**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 ВТ-231

Масленников Даниил Александрович

Проверили:

Рязанов Юрий Дмитриевич

Белгород 2024 г.

Л а б о р а т о р н а я р а б о т а № 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.

// Алгоритм Краскала для построения покрывающего леса

matrix\* kruskalAlgorithm(matrix\* G, int\* num\_components) {

// Инициализируем массив компонентов связности

int\* components = (int\*) malloc(sizeof(int) \* G->nRows);

if (!components) {

printf("Ошибка при выделении памяти\n");

exit(EXIT\_FAILURE); // Прерывание программы в случае ошибки

}

for (int i = 0; i < G->nRows; ++i) {

components[i] = i + 1; // Каждая вершина изначально принадлежит своей компоненте

}

// Создаем новую пустую матрицу для результата

matrix\* result = createEmptyMatrix(G->nRows, G->nCols);

// Проходим по всем рёбрам графа

for (int j = 0; j < G->nCols; ++j) {

int v1 = -1, v2 = -1;

// Находим два конца ребра

for (int i = 0; i < G->nRows; ++i) {

if (G->values[i][j]) {

if (v1 == -1) {

v1 = i;

} else if (v2 == -1) {

v2 = i;

break;

}

}

}

// Если оба конца найдены и они принадлежат разным компонентам

if (v1 != -1 && v2 != -1 && components[v1] != components[v2]) {

// Объединяем компоненты

int comp\_to\_merge = components[v2];

for (int i = 0; i < G->nRows; ++i) {

if (components[i] == comp\_to\_merge) {

components[i] = components[v1]; // Присваиваем всем элементам второй компоненты первую

}

}

// Добавляем ребро в результат

result->values[v1][j] = 1;

result->values[v2][j] = 1;

}

}

// Подсчитываем количество компонент связности

\*num\_components = 1;

int current\_component = components[0];

for (int i = 1; i < G->nRows; ++i) {

if (current\_component != components[i]) {

(\*num\_components)++;

current\_component = components[i];

}

}

// Освобождаем память

free(components);

return result;

}

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

Найти все точки сочленения в связном графе.

// Структура графа

typedef struct Graph {

int vertices;

int\*\* adjMatrix;

} Graph;

// Создание нового графа

Graph\* createGraph(int vertices) {

Graph\* graph = (Graph\*)malloc(sizeof(Graph));

graph->vertices = vertices;

graph->adjMatrix = (int\*\*)malloc(vertices \* sizeof(int\*));

for (int i = 0; i < vertices; i++) {

graph->adjMatrix[i] = (int\*)calloc(vertices, sizeof(int));

}

return graph;

}

// Функция для поиска родителя узла

int find(int parent[], int node) {

if (parent[node] != node) {

parent[node] = find(parent, parent[node]); // путь сжатия

}

return parent[node];

}

// Объединение двух множеств

void unionSets(int parent[], int rank[], int u, int v) {

int rootU = find(parent, u);

int rootV = find(parent, v);

if (rootU != rootV) {

if (rank[rootU] < rank[rootV]) {

parent[rootU] = rootV;

} else if (rank[rootU] > rank[rootV]) {

parent[rootV] = rootU;

} else {

parent[rootV] = rootU;

rank[rootU]++;

}

}

}

// Алгоритм Краскала для построения минимального остовного дерева

Graph\* kruskal(Graph\* graph) {

Graph\* mst = createGraph(graph->vertices);

int\* parent = (int\*)malloc(graph->vertices \* sizeof(int));

int\* rank = (int\*)malloc(graph->vertices \* sizeof(int));

for (int i = 0; i < graph->vertices; ++i) {

parent[i] = i;

rank[i] = 0;

