ФГБОУ ВО “Чувашский государственный университет им.

И.Н. Ульянова”

Кафедра вычислительной техники

Лабораторная работа №6

«Алгоритмы на графах»

Вариант 9.

Выполнил:

Студент гр. ИВТ-41-22

Иванов Владимир Сергеевич

Проверил:

Павлов Леонид Александрович

Чебоксары, 2024

***Цель работы*:** изучить основные алгоритмы на графах, получить практические навыки программирования задач, которые формулируются в терминах теории графов.

Выполнение работы:

Алгоритм перехода от матрицы смежности к структуре смежности:

Процедура конвертации\_в\_структуру\_смежности(матрица\_смежности):

// Инициализация пустых списков смежности для каждой вершины

Для каждой вершины v:

список\_смежности[v] = Пустой список

// Проход по матрице смежности

Для каждой вершины i:

Для каждой вершины j:

Если матрица\_смежности[i][j] == 1:

Добавить вершину j в список\_смежности[i]

return список\_смежности

Алгоритм перехода структуры смежности в матрицу смежности:

Процедура конвертации\_в\_матрицу\_смежности(структура\_смежности):

// Инициализация пустой матрицы смежности

Инициализировать матрицу\_смежности размером N x N, где N - количество вершин

// Заполнение матрицы смежности

Для каждой вершины i:

Для каждой вершины j в списке\_смежности[i]:

матрица\_смежности[i][j] = 1 // Или другое значение, обозначающее наличие ребра

return матрицу\_смежности

Реализовать алгоритм Крускала нахождения минимального остовного дерева взвешенного графа:

void kruskal() {

int i, uRep, vRep;

sort(G.begin(), G.end()); // increasing weight

for (i = 0; i < G.size(); i++) {

uRep = find\_set(G[i].second.first);

vRep = find\_set(G[i].second.second);

if (uRep != vRep) {

T\_krus.push\_back(G[i]); // add to tree

union\_set(uRep, vRep);

}

}

Реализовать алгоритм Дейкстры-Прима нахождения минимального остовного дерева взвешенного графа:

void prim() {

vector<bool> inMST(V, false);

vector<int> parent(V, -1); // Родительская вершина в остовном дереве

priority\_queue<pair<int, edge>, vector<pair<int, edge>>, greater<pair<int, edge>>> pq;

pq.push(make\_pair(0, edge(-1, 0))); // Start from vertex 0

while (!pq.empty()) {

int u = pq.top().second.second;

int parent\_u = pq.top().second.first;

int weight = pq.top().first;

pq.pop();

if (inMST[u]) continue; // Skip if u is already in MST

inMST[u] = true;

if (parent\_u != -1) { // Skip if u is the starting vertex

T\_prim.push\_back(make\_pair(weight, edge(parent\_u, u)));

}

for (auto& e : G) {

int v = e.second.second;

if (e.second.first == u && !inMST[v]) {

parent[v] = u;

pq.push(make\_pair(e.first, edge(u, v))); // corrected weight here

}

}

}

}

Реализовать метод построения фундаментального множества циклов неориентированного графа:

void dfs(int v, int parent) {

visited[v] = true;

path.push\_back(v);

for (int u : adj[v]) {

if (!visited[u]) {

dfs(u, v);

}

else if (u != parent) {

// Найден цикл

vector<int> cycle;

// Найдите индекс первого вхождения u в path

int idx = find(path.begin(), path.end(), u) - path.begin();

// Запишите вершины в цикл в порядке, обходя вершины по пути от idx до path.size() - 1

for (int j = idx; j < path.size(); ++j) {

cycle.push\_back(path[j]);

}

// Добавьте вершины от начала пути до u

for (int j = 0; j < idx; ++j) {

cycle.push\_back(path[j]);

}

// Добавьте цикл в список циклов

cycles.push\_back(cycle);

}

}

path.pop\_back();

}

Реализовать поиск кратчайших путей от фиксированной вершины до всех остальных вершин орграфа:

void BFS(int startVertex) {

vector<bool> visited(V, false); // Локальный массив для отслеживания посещенных вершин

queue<int> queue;

visited[startVertex] = true;

queue.push(startVertex);

int k = 0;

while (!queue.empty()) {

int currVertex = queue.front();

cout << currVertex ;

k++;

if(k != V) cout << " <-> ";

queue.pop();

for (int adjVertex : adj[currVertex]) {

if (!visited[adjVertex]) {

visited[adjVertex] = true;

queue.push(adjVertex);

}

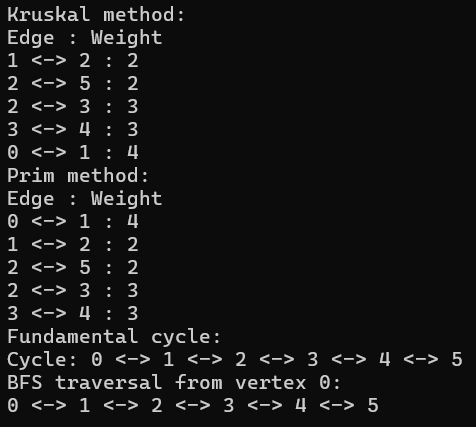
}

}

cout << endl;

}

Вывод программы:



Вывод: изучил основные алгоритмы на графах, получил практические навыки программирования задач, которые формулируются в терминах теории графов.