Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №3
По дисциплине « ***Алгоритмы и структуры данных*** »
Тема: ***«Нахождение минимального остовного дерева связанного неориентированного графа.»***

Выполнил:
Студент 2 курса
Группы ПО-11(2)
Сымоник И.А
Проверила:
Глущенко Т.А

Брест 2023

Цель работы: изучить алгоритмы Прима и Краскала для нахождения минимального остовного дерева связанного неориентированного графа .

Вариант 6
Ход работы

Задание 1. Найти минимальное остовное дерево для заданного графа *G* алгоритмом *Прима* и *Крускаля*. Варианты графов указаны в *таблице 1.* Граф задан списком ребер.

№	*Кол. верш.*	*Кол. ребер*	*Список ребер*	*Веса ребер*
6.	7	12	(a,b),(a,c),(a,d),(a,f), (b,e),(b,d),(b,g), (c,f),(d,f),(d,g),(e,g),(f,g)	2,4,7,5,3,6, 8,6.4,6,7,6

Исходный код:

```
#include <list>
#include <vector>
#include <queue>
#include <Windows.h>
#include <iostream>
#include <algorithm>

class DSU
{
public:

    DSU(int size)
    {
        for (int i = 0; i < size; i++)
        {
            parent.push_back(i);
            rank.push_back(0);
        }
    };

    int Find(int n)
    {
        return (n == parent[n]) ? (n) : (parent[n] = Find(parent[n]));
    };

    void Union(int first, int second)
    {
        first = Find(first);
        second = Find(second);

        if (first != second)
```

```
        {
            if(rank[first] < rank[second])
                std::swap(first, second);

            parent[first] = second;
            if (rank[first] == rank[second])
                rank[first]++;

        }

    };


private:
    std::vector<int> parent;
    std::vector<int> rank;
};

struct Edge
{
    Edge(int v, int u, int w)
        :weight(w), u(u), v(v)
    {

    }

    bool operator<(Edge const& oth)
    {
        return this->weight < oth.weight;
    };

    int weight;
    int v;
    int u;
};



void Craskal(std::vector<Edge>& edges, int countOfEdges)
{
    DSU un(countOfEdges);

    std::vector<Edge> res;

    int cost = 0;

    std::sort(edges.begin(), edges.end());

    for (auto& i : edges) {
        if (un.Find(i.u) != un.Find(i.v)) {
            cost += i.weight;
            res.push_back(i);
            un.Union(i.u, i.v);
        }
    }

    for (auto& i : res)
    {
        std::cout << static_cast<char>(i.v + 97) << " -> " << 
static_cast<char>(i.u + 97) << " Weight: " << i.weight << std::endl;
    }

}
```

```
std::list<std::pair<int, int>>* CreateMatrix(const std::vector<std::pair<char,
char>>& edges, const std::vector<int>& weights, int countOfVert)
{

	std::list<std::pair<int, int>>* edg = new std::list<std::pair<int,
int>>[countOfVert];

	int j = 0;
	for (auto& i : edges)
	{
		edg[i.first - 97].push_back(std::make_pair(i.second - 97,
weights[j]));
		edg[i.second - 97].push_back(std::make_pair(i.first - 97,
weights[j]));

		j++;
	}

	return edg;
}



void primMST(std::list<std::pair<int, int>> adj[], int countOfVert)
{
    std::priority_queue< std::pair<int,int>, std::vector <std::pair<int,int>>,
std::greater<std::pair<int,int>> > pq;
    int src = 0;
    std::vector<int> key(countOfVert, INT_MAX);
    std::vector<int> parent(countOfVert, -1);
    std::vector<bool> inMST(countOfVert, false);
    pq.push(std::make_pair(0, src));
    key[src] = 0;
    while (!pq.empty())
    {

        int u = pq.top().second;
        pq.pop();
        if (inMST[u] == true) {
            continue;
        }

        inMST[u] = true;
        for (auto& x : adj[u])
        {
            int v = x.first;
            int weight = x.second;
            if (inMST[v] == false && key[v] > weight)
            {
                key[v] = weight;
                pq.push(std::make_pair(key[v], v));
                parent[v] = u;
            }
        }
    }
    for (int i = 1; i < countOfVert; ++i)
        printf("%c -> %c  Weight: %d\n", parent[i] + 'a', i + 'a', key[i]);
}


int main()
```

```
{
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);

	std::vector<std::pair<char, char>> edges = {
{'a','b'},{'a','c'},{'a','d'},{'a','f'},{'b','e'},{'b','d'},
			{'b','g'},{'c','f'},{'d','f'},{'d','g'},{'e','g'},{'f','g'} };

	std::vector<int> weight = { 2,4,7,5,3,6,8,6,4,6,7,6 };

    std::vector<Edge> ed;

    for (int i = 0; i < edges.size(); i++)
    {
        ed.push_back(Edge(static_cast<int>(edges[i].first - 97),
static_cast<int>(edges[i].second - 97), weight[i]));
    }

    std::cout << "Алгоритм Прима" << std::endl;

    // PRIMA
    auto mat = CreateMatrix(edges, weight, 7);
    primMST(mat,7);
    // PRIMA

    std::cout << std::endl << std::endl;
    std::cout << "Алгоритм Краскала" << std::endl;
    // CRASKAL
    Craskal(ed, 7);
    // CRASKAL
}
```