}

int edgeCount = 0;

for (int i = 0; i < graph->vertices; i++) {

for (int j = i + 1; j < graph->vertices; j++) {

if (graph->adjMatrix[i][j] != 0) {

if (find(parent, i) != find(parent, j)) {

unionSets(parent, rank, i, j);

mst->adjMatrix[i][j] = graph->adjMatrix[i][j];

mst->adjMatrix[j][i] = graph->adjMatrix[i][j];

edgeCount++;

}

}

}

}

// Освобождаем память

free(parent);

free(rank);

return mst;

}

// Подсчет количества связанных компонентов

int countConnectedComponents(Graph\* graph, int nodeToRemove) {

int\* visited = (int\*)calloc(graph->vertices, sizeof(int));

int count = 0;

void dfs(int node) {

visited[node] = 1;

for (int i = 0; i < graph->vertices; i++) {

if (graph->adjMatrix[node][i] != 0 && !visited[i]) {

dfs(i);

}

}

}

for (int i = 0; i < graph->vertices; i++) {

if (!visited[i] && i != nodeToRemove) {

dfs(i);

count++;

}

}

free(visited);

return count;

}

// Поиск точек сочленения

void findArticulationPoints(Graph\* graph) {

Graph\* mst = kruskal(graph);

int\* articulationPoints = (int\*)malloc(graph->vertices \* sizeof(int));

int count = 0;

for (int node = 0; node < graph->vertices; node++) {

// Создаем временный граф, исключив текущий узел

Graph\* tempGraph = createGraph(graph->vertices);

for (int i = 0; i < graph->vertices; i++) {

for (int j = 0; j < graph->vertices; j++) {

if (i != node && j != node) {

tempGraph->adjMatrix[i][j] = mst->adjMatrix[i][j];

}

}

}

if (countConnectedComponents(tempGraph, node) > countConnectedComponents(mst, -1)) {

articulationPoints[count++] = node; // Добавляем точку сочленения

}

for (int i = 0; i < graph->vertices; i++) {

free(tempGraph->adjMatrix[i]);

}

free(tempGraph->adjMatrix);

free(tempGraph);

}

printf("Точки сочленения: ");

for (int i = 0; i < count; i++) {

printf("%d ", articulationPoints[i] + 1);

}

printf("\n");

free(articulationPoints);

for (int i = 0; i < graph->vertices; i++) {

free(mst->adjMatrix[i]);

}

free(mst->adjMatrix);

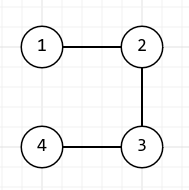
free(mst);

}

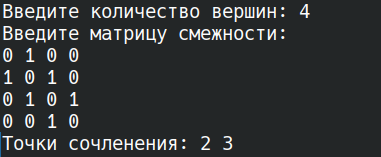
1. Подобрать тестовые данные. Результат представить в виде диаграммы графа:

Тест 1:

Изначальный Граф:

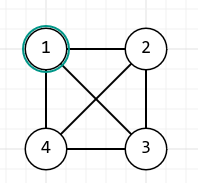


Результат работы программы:

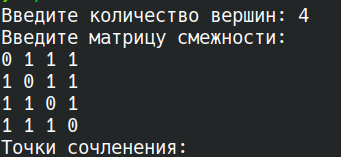


Тест 2:

Изначальный Граф:

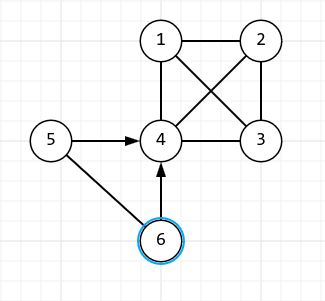


Результат работы программы:

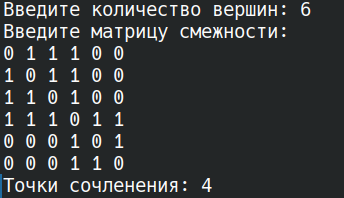


Тест 3:

Изначальный Граф:



Результат работы программы:



**Вывод**

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