TCP2101 ALGORITHM DESIGN & ANALYSIS
SEMESTER 2,
YEAR 2020/2021

ASSIGNMENT 2

GROUP 3

NAME	**STUDENT ID**
CHANG KAI BOON (Team Leader)	1181101282
ANG KELVIN	1181101297
PRITESH PATEL	1181101645

Table of Content

1.0 Display and Sort

1.1 Program 1

Planet
- x : int - y : int - z : int - iWeight : int - iProfit : int - name : string
+ Planet () +Planet (name : string, x : int , y : int, z: int , iWeight : int , iProfit : int) + getX () : int + getY () : int + getZ () : int + getIweight () : int + getIprofit () : int + getName () : string + getDistance(B : Planet) : int

(Figure 1 : Planet Class Diagram)

The figure (Figure 1 : Planet Class Diagram) above shows a class diagram of the Planet class. The class is needed to store information about the planet.There are 6 private variables x , y . z , iWeight, iProfit and name and each of the private variables have their own public accessor and they are getX() ,getY() ,getZ() ,getIweight() ,getIprofit(), and getName() respectively. The class also has a public method which is getDistance(B : Planet), it is needed to calculate the distance between 2 planets, The algorithm of the method is shown in the pseudo code below :

Algorithm 1 getDistance

procedure getDistance(Planet B)
    x2 = B.getX()
    y2 = B.getY()
    z2 = B.getZ()
    total_dis = $(x-x2)^2 + (y-y2)^2 + (z-z2)^2$
    return $\text{total\_dis}^{1/2}$
end procedure

Edge
-weight : int -name : string
+Edge () +Edge(name : string, weight : int) +getEdgeName() : string +getEdgeWeight() : int

(Figure 2 : Edge Class Diagram)

The figure (Figure 2 : Edge Class Diagram) above shows a class diagram of edge class. The edge class is needed to store information about the edges of the planet. There are 2 private variables inside the class, weight and name. Each of these variables have their own accessor, getEdgeName() and getEdgeWeight respectively.

Algorithm 2 add_edge

procedure add_edge(list p,int u, int v)
    adjMatrix[u][v] = p[u].getDistance(p[v]);
    adjMatrix[u][v] = p[u].getDistance(p[v]);
end procedure

Algorithm 3 merge

procedure merge(list edgeList,int l, int m, int r)
    int n1 = m - l + 1;

```
int n2 = r - m;
Edge L[n1];
Edge R[n2];
for i = 0 until i < n1 do
    L[i] = edgeList[l + i]
endfor
for j = 0 until j < n2 do
     R[j] = edgeList[m + 1 + j]
endfor
i = 0
j = 0
k = l
while i < n1 and j < n2 do
    if L[i].getEdgeWeight() <= R[j].getEdgeWeight() then
        edgeList[k] = L[i];
        i++
    else then
        edgeList[k] = R[j];
        j++
    endIf
    k++
    while i < n1 do
        edgeList[k] = L[i]
        i++
        k++
    endWhile
    while j < n2 do
        edgeList[k] = R[j]
        j++
        k++
    endWhile
endWhile
end procedure
```

Algorithm 4 mergeSort

```
procedure mergeSort(list edgeList, int l, int r)
    if l >= r then
        return
    endIf
    int m = l+(r-l)/2
    mergeSort(edgeList,l,m)
    mergeSort(edgeList,m+1,r)
    merge(edgeList,l,m,r)
end procedure
```

Algorithm 3 mergePlanet

```
procedure mergePlanet(list planetList,int l, int m, int r)
    int n1 = m - l + 1;
    int n2 = r - m;
    Edge L[n1];
    Edge R[n2];
    for i = 0 until i < n1 do
        L[i] = planetList[l + i]
    endfor
    for j = 0 until j < n2 do
        R[j] = planetList[m + 1 + j]
    endfor
    i = 0
    j = 0
    k = l
    while i < n1 and j < n2 do
        if L[i].getEdgeWeight() <= R[j].getEdgeWeight() then
            planetList[k] = L[i];
            i++
        else then
            planetList[k] = R[j];
            j++
        endIf
        k++
```

```
    while i < n1 do
        planetList[k] = L[i]
        i++
        k++
    endWhile
    while j < n2 do
        planetList[k] = R[j]
        j++
        k++
    endWhile
endWhile
end procedure
```

Algorithm 4 mergeSortPlanet

```
procedure mergeSort(list planetList, int l, int r)
    if l >= r then
        return
    endIf
    int m = l+(r-l)/2
    mergeSort(planetList,l,m)
    mergeSort(planetList,m+1,r)
    merge(planetList,l,m,r)
end procedure
```

1.2 Source Code

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <sstream>
#include <cmath>
#include <stdio.h>
#include <limits.h>
#include <fstream>
#include<bits/stdc++.h>
```

```
#include <iostream>
#include <string>
#include <chrono>
#include <map>
#include <vector>
using namespace std::chrono;
using namespace std;

int adjMatrix[10][10];

class Planet{
    int x;
    int y;
    int z;
    int iWeight;
    int iProfit;
    string name;

    public:
        Planet(){}

        Planet(string name,int x,int y, int z, int iWeight, int iProfit){
            this->name = name;
            this->x = x;
            this->y = y;
            this->z = z;
            this->iWeight = iWeight;
            this->iProfit = iProfit;
        }

        int getX(){
            return x;
        }

        int getY(){
            return y;
        }

        int getZ(){
            return z;
        }

        int getIweight(){
            return iWeight;
        }

        int getIprofit(){
            return iProfit;
        }
```

```
        string getName(){
            return name;
        }

        int getDistance(Planet B){
            int total_dis = pow((x - B.x),2) + pow((y - B.y),2) + pow((z - B.z),2);
            return sqrt(total_dis);
        }


