

**Data Structuers and algorithms (CS09203)**

**Lab Report**

Name: Wajih-ul-hassan

Registration #: SEU-F16-151

Lab Report #: 11

Dated: 27-06-2018

Submitted To: Mr. Usman Ahmed

The University of Lahore, Islamabad Campus

Department of Computer Science & Information Technology

**Experiment # 11**

**Kruskal’s algorithml**

**Objective**

The objective of this session is to show the representation of trees using C++.

**Software Tool**

1. Code Blocks with GCC compiler.

# Theory

Kruskal’s algorithm is a minimum-spanning-tree algorithm which finds an edge of the least possible weight that connects any two trees in the forest.[1] It is a greedy algorithm in graph theory as it finds a minimum spanning tree for a connected weighted graph adding increasing cost arcs at each step.[1] This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. If the graph is not connected, then it finds a minimum spanning forest (a minimum spanning tree for each connected component).

# Task

