Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №3

за 3 семестр

По дисциплине: «Дискретная математика»

Тема: «Деревья. Графы»

Выполнил:

Студент 2 курса

Группы ПО-6(1)

Мартынович Д. М.

Проверил:

Глущенко Т. А.

Брест 2021

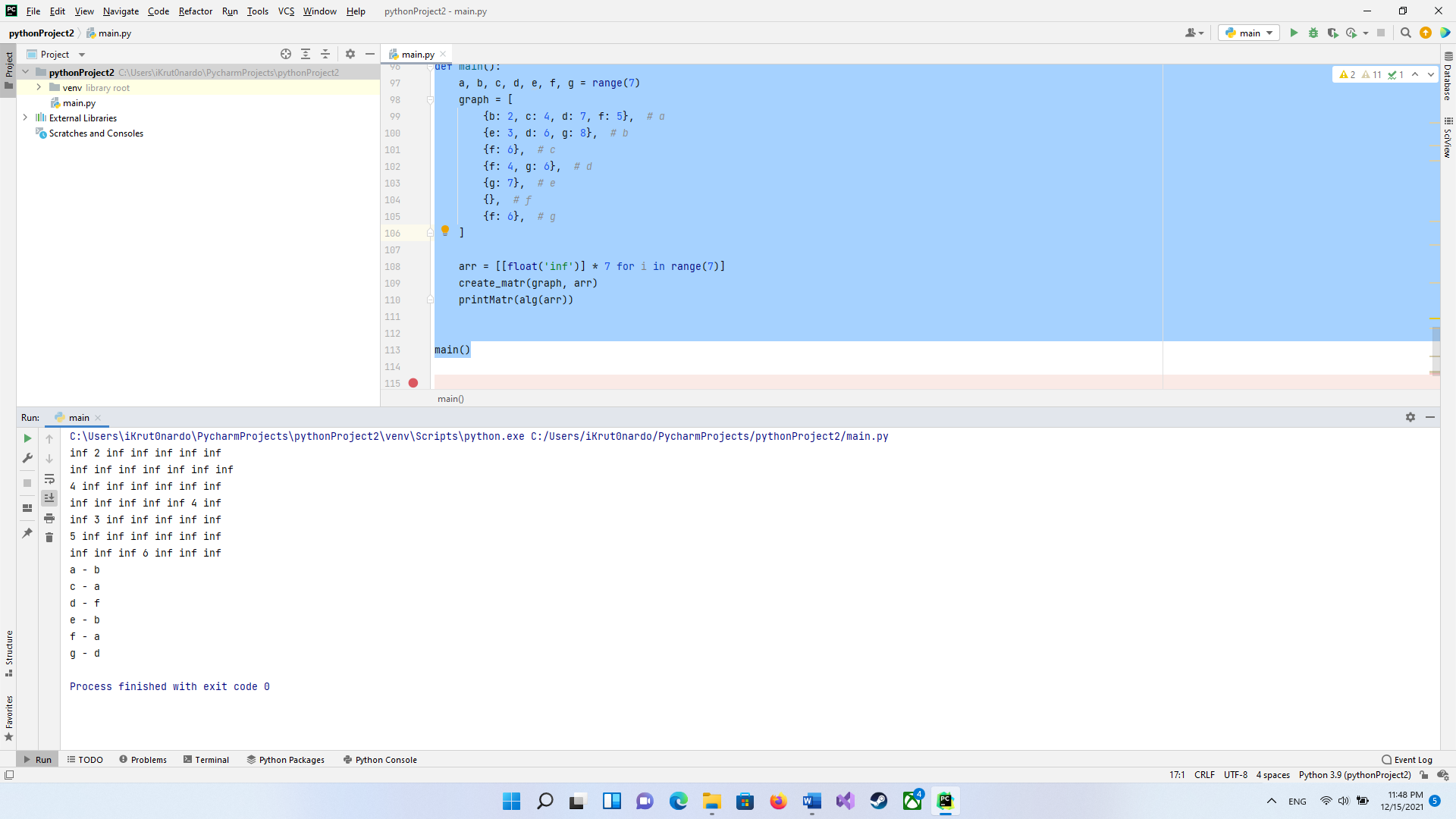
Цель: изучить теорию по теме «Деревья. Графы», ознакомиться с понятиями, данными в теории. По данному варианту программно реализовать предложенные задачи.