};

class Edge{
    int weight;
    string name;

    public:
        Edge(){}

        Edge(string name,int weight){
            this->name = name;
            this->weight = weight;
        }

        string getEdgeName(){
            return name;
        }

        int getEdgeWeight(){
            return weight;
        }

};

void add_edge(vector<Planet> p,int u,int v){
    adjMatrix[u][v] = p[u].getDistance(p[v]);
    adjMatrix[v][u] = p[u].getDistance(p[v]);
}

void merge(vector<Edge>& edgeList, int l, int m, int r)
{
    int n1 = m - l + 1;
    int n2 = r - m;

    Edge L[n1];
    Edge R[n2];

    for (int i = 0; i < n1; i++)
        L[i] = edgeList[l + i];
    for (int j = 0; j < n2; j++)
```

```
        R[j] = edgeList[m + 1 + j];


    int i = 0;

    int j = 0;

    int k = l;

    while (i < n1 && j < n2) {
        if (L[i].getEdgeWeight() <= R[j].getEdgeWeight() ) {
            edgeList[k] = L[i];
            i++;
        }
        else {
            edgeList[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
        edgeList[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
        edgeList[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(vector<Edge>& edgeList,int l,int r){
    if(l>=r){
        return;
    }
    int m =l+ (r-l)/2;
    mergeSort(edgeList,l,m);
    mergeSort(edgeList,m+1,r);
    merge(edgeList,l,m,r);
}

void printArray(vector<Edge> edgeList, int size)
{
    cout << left << setw(3) << "No";
    cout << right << setw(3) << "Edge" << "  ";
    cout << right << setw(3) << "Distance" << "  ";
    cout <<endl;
```

```
    for (int i = 0; i < size; i++){
        cout << left << setw(3) << i+1 ;
        cout << right << setw(3) <<edgeList[i].getEdgeName() << "  ";
        cout << right << setw(3) <<edgeList[i].getEdgeWeight() << "  ";

        cout <<endl;
    }
}

void mergePlanet(vector<Planet>& planets, int l, int m, int r)
{
    int n1 = m - l + 1;
    int n2 = r - m;

    Planet L[n1];
    Planet R[n2];

    for (int i = 0; i < n1; i++)
        L[i] = planets[l + i];
    for (int j = 0; j < n2; j++)
        R[j] = planets[m + 1 + j];


    int i = 0;

    int j = 0;

    int k = l;

    while (i < n1 && j < n2) {
        if (L[i].getIweight() >= R[j].getIweight() ) {
            planets[k] = L[i];
            i++;
        }
        else {
            planets[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
        planets[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
        planets[k] = R[j];
```

```
        j++;
        k++;
    }
}

void mergeSortPlanet(vector<Planet>& planets,int l,int r){
    if(l>=r){
        return;
    }
    int m =l+ (r-l)/2;
    mergeSortPlanet(planets,l,m);
    mergeSortPlanet(planets,m+1,r);
    mergePlanet(planets,l,m,r);
}

void printPlanet(vector<Planet> planets, int size)
{
    cout << left << setw(3) << "No";
    cout << right << setw(3) << "Planet" << "  ";
    cout << right << setw(3) << "Weight" << "  ";
    cout <<endl;

    for (int i = 0; i < size; i++){
        cout << left << setw(3) << i+1 ;
        cout << right << setw(3) <<planets[i].getName() << "  ";
        cout << right << setw(3) <<planets[i].getIweight() << "  ";
        cout <<endl;
    }
}



int main()
{
    ifstream File("A2planets_TT8V_Group3.txt");
    vector<Planet> planets;
    string a;
    int b,c,d,e,f;

    while (File >> a >> b >> c >> d >> e >> f)
    {
        Planet temp(a,b,c,d,e,f);
        planets.push_back(temp);
    }

    for(int i=0; i<planets.size(); ++i){
        cout << planets[i].getName() << " " << planets[i].getX() << " " << planets[i].getY() << " "
        << planets[i].getZ() << " " << planets[i].getIweight() << " " << planets[i].getIprofit() <<
endl;
    }
```

```
    vector<vector<Planet>> connected;
    connected.push_back({planets[3],planets[9],planets[7],planets[5]});
    connected.push_back({planets[3],planets[6],planets[4]});
    connected.push_back({planets[4],planets[8],planets[5]});
    connected.push_back({planets[1],planets[9],planets[0]});
    connected.push_back({planets[1],planets[6],planets[8],planets[2]});
    connected.push_back({planets[0],planets[7],planets[2]});
    connected.push_back({planets[1],planets[4],planets[8],planets[9]});
    connected.push_back({planets[9],planets[0],planets[8],planets[5]});
    connected.push_back({planets[6],planets[4],planets[2],planets[7]});
    connected.push_back({planets[3],planets[6],planets[7],planets[0]});
    cout << endl;

    cout << "Adjacency List :" << endl;


    for(int i=0 ; i<connected.size();i++){
        cout<< planets[i].getName();
        for(int j = 0; j<connected[i].size();j++)
            cout << " --> " << connected[i][j].getName() << " [W = " <<
connected[i][j].getDistance(planets[i]) << "]" ;
        cout << endl;
    }

    add_edge(planets,0, 3);
    add_edge(planets,0, 9);
    add_edge(planets,0, 7);
    add_edge(planets,0, 5);

    add_edge(planets,1, 3);
    add_edge(planets,1, 6);
    add_edge(planets,1, 4);

    add_edge(planets,2, 4);
    add_edge(planets,2, 8);
    add_edge(planets,2, 5);

    add_edge(planets,3, 0);
    add_edge(planets,3, 9);
    add_edge(planets,3, 1);

    add_edge(planets,4, 1);
    add_edge(planets,4, 6);
    add_edge(planets,4, 8);
    add_edge(planets,4, 2);

    add_edge(planets,5, 0);
    add_edge(planets,5, 7);
```

```
add_edge(planets,5, 8);
add_edge(planets,5, 2);

