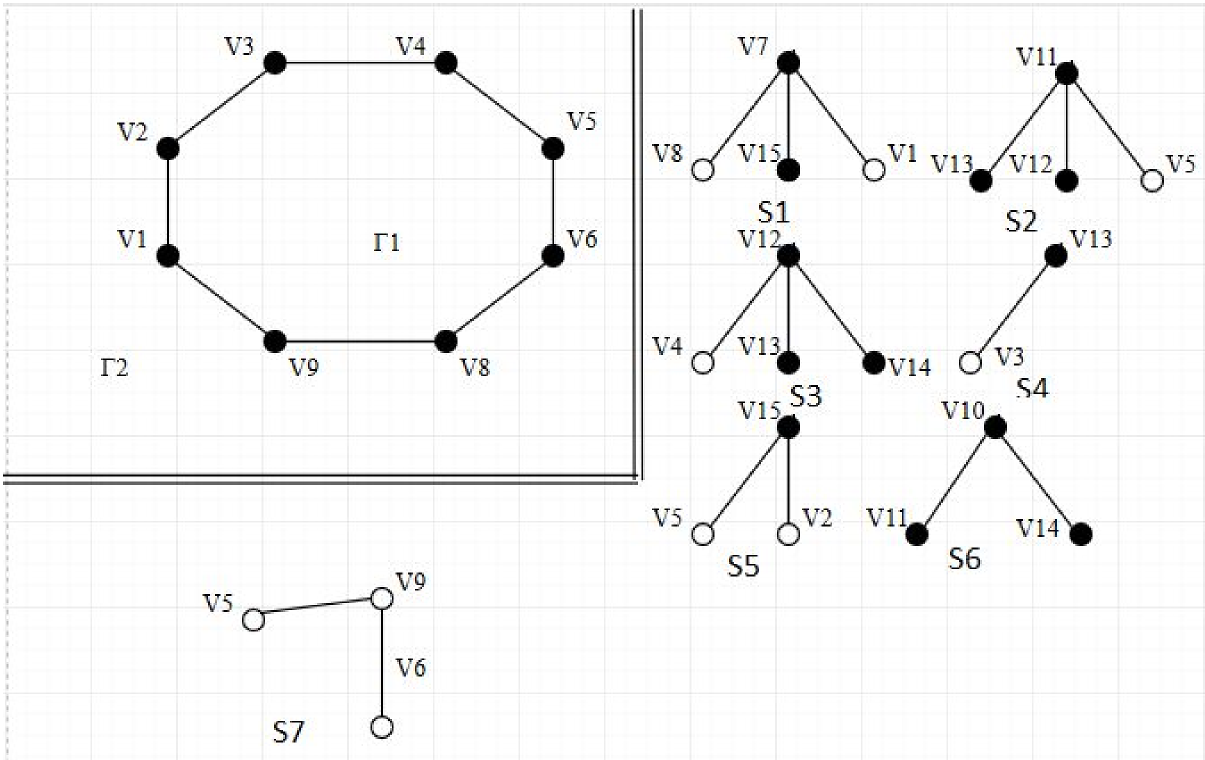


Дерево найближчих вершин виділено на рисунку жирними лініями і є кістяковим деревом, тому що містить усі вершини графа. Шуканий найкоротший ланцюг: $[v_0,\ v_1,\ v_2,\ v_3,\ v_8,\ v_{15},\ v_{19},\ v_{26},\ v_{27},\ v^*]$, довжина ланцюга $l = l(v^*) = 24$.