***Задание 1.***

1. Найти минимальное остовное дерево для заданного графа *G* алгоритмом *Прима* и *Крускаля*. Варианты графов указаны в *таблице 1.* Граф задан списком ребер.
2. Ответить на поставленные вопросы.
3. Графически изобразить граф и его минимальное остовное дерево.

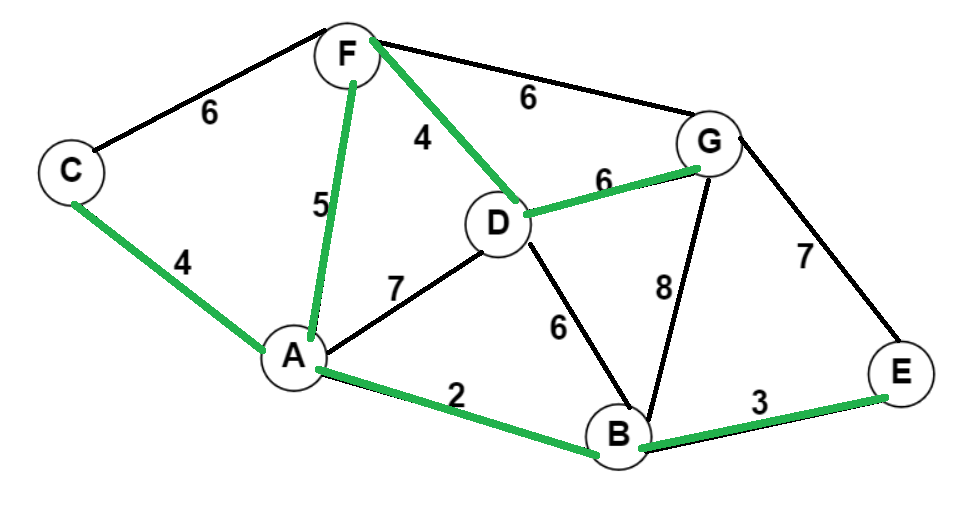
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 10 | 7 | 12 | (a,b),(a,c),(a,d),(a,f),( b,e),(b,d),  (b,g),( c,f),(d,f),(d,g),(e,g),(f,g) | 2,4,7,5,3,6,  8,6.4,6,7,6 |

from math import \*  
  
  
*# формируем матрицу смежности графа*def create\_matr(graph, arr):  
 i = 0  
 for value in graph:  
 for key in value:  
 arr[i][key] = value[key]  
 arr[key][i] = value[key]  
 i += 1  
  
  
*# минимальное остове дерево*def alg(arr):  
 newArr = [[float('inf')] \* 7 for i in range(7)]  
 list = []  
 for temp in range(7):  
 *# ищем минмальный элемент* minimum = float('inf')  
 i = 0  
 j = 0  
  
 *# формируем массив помеченных вершин* tempArr = [[float('inf')] \* 7 for i in range(7)]  
 if len(list) != 0:  
 for i in range(7):  
 for value in list:  
 tempArr[i][value] = arr[i][value]  
 else:  
 tempArr = arr  
  
 for value in tempArr:  
 if min(value) < minimum:  
 minimum = min(value)  
 i = tempArr.index(value)  
 j = value.index(minimum)  
  
 *# помечаем пройденные вершины* if list.count(i) == 0:  
 list.append(i)  
 if list.count(j) == 0:  
 list.append(j)  
  
