Министерство образования Республики Беларусь
Учреждение образования
“Брестский Государственный технический университет”
Кафедра ИИТ

Лабораторная работа №6

По дисциплине “Математические основы интеллектуальных систем”
Тема: “Определение компонент сильной связности в орграфе.
Определение компонент двусвязности для неориентированного графа”

Выполнил:
Студент 2 курса
Группы ИИ-21
Кирилович А. А.
Проверил:
Козинский А. А.

Брест 2022

Цель: научиться определять компонент сильной связности в орграфе и компонент двусвязности для неориентированного графа.

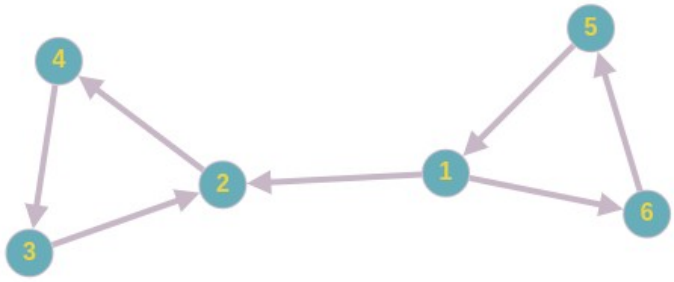
Ход работы:

Задание 1

Написать программу нахождения компонент сильной связности орграфа. Изобразить граф, пометив его компоненты сильной связности.

(1,4),(2,1),(3,2),(4,3),(4,5),(5,6),(6,7),(7,5)

```
#include "../graph_LIB.hh"

int main()
{
        convert c;
        std::string file_path = "connections1.txt";
        VEC1 nodes = c.reading_file(file_path);

        int max_node = c.count_of_nodes(nodes);
        VEC2 adjacencyMatrix = c.di_adjancy(nodes,
max_node);

    alg search;

    VEC1 n_comps;
    for (int i = 0; i < max_node; i++)
    {
        n_comps.push_back(search.conCompDFS_strong(adjacencyMatrix, i));
    }
    int max_n_comp = *max_element(n_comps.begin(), n_comps.end());
    int index = std::distance(n_comps.begin(), std::find(n_comps.begin(), n_comps.end(), max_n_comp));

    VEC2 comps(max_n_comp);
    std::cout << "count = " << max_n_comp << std::endl;
    search.conCompDFS_strong(adjacencyMatrix, comps, index);
    for (int i = 0; i < max_n_comp; i++)
    {
        std::cout << "Component " << i + 1 << ": ";
        for (int j = 0; j < comps[i].size(); j++)
        {
            std::cout << comps[i][j] << " ";
        }
        std::cout << std::endl;
    }
    std::cout << max_n_comp;
}
```




```
count = 2
Component 1: 0 1 2 3
Component 2: 4 5 6
```

Реализация функции conCompDFS_strong из файла Graph_LIB:

```
    int conCompDFS_strong(VEC2 &adjacencyMatrix, VEC2 &comps, int start) {
        int max_node = adjacencyMatrix.size();
        int count = 0;
        VEC1 visited(max_node);
        VEC1 buffer_visited(max_node);
        for (int i = 0; i < max_node; i++) {
            visited[i] = 0;
        }
        for (int i = start; i < max_node; i++) {
            if (visited[i] == 0) {
                buffer_visited = visited;

                DFS(adjacencyMatrix, visited, i);

                for (int i = 0; i < max_node; i++) {
                    visited[i] = visited[i] - buffer_visited[i];
                }
                for (int i = 0; i < max_node; i++) {
                    if (visited[i]) {
```

