МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

Лабораторная работа №4.3

по дисциплине: Дискретная математика
тема: «Связность»

Выполнил: ст. группы ПВ-221
Лоёк Никита Викторович

Проверили:
Бондаренко Татьяна Владимировна
Рязанов Юрий Дмитриевич

Белгород 2023 г.

Лабораторная работа № 4.3

Цель работы: изучить алгоритм Краскала построения покрывающего леса, научиться использовать его при решении различных задач.

Задания

1. Реализовать алгоритм Краскала построения покрывающего леса.

```
vector<vector<bool>> GetCoveringTree(vector<vector<bool>> matrix) {
    vector<int> b(matrix.size());
    for (int i = 0; i < b.size(); ++i) {
        b[i] = i + 1;
    }

    for (int i = 0; i < matrix.size(); ++i) {
        for (int j = i + 1; j < matrix.size(); ++j) {
            if (matrix[i][j]) {

                if (b[i] != b[j]) {
                    int groupBouquet = b[j];
                    for (int k = 0; k < b.size(); ++k) {
                        if (b[k] == groupBouquet) {
                            b[k] = b[i];
                        }
                    }
                } else {
                    matrix[i][j] = false;
                    matrix[j][i] = false;
                }
            }
        }
    }

    return matrix;
}
```

2. Используя алгоритм Краскала, разработать и реализовать алгоритм решения задачи (см. варианты заданий).

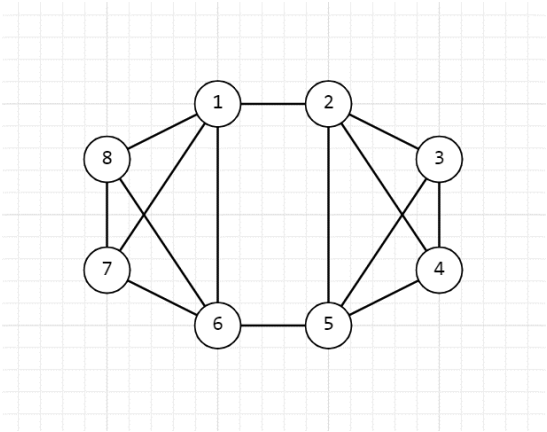
Найти минимальное множество ребер, удаление которых из связного графа делает его несвязным:

```
int CountBouquet(vector<vector<bool>> &matrix) {
    vector<int> bouquet(matrix.size());
    for (int i = 0; i < bouquet.size(); ++i) {
        bouquet[i] = i + 1;
    }
    int counter = matrix.size();

    for (int i = 0; i < matrix.size(); ++i) {
        for (int j = i + 1; j < matrix.size(); ++j) {
            if (matrix[i][j]) {

                if (bouquet[i] != bouquet[j]) {
                    counter--;
                    int groupBouquet = bouquet[j];
                    for (int k = 0; k < bouquet.size(); ++k) {
                        if (bouquet[k] == groupBouquet) {
                            bouquet[k] = bouquet[i];
                        }
                    }
                }
            }
        }
    }
    return counter;
}

void GetAllCombinations(vector<vector<int>> &matrix,
                        vector<vector<int>> edges,
                        int start,
                        int n,
                        vector<vector<vector<int>>> &res) {
    for (int i = start; i < matrix.size() - n && n > 0; ++i) {
        edges.push_back(matrix[i]);
        GetAllCombinations(matrix, edges, i + 1, n - 1, res);
        edges.pop_back();
    }

    if (n == 0) {
        res.push_back(edges);
    }
}

vector<vector<int>> GetSetUnlinkedEdges(vector<vector<bool>> &matrix) {
    vector<vector<int>> res = {};
    int countBouquet = CountBouquet(matrix);
    if (countBouquet == 1) {
        vector<vector<int>> pairV;
        for (int i = 0; i < matrix.size(); ++i) {
            for (int j = i + 1; j < matrix.size(); ++j) {
                if (matrix[i][j]) {
                    vector<int> buf = {i + 1, j + 1};
                    pairV.push_back(buf);
                }
            }
        }
        int n = 1;
        while (countBouquet == 1 && n < matrix.size()) {
            vector<vector<vector<int>>> buf;
            GetAllCombinations(pairV, vector<vector<int>>{}, 0, n, buf);
            vector<vector<bool>> matrixCopy;

            for (int i = 0; i < buf.size() && countBouquet == 1; ++i) {
                matrixCopy.assign(matrix.begin(), matrix.end());
                res.assign(buf[i].cbegin(), buf[i].cend());
                for (int j = 0; j < buf[i].size(); ++j) {
                    matrixCopy[res[j][0] - 1][res[j][1] - 1] = false;
                    matrixCopy[res[j][1] - 1][res[j][0] - 1] = false;
                }

                countBouquet = CountBouquet(matrixCopy);
            }
            n++;
        }
    }

    return res;
}
```

3. Подобрать тестовые данные. Результат представить в виде диаграммы графа:

Тест 1:

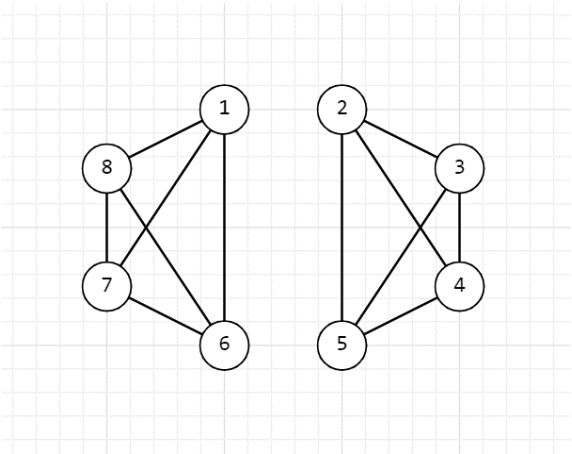
Изначальный Граф:



Результат работы программы:

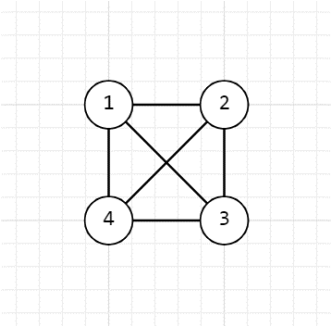
```
0   1   0   0   0   1   1   1
1   0   1   1   1   0   0   0
0   1   0   1   1   0   0   0
0   1   1   0   1   0   0   0
0   1   1   1   0   1   0   0
1   0   0   0   1   0   1   1
1   0   0   0   0   1   0   1
1   0   0   0   0   1   1   0
1 2
5 6
```

Полученный граф:



Тест 2:

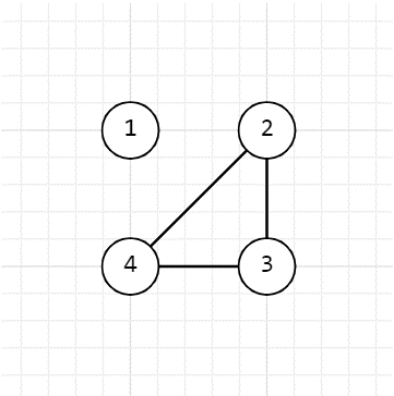
Изначальный Граф:



Результат работы программы:

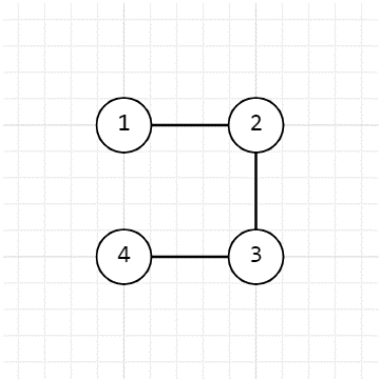
```
0 1 1 1
1 0 1 1
1 1 0 1
1 1 1 0
1 2
1 3
1 4
```

Полученный граф:



Тест 3:

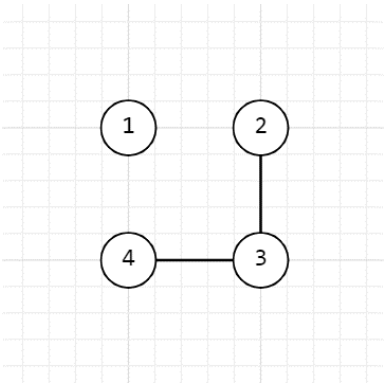
Изначальный Граф:



Результат работы программы:

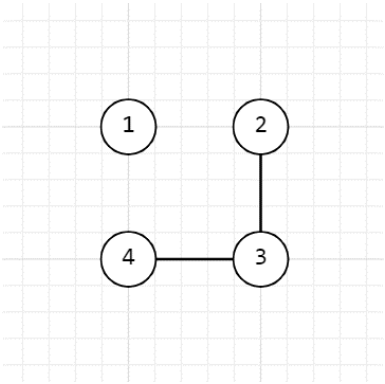
```
0 1 0 0
1 0 1 0
0 1 0 1
0 0 1 0
1 2
```

Полученный граф:



Тест 4:

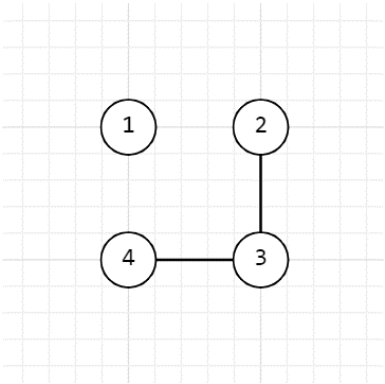
Изначальный Граф:



Результат работы программы:

```
0   0   0   0
0   0   1   0
0   1   0   1
0   0   1   0

Process finished with exit code 0
```

Полученный граф:



Вывод

Вывод: в ходе работы я изучил алгоритм Краскала построения покрывающего леса, научился использовать его при решении различных задач.