 *# формируем матрицу минамльного остoвого дерева* newArr[i][j] = minimum  
  
 *# удалаем значения из строк* for k in range(7):  
 arr[i][k] = float(inf)  
 arr[j][k] = float(inf)  
 return newArr  
  
  
def printMatr(newArr):  
 for i in range(7):  
 for j in range(7):  
 print(newArr[i][j], end=" ")  
 print()  
  
 for i in range(7):  
 for j in range(7):  
 if newArr[i][j] != float('inf'):  
 if i == 0:  
 a = "a"  
 elif i == 1:  
 a = "b"  
 elif i == 2:  
 a = "c"  
 elif i == 3:  
 a = "d"  
 elif i == 4:  
 a = "e"  
 elif i == 5:  
 a = "f"  
 elif i == 6:  
 a = "g"  
  
 if j == 0:  
 b = "a"  
 elif j == 1:  
 b = "b"  
 elif j == 2:  
 b = "c"  
 elif j == 3:  
 b = "d"  
 elif j == 4:  
 b = "e"  
 elif j == 5:  
 b = "f"  
 elif j == 6:  
 b = "g"  
 print(a, "-", b)  
  
  
def main():  
 a, b, c, d, e, f, g = range(7)  
 graph = [  
 {b: 2, c: 4, d: 7, f: 5}, *# a* {e: 3, d: 6, g: 8}, *# b* {f: 6}, *# c* {f: 4, g: 6}, *# d* {g: 7}, *# e* {}, *# f* {f: 6}, *# g* ]  
  
 arr = [[float('inf')] \* 7 for i in range(7)]  
 create\_matr(graph, arr)  
 printMatr(alg(arr))  
  
  
main()



Изображение выглядит как ножницы, инструмент

Автоматически созданное описание



***Вопросы.***

1. Для какого графа определяет число остовных деревьев формула *Кэли*.

Для полных помеченных (взвешенных) графов

1. Какое остовное дерево находится алгоритмом *Дейкстры*?

Обход вершин, которого стоит меньше всего (минимальное остовное дерево).

1. Может ли быть несколько *минимальных* остовных деревьев?

Если каждое ребро имеет отдельный вес, тогда будет только одно уникальное

минимальное остовное дерево. В противном случае может существовать несколько минимальных остовов.

***Задание.***

1. Построить *матрицу смежности* и *инцидентности* для заданного графа. Изобразить граф.
2. По матрице смежности (инцидентности) для каждой из вершин вычислить ее *степень*.
3. Используя поиск в глубину и поиск в ширину написать программу, определяющую *число компонент связности* графа. Методы представляются в виде отдельных функций (или классов).
4. Построить *деревья поиска в ширину и глубину*.
5. Из заданного неориентированного графа построить произвольным образом ориентированный граф (добавить к каждому ребру стрелку). Для полученного таким образом ориентированного графа построить *матрицу смежности и инцидентности*.
6. Из заданного неориентированного графа построить произвольным образом *псевдограф*.
7. Варианты заданий указаны в *таблице 1*.
8. В таблице граф задан списком ребер, например, запись *(1,2)*означает, что существует ребро, соединяющее вершину *1* с вершиной 2.

**Графы**

Изображение выглядит как текст

Автоматически созданное описание

1. Построить *матрицу смежности* и *инцидентности* для заданного графа. Изобразить граф.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 4 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 6 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 4 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 6 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Матрица смежности Матрица инцидентности

Изображение выглядит как катается на лыжах

Автоматически созданное описание

1. По матрице смежности (инцидентности) для каждой из вершин вычислить ее *степень*.

Вершина 1 - **1**

Вершина 2 - **2**

Вершина 3 - **1**

Вершина 4 - **3**

Вершина 5 - **2**

Вершина 6 - **3**

Вершина 7 - **1**

Вершина 8 - **1**

1. Используя поиск в глубину и поиск в ширину написать программу, определяющую *число компонент связности* графа. Методы представляются в виде отдельных функций (или классов).

