МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ УЧРЕЖДЕНИЯ ОБРАЗОВАНИЯ “БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”

КАФЕДРА ИНТЕЛЛЕКТУАЛЬНЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Лабораторная работа №4

По дисциплине “Теоретико-множественные основы ИС”

Тема: “*Нахождение минимального остового дерева связанного неориентированного графа.*”

Выполнил:

Студент 1 курса Группы ИИ-23

Макаревич Н.Р.

Проверила:

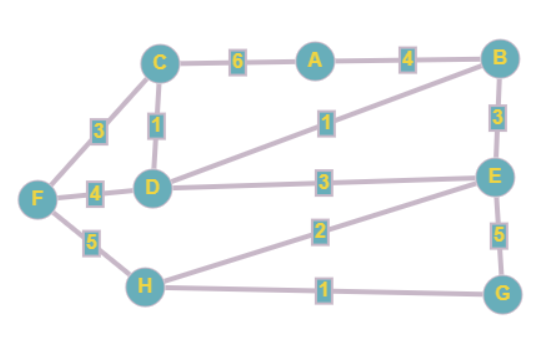
Глущенко Т.А.

Брест 2023

***Задание.***

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.

Программно реализовать алгоритм решения задачи **11. Container With Most Water**

****

Текст программы задания :

#include <stdio.h>

#include <iostream>

#include <stdlib.h>

struct edge {

int u;

int v;

int w;

};

struct graph {

int n;

int m;

struct edge\* edges;

};

int compare\_edges(const void\* a, const void\* b) {

struct edge\* e1 = (struct edge\*)a;

struct edge\* e2 = (struct edge\*)b;

return e1->w - e2->w;

}

int find(int x, int\* parent) {

if (parent[x] == x) {

return x;

}

else {

return find(parent[x], parent);

}

}

void union\_sets(int x, int y, int\* parent, int\* rank) {

int x\_root = find(x, parent);

int y\_root = find(y, parent);

if (x\_root != y\_root) {

if (rank[x\_root] < rank[y\_root]) {

parent[x\_root] = y\_root;

}

else if (rank[x\_root] > rank[y\_root]) {

parent[y\_root] = x\_root;

}

else {

parent[y\_root] = x\_root;

rank[x\_root]++;

}

}

}

struct graph\* kruskal(struct graph\* g) {

struct graph\* t = (struct graph\*)malloc(sizeof(struct graph));

t->n = g->n;

t->m = 0;

t->edges = (struct edge\*)malloc(g->n \* sizeof(struct edge));

qsort(g->edges, g->m, sizeof(struct edge), compare\_edges);

int\* parent = (int\*)malloc(g->n \* sizeof(int));

int\* rank = (int\*)malloc(g->n \* sizeof(int));

for (int i = 0; i < g->n; i++) {

parent[i] = i;

rank[i] = 0;

}

for (int i = 0; i < g->m; i++) {

struct edge e = g->edges[i];

if (find(e.u, parent) != find(e.v, parent)) {

t->edges[t->m++] = e;

union\_sets(e.u, e.v, parent, rank);

}

}

free(parent);

free(rank);

return t;

}

struct graph\* prim(struct graph\* g) {

struct graph\* t = (struct graph\*)malloc(sizeof(struct graph));

t->n = g->n; t->m = 0;

t->edges = (struct edge\*)malloc(g->n \* sizeof(struct edge));

int\* visited = (int\*)malloc(g->n \* sizeof(int));

for (int i = 0; i < g->n; i++) {

visited[i] = 0;

}

int start = rand() % g->n;

visited[start] = 1;

for (int i = 0; i < g->n - 1; i++) {

struct edge min\_edge;

min\_edge.w = INT\_MAX;

for (int j = 0; j < g->m; j++) {

struct edge e = g->edges[j];

if ((visited[e.u] && !visited[e.v]) || (!visited[e.u] &&

visited[e.v])) {

if (e.w < min\_edge.w) {

min\_edge = e;