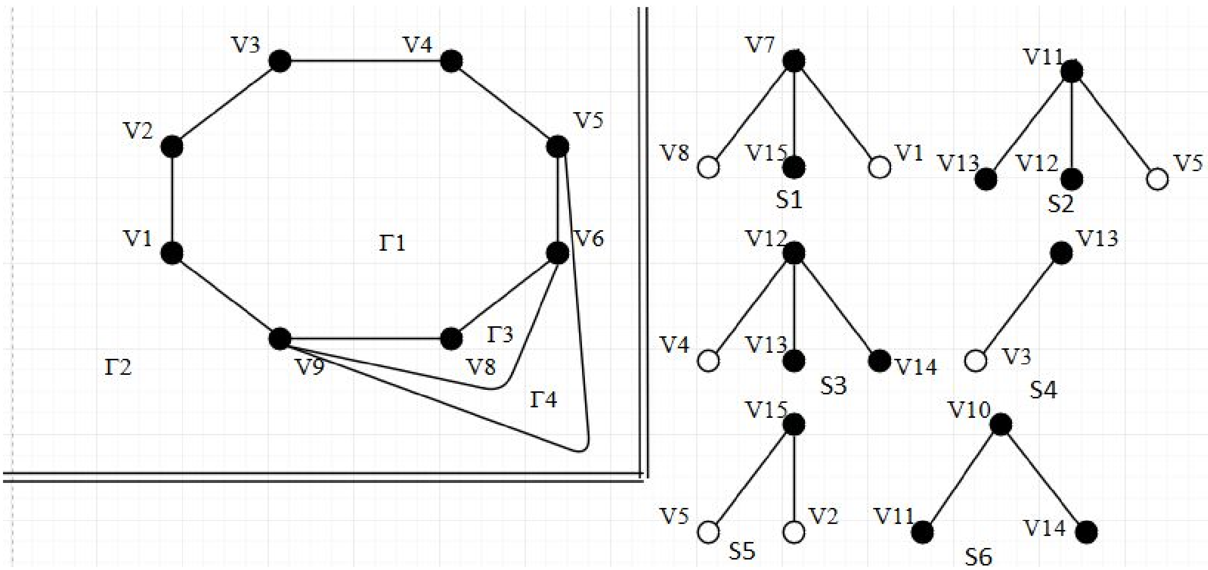
За допомогою γ -алгоритма зробити укладку графа у площині, або довести що вона неможлива.



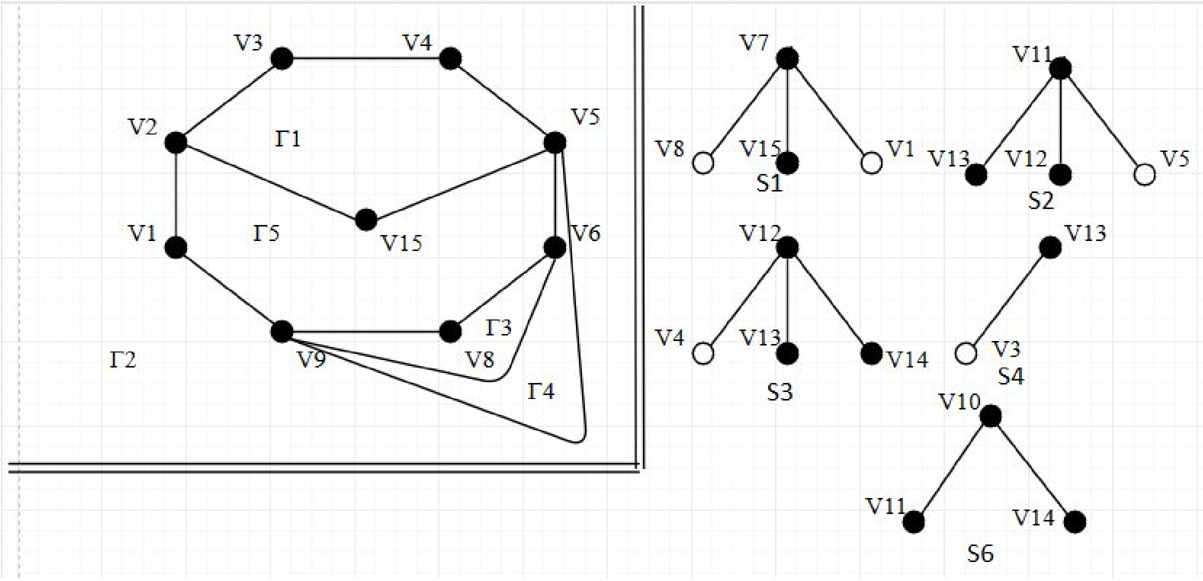
Розв’язання: Укладемо спочатку цикл [1, 2, 3, 4, 5, 6, 8, 9] що розіб’є площину на дві грані Г1 та Г2. Запишемо сегменти S1, S2, S3, S4, S5, S6 та S7.



