МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНОМУ УНІВЕРСИТЕТІ “ЛЬВІВСЬКА ПОЛІТЕХНІКА“

Кафедра систем штучного інтелекту

Розрахунково-графічні завдання
З дисципліни
«Дискретна математика»

Виконала:
Студентка групи КН-115
Галік Вікторія
Викладач:
Мельникова Н. І.

Львів – 2019р.

ТЕОРЕТИЧНІ ПИТАННЯ ТЕОРІЯ ГРАФІВ.

1. Неформальне означення графа.

2. Ізоморфізм графів.

3. Математичне означення графів.

4. Поняття суміжності вершин та ребер.

5. Побудова матриці суміжності для орієнтованого та неорієнтованого графів.

6. Який граф називається простим?

7. Означення степені вершини, яка вершина називається висячою, ізольованою?

8. Чому дорівнює сума степеній вершин для графа з m-ребер?

9. Чому дорівнює кількість вершин непарного степеня?

10. Означення під графа.

11. Означення субграфа.

12. Який граф називається доповненням до графу G?

13. Який граф називається доповненням до підграфу G’ в графі G?

14. Надайте означення орієнтованого графа, псевдографа, мультиграфа.

15. Маршрутом з вершини v 1 у вершину. v 2 називається ... . Довжиною маршруту називають … .

16. Означення ланцюгу, простого ланцюгу, простого шляху, простого циклу.

17. Означення відстані між вершинами.

18. Діаметр графа.

19. Який граф називається зв’язним?

20. Означення мосту графа.

21. Операції над графами (9 шт.).

22. Означення повного графа.

23. Чому дорівнює кількість ребер у повному графі?

24. Дводольний та повний дводольний графи.

25. Напівстепінь виходження та заходження.

26. Теорема о кількості вхідних та вихідних дуг.

27. Який орграф називається сильнозв’язним, мінімальнозв’язним?

28. Означення дерева, лісу, остову, кодерева.

29. Теорема про дерево.

30. Означення n-дерева, остового n-дерева.

31. Яка вершина називається коренем в орграфі?

32. Означення орієнтованого дерева, орієнтованого остову.

33. Який граф називається квазісильно-зв’язним?

34. Загальна постановка екстремальної задачі на графі.

35. Постановка задачі про найкоротше остове дерево.

36. Постановка задачі про найкоротший ланцюг.

37. Який ланцюг називається ейлеровим, відкритим ейлеровим?

38. Який граф називається ейлеровим?

39. Теорема про ейлеровий граф.

40. Що таке оріентований ейлерів цикл?

41. Означення орієнтованого ейлерового графа.

42. Постановка задачі комівояжера.

43. Що таке гамільтонів цикл?

44. Який граф називається гамільтоновим?

45. Який граф називається сіткою?

46. Дайте означення потоку в сітці.

47. Що таке розмір потоку?

48. Яка дуга називається насиченою?

49. Поток називається повним, якщо ... .

50. Який поток називається максимальним?

51. Що таке розріз в сітці D відносно множини вершин V 1 ?

52. Що таке пропускна здібність розрізу?

53. Теорема Форда-Фалкерсона про максимальний потік.

54. Означення паросполучення графа.

55. Що таке розфарбування графу?

56. Який граф називається плоским, планарним?

57. Теорема Ейлера (на плоских графах)

58. Які графи непланарні (слідство з т. Ейлера)?

59. Теорема Куратовського о планарності графа.

Напишіть алгоритм

60. Обхід графа вглиб та вшир.

61. Прима знаходження найменшого остову.

62. Краскала знаходження найменшого остову.

63. Дейкстра знаходження найкоротшого ланцюга між парою вершин.

64. «Іди в найближчий» для розв'язання задачі комівояжера.

65. Флері та елементарних циклів знаходження ейлерового ланцюга в ейлеровому графі.

ІНДИВІДУАЛЬНІ ЗАВДАННЯ

Варіант 22

Завдання № 1

Виконати наступні операції над графами:

1) знайти доповнення до першого графу,

2) об'єднання графів,

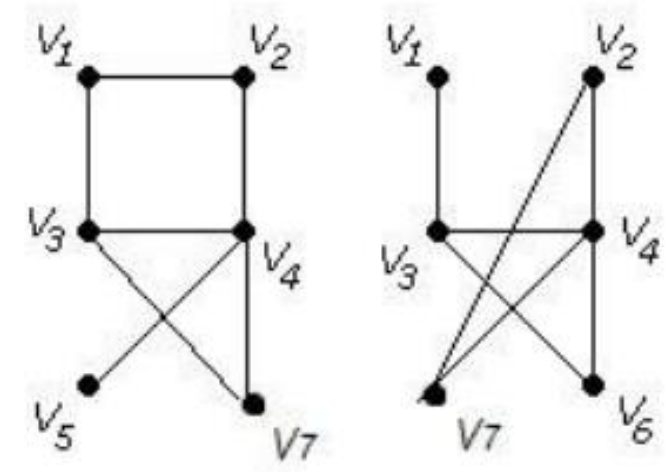
3) кільцеву суму G1 та G2 (G1+G2),

4) розмножити вершину у другому графі,

5) виділити підграф А - що складається з 3-х вершин в G1, стягнути в граф в G1

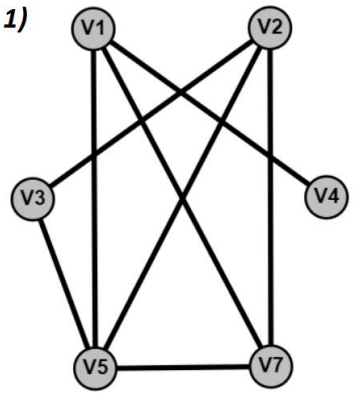
6) добуток графів.

22)

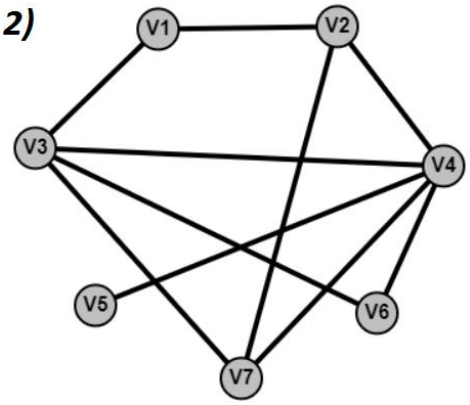


Розв’язання

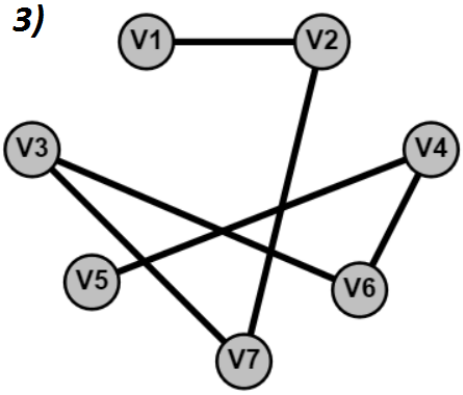
1)