Текст программы:

1. Обход в ширину:

#include <iostream>

#include <queue> // стек

using namespace std;

int main() {

system("color f0");

queue<int> Queue;

int mas[8][8] = { { 0, 0, 0, 1, 0, 0, 0, 0 }, // матрица смежности

{ 0, 0, 0, 1, 0, 1, 0, 0 },

{ 0, 0, 0, 0, 0, 1, 0, 0 },

{ 1, 1, 0, 0, 1, 0, 0, 0 },

{ 0, 0, 0, 1, 0, 0, 0, 0 },

{ 0, 1, 1, 0, 1, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 1 },

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

int nodes[8]; // вершины графа

for (int i = 0; i < 8; i++) // исходно все вершины равны 0

nodes[i] = 0;

for (int f = 0; f < 8; f++) {

Queue.push(f); // помещаем в очередь первую вершину

while (!Queue.empty()) { // пока стек не пуст

int node = Queue.front(); // извлекаем вершину

Queue.pop();

if (nodes[node] == 2) continue;

nodes[node] = 2; // отмечаем ее как посещенную

for (int j = 7; j >= 0; j--)

{ // проверяем для нее все смежные вершины

if (mas[node][j] == 1 && nodes[j] != 2)

{ // если вершина смежная и не обнаружена

Queue.push(j); // добавляем ее в cтек

nodes[j] = 1; // отмечаем вершину как обнаруженную

}

}

cout << node + 1 << " - ";// выводим номер вершины

}

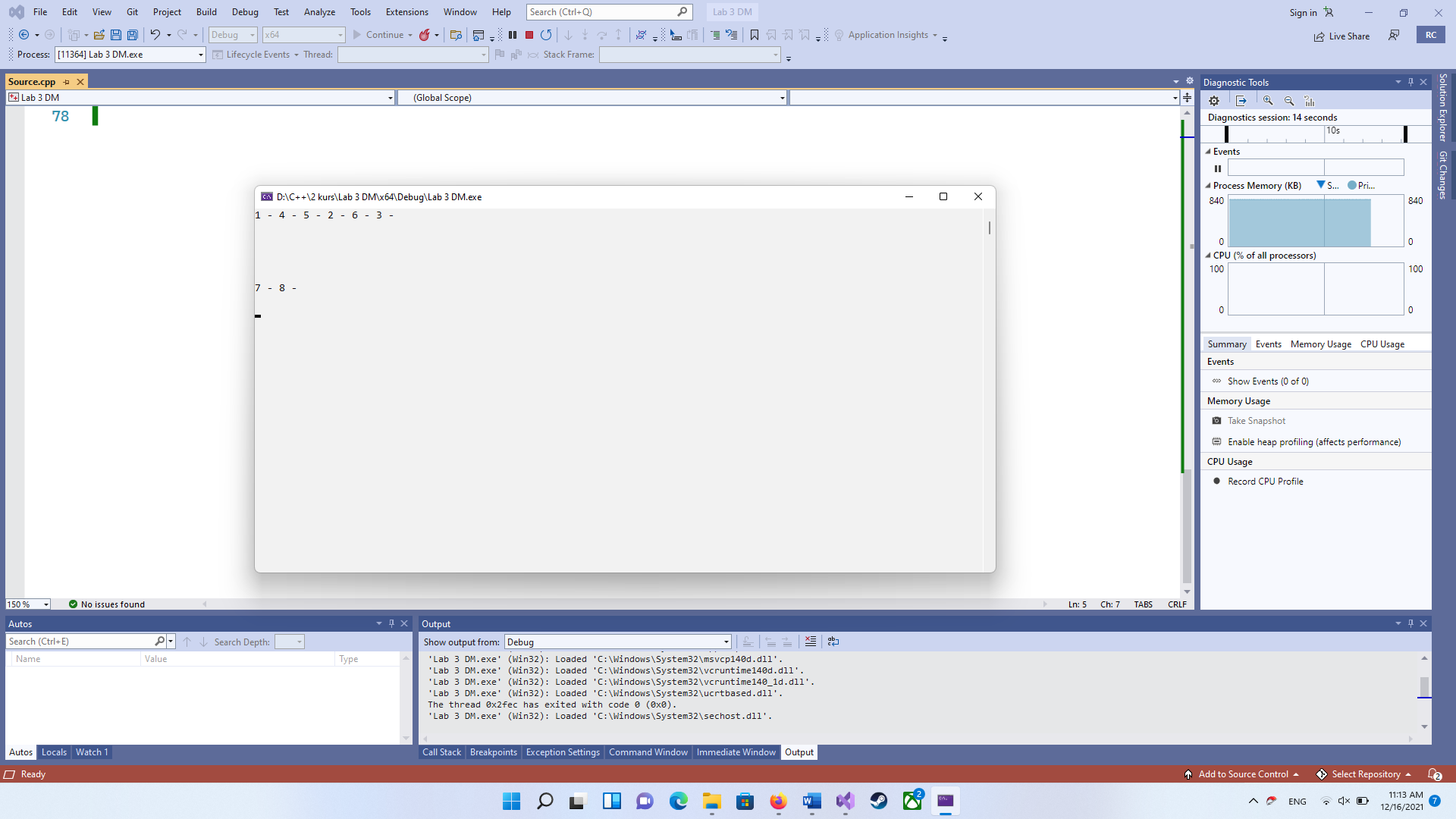
cout << endl;