}

}

}

t->edges[t->m++] = min\_edge;

visited[min\_edge.u] = 1;

visited[min\_edge.v] = 1;

}

free(visited);

return t;

}

void print\_graph(struct graph\* g) {

for (int i = 0; i < g->m; i++) {

printf("(%c, %c) - %d\n", g->edges[i].u + 'A', g->edges[i].v + 'A', g->edges[i].w);

}

}

void free\_graph(struct graph\* g) {

free(g->edges);

free(g);

}

struct graph\* create\_test\_graph\_from\_matrix(int matrix[][8], int n) {

struct graph\* g = (struct graph\*)malloc(sizeof(struct graph));

g->n = n;

g->m = 0;

g->edges = (struct edge\*)malloc(n \* n \* sizeof(struct edge));

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

for (int j = i + 1; j < n; j++) {

if (matrix[i][j] != 0) {

struct edge e;

e.u = i;

e.v = j;

e.w = matrix[i][j];

g->edges[g->m++] = e;

}

}

}

return g;

}

int main() {

int n = 8;

int matrix[8][8] = {

{0, 4, 6, 0, 0, 0, 0, 0},

{4, 0, 0, 1, 3, 0, 0, 0},

{6, 0, 0, 1, 0, 3, 0, 0},

{0, 1, 1, 0, 3, 4, 0, 0},

{0, 3, 0, 3, 0, 0, 5, 2},

{0, 0, 3, 4, 0, 0, 0, 5},

{0, 0, 0, 0, 5, 0, 0, 1},

{0, 0, 0, 0, 2, 5, 1, 0}, };

struct graph\* G = create\_test\_graph\_from\_matrix(matrix, n);

printf("\nKruskal:\n");

struct graph\* T1 = kruskal(G);

print\_graph(T1);

printf("\nPrima:\n");

struct graph\* T2 = prim(G);

print\_graph(T2);

free\_graph(G);

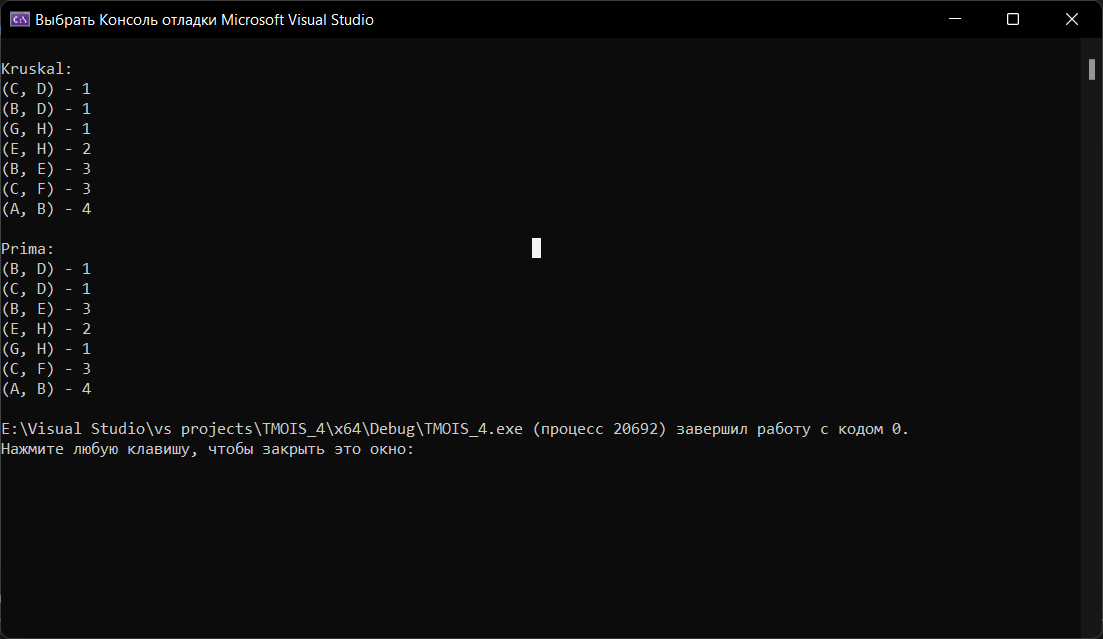
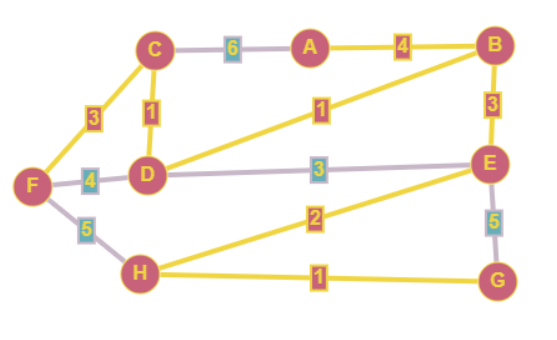
free\_graph(T1);