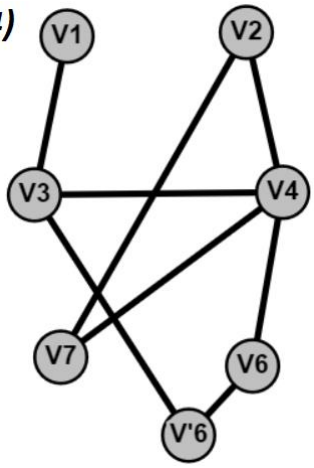
2)



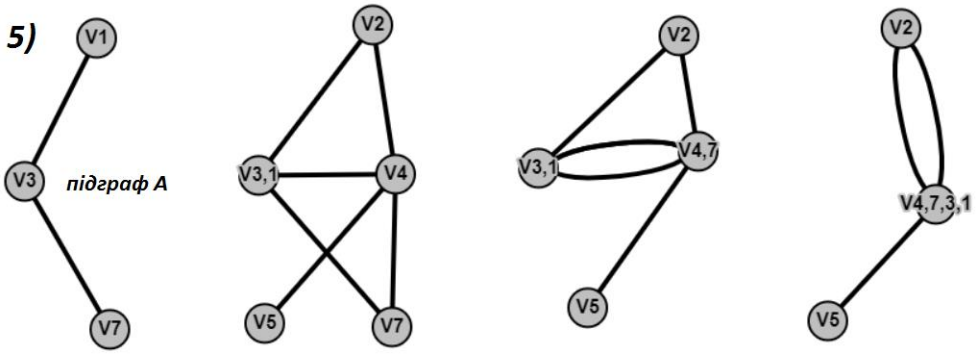
3)



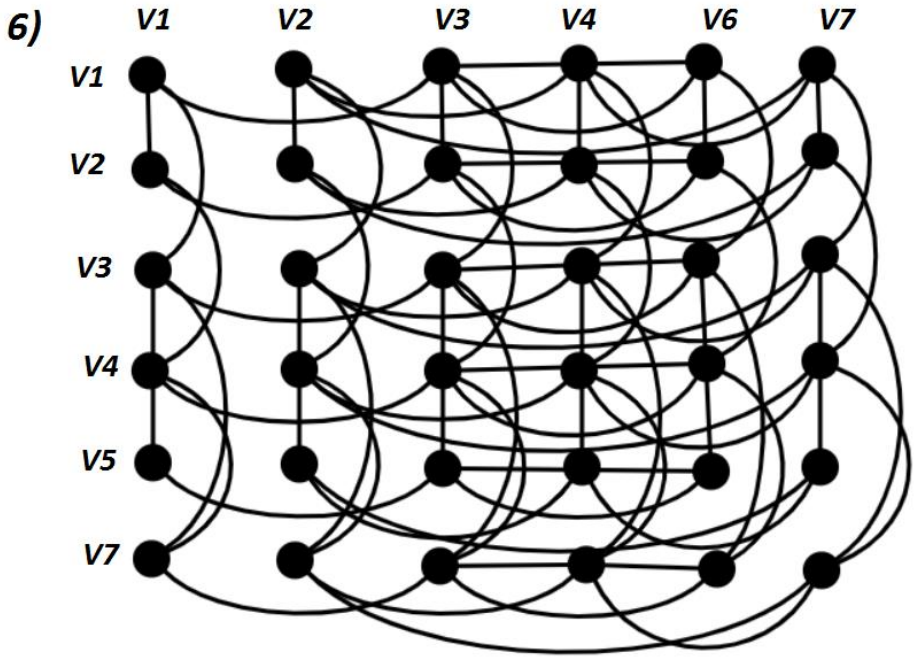
4)



5)

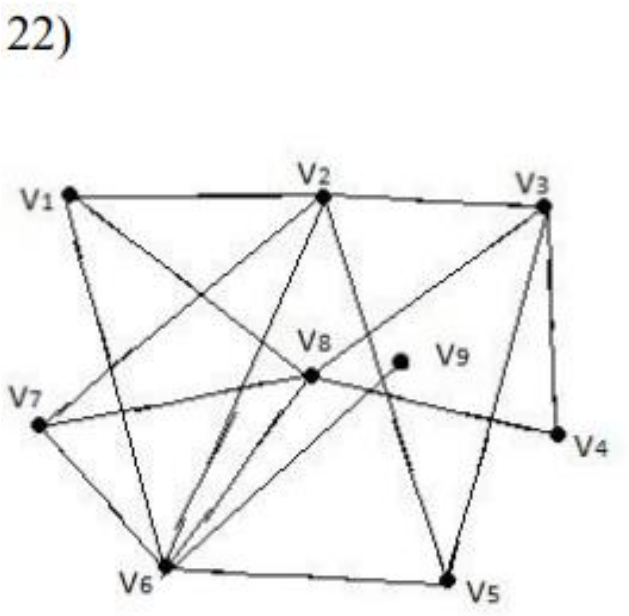


6)



Завдання № 2

Скласти таблицю суміжності для графа



Розв’язання

	V1	***V2***	***V3***	***V4***	***V5***	***V6***	***V7***	***V8***	***V9***
V1	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**0**
V2	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**0**
V3	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**0**
V4	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**0**
V5	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**0**
V6	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**1**
V7	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**0**
V8	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**0**
V9	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**

Завдання № 3

Для графа з другого завдання знайти діаметр.

Розв’язання

Діаметр = **3** (**V4** -> V8 -> V6 -> **V9**)

Завдання № 4

Для графа з другого завдання виконати обхід дерева вглиб (варіант закінчується на непарне число) або **вшир** (закінчується на парне число).

Розв'язання

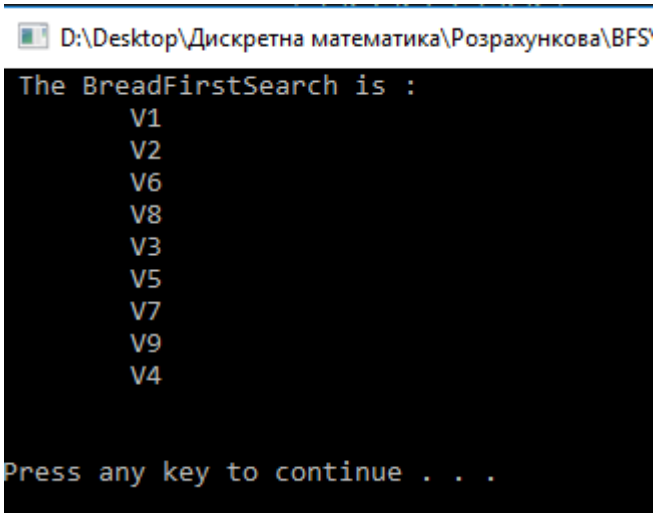
Вершина	BFS - номер	Вміст черги
V1	1	V1
V2	2	V1, V2
V8	3	V1, V2, V8
V6	4	V1, V2, V8, V6
-	-	V2, V8, V6
V3	5	V2, V8, V6, V3
V5	6	V2, V8, V6, V3, V5
V7	7	V2, V8, V6, V3, V5, V7
-	-	V8, V6, V3, V5, V7
V4	8	V8, V6, V3, V5, V7, V4
-	-	V6, V3, V5, V7, V4
V9	9	V6, V3, V5, V7, V4, V9
-	-	V3, V5, V7, V4, V9
-	-	V5, V7, V4, V9
-	-	V7, V4, V9
-	-	V4, V9
-	-	V9
-	-	Ø