Вывод программы:

Алгоритм Прима

a -> b Weight: 2

a -> c Weight: 4

f -> d Weight: 4

b -> e Weight: 3

a -> f Weight: 5

f -> g Weight: 6

Алгоритм Краскала

a -> b Weight: 2

b -> e Weight: 3

a -> c Weight: 4

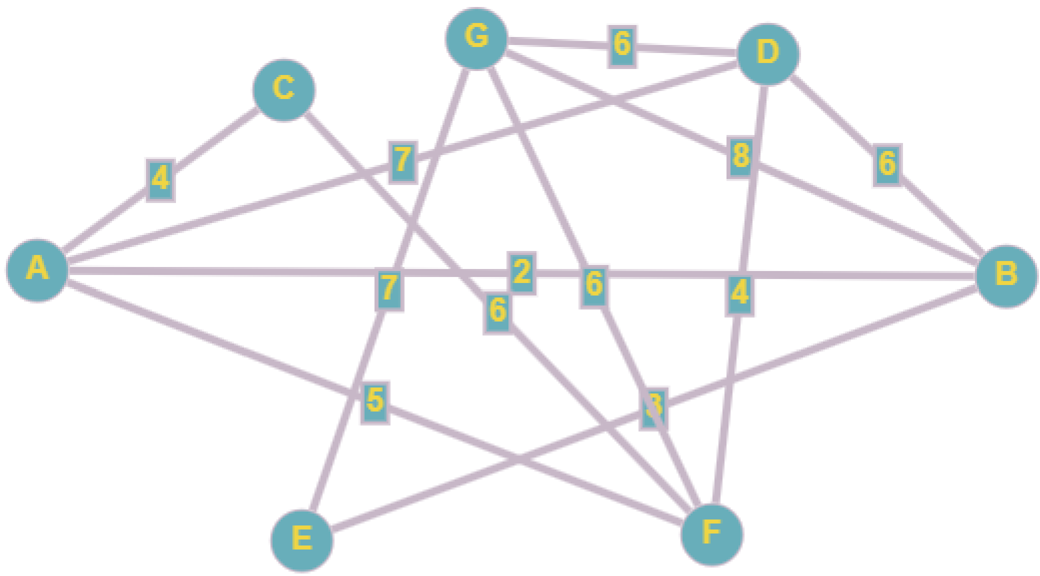
d -> f Weight: 4

a -> f Weight: 5

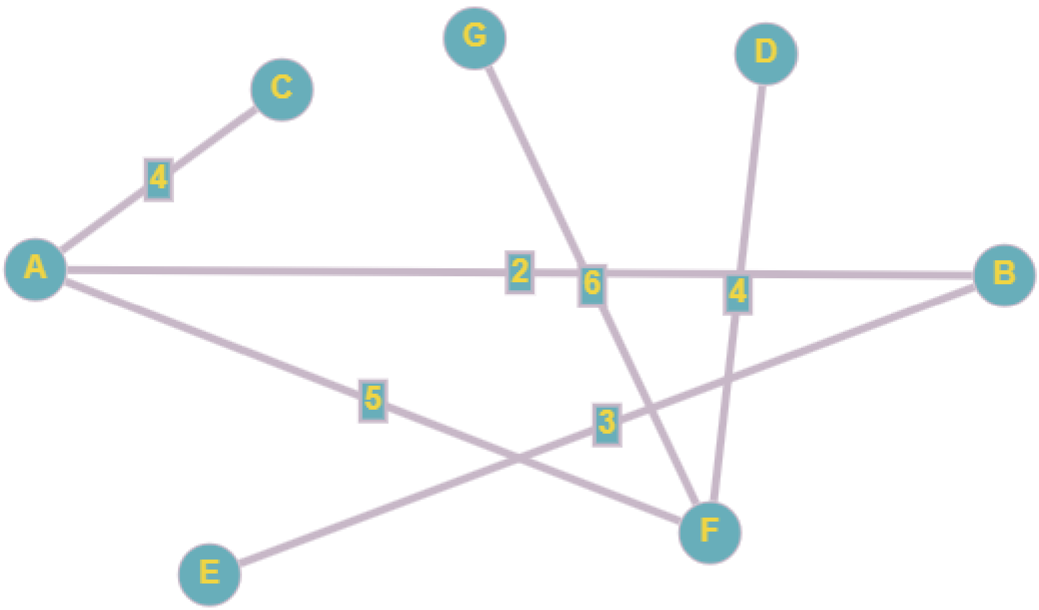
d -> g Weight: 6

Задание 2. Графически изобразить граф и его минимальное остовное дерево.

Изображение графа:



Изображение минимального остовного графа:



Задание 3. Решить задачу *223.Rectangle Area ресурса LeetCode* (рассмотреть все возможные случаи расположения прямоугольников относительно друг друга и правильно составить соответствующие *оптимальные* условия).

Профиль: https://leetcode.com/DOXECEES/

```
class Solution {
public:
    int computeArea(int ax1, int ay1, int ax2, int ay2, int bx1, int by1, int
bx2, int by2) {

        int width1  = ax2 - ax1;
        int height1 = ay2 - ay1;

        int width2  = bx2 - bx1;
        int height2 = by2 - by1;

        if (bx1 >= ax2 || bx2 <= ax1 || by1 >= ay2 || by2 <= ay1) {
            return ((width1 * height1) + (width2 * height2));
        }
```

```
        int unW = std::min(ax2, bx2) - std::max(ax1, bx1);
        int unH = std::min(ay2, by2) - std::max(ay1, by1);

        return ((width1 * height1) + (width2 * height2) - (unW * unH));
    }
};
```

Runtime

0ms

Beats 100.00%of users with C++

Memory

6.24MB

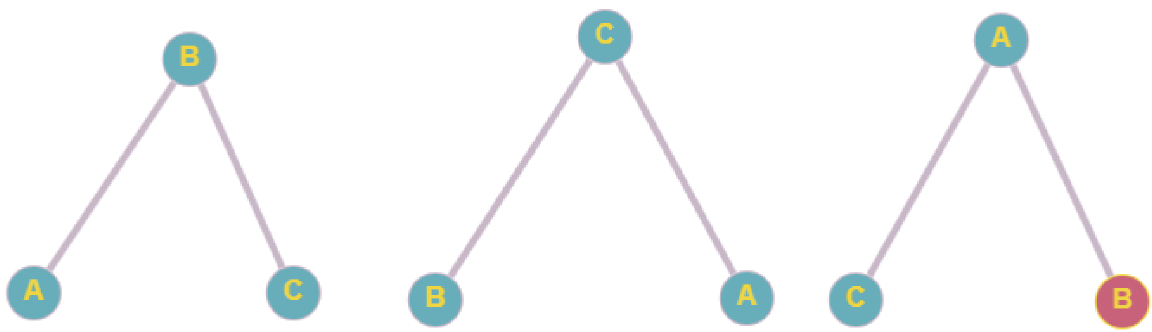
Beats 57.80%of users with C++

Вопросы к лабораторной работе:

1. Для какого графа определяет число остовных деревьев формула *Кэли*.

