**İ.T.Ü.**

**Bilgisayar ve Bilişim Fakültesi**

**Bilgisayar Mühendisliği Bölümü**



**ANALYSIS OF ALGORITHMS**

**HW2**

**Aykut Akın**

**040080177**

**18.04.2011**

**Description of how my program should be compiled and run:**

I use Microsoft Visual C++ 2010 Express to develop the program. Program can be compiled and run on Linux/Unix using g++.

You should write a command like that for compile the Lightening a City solution Linux terminal.

g++ city.cpp graph.cpp –o city

You should give a command like that for run the Lightening a City solution after compilation.

./city

**Description of homework**

Economic crisis affected many countries. AOACountry is one of these countries. The government wants to make some savings. In the cities of AOACountry, street lights are turned on all night. The cost of a turned on street light is 1 AOALiras. The government decides to turn off some lights but they still want to establish a safe, peaceful environment. Therefore, the government wants to supply at least one lighted road from a crossroad to another crossroad even if the road is not the shortest path between the crossroads. Roads consist of one or more streets. Crossroads connect the streets to each other. Streets are dual carriageway; cars both go and come back from a street.

You are going to be given a city plan of some cities in AOACountry which holds crossroads and number of lights on the streets. By supplying the restriction of the government maximum how many AOALiras can be saved for a day?

City plan documentation format:

A city consists of m crossroads and n streets. You are going to be given an input file called “cityplan.txt”. The first line of the input file holds the numbers of m and n respectively. The following lines hold mi. and mj. crossroads and the number of the lights on the street between these crossroads (i != j).

As an output, print the lighted streets by representing them with crossroads that they connect. Give the total savings in AOALiras, and calculate and print the percentage of gain. In the last line, give the running time of the algorithm in milliseconds. You can run the algorithm for 1000 times in order to obtain a statistically significant value.

Input file format:

7 11

0 1 7

0 3 5

1 2 8

1 3 9

1 4 7

2 4 5

3 4 15

3 5 6

4 5 8

4 6 9

5 6 10

Output format:

0 1

0 3

1 4

2 4

3 5

4 6

Total savings: 50 AOALiras

Total gain: (39 / 89) 43.82 %

Running time: 5 ms

**Solution of the problem**

We can consider roads as a edges of graphs, crossroads as a nodes of graphs and street lights as a weight of an edge. Streets are dual carriageway; cars both go and come back from a street so graph is undirected.

5

10

8

9

15

6

8

7

9

5

7

**Graph of the given input**



The government wants to supply at least one lighted road from a crossroad to another crossroad. The government decides to turn off some lights but they still want to establish a safe, peaceful environment. If we think this situation on graphs, they want to cover every node. They also want to save maximum money from this operation. So if we make a tree which includes all nodes and if the tree has minimum weight we solve the problem.

We can use Minimum Spanning Tree algorithms to solve this problem. I choose Kruskal algorithm and solve this. You can see the solution below.

5

10

8

9

15

6

8

4

9

5

7

**Minimum Spanning Tree of the given input**



**Kruskal algorithm in C++ code**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

void Graph::kruskal(){

int \*cycles = new int[m];

vector<Edge\*> edgescopy = edges; //not to loose original graph

sort(edgescopy.begin(), edgescopy.end(), sortFunc);//sort edges according to

sortFunc function

for(int i=0; i<m; i++)

cycles[i] = i;

while(mst.size()<(m-1) && edgescopy.size()){//if mst did not reach the maximum

size and there is still edge in graph

if(cycles[edgescopy[0]->node1]!=cycles[edgescopy[0]->node2]){//if there is no cycle

for(int i=0; i<m; i++){//for all nodes

if(cycles[i]==cycles[edgescopy[0]->node2])

cycles[i] = cycles[edgescopy[0]->node1];//rearrange cycles

(write the oldest connected edge)

}

mst.push\_back(edgescopy[0]);//add to mst

}

edgescopy.erase(edgescopy.begin(),edgescopy.begin()+1);//delete checked edge

}

for(int i=0;i<mst.size();i++)

totalmst += mst[i]->weight;

}

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Sorting = O(mlogm)

Make a set containing singleton u = O(n) So, algorithm runs O(mlogm) time.

Look for u and v are different set = O(logn)

**My header files & explanations**

This part of the report contains explanation about classes and necessary methods of the program.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_edge.h\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#ifndef \_\_EDGE\_H\_\_

#define \_\_EDGE\_H\_\_

#include <iostream>

#include <iomanip>

#include <cstdlib>

#include <fstream>

#include <string>

using namespace std;

class Edge{ //This class is a simple edge class.

//just include start node, finish

//node and weight of the edge

public:

int node1;

int node2;

int weight;

};

#endif

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_graph.h\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#ifndef \_\_GRAPH\_H\_\_

#define \_\_GRAPH\_H\_\_

#include "edge.h"

#include <vector>

using namespace std;

class Graph{ //This class is a graph class

vector<Edge\*> edges; //Graph with contains edges

vector<Edge\*> mst; //Minimum spanning tree of a graph

int m; //node number

int n; //edge number

int totalmst; //minimum weight

int total; //weight

void readData(string fileName); //read data from txt file

public:

Graph(string fileName); //constructor

~Graph(); //destructor

void kruskal(); //Kruskal algorithm which finds MST

void writeData(int time); //generate output file

void clear(); //clears values for rerunning

};

#endif

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_