Програмна реалізація :

```
#include <iostream>
#include <queue>
#define SIZE 9
using namespace std;
int main()
{
	queue<int> Queue;
	int graf[SIZE][SIZE] = {                  { 0,1,0,0,0,1,0,1,0 },
                                                  { 1,0,1,0,1,1,1,0,0 },
                                                  { 0,1,0,1,1,0,0,1,0 },
                                                  { 0,0,1,0,0,0,0,1,0 },
                                                  { 0,1,1,0,0,1,0,0,0 },
                                                  { 1,1,0,0,1,0,1,1,1 },
                                                  { 0,1,0,0,0,1,0,1,0 },
                                                 { 1,0,1,1,0,1,1,0,0 },
                                                 { 0,0,0,0,0,1,0,0,0 },
	};
```

```
int vertices[SIZE];
for (int i = 0; i < SIZE; i++)
	vertices[i] = 0;
Queue.push(0);                          // поміщаємо в чергу першу вершину
cout << " The BreadFirstSearch is : " << endl;
while (!Queue.empty())                  // поки черга не порожня
{
	int node = Queue.front();           // вилучаємо вершину
	Queue.pop();
	vertices[node] = 2;                 // позначаємо її як пройдену
	for (int j = 0; j < SIZE; j++)
	{                                   // перевірка вершини на суміжні вершини
		if (graf[node][j] == 1 && vertices[j] == 0)
		{                                  // якщо вершина не суміжна та не знайдена
			Queue.push(j);                 // додаємо її в чергу
			vertices[j] = 1;               // позначаємо вершину як знайдену
		}
	}
	cout << "\tV"<< node + 1 << endl;        // номер вершини
}
cout << endl << endl;
system("pause");
return 0;
                                        }
```

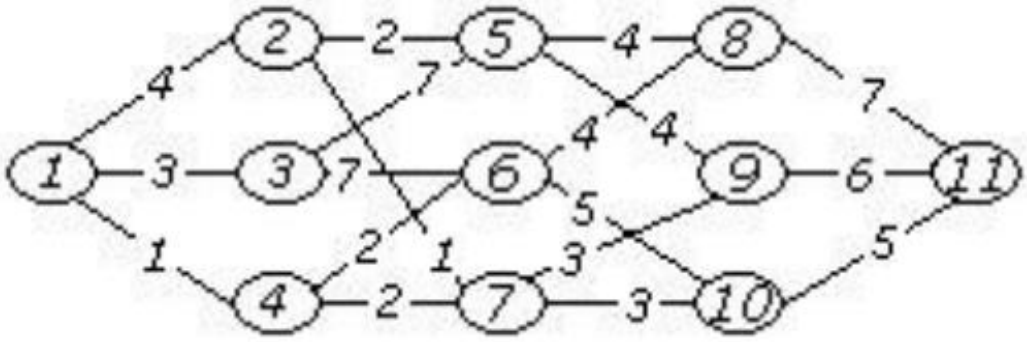
Результат :



Завдання № 5

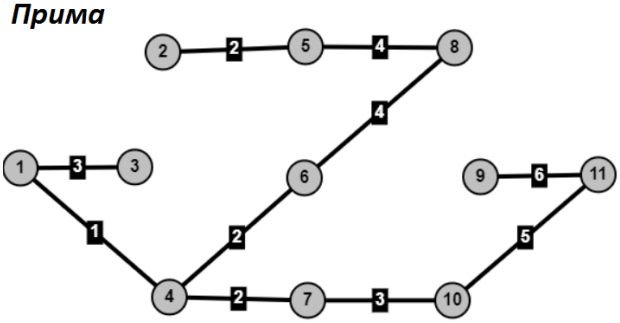
Знайти двома методами (Краскала і Прима) мінімальне остове дерево графа.

22)

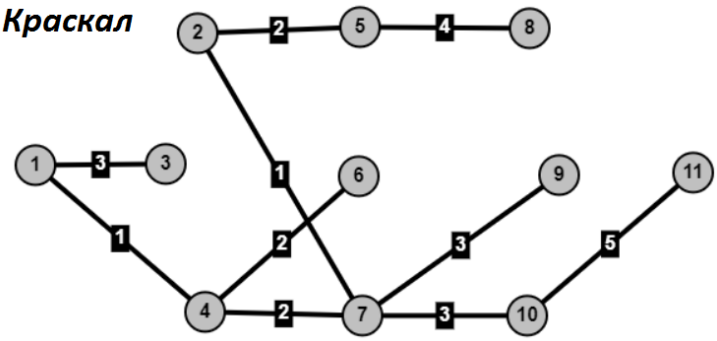


Розв’язання

Прима



Краскал



Прима :