}

cin.get();

return 0;

}



1. Обход в глубину:

#include <iostream>

using namespace std;

const int n = 8;

int i, j;

bool\* visited = new bool[n];

//матрица смежности графа

int graph[n][n] =

{ { 0, 0, 0, 1, 0, 0, 0, 0 }, // матрица смежности

{ 0, 0, 0, 1, 0, 1, 0, 0 },

{ 0, 0, 0, 0, 0, 1, 0, 0 },

{ 1, 1, 0, 0, 1, 0, 0, 0 },

{ 0, 0, 0, 1, 0, 0, 0, 0 },

{ 0, 1, 1, 0, 1, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 1 },

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

//поиск в глубину

void DFS(int st)

{

int r;

cout << st + 1 << " ";

visited[st] = true;

for (r = 0; r <= n; r++)

if ((graph[st][r] != 0) && (!visited[r]))

DFS(r);

}

//главная функция

void main()

{

system("color f0");

setlocale(LC\_ALL, "Rus");

int start;

cout << "Матрица смежности графа: " << endl;

for (i = 0; i < n; i++)

{

visited[i] = false;

for (j = 0; j < n; j++)

cout << " " << graph[i][j];

cout << endl;

}

cout << "Стартовая вершина >> "; cin >> start;

//массив посещенных вершин

bool\* vis = new bool[n];