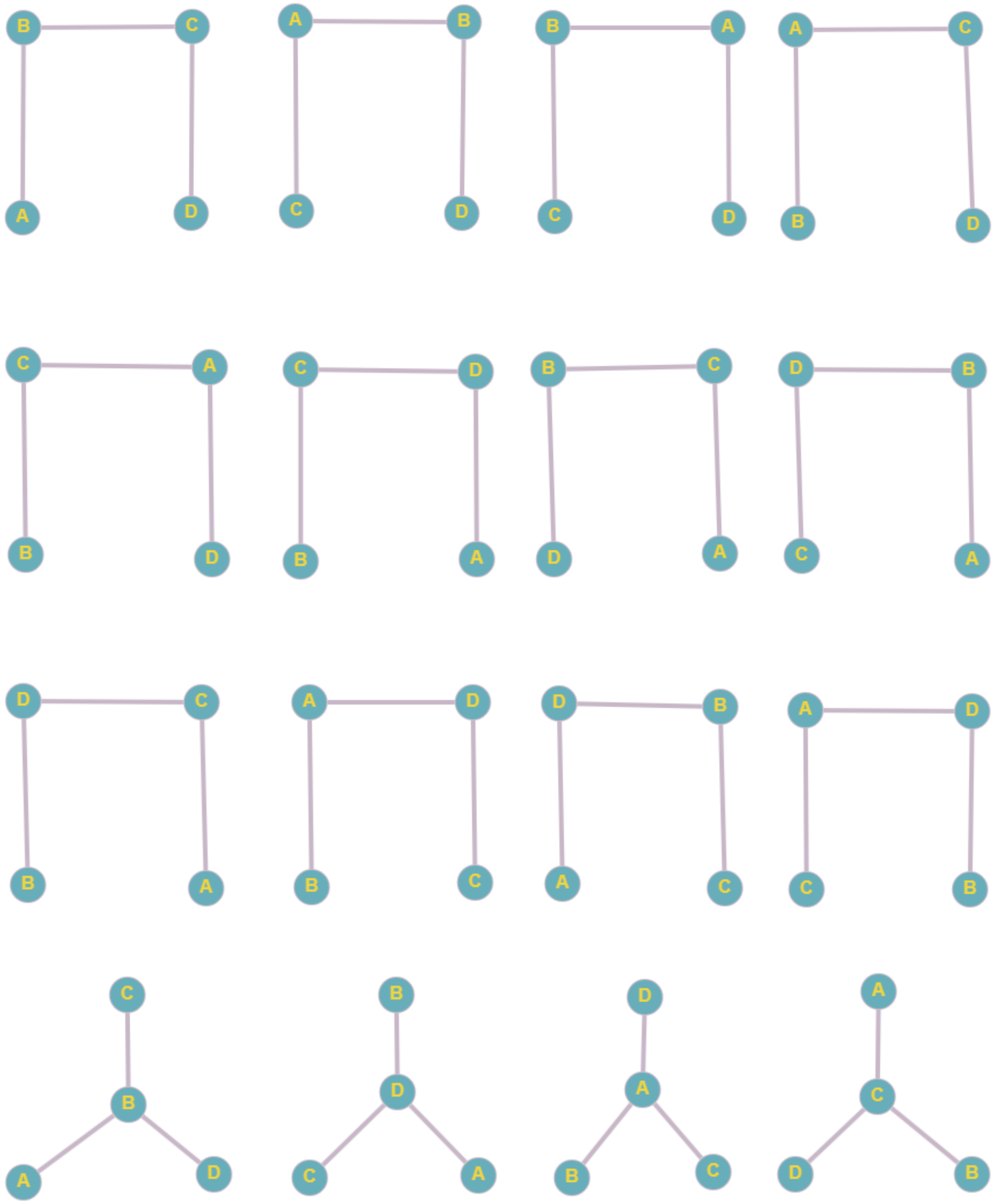
Формула Кэли определяет число остовных деревьев для полного графа.

2. Подсчитать по формуле *Кэли* и нарисовать число остовных деревьев для *n = 3*.
Формула Кэли : $N = n^{n-2} = 3^{3-2} = 3$.



3. Подсчитать по формуле *Кэли* и нарисовать число остовных деревьев для $n = 4$.

Формула Кэли : $N = n^{n-2} = 4^{4-2} = 16$.



4. Какое остовное дерево находится алгоритмом *Дейкстры*?

Алгоритм Дейкстры находит кратчайшее остовное дерево взвешенного графа, соединяющее все вершины с минимальной общей стоимостью путей между ними.

5. Может ли быть несколько *минимальных* остовных деревьев?

Да может. Это происходит, когда существует несколько путей между вершинами с одинаковыми весами, и при выборе одного из этих путей для построения остовного дерева получается несколько различных минимальных остовных деревьев.

6. Приведите оценку *временной сложности* обеих алгоритмов (*«О большое»*).

Алгоритм Краскала – O(m * log(n)) – при использовании бинарной кучи

Алгоритм Прима – O(n^2), при использовании массива

O(n * log(n)), при использовании системы не пересекающихся множеств

7. Являются ли алгоритмы *жадными*, если да, то почему?

Да, являются, так как на каждом шаге они пытаются найти локально оптимальный вариант.

8. Какие структуры данных позволяют оптимизировать алгоритмы?

Алгоритм Краскала можно оптимизировать, если использовать систему непересекающихся множеств.

Алгоритм Прима можно оптимизировать, если использовать бинарную или Фибоначчиеву кучу.

Вывод: Изучили алгоритмы Прима и Краскала для нахождения минимального остовного дерева связанного неориентированного графа .

Брест 2023