V = { 1, 4, 6, 8, 5, 2, 7, 10, 11, 9, 3}

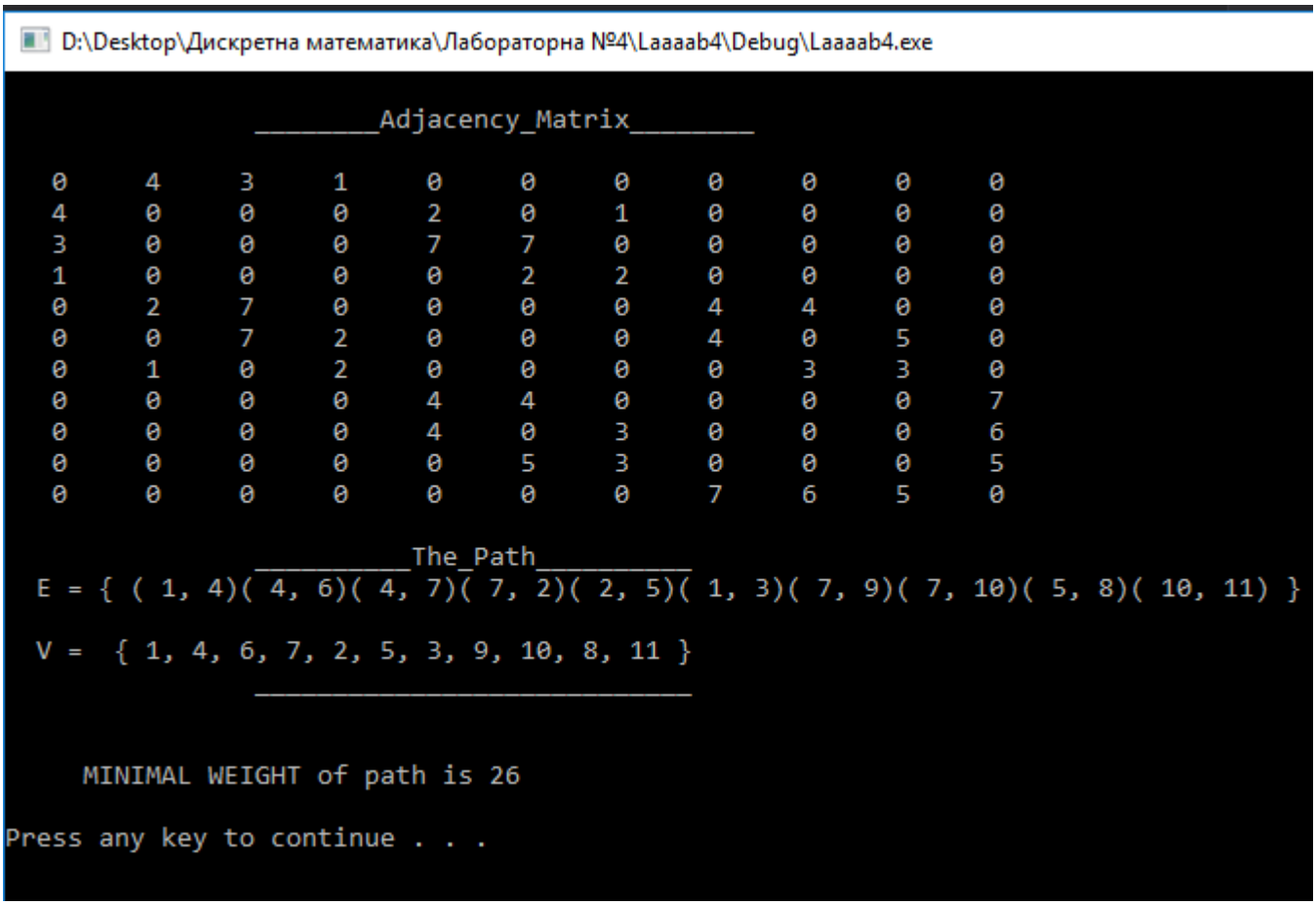
E = { (1,4), (4,6), (6,8), (8,5), (5,2), (4,7), (7,10), (10,11), (11,9), (1,3) }

Програмна реалізація :

```
#include<conio.h>
#include<iostream>

using namespace std;

int main(){

	int a, b, u, n, v, i, j;
	int    no = 1;
	int visited[20] = { 0 };                                  // пройдені вершини
	int min;                                                  // мінімальна вага
	int minweight = 0;
	int path[100] = { 0 };                                    // шлях по вершинах

	int path_index = 0;

	cout << endl << "\t\t________Adjacency_Matrix________" << endl << endl;

	n = 11;
	int weight[12][12] = { {0,0,0,0,0,0,0,0,0,0,0,0},      // матриця інцидентності
	                       {0,0,4,3,1,0,0,0,0,0,0,0},
	                                  {0,4,0,0,0,2,0,1,0,0,0,0},
	                                  {0,3,0,0,0,7,7,0,0,0,0,0},
	                                  {0,1,0,0,0,0,2,2,0,0,0,0},
	                                  {0,0,2,7,0,0,0,0,4,4,0,0},
	                                  {0,0,0,7,2,0,0,0,4,0,5,0},
	                                  {0,0,1,0,2,0,0,0,0,3,3,0},
	                                  {0,0,0,0,0,4,4,0,0,0,0,7},
	                                  {0,0,0,0,0,4,0,3,0,0,0,6},
	                                   {0,0,0,0,0,0,5,3,0,0,0,5},
	                                  {0,0,0,0,0,0,0,0,0,7,6,5,0}
	};

   for (int i = 1; i <=  n; i++)            // вивід матриці
	{
		for ( int j = 1; j <= n ; j++)
		{
			cout << "   " << weight[i][j] << "  ";
		}
		cout << endl;
	}

	for (int i = 1; i <= n; i++)
	{
```

```
		for (int j = 1; j <= n; j++)
		{
			if (weight[i][j] == 0) {
				weight[i][j] = 100;		// позначення не інцидентного ребра
			}
		}
	}
	visited[1] = 1;		//початок шляху
	cout << endl << "\t\t__________The_Path__________" << endl;
	cout << "  E = { ";                                  //вивід множини ребер
	while (no < n){
	 for (i = 1, min = 100; i <= n; i++) {

			for (j = 1; j <=n; j++) {

				if (weight[i][j] < min) {		// перевірка на мінімальну вагу

					if (visited[i] != 0){		// перевірка чи вершина пройдена

						min = weight[i][j];

						a = u = i;
						b = v = j;

					}
				}
			}
		}

		if (visited[u] == 0 || visited[v] == 0){	// перевірка на цикл
			path[path_index] = b;
			path_index++;
			no++;

			minweight += min;
			visited[b] = 1;                         // позначення вершини як пройденої
			cout << "( " << a << ", " << b << ")";

		}
		weight[a][b] = weight[b][a] = 100;
	}
	cout << " }" << endl;
	cout << endl;

	cout << "  V =  { ";		// вивід множини вершин
	cout << 1 << ", ";
	for (int i = 0; i < n - 1; i++)
	{
		cout << path[i];
		if (i < n - 2) cout << ", ";
	}
	cout << " }" << endl;

	cout << "\t\t_____________________________" << endl << endl;
	cout << endl << "     MINIMAL WEIGHT of path is " << minweight << endl << endl;	// сума
шляху

	system("pause");
	return 0;

                                        }
```

Результат :



Краскал :