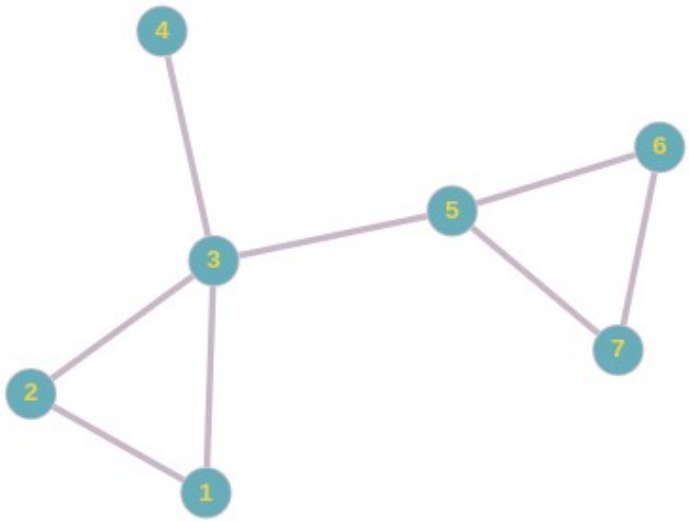
```
                    comps[count].push_back(i);
                }
            }
            for (int i = 0; i < max_node; i++) {
                visited[i] = visited[i] + buffer_visited[i];
            }

            count++;
        }
    }
    for (int i = 0; i < start; i++) {
        if (visited[i] == 0) {
            buffer_visited = visited;

            DFS(adjacencyMatrix, visited, i);

            for (int i = 0; i < max_node; i++) {
                visited[i] = visited[i] - buffer_visited[i];
            }
            for (int i = 0; i < max_node; i++) {
                if (visited[i]) {
                    comps[count].push_back(i);
                }
            }
            for (int i = 0; i < max_node; i++) {
                visited[i] = visited[i] + buffer_visited[i];
            }

            count++;
        }
    }
    return count;
}
```

Задание 2

Написать программу нахождения компонент двусвязности для неориентированного графа. Изобразить граф, пометив точки сочленения и мосты, и его компоненты двусвязности.

(1,2),(1,3),(2,3),(3,4),(3,5),(5,6),(5,7),(6,7)

```
#include "../graph_LIB.hh"

int main() {
    convert c;
    std::string file_path = "connections2.txt";
    VEC1 nodes = c.reading_file(file_path);
    int max_node = c.count_of_nodes(nodes);
    VEC2 adjacencyMatrix = c.adjancy(nodes,
max_node);
    VEC2 bufferMatrix = adjacencyMatrix;

    alg search;
    VEC1 visited(max_node);
    int count;
    count = search.conCompBFS(adjacencyMatrix);
    int k;
    VEC1 articulationPoints;
    for (int i = 0; i < max_node; i++) {
        for (int j = 0; j <
bufferMatrix.size(); j++) {


bufferMatrix[j].erase(bufferMatrix[j].begin() +
i);
        }
        bufferMatrix.erase(bufferMatrix.begin() + i);

        k = search.conCompBFS(bufferMatrix);
        if (k > count) {
            articulationPoints.push_back(i);
```



```
            }
            bufferMatrix = adjacencyMatrix;
        }

        std::cout << "Articulation points: ";
        for (int i = 0; i < articulationPoints.size(); i++) {
            std::cout << articulationPoints[i] + 1 << " ";
        }

        VEC1 bridges;
        VEC1 vis(max_node);
        for (int i = 0; i < articulationPoints.size(); i++) {
            for (int j = 0; j < max_node; j++) {
                bufferMatrix[articulationPoints[i]][j] = 0;
                bufferMatrix[j][articulationPoints[i]] = 0;

                k = search.conCompBFS(bufferMatrix);
                if (k > count) {
                    if (vis[j] && vis[articulationPoints[i]]) {
                        break;
                    }
                    bridges.push_back(articulationPoints[i]);
                    bridges.push_back(j);
                    vis[articulationPoints[i]] = 1;
                    vis[j] = 1;
                }
                bufferMatrix = adjacencyMatrix;
            }
        }

        std::cout << std::endl << "Bridges: ";
        c.print(bridges);
}
```

```
Articulation points: 3 5
Bridges: (3,4),(3,5)
```

Вывод: в ходе лабораторной работы я научился находить минимальное остовное дерево во взвешенном связном графе с помощью алгоритмов Прима и Краскала.