add_edge(planets,6, 1);
add_edge(planets,6, 9);
add_edge(planets,6, 8);
add_edge(planets,6, 4);

add_edge(planets,7, 0);
add_edge(planets,7, 9);
add_edge(planets,7, 8);
add_edge(planets,7, 5);

add_edge(planets,8, 7);
add_edge(planets,8, 6);
add_edge(planets,8, 4);
add_edge(planets,8, 2);

add_edge(planets,9, 0);
add_edge(planets,9, 7);
add_edge(planets,9, 6);
add_edge(planets,9, 3);
cout << endl;

cout << "Adjacency Matrix :" << endl;
string name[10] = { "A","B","C","D","E","F","G","H","I","J" };
cout << left << setw(4) << " " << " ";
for(int i = 0;i<10;i++){
    cout << left << setw(4) << name[i] << " ";
}
cout<<endl;
for(int i = 0; i < 10; i++) {
    cout << left << setw(4) << name[i] << " ";
    for(int j = 0; j < 10; j++) {
        cout << left << setw(4) << adjMatrix[i][j] << " ";
    }
    cout << endl;
}

vector<Edge> edgeList;
for(int i=0;i<10;i++){
    string tempRowName = name[i];
    for(int j=0;j < 10;j++){
        string tempColName = name[j];
        if(adjMatrix[i][j] > 0){
            string cat = tempRowName + tempColName;
            Edge e(cat,adjMatrix[i][j]);
            edgeList.push_back(e);
        }
    }
```

```
    }

    cout << endl;
    cout << "List of edges before merge sorting:" << endl;
    printArray(edgeList, edgeList.size());


    cout << endl;
    cout << "List of edges after merge sorting in ascending order of distance:" << endl;
    mergeSort(edgeList, 0, edgeList.size()-1);
    printArray(edgeList, edgeList.size());

    cout << endl;
    cout << "List of planets before merge sorting:" << endl;
    printPlanet(planets, planets.size());


    cout << endl;
    cout << "List of planets after merge sorting in descending order item weight:" << endl;
    mergeSortPlanet(planets, 0, planets.size()-1);
    printPlanet(planets, planets.size());
}
```

1.3 Program Outputs

```
Adjacency List :
Planet_A --> Planet_D [W = 13] --> Planet_J [W = 717] --> Planet_H [W = 580] --> Planet_F [W = 181]
Planet_B --> Planet_D [W = 122] --> Planet_G [W = 76] --> Planet_E [W = 138]
Planet_C --> Planet_E [W = 1229] --> Planet_I [W = 394] --> Planet_F [W = 1381]
Planet_D --> Planet_B [W = 122] --> Planet_J [W = 729] --> Planet_A [W = 13]
Planet_E --> Planet_B [W = 138] --> Planet_G [W = 74] --> Planet_I [W = 1182] --> Planet_C [W = 1229]
Planet_F --> Planet_A [W = 181] --> Planet_H [W = 741] --> Planet_C [W = 1381]
Planet_G --> Planet_B [W = 76] --> Planet_E [W = 74] --> Planet_I [W = 1127] --> Planet_J [W = 682]
Planet_H --> Planet_J [W = 440] --> Planet_A [W = 580] --> Planet_I [W = 884] --> Planet_F [W = 741]
Planet_I --> Planet_G [W = 1127] --> Planet_E [W = 1182] --> Planet_C [W = 394] --> Planet_H [W = 884]
Planet_J --> Planet_D [W = 729] --> Planet_G [W = 682] --> Planet_H [W = 440] --> Planet_A [W = 717]

Adjacency Matrix :
     A    B    C    D    E    F    G    H    I    J
A    0    0    0    13   0    181  0    580  0    717
B    0    0    0    122  138  0    76   0    0    0
C    0    0    0    0    1229 1381 0    0    394  0
D    13   122  0    0    0    0    0    0    0    729
E    0    138  1229 0    0    0    74   0    1182 0
F    181  0    1381 0    0    0    0    741  1327 0
G    0    76   0    0    74   0    0    0    1127 682
H    580  0    0    0    0    741  0    0    884  440
I    0    0    394  0    1182 1327 1127 884  0    0
J    717  0    0    729  0    0    682  440  0    0
```

Figure 3 : Program outputs, Adjacency Matrix and List

```
List of edges before merge sorting:
No Edge  Distance
1   AD   13
2   AF  181
3   AH  580
4   AJ  717
5   BD  122
6   BE  138
7   BG   76
8   CE  1229
9   CF  1381
10  CI  394
11  DA   13
12  DB  122
13  DJ  729
14  EB  138
15  EC  1229
16  EG   74
17  EI  1182
18  FA  181
19  FC  1381
20  FH  741
21  FI  1327
22  GB   76
23  GE   74
24  GI  1127
25  GJ  682
26  HA  580
27  HF  741
28  HI  884
29  HJ  440
30  IC  394
31  IE  1182
32  IF  1327
33  IG  1127
34  IH  884
35  JA  717
36  JD  729
37  JG  682
38  JH  440
```

```
List of edges after merge sorting in ascending order of distance:
No Edge  Distance
1   AD   13
2   DA   13
3   EG   74
4   GE   74
5   BG   76
6   GB   76
7   BD  122
8   DB  122
9   BE  138
10  EB  138
11  AF  181
12  FA  181
13  CI  394
14  IC  394
15  HJ  440
16  JH  440
17  AH  580
18  HA  580
19  GJ  682
20  JG  682
21  AJ  717
22  JA  717
23  DJ  729
24  JD  729
25  FH  741
26  HF  741
27  HI  884
28  IH  884
29  GI  1127
30  IG  1127
31  EI  1182
32  IE  1182
33  CE  1229
34  EC  1229
35  FI  1327
36  IF  1327
37  CF  1381
38  FC  1381
```

Figure 4 : Program outputs, Sorted List of edges