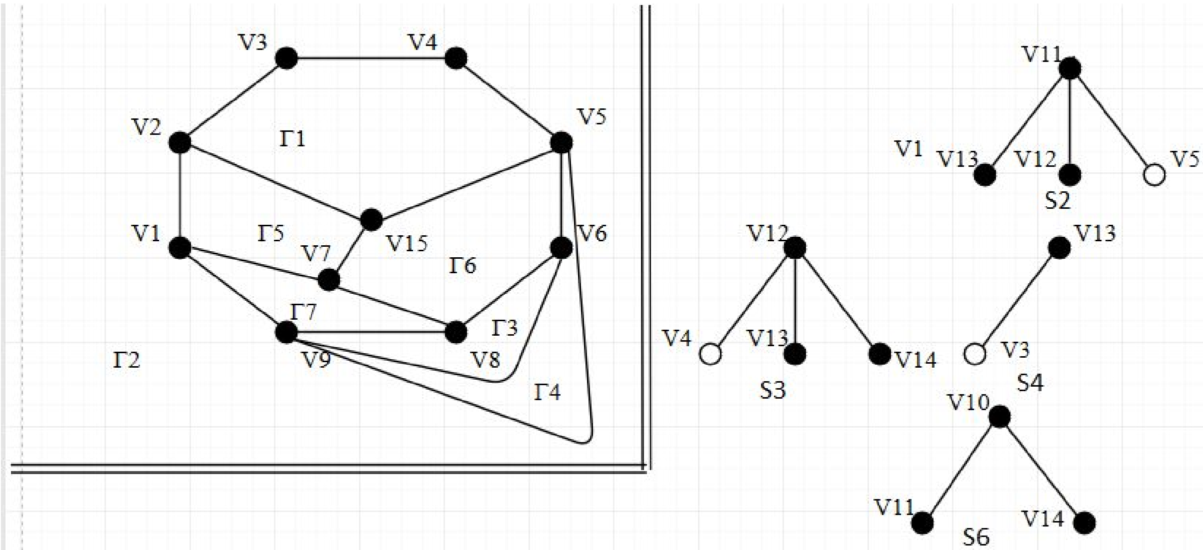
Укладемо сегмент S7 в грань Г2. Таким чином Г2 буде розбита на грані Г2, Г3 та Г4.



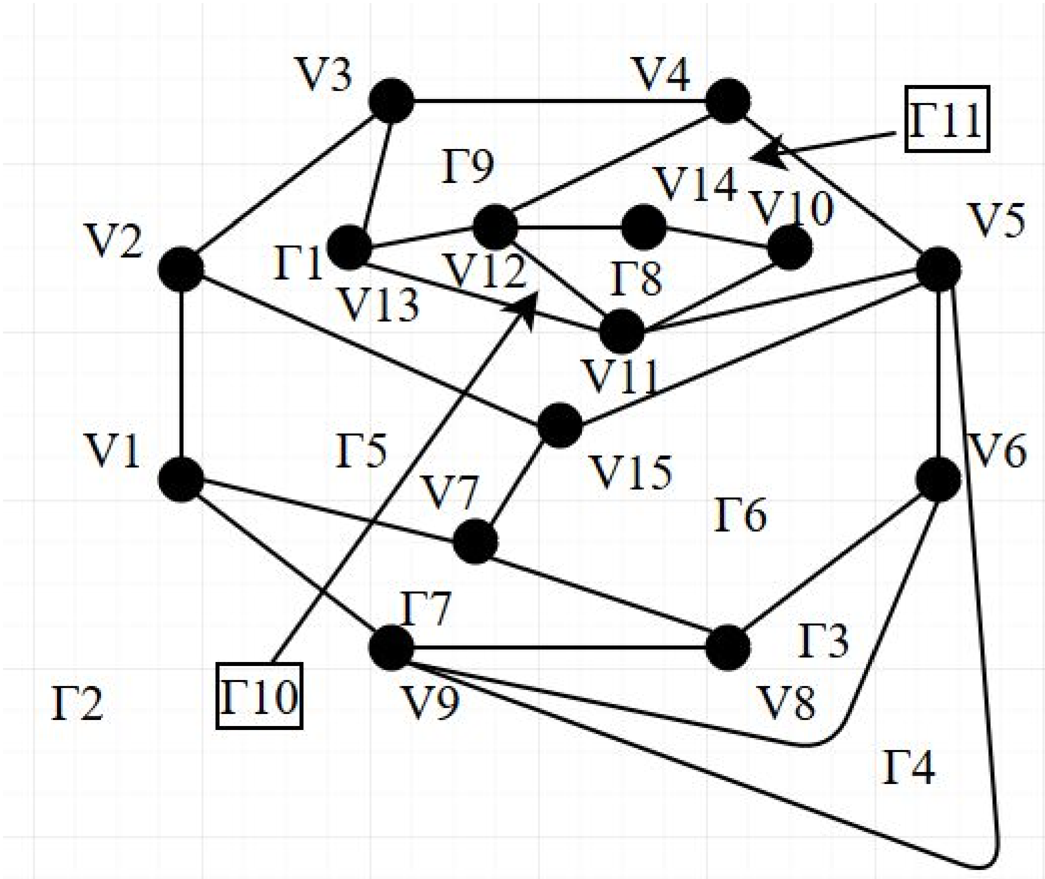
Тоді укладемо сегмент S5 у грань Г1, яка розіб’ється на грані Г1 та Г5.