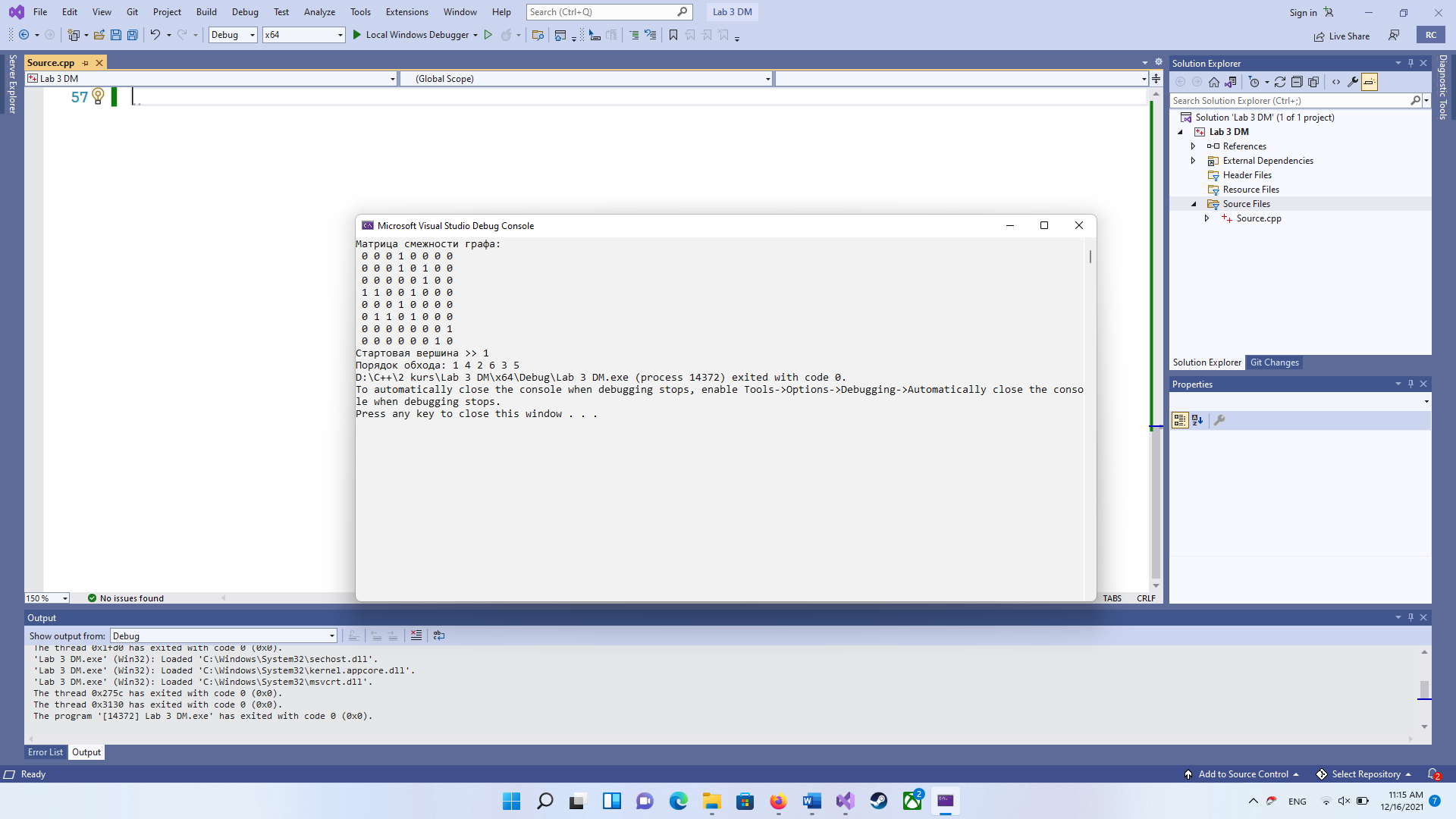
cout << "Порядок обхода: ";

DFS(start - 1);

delete[]visited;

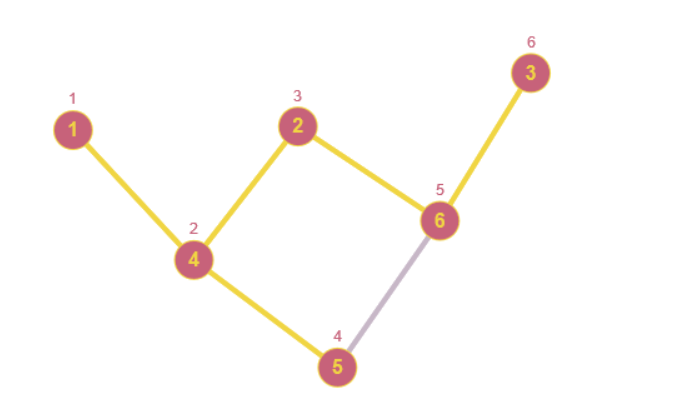
system("pause>>void");

}

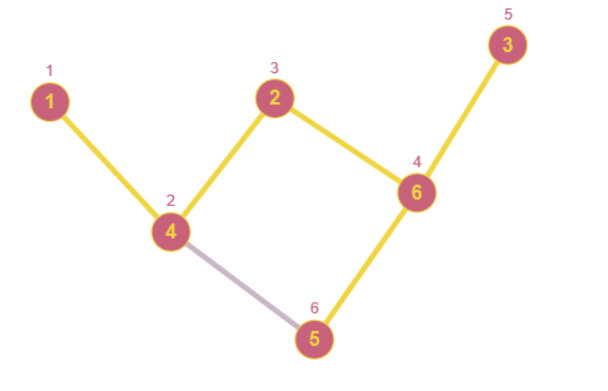


1. Построить *деревья поиска в ширину и глубину*.