free\_graph(T2);

return 0;

}

Результат действия программы:

Решение задачи с LeetCode:

int maxArea(int\* height, int heightSize) {

int left = 0;

int right = heightSize - 1;

int max\_area = 0;

while (left < right) {

int current\_area = (right - left) \* (height[left] < height[right] ?

height[left] : height[right]);

if (current\_area > max\_area) {

max\_area = current\_area;

}

if (height[left] < height[right]) {

left++;

}

else {

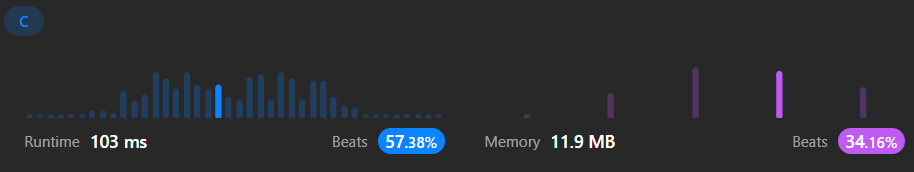
right--;

}

}

return max\_area;

}



1. Формула Кэли — это теорема, утверждающая, что число деревьев с пронумерованными вершинами равно n^(n-2), где n — число вершин. Эта формула определяет число остовных деревьев для полного графа K\_n, то есть графа, в котором каждая вершина соединена ребром с каждой другой.

2. По формуле Кэли, число остовных деревьев для n = 3 равно 3^(3-2) = 3.

3. По формуле Кэли, число остовных деревьев для n = 4 равно 4^(4-2) = 16.

4. Алгоритм Дейкстры не находит остовное дерево, а лишь кратчайшие пути. Однако, если мы захотим построить остовное дерево на основе этих путей, то мы можем взять все рёбра, которые использовались в кратчайших путях от начальной вершины до всех остальных. Такое дерево будет называться деревом кратчайших путей. Оно не обязательно будет минимальным остовным деревом, то есть таким, у которого сумма весов рёбер минимальна.

5. Да, может быть несколько минимальных остовных деревьев для одного и того же графа. Это происходит, когда есть несколько рёбер с одинаковым минимальным весом, и можно выбрать любое из них для построения остовного дерева.

6. Да, алгоритмы Прима и Крускаля являются жадными алгоритмами. Это означает, что они на каждом шаге делают локально оптимальный выбор, не учитывая последствия для всего решения. В случае поиска минимального остовного дерева, жадный выбор заключается в том, чтобы добавлять ребро с минимальным весом, которое не создает цикл в текущем дереве.

7. Алгоритм Прима требует отсортировать рёбра по весу и поддерживать очередь с приоритетом, в которой хранятся кандидаты для добавления в остовное дерево. Сложность сортировки рёбер составляет O(E log E), где E - число рёбер в графе. Алгоритм Крускала также требует отсортировать рёбра по весу, что занимает O(E log E) времени.

Вывод: В ходе данной лабораторной работы мы изучили основные алгоритмы нахождения кратчайшего пути в связном графе.