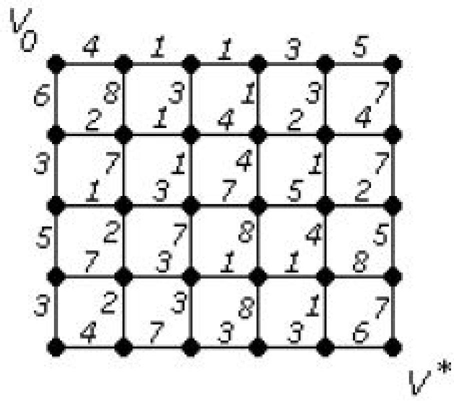
Укладаємо сегмент S1 у грань Г5. Тоді ця грань розіб'ється на грані Г5, Г6 та Г7.



Таким самим чином послідовно укладемо решту сегментів у грань Г1, які в кінцевому результаті розіб'ють цю грань на Г1, Г8, Г9, Г10 та Г11. Одержуємо укладання графа на площині.



Завдання 2:Написати програму, яка реалізує алгоритм Дейкстри знаходження найкоротшого шляху між парою вершин у графі. Протестувати розроблену програму на графі згідно свого варіанту.



Код програми:

```
#include<stdio.h>

#define INFINITY 9999
#define MAX 30

void dejkstra(int G[MAX][MAX], int n, int startnode);

int main()
```

```
{
	int G[MAX][MAX], i, j, n, u;
	printf("Enter no. of vertices:");
	scanf("%d", &n);
	printf("\nEnter the adjacency matrix:\n");

	for (i = 0; i < n; i++)
		for (j = 0; j < n; j++)
		{
			printf("G[%d][%d]=", i + 1, j + 1);
			scanf("%d", &G[i][j]);
		}

	printf("\nEnter the starting node:");
	scanf("%d", &u);
	dejkstra(G, n, u);

	return 0;
}

void dejkstra(int G[MAX][MAX], int n, int startnode)
{

	int cost[MAX][MAX], distance[MAX], pred[MAX];
	int visited[MAX], count, mindistance, nextnode, i, j;

	//pred[] stores the predecessor of each node
	//count gives the number of nodes that we already seen

	//create the cost matrix
	for (i = 0; i < n; i++)
		for (j = 0; j < n; j++)
			if (G[i][j] == 0)
				cost[i][j] = INFINITY;
			else
				cost[i][j] = G[i][j];

	//initialize pred[],distance[] and visited[]
	for(i = 0; i < n; i++)
	{
		distance[i] = cost[startnode][i];
		pred[i] = startnode;
		visited[i] = 0;
	}

	distance[startnode] = 0;
```

```
	visited[startnode] = 1;
	count = 1;

	while (count < n - 1)
	{
		mindistance = INFINITY;

		//nextnode gives the node at minimum distance
		for (i = 0; i < n; i++)
			if (distance[i] < mindistance && !visited[i])
			{
				mindistance = distance[i];
				nextnode = i;
			}

		//check if a better path exists through nextnode
		visited[nextnode] = 1;
		for (i = 0; i < n; i++)
			if (!visited[i])
				if (mindistance + cost[nextnode][i] < distance[i])
				{
					distance[i] = mindistance + cost[nextnode][i];
					pred[i] = nextnode;
				}
		count++;
	}

	//print the path and distance of each node
	for (i = 0; i < n; i++)
		if(i != startnode)
		{
			printf("\nDistance of node%d=%d", i, distance[i]);
			printf("\nPath=%d", i);

			j = i;
			do
			{
				j = pred[j];
				printf("<-%d",j);
			} while (j != startnode);
		}
}
```