Поиск в ширину:



Поиск в глубину:



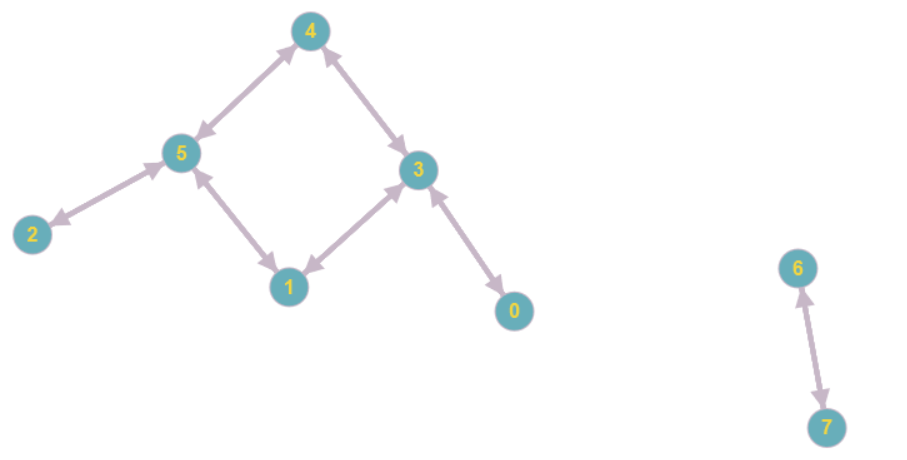
1. Из заданного неориентированного графа построить произвольным образом ориентированный граф (добавить к каждому ребру стрелку). Для полученного таким образом ориентированного графа построить *матрицу смежности и инцидентности*.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | -1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | -1 | -1 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | -1 | 0 | 0 | 0 |
| 4 | 1 | 1 | 0 | 0 | -1 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 1 | -1 | 0 |
| 6 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | -1 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Матрица смежности Матрица инцидентности

1. Из заданного неориентированного графа построить произвольным образом *псевдограф*.



Вопросы:

* В каком из методов обхода графа путь в дереве поиска соответствует кратчайшему (т.е. содержащему наименьшее количество ребер) пути от вершины s до вершины v.

В методе обхода в ширину в дереве поиска показывается кратчайший путь от вершины s до вершины и все вершины, связанные с выбранной, помечаются как доступные, что отображается на дереве поиска. При методе обхода в глубину выбирается первая доступная вершина для выбранной и делается обход, поэтому количество ребер, необходимых для доступа в вершину, может увеличиться.

* Для какого из обходов строится единственное (с точностью до изоморфизма) дерево поиска, а для какого можно построить их несколько.

Для обхода в ширину можно построить единственное дерево поиска, а для поиска в глубину – несколько, т.к. обход может выбираться по-разному.

* Какое из утверждений является верным:

а) Полный граф всегда является регулярным графом;

б) Регулярный граф всегда является полным графом.

Полный граф — [простой](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%81%D1%82%D0%BE%D0%B9_%D0%B3%D1%80%D0%B0%D1%84) неориентированный [граф](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), в котором каждая пара различных вершин смежна.

Регулярный (однородный) граф — [граф](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), [степени всех вершин](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B5%D0%BF%D0%B5%D0%BD%D1%8C_%D0%B2%D0%B5%D1%80%D1%88%D0%B8%D0%BD%D1%8B_(%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2)) которого равны, то есть каждая вершина имеет одинаковое количество соседей.

Верно А.

* Что показывает сумма строк, столбцов матриц смежности и инцидентности для простого графа?

Сумма элементов матрицы по i-й строке равна степени соответствующей вершины. То есть, она показывает, сколько ребер выходит из данной вершины. Сумма элементов по столбцам показывает, сколько ребер входит в данную вершину.

Вывод: изучил принципы работы с деревьями и графами.