E = { (1,4), (2,7), (4,7), (4,6), (2,5), (1,3), (7,9), (7,10), (5,8), (8,11) }

Програмна реалізація :

```
#include <iostream>
using namespace std;
struct Rib
{
	int v1, v2, weight;
}Graph[100];

struct sort_rib {
	int v1;
	int v2;
	int weight;
}sort;

void Fill_Struct(int number_of_ribs) {
	for (int i = 0; i < number_of_ribs; i++) {
		cout << "Firts point: ";
		cin >> Graph[i].v1;
		cout << "Second point: ";
		cin >> Graph[i].v2;
		int sort;
		if (Graph[i].v1 > Graph[i].v2) {
			sort = Graph[i].v1;
			Graph[i].v1 = Graph[i].v2;
			Graph[i].v2 = sort;
		}
		cout << "The rib [" << Graph[i].v1 << ";" << Graph[i].v2 << "] = ";
		cin >> Graph[i].weight;
```

```
			cout << endl;
		}
}
void Sort_Sructure(int number_of_ribs) {
		for (int s = 1; s < number_of_ribs; s++) {
			for (int i = 0; i < number_of_ribs - s; i++) {
				if (Graph[i].weight > Graph[i + 1].weight) {
					sort.v1 = Graph[i].v1;
					sort.v2 = Graph[i].v2;
					sort.weight = Graph[i].weight;
					Graph[i].v1 = Graph[i + 1].v1;
					Graph[i].v2 = Graph[i + 1].v2;
					Graph[i].weight = Graph[i + 1].weight;
					Graph[i + 1].v1 = sort.v1;
					Graph[i + 1].v2 = sort.v2;
					Graph[i + 1].weight = sort.weight;
				}
			}
		}
}
void Show_Struct(int number_of_ribs) {
		for (int i = 0; i < number_of_ribs; i++) {
			cout << "The rib [" << Graph[i].v1 << ";" << Graph[i].v2 << "] = " <<
				Graph[i].weight << endl;
		}
}
void Algo_Kraskala(int number_of_ribs, int amount_of_points)
{
		int weighttree = 0;
		int* parent = new int[amount_of_points];
		int v1, v2, weight;
		int to_change, changed;
		for (int i = 0; i < amount_of_points; i++)
		{
			parent[i] = i;
		}
		for (int i = 0; i < number_of_ribs; i++)
		{
			v1 = Graph[i].v1;
			v2 = Graph[i].v2;
			weight = Graph[i].weight;
			if (parent[v2] != parent[v1])
			{
				cout << "\tThe rib (" << Graph[i].v1 << ";" << Graph[i].v2 << ")	Weight =
" <<
					Graph[i].weight << endl;
				weighttree += weight;
				to_change = parent[v1];
				changed = parent[v2];
				for (int j = 0; j < amount_of_points; j++)
				{
					if (parent[j] == changed)
					{
						parent[j] = to_change;
					}
				}
			}
		}
		delete[] parent;
		cout << "The weight of the tree: " << weighttree;
}
int main() {
		cout << "Enter an amount of points" << endl;
		int q;
		cin >> q;
		int amount_of_points = q + 1;
		cout << "Enter a number of ribs" << endl;
		int number_of_ribs;
		cin >> number_of_ribs;
```

```
Fill_Struct(number_of_ribs);
Show_Struct(number_of_ribs);
Sort_Sructure(number_of_ribs);
cout << "After sorting" << endl;
Show_Struct(number_of_ribs);
cout << "Tree" << endl;
Algo_Kraskala(number_of_ribs, amount_of_points);
                                                }
```

Результат :

```
Tree
        The rib (1;4)       Weight = 1
        The rib (2;7)       Weight = 1
        The rib (2;5)       Weight = 2
        The rib (4;6)       Weight = 2
        The rib (4;7)       Weight = 2
        The rib (1;3)       Weight = 3
        The rib (7;9)       Weight = 3
        The rib (7;10)       Weight = 3
        The rib (5;8)       Weight = 4
        The rib (10;11)       Weight = 5
The weight of the tree: 26
```

Завдання № 6

Розв’язати задачу комівояжера для повного 8-ми вершинного графа методом «іди у найближчий», матриця вагів якого має вигляд:

22)

	1	2	3	4	5	6	7	8
1	∞	5	5	1	5	6	5	5
2	5	∞	2	3	5	1	2	3
3	5	2	∞	3	4	5	1	2
4	1	3	3	∞	5	5	5	5
5	5	5	4	5	∞	4	4	4
6	6	1	5	5	4	∞	3	3
7	5	2	1	5	4	3	∞	2
8	5	3	2	5	4	3	2	∞

Розв’язання

№1	1	2	3	4	5	6	7	8
1	∞	5	5	1	5	6	5	5
2	5	∞	2	3	5	1	2	3
3	5	2	∞	3	4	5	1	2
4	1	3	3	∞	5	5	5	5
5	5	5	4	5	∞	4	4	4
6	6	1	5	5	4	∞	3	3
7	5	2	1	5	4	3	∞	2
8	5	3	2	5	4	3	2	∞

№2	2	3	4,1	5	6	7	8
2	∞	2	3	5	1	2	3
3	2	∞	3	4	5	1	2
4,1	3	3	∞	5	5	5	5
5	5	4	5	∞	4	4	4
6	1	5	5	4	∞	3	3
7	2	1	5	4	3	∞	2
8	3	2	5	4	3	2	∞

№3	2	3	5	6,4,1	7	8
2	∞	2	5	1	2	3
3	2	∞	4	5	1	2
5	5	4	∞	4	4	4
6,4,1	1	5	4	∞	3	3
7	2	1	4	3	∞	2
8	3	2	4	3	2	∞

№4	2	4,1,6,3	5	7	8
2	∞	2	5	2	3
4,1,6,3	2	∞	4	1	2
5	5	4	∞	4	4
7	2	1	4	∞	2
8	3	2	4	2	∞

№5	2	5	4,1,6,3,7	8
2	∞	5	2	3
5	5	∞	4	4
4,1,6,3,7	2	4	∞	2
8	3	4	2	∞

№6	2	5	4,1,6,3,7,8
2	∞	5	3
5	5	∞	4
4,1,6,3,7,8	3	4	∞

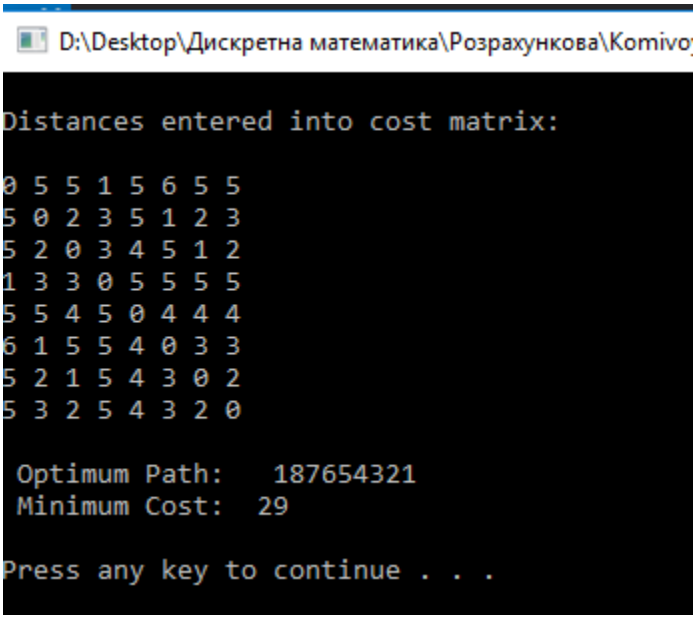
№7	2	4,1,6,3,7,8,5
2	∞	5
4,1,6,3,7,8,5	5	∞

Програмна реалізація :

```
#include <iostream>

using namespace std;

int vistedVertices[8];
int numVertices = 8;
int cost = 0;

int costMatrix[8][8] =
{
	{0,5,5,1,5,6,5,5},
	{5,0,2,3,5,1,2,3},
	{5,2,0,3,4,5,1,2},
	{1,3,3,0,5,5,5,5},
	{5,5,4,5,0,4,4,4},
	{6,1,5,5,4,0,3,3},
	{5,2,1,5,4,3,0,2},
	{5,3,2,5,4,3,2,0},
};

int tsp(int city1) {		// алгоритм комівояжера
	int counter;
	int nearestVertix = 999;
	int mini = 999;
	int temp;
	for (counter = 0; counter < numVertices; counter++) {
		if ((costMatrix[city1][counter] != 0) && (vistedVertices[counter] == 0)) {
			if (costMatrix[city1][counter] < mini) {
				mini = costMatrix[counter][0] + costMatrix[city1][counter];
			}
			temp = costMatrix[city1][counter];
			nearestVertix = counter;
		}
	}
	if (mini != 999)  cost = cost + temp;
	return nearestVertix;
}

void minCost(int city) {
	int nearestVertix;
	vistedVertices[city] = 1;
	cout << city + 1;
	nearestVertix = tsp(city);
	if (nearestVertix == 999) {
		nearestVertix = 0;
		cout << nearestVertix + 1;
		cost = cost + costMatrix[city][nearestVertix];
		return;
	}
	minCost(nearestVertix);
}


int main() {

	int i;
	int j;
	cout << "\nDistances entered into cost matrix:\n";
	for (i = 0; i < numVertices; i++) {
		cout << endl;
		for (j = 0; j < numVertices; j++) {
			cout << costMatrix[i][j] << " ";
		}
	}
```

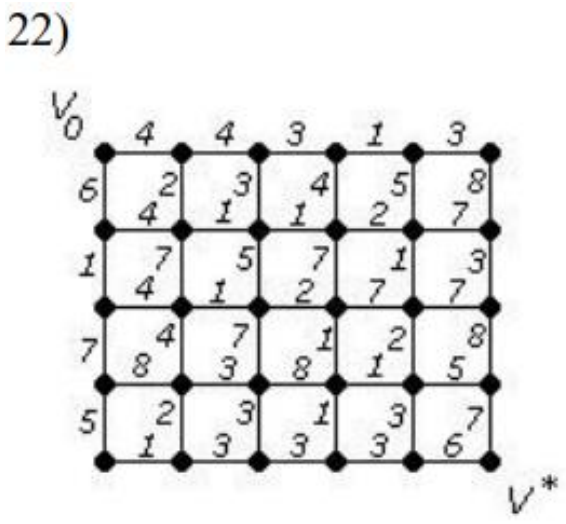
```
        cout << "\n\n Optimum Path: \t ";
        minCost(0);
        cout << "\n Minimum Cost: \t";
        cout << cost;
        cout << endl << endl;
        system("pause");
        return 0;
}
```

Результат :

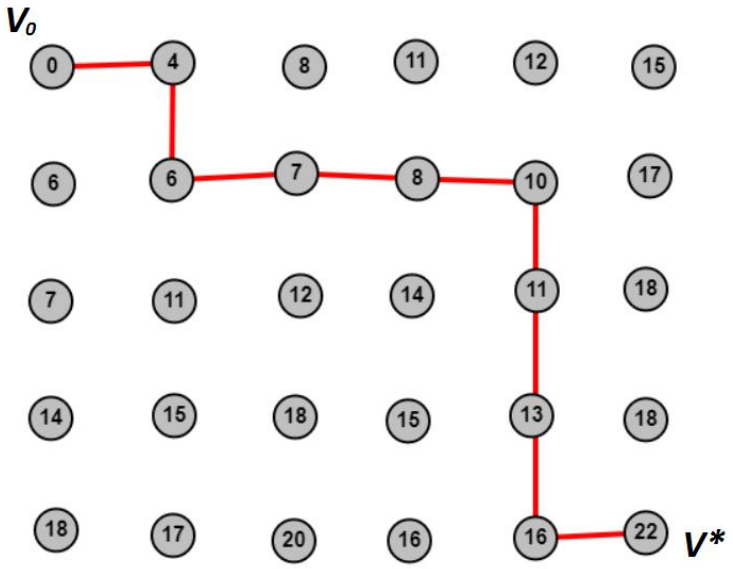


Завдання № 7

За допомогою алгоритму Дейкстри знайти найкоротший шлях у графі між парою вершин V0 і V * .



Розв’язання



Найкоротший шлях : 4 + 2 + 1 + 1 + 2 + 1 + 2 + 3 + 6 = *22*

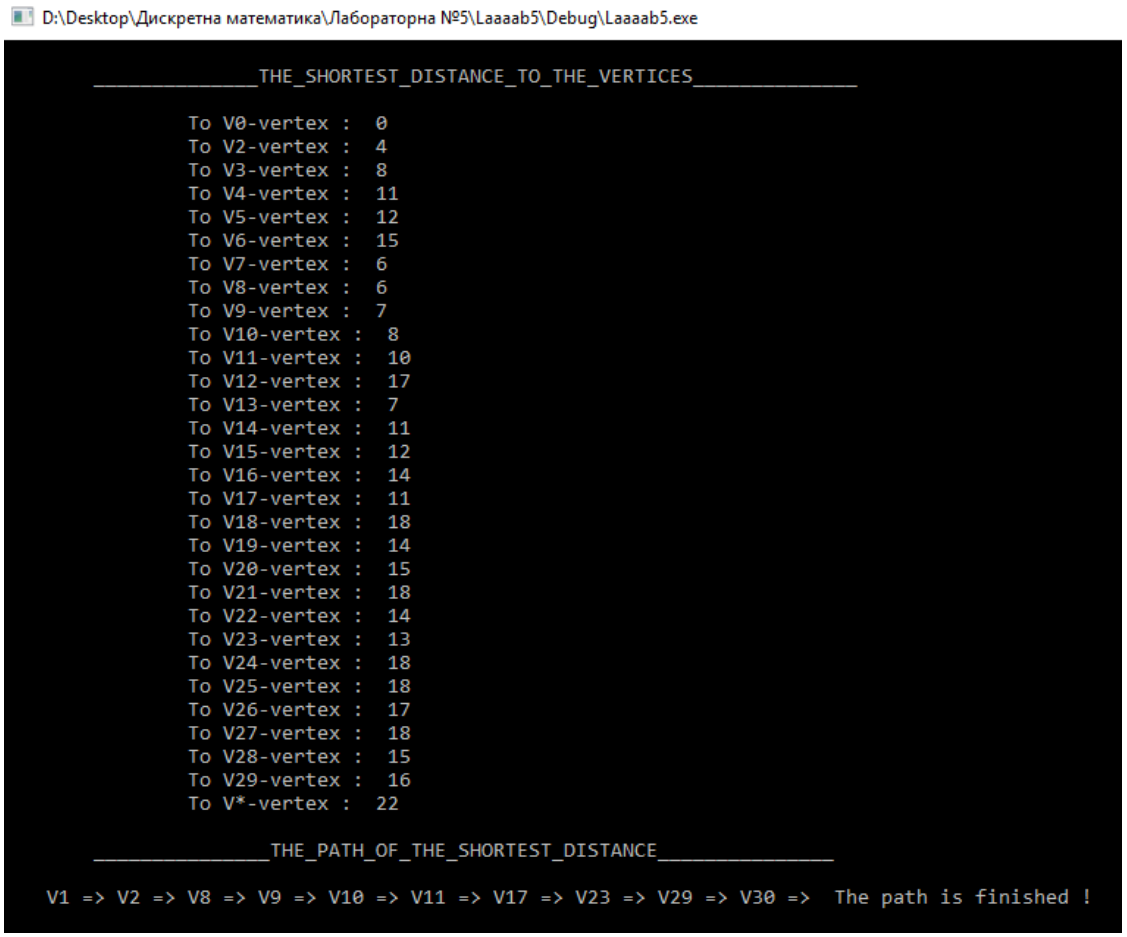
Програмна реалізація :

```
#include <iostream>

#define SIZE 30
#define INF 1000

using namespace std;

int main()
{
	int adjacency[SIZE][SIZE] = {
	{0,4,0,0,0,0,6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
	{4,0,4,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
	{0,4,0,3,0,0,0,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
	{0,0,3,0,1,0,0,0,0,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
	{0,0,0,1,0,3,0,0,0,0,5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
	{0,0,0,0,3,0,0,0,0,0,0,8,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
	{6,0,0,0,0,0,0,4,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
	{0,2,0,0,0,0,4,0,1,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
	{0,0,3,0,0,0,0,1,0,1,0,0,0,0,5,0,0,0,0,0,0,0,0,0,0,0,0,0},
	{0,0,0,4,0,0,0,0,1,0,2,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,0},
	{0,0,0,0,5,0,0,0,0,2,0,7,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0},
	{0,0,0,0,0,8,0,0,0,0,7,0,0,0,0,0,0,3,0,0,0,0,0,0,0,0,0,0},
	{0,0,0,0,0,0,1,0,0,0,0,0,0,4,0,0,0,0,7,0,0,0,0,0,0,0,0,0},
	{0,0,0,0,0,0,0,7,0,0,0,0,4,0,1,0,0,0,0,4,0,0,0,0,0,0,0,0},
	{0,0,0,0,0,0,0,0,5,0,0,0,0,1,0,2,0,0,0,0,7,0,0,0,0,0,0,0},
	{0,0,0,0,0,0,0,0,0,7,0,0,0,0,2,0,7,0,0,0,0,1,0,0,0,0,0,0},
	{0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,7,0,7,0,0,0,0,2,0,0,0,0,0},
	{0,0,0,0,0,0,0,0,0,0,0,3,0,0,0,0,7,0,0,0,0,0,0,8,0,0,0,0},
	{0,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,0,8,0,0,0,0,5,0,0,0},
	{0,0,0,0,0,0,0,0,0,0,0,0,0,4,0,0,0,0,8,0,3,0,0,0,0,2,0,0},
	{0,0,0,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,3,0,8,0,0,0,0,3,0},
	{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,8,0,1,0,0,0,0,1},
	{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,1,0,5,0,0,0,3},
	{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,8,0,0,0,0,5,0,0,0,0,7},
	{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,5,0,0,0,0,0,1,0,0,0},
	{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,1,0,3,0},
	{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,0,0,0,0,3,0,3},
	{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,3,0},
	{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,0,0,0,0,3},
	{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0},
	};
```

```
	int mindistance[SIZE];                          // мінімальна відстань
	int visited[SIZE];                              // пройдені вершини
	int min, minindex, temp;
	int begin_index = 0;                            // початкова вершина

	for (int i = 0; i < SIZE; i++)
	{
		mindistance[i] = INF;                       // ініціалізація відстаней як невідомих
		visited[i] = 1;                             // 1 - непройдена вершина
	}
	mindistance[begin_index] = 0;                   // відстань до початкової вершини

	do {
		minindex = INF;
		min = INF;
		for (int i = 0; i < SIZE; i++)
		{
			if ((visited[i] == 1) && (mindistance[i] < min))              //перевірка чи
вершина пройдена
			{
				min = mindistance[i];                                      //
переприсвоєння мінімального значення
				minindex = i;
			}
		}

		if (minindex != INF)
		{
			for (int i = 0; i < SIZE;  i++)
			{
				if (adjacency[minindex][i] > 0)
				{
					temp = min + adjacency[minindex][i];                  // додаємо
знайдену мінімальну відстань до поточної відстані
					if (temp < mindistance[i])
					{
						mindistance[i] = temp;                             //
остаточна мінімальна відстань до і-тої вершини
					}
				}
			}
			visited[minindex] = 0;                                        // позначення
вершини як пройденої
		}
	} while (minindex < INF);


	cout << endl << "\t_____________THE_SHORTEST_DISTANCE_TO_THE_VERTICES_____________" << endl
<< endl;
	for (int i = 0; i < SIZE; i++) {
		if (i == 0)
		{
			cout << "\t\tTo V0-vertex :  " << mindistance[i] << "\n";
			continue;
		}
		if (i == SIZE-1)
		{
			cout << "\t\tTo V*-vertex :  " << mindistance[i] << "\n";
			break;
		}
		cout << "\t\tTo V" << i + 1 << "-vertex :  " << mindistance[i] << "\n";
	}


	int vis[SIZE];                                                  // пройдені вершини
	int end = SIZE - 1;                                             // індекс кінцевої вершини
	int prev = 1;
	int weight = mindistance[end];                                  // відстань до кінцевої
вершини
```

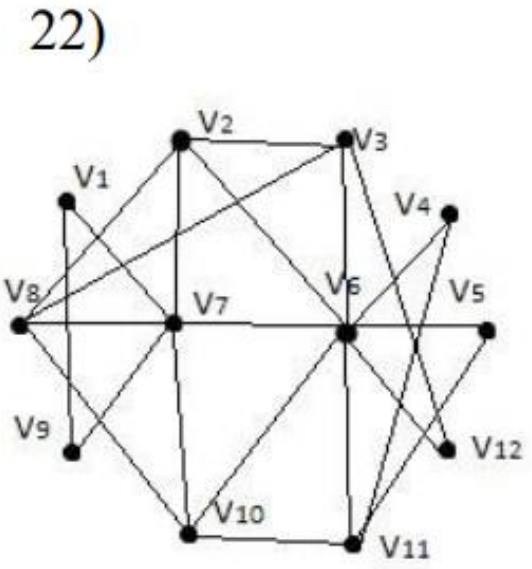
```
	vis[0] = end + 1;                                          // початок - кінцева вершина

	while (end != begin_index)
	{
		for (int i = 0; i < SIZE; i++) {
			if (adjacency[end][i] != 0)                        // якщо є сумісність -
перевірка на знаходження відстані, знайденої раніше
			{
				int temp = weight - adjacency[end][i];
				if (temp == mindistance[i])
				{
					weight = temp;
					end = i;
					vis[prev] = i + 1;                         // збереження
відстані, через яку був перехід у найкоротшому шляху
					prev++;
				}
			}
		}
	}

	cout << endl << "\t_______________THE_PATH_OF_THE_SHORTEST_DISTANCE_______________" << endl
<< endl;
	cout << "    ";
	for (int i = prev - 1; i >= 0; i--) {

		cout << "V" << vis[i] << " => ";
	}
	cout << " The path is finished !";
	cout << endl << endl << endl;
	system("pause");
	return 0;
}
```

Результат:



Завдання № 8

Знайти ейлеровий цикл в ейлеровому графі двома методами: а) Флері; б) елементарних циклів.



Метод Флері :

Нехай V8 – початок циклу, рухаємось по ребрах і перевіряємо чи вони не є мостами, якщо не є мостом – викреслюємо ребро.

(V8, V2) – не є мостом, отже V8 => V2 ,

Аналогічно проходимо весь цикл :

V8 => V2 => V3 => V12 => V6 => V4 => V11 => V5 => V6 => V3 => V8 => V10 => V7 => V1 => V9 => V7 => V2 => V6 => V11 => V10 => V6 => V7 => **V8**

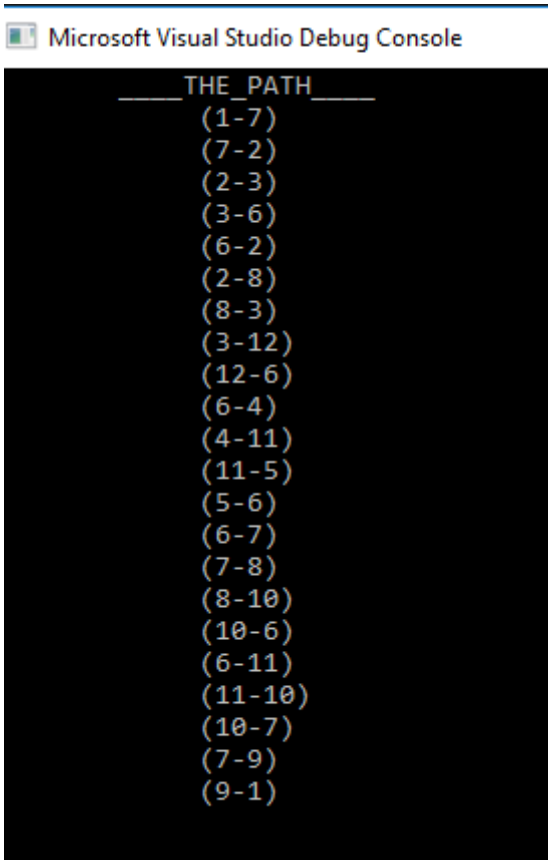
Програмна реалізація :

```
#include<iostream>
#define SIZE 12
using namespace std;

int StartV();
bool IsBridge(int, int);
int edgeCount();
void Fleury(int);

int graph[SIZE][SIZE] = {
{0,0,0,0,0,0,1,0,1,0,0,0},
{0,0,1,0,0,1,1,1,0,0,0,0},
{0,1,0,0,0,1,0,1,0,0,0,1},
{0,0,0,0,0,1,0,0,0,0,1,0},
{0,0,0,0,0,1,0,0,0,0,1,0},
{0,1,1,1,1,0,1,0,0,1,1,1},
{1,1,0,0,0,1,0,1,1,1,0,0},
{0,1,1,0,0,0,1,0,0,1,0,0},
{1,0,0,0,0,0,1,0,0,0,0,0},
{0,0,0,0,0,1,1,1,0,0,1,0},
{0,0,0,1,1,1,0,0,0,1,0,0},
```

```
{0,0,1,0,0,1,0,0,0,0,0,0},
};

int A[SIZE][SIZE];


int main() {
	for (int i = 0; i < SIZE; i++) {
		for (int j = 0; j < SIZE; j++) {
			A[i][j] = graph[i][j];
		}
	}
	cout << "\t____THE_PATH____ " << endl;
	Fleury(StartV());
	cout << endl << endl;
}
int StartV() {
	for (int i = 0; i < SIZE; i++) {
		int d = 0;
		for (int j = 0; j < SIZE; j++) {
			if (A[i][j]) {
				d++;
			}
		}
		if (d % 2 != 0)
			return i;
	}
	return 0;
}
bool IsBridge(int u, int v) {
	int d = 0;
	for (int i = 0; i < SIZE; i++) {
		if (A[v][i]) {
			d++;
		}
	}
	if (d > 1) {
		return false;
	}
	return true;
}
int edgeCount() {
	int count = 0;
	for (int i = 0; i < SIZE; i++) {
		for (int j = i; j < SIZE; j++) {
			if (A[i][j]) {
				count++;
			}
		}
	}
	return count;
}
void Fleury(int start) {
	static int edge = edgeCount();
	for (int v = 0; v < SIZE; v++) {
		if (A[start][v]) {
			if (edge <= 1 || !IsBridge(start, v)) {
				cout << "\t     (" << start + 1 << "-" << v + 1 << ")    " << endl;
				A[start][v] = A[v][start] = 0;
				edge--;
				Fleury(v);
			}
		}
	}
```

}

Результат :



Метод елементарних циклів :

Для пошуку ейлерового циклу знайдемо всі прості цикли та об’єднаємо їх:

Виділимо перший початковий цикл. Нехай це буде цикл (V1 – V7 – V9),

Виділимо наступний простий цикл, що починається, наприклад, у вершині 7 та об’єднаємо його з попереднім :

Цикл (V7 – V2 - V3 – V6 – V10 – V7)

Отримаємо : (V1 – V7 – V2 - V3 – V6 – V10 – V7 - V9),

Аналогічно:

(V10 – V8 – V7 – V6 – V11 - V10) =>

(V1 – V7 – V2 - V3 – V6 – V10 – V8 – V7 – V6 – V11 - V10 – V7 - V9)

(V8 – V2 – V6 – V12 – V3 – V8) =>

(V1 – V7 – V2 - V3 – V6 – V10 – V8 – V2 – V6 – V12 – V3 – V8 – V7 – V6 – V11 - V10 – V7 - V9)

(V6 – V5 – V11 – V4 – V6) =>

(**V1** – V7 – V2 - V3 – V6 – V5 – V11 – V4 – V6 – V10 – V8 – V2 – V6 – V12 – V3 – V8 – V7 – V6 – V11 - V10 – V7 – V9 – **V1**)

Ми знайшли цикл : (**V1** – V7 – V2 - V3 – V6 – V5 – V11 – V4 – V6 – V10 – V8 – V2 – V6 – V12 – V3 – V8 – V7 – V6 – V11 - V10 – V7 – V9 – **V1**)

Завдання №9

Спростити формули (привести їх до скороченої ДНФ).

22. $\overline{x\bar{y} \vee (x\bar{y}\bar{z}) \vee \bar{x}\bar{z}}$

= ¬x ᵛ y ^ (¬x ᵛ y ᵛ z)^ ¬x ᵛ ¬z - за законом де Моргана

= ¬x ᵛ y ^ ¬x ᵛ ¬z - за законом поглинання

= ¬x ᵛ ¬z - за законом ідемпотентності