```
List of planets before merge sorting:
No Planet  Weight
1  Planet_A    0
2  Planet_B    8
3  Planet_C   14
4  Planet_D   20
5  Planet_E   13
6  Planet_F   13
7  Planet_G    8
8  Planet_H   14
9  Planet_I   11
10 Planet_J   15
```

```
List of planets after merge sorting in descending order item weight:
No Planet  Weight
1  Planet_D   20
2  Planet_J   15
3  Planet_C   14
4  Planet_H   14
5  Planet_E   13
6  Planet_F   13
7  Planet_I   11
8  Planet_B    8
9  Planet_G    8
10 Planet_A    0
```

Figure 5 : Program outputs, Sorted List of planets

2.0 Shortest Paths

2.1 Program 2

Algorithm 1 CalDisPla

```
procedure CalDisPla(Planet A ,Planet B)
    int total_distance = (A.x-B.x)^2 +  (A.y-B.y)^2 +(A.z-B.z)^2;
    return total_distance;
end procedure
```

Algorithm 2 minDistance

```
procedure minDistance(int dist[ ],bool sptSet[ ] )
    int min = INT_MAX,MIN_INDEX;
    for v=0 until v < size do
        if sptSet[v] == False and dist[v] <= min
            min = dist[v], min_index = v
        endif
    endfor
end procedure
```

Algorithm 3 printPath

```
procedure printPath(int parent[ ], int j)
    string name[10] = { "A","B","C","D","E","F","G","H","I","J" }
    if parent[j] == -1
        return
    endif
    printPath(parent, parent[j])
    string n = name[j]
    display n
end procedure
```

Algorithm 4 printSolution

```
procedure printSolution(int dist[ ],int n, int parent [ ])
    int src = 0
    char name[10] = { "A","B","C","D","E","F","G","H","I","J" }
    for i = 1 until i < size do
        char s = name [i]
        display s, dist[i]
        printPath(parent,i)
    endfor
end procedure
```

Algorithm 5 dijkstra

```
procedure dijkstra(int graph[size][size],int src )
    int dist[size]
    bool sptSet [size]
    int parent[size]
    for i = 0 until i < size do
        parent[0] = -1
        dist[i] = INT_MAX
        sptSet[i] = false
    endfor
    dist[src] = 0
    for count = 0 until count < size do
        int u = minDistance(dist, sptSet)
        sptSet[u] = true
        for v = 0 until v < size do
            if  not sptSet[v] and  graph[u][v] and dist[u] + graph[u][v] < dist[v]
                parent[v] = u
                dist[v] = dist[u] + graph[u][v]
            endif
        endfor
    endfor
end procedure
```

2.2 Source Code

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <sstream>
#include <cmath>
#include <stdio.h>
#include <limits.h>
using namespace std;

#define size 10

class Planet {
    public:
        int x;
        int y;
        int z;
        int weight;
        int profit;
        string name;
};

int calDisPla(Planet A, Planet B){
    int total_dis = pow((A.x - B.x),2) + pow((A.y - B.y),2) + pow((A.z - B.z),2);
    return sqrt(total_dis);
}

// Function to find the vertex with minimum distance value
int minDistance(int dist[],bool sptSet[]){
    // Initialize min value
    int min = INT_MAX, min_index;

    for (int v = 0; v < size; v++){
        if (sptSet[v] == false && dist[v] <= min){
            min = dist[v], min_index = v;
        }
    }
    return min_index;
}

// Function to print shortest path from source to j using parent array
void printPath(int parent[], int j){
    string name[10] = { "A","B","C","D","E","F","G","H","I","J" };
    // Base Case : If j is source
    if (parent[j] == - 1)
        return;
```

```
    printPath(parent, parent[j]);

    string n = name[j];
    //printf("%d ", j);
    //printf("{}", n);
    cout << n << " ";
}

// Function to print constructed distance array
void printSolution(int dist[], int n, int parent[]){
    int src = 0;
    char name[10] = { 'A','B','C','D','E','F','G','H','I','J'};
    printf("Vertex\t\t\t Distance\tPath");
    for (int i = 1; i < size; i++){
        char s  = name[i];
        printf("\n A -> %C \t\t %d\t\t A ", s, dist[i]);
        printPath(parent, i);
    }
}

// Function that implements Dijkstra's single source shortest path algorithm for a graph
represented using adjacency matrix representation
void dijkstra(int graph[size][size], int src) {
    // The output array. dist[i] will hold the shortest distance from src to i
    int dist[size];

    // sptSet[i] will true if vertex i is included / in shortest path tree or shortest distance from src
to i is finalized
    bool sptSet[size];

    // Parent array to store shortest path tree
    int parent[size];

    // Initialize all distances as
    // INFINITE and stpSet[] as false
    for (int i = 0; i < size; i++){
        parent[0] = -1;
        dist[i] = INT_MAX;
        sptSet[i] = false;
    }

    // Distance of source vertex from itself is always 0
    dist[src] = 0;

    // Find shortest path for all vertices
    for (int count = 0; count < size - 1; count++) {
        // Pick the minimum distance vertex from the set of vertices not yet processed. u is
always equal to src in the first iteration.
        int u = minDistance(dist, sptSet);
```

```
        // Mark the picked vertex as processed
        sptSet[u] = true;

        // Update dist value of the adjacent vertices of the picked vertex.
        for (int v = 0; v < size; v++)
            // Update dist[v] only if is not in sptSet, there is an edge from u to v, and total weight of
path from src to v through u is smaller than current value of dist[v]
            if (!sptSet[v] && graph[u][v] && dist[u] + graph[u][v] < dist[v]){
                parent[v] = u;
                dist[v] = dist[u] + graph[u][v];
            }
    }

    // print the constructed distance array
    printSolution(dist, size, parent);
}

//Plot graph
void initmap(char m[7][7]){
    for(int i=0; i<7; i++){
        for(int j=0; j<7; j++){
            m[i][j] = ' ';
        }
    }
}

void graph_planets(char m[7][7]){
    m[0][3] = 'A';
    m[4][0] = 'B';
    m[4][6] = 'C';
    m[2][0] = 'D';
    m[6][3] = 'E';
    m[2][6] = 'F';
    m[4][2] = 'G';
    m[2][4] = 'H';
    m[4][4] = 'I';
    m[2][2] = 'J';
}

void graph_connect(char m[7][7], int a, int b){
    switch (a) {
    case 1:     // A
        if (b == 4)     // connect to D
        {
            m[0][0] = '+';
            m[0][1] = '-';
            m[0][2] = '-';
            m[1][0] = '|';
        }
```

```
    if (b == 6)     // connect to F
    {
        m[0][6] = '+';
        m[0][5] = '-';
        m[1][6] = '|';
    }
    if (b == 10)     // connect to J
    {
        m[0][2] = '+';
        m[1][2] = '|';
    }
    if (b == 8)      // connect to H
    {
        m[0][4] = '+';
        m[1][4] = '|';
    }
    break;
case 2:     // B
    if (b == 4)     // connect to D
    {
        m[3][0] = '|';
    }
    if (b == 5)     // connect to E
    {
        m[6][0] = '+';
        m[6][1] = '-';
        m[6][2] = '-';
        m[5][0] = '|';
    }
    if (b == 7)     // connect to G
    {
        m[4][1] = '-';
    }
    break;
case 3:     // C
    if (b == 6)     // connect to F
    {
        m[3][6] = '|';
    }
    if (b == 5)     // connect to E
    {
        m[6][6] = '+';
        m[6][5] = '-';
        m[6][4] = '-';
        m[5][6] = '|';
    }
    if (b == 9)     // connect to I
    {
        m[4][5] = '-';
    }
```

```
        break;
    case 4:     // D
        if (b == 10)     // connect to J
        {
            m[2][1] = '-';
        }
        break;
    case 5:     // E
        if (b == 7)     // connect to G
        {
            m[6][2] = '+';
            m[5][2] = '|';
        }
        if (b == 9)     // connect to I
        {
            m[6][4] = '+';
            m[5][4] = '|';
        }
        break;
    case 6:     //
        if (b == 8)     // connect to H
        {
            m[2][5] = '-';
        }
        break;
    case 7:     // G
        if (b == 10)     // connect to J
        {
            m[3][2] = '|';
        }
        if (b == 9)     // connect to I
        {
            m[4][3] = '-';
        }
        break;
    case 8:     // H
        if (b == 10)     // connect to J
        {
            m[2][3] = '-';
        }
        if (b == 9)     // connect to I
        {
            m[3][4] = '|';
        }
        break;

    }

}
```

```
void graph_edges(char m[7][7]){
   graph_connect(m,1,4);     // A-D
   graph_connect(m,1,10);    // A-j
   graph_connect(m,1,8);     // A-H
   graph_connect(m,1,6);     // A-F
   graph_connect(m,2,4);     // B-D
   graph_connect(m,2,5);     // B-E
   graph_connect(m,2,7);     // B-G
   graph_connect(m,3,5);     // C-E
   graph_connect(m,7,9);     // G-I
}

void graph_display(char m[7][7]){
   cout << endl;
   for (int i=0; i<7; i++)
   {
      cout << "  ";
      for (int j=0; j<7; j++)
        cout << m[i][j];
      cout << endl;
   }
}

int main()
{
   string array[60];
   ifstream MyReadFile("A2planets_TT8V_Group3.txt");
   Planet planet[size];
   string tempString;
   for (int i = 0; i < 11; i++){
      getline(MyReadFile, tempString);

      istringstream read(tempString);

      read >> planet[i].name;
      read >> planet[i].x;
      read >> planet[i].y;
      read >> planet[i].z;
      read >> planet[i].weight;
      read >> planet[i].profit;
   }


   int adjMatrix[size][size] = {{0, 0, 0, calDisPla(planet[0],planet[3]), 0,
calDisPla(planet[0],planet[5]), 0, calDisPla(planet[0],planet[7]), 0,
calDisPla(planet[0],planet[9])},
      {0, 0, 0, calDisPla(planet[1],planet[3]), calDisPla(planet[1],planet[4]), 0,
calDisPla(planet[1],planet[6]), 0, 0, 0},
      {0, 0, 0, 0, calDisPla(planet[2],planet[4]), calDisPla(planet[2],planet[5]), 0, 0,
calDisPla(planet[2],planet[8]), 0},
```

```
        {calDisPla(planet[3],planet[0]), calDisPla(planet[3],planet[1]), 0, 0, 0, 0, 0, 0, 0,
calDisPla(planet[3],planet[9])},
        {0, calDisPla(planet[4],planet[1]), calDisPla(planet[4],planet[2]), 0 , 0, 0,
calDisPla(planet[4],planet[6]), 0, calDisPla(planet[4],planet[8]), 0},
        {calDisPla(planet[5],planet[0]), 0, calDisPla(planet[5],planet[2]), 0, 0, 0, 0,
calDisPla(planet[1],planet[7]), 0, 0},
        {0, calDisPla(planet[6],planet[1]), 0, 0, calDisPla(planet[6],planet[4]), 0 , 0, 0,
calDisPla(planet[6],planet[8]), calDisPla(planet[6],planet[9])},
        {calDisPla(planet[7],planet[0]), 0, 0, 0, 0, calDisPla(planet[7],planet[5]), 0, 0,
calDisPla(planet[7],planet[8]), calDisPla(planet[7],planet[9])},
        {0, 0, calDisPla(planet[8],planet[2]), 0, calDisPla(planet[8],planet[4]), 0,
calDisPla(planet[8],planet[6]), calDisPla(planet[8],planet[7]), 0, 0},
        {calDisPla(planet[9],planet[0]), 0, 0, calDisPla(planet[9],planet[3]), 0, 0,
calDisPla(planet[9],planet[6]), calDisPla(planet[9],planet[7]), 0, 0}
    };

    dijkstra(adjMatrix, 0);

    cout << " " << endl;
    cout << " " << endl;
    cout << "Display graph: " << endl;

    char map[7][7];
    initmap(map);
    graph_planets(map);
    graph_edges(map);
    graph_display(map);

    return 0;
}
```

2.3 Program Outputs

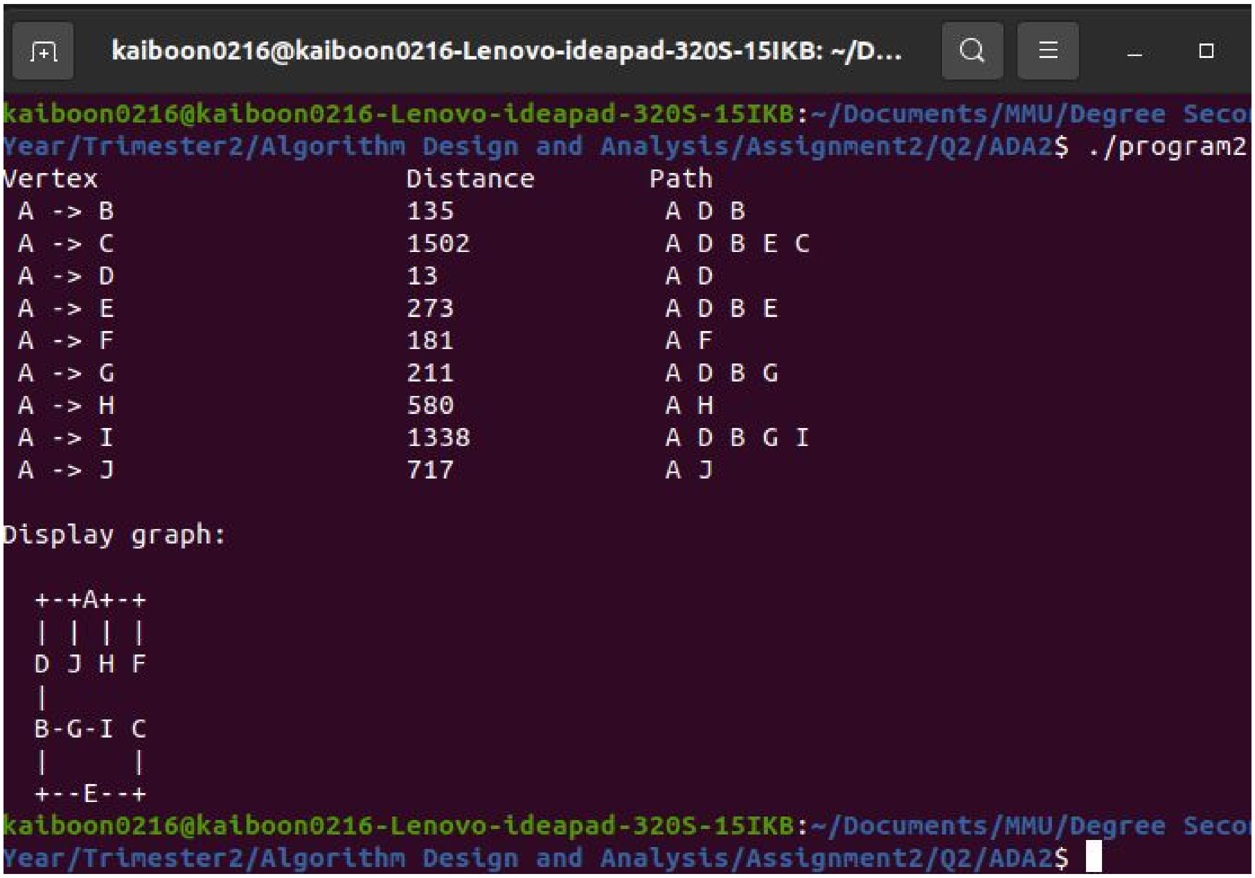

Figure 6 : Output of shortest path

3.0 Minimum Spanning Tree

3.1 Program 3

Algorithm 1 distance

```
procedure distance(int x, int y, int z,int x1, int y1, int z1)
 return sqrt(pow(x-x1,2)+pow(y-y1,2)+pow(z-z1,2))
```

Struct Graph

```
struct Graph
    int V, E
    vector<pair<int,pair> edges;
    Graph(int V, int E){
          this→V = V;
          this→E = E; }
   void addEdge(int u,intv,int w) {
         edges.push_back({w, {u, v}});  //u = node1 v=node2 w=distance
   int kruskalMST();
  void display(char m[7][7]);
  void connection(char m[7][7],int x, int y);
 void connect(char m[7][7], int a, int b);
 void planets(char m[7][7]);
 void initmap(char m[7][7]);
end struct
```

Struct DisjointSets

```
struct DisjointSets
    int parent, rnk;
    int n
    Disjoint(int n){
        this→n = n;
        parent = new int[n+1]
        rnk = new int[n+1]
      loop i<n
          rnk[i] = 0;
          parent[i] = i;
      end loop
    int find(int u)
      if u != parent
          find (parent[u])
     return parent[u]
   void merge()
     find(x),find(y)
     if(rank x >  rank y)
         parent of y = x;
     else
         parent of x  = y;
     end if
end struct
```

Algorithm Graph::kruskalMST()

```
procedure KruskalMST()
    int spanning tree weight = 0
    char map[7][7]
    initmap(map)
    planets(map);
    sort all the edges;
    DisjointSets ds(V);
    vector::iterator i;
      loop(i = start of edge; until i == end of edge; i++)
```

```
            u = second.first
            v = second.second
            cout << u << " - " << v << endl;
        end loop
    end iterator
    return mst_wt
end procedure
```

3.2 Source Code

```
// C++ program for Kruskal's algorithm to find Minimum
// Spanning Tree of a given connected, undirected and
// weighted graph
#include<bits/stdc++.h>
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <array>
#include <cmath>

using namespace std;

class Planet {
    public:
        int x;
        int y;
        int z;
        int weight;
        int profit;
        string name;
};


int distance (int x, int y, int z,int x1, int y1, int z1){
    return(sqrt(pow(x-x1,2)+pow(y-y1,2)+pow(z-z1,2)));
}

typedef pair<int, int> iPair;

struct Graph
{
            int V, E;
            vector< pair<int, iPair> > edges;
```

```
        Graph(int V, int E)
        {
                this->V = V;
                this->E = E;
        }

        void addEdge(int u, int v, int w)
        {
                edges.push_back({w, {u, v}});
        }

        int kruskalMST();
        void display(char m[7][7]);
        void connection(char m[7][7],int x, int y);
        void connect(char m[7][7], int a, int b);
        void planets(char m[7][7]);
        void initmap(char m[7][7]);
};

void Graph::display(char m[7][7])
{
   cout << endl;
   for (int i=0; i<7; i++)
   {
      cout << "  ";
      for (int j=0; j<7; j++)
         cout << m[i][j];
      cout << endl;
   }
}

void Graph::connection(char m[7][7], int x, int y)
{
   connect(m,x,y);     // A-D
}

void Graph::connect(char m[7][7], int a, int b)
{
   switch (a) {
   case 1:     // A
      if (b == 4)     // connect to D
      {
         m[0][0] = '+';
         m[0][1] = '-';
         m[0][2] = '-';
         m[1][0] = '|';
      }
      if (b == 6)     // connect to F
      {
         m[0][6] = '+';
```

```
        m[0][5] = '-';
        m[1][6] = '|';
    }
    if (b == 10)     // connect to J
    {
        m[0][2] = '+';
        m[1][2] = '|';
    }
    if (b == 8)      // connect to H
    {
        m[0][4] = '+';
        m[1][4] = '|';
    }
    break;
case 2:     // B
    if (b == 4)     // connect to D
    {
        m[3][0] = '|';
    }
    if (b == 5)     // connect to E
    {
        m[6][0] = '+';
        m[6][1] = '-';
        m[6][2] = '-';
        m[5][0] = '|';
    }
    if (b == 7)     // connect to G
    {
        m[4][1] = '-';
    }
    break;
case 3:     // C
    if (b == 6)     // connect to F
    {
        m[3][6] = '|';
    }
    if (b == 5)     // connect to E
    {
        m[6][6] = '+';
        m[6][5] = '-';
        m[6][4] = '-';
        m[5][6] = '|';
    }
    if (b == 9)     // connect to I
    {
        m[4][5] = '-';
    }
    break;
case 4:     // D
    if (b == 10)     // connect to J
```

```
      {
         m[2][1] = '-';
      }

      break;
   case 5:    // E
      if (b == 7)    // connect to G
      {
         m[6][2] = '+';
         m[5][2] = '|';
      }
      if (b == 9)    // connect to I
      {
         m[6][4] = '+';
         m[5][4] = '|';
      }
      break;
   case 6:    //
      if (b == 8)    // connect to H
      {
         m[2][5] = '-';
      }
      break;
   case 7:    // G
      if (b == 10)    // connect to J
      {
         m[3][2] = '|';
      }
      if (b == 9)    // connect to I
      {
         m[4][3] = '-';
      }
      break;
   case 8:    // H
      if (b == 10)    // connect to J
      {
         m[2][3] = '-';
      }
      if (b == 9)    // connect to I
      {
         m[3][4] = '|';
      }
      break;

   }

}

void Graph::initmap(char m[7][7])
{
```

```
    for (int i=0; i<7; i++)
        for (int j=0; j<7; j++)
            m[i][j] = ' ';
}

void Graph::planets(char m[7][7])
{
    m[0][3] = 'A';
    m[4][0] = 'B';
    m[4][6] = 'C';
    m[2][0] = 'D';
    m[6][3] = 'E';
    m[2][6] = 'F';
    m[4][2] = 'G';
    m[2][4] = 'H';
    m[4][4] = 'I';
    m[2][2] = 'J';
}

struct DisjointSets
{
        int *parent, *rnk;
        int n;

        DisjointSets(int n)
        {
                this->n = n;
                parent = new int[n+1];
                rnk = new int[n+1];

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

                        parent[i] = i;
                }
        }

        int find(int u)
        {
                if (u != parent[u])
                        parent[u] = find(parent[u]);
                return parent[u];
        }

        void merge(int x, int y)
        {
                x = find(x), y = find(y);

                if (rnk[x] > rnk[y])
```

```
				parent[y] = x;
			else
				parent[x] = y;

			if (rnk[x] == rnk[y])
				rnk[y]++;
		}
};

int Graph::kruskalMST()
{
		int mst_wt = 0;
		char map[7][7];
    initmap(map);
    planets(map);

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


		DisjointSets ds(V);

		vector< pair<int, iPair> >::iterator it;
		for (it=edges.begin(); it!=edges.end(); it++)
		{
			int u = it->second.first;
			int v = it->second.second;
			char u1;
			char v1;

			int set_u = ds.find(u);
			int set_v = ds.find(v);


			if (set_u != set_v)
			{

				if(u == 0)
					u1 = 'A';
				if(u == 1)
					u1 = 'B';
				if(u == 2)
					u1 = 'C';
				if(u == 3)
					u1 = 'D';
				if(u == 4)
					u1 = 'E';
				if(u == 5)
					u1 = 'F';
				if(u == 6)
					u1 = 'G';
```

```
			if(u == 7)
				u1 = 'H';
			if(u == 8)
				u1 = 'I';
			if(u == 9)
				u1 = 'J';

			if(v == 0)
				v1 = 'A';
			if( v == 1)
				v1 = 'B';
			if( v == 2)
				v1 = 'C';
			if( v == 3)
				v1 = 'D';
			if( v == 4)
				v1 = 'E';
			if( v == 5)
				v1 = 'F';
			if( v == 6)
				v1 = 'G';
			if( v == 7)
				v1 = 'H';
			if( v == 8)
				v1 = 'I';
			if( v == 9)
				v1 = 'J';
			cout << u1 << " - " << v1 << endl;
			connection(map,v+1,u+1);
			connection(map,u+1,v+1);
			mst_wt += it->first;

			ds.merge(set_u, set_v);
		}
	}

	display(map);

	return mst_wt;
}




int main()
{

	int V = 10, E = 18;
	Graph g(V, E);
```

```
string array[60];
ifstream MyReadFile("A2planets_TT8V_Group3.txt");
Planet planet[10];
string tempString;
for (int i = 0; i < 11; i++)
{
   getline(MyReadFile, tempString);

   istringstream pp(tempString);

   pp >> planet[i].name;
   pp >> planet[i].x;
   pp >> planet[i].y;
   pp >> planet[i].z;
   pp >> planet[i].weight;
   pp >> planet[i].profit;
}

      int AD = distance(planet[0].x,planet[0].y,planet[0].z,planet[3].x,planet[3].y,planet[3].z);
int AJ = distance(planet[0].x,planet[0].y,planet[0].z,planet[9].x,planet[9].y,planet[9].z);
int AH = distance(planet[0].x,planet[0].y,planet[0].z,planet[7].x,planet[7].y,planet[7].z);
int AF = distance(planet[0].x,planet[0].y,planet[0].z,planet[5].x,planet[5].y,planet[5].z);
int DB = distance(planet[3].x,planet[3].y,planet[3].z,planet[1].x,planet[1].y,planet[1].z);
int DJ = distance(planet[3].x,planet[3].y,planet[3].z,planet[9].x,planet[9].y,planet[9].z);
int JG = distance(planet[9].x,planet[9].y,planet[9].z,planet[6].x,planet[6].y,planet[6].z);
int JH = distance(planet[9].x,planet[9].y,planet[9].z,planet[7].x,planet[7].y,planet[7].z);
int HI = distance(planet[7].x,planet[7].y,planet[7].z,planet[8].x,planet[8].y,planet[8].z);
int HF = distance(planet[7].x,planet[7].y,planet[7].z,planet[5].x,planet[5].y,planet[5].z);
int BE = distance(planet[1].x,planet[1].y,planet[1].z,planet[4].x,planet[4].y,planet[4].z);
int BG = distance(planet[1].x,planet[1].y,planet[1].z,planet[6].x,planet[6].y,planet[6].z);
int GE = distance(planet[6].x,planet[6].y,planet[6].z,planet[4].x,planet[4].y,planet[4].z);
int GI = distance(planet[6].x,planet[6].y,planet[6].z,planet[8].x,planet[8].y,planet[8].z);
int IE = distance(planet[8].x,planet[8].y,planet[8].z,planet[4].x,planet[4].y,planet[4].z);
int IC = distance(planet[8].x,planet[8].y,planet[8].z,planet[2].x,planet[2].y,planet[2].z);
int CE = distance(planet[2].x,planet[2].y,planet[2].z,planet[4].x,planet[4].y,planet[4].z);
int CF = distance(planet[2].x,planet[2].y,planet[2].z,planet[5].x,planet[5].y,planet[5].z);

      // making above shown graph
      g.addEdge(0, 3, AD);
      g.addEdge(0, 9,AJ);
      g.addEdge(0, 7,AH);
      g.addEdge(0, 5,AF);
      g.addEdge(3, 1,DB);
      g.addEdge(3, 9,DJ);
      g.addEdge(9, 6,JG);
      g.addEdge(9, 7,JH);
      g.addEdge(7, 8,HI);
      g.addEdge(7, 5,HF);
      g.addEdge(1, 4,BE);
      g.addEdge(1, 6,BG);
```

```
        g.addEdge(6, 4,GE);
        g.addEdge(6, 8,GI);
        g.addEdge(8, 4,IE);
        g.addEdge(8, 2,IC);
        g.addEdge(2, 4,CE);
        g.addEdge(2, 5,CF);





        cout << "Edges of MST are \n";
        int mst_wt = g.kruskalMST();
        cout << "\nWeight of MST is " << mst_wt;

        return 0;
}
```

3.3 Program Outputs

```
Edges of MST are
A - D
G - E
B - G
D - B
A - F
I - C
J - H
A - H
H - I

  +--A+-+
  |   | |
  D J-H F
  |   |
  B-G I-C
    |
    +E

Weight of MST is 2764
```

Figure 7: Output of Minimum Spanning Treee

4.0 Conclusion

It was a wonderful learning experience for the team while working on this assignment. Through the effort of a few weeks, the team has successfully completed and created the Adjacency Matrix and List of planets using merge-sort, graph of shortest paths using Dijkstra's Algorithm and minimum spanning tree using Kruskal's Algorithm.

In the process of writing and understanding the algorithms, the team has faced many challenges. Challenges such as stackoverflow problems and program crash issues. With extensive research and with the help of lecture slides and lab exercises, the team managed to fix the issues and get the desired output of each question.

Last but not least, the team would like to thank Dr Yeoh for the guidance and consultation. Without the guidance from Dr Yeoh, understanding the algorithms would have opposed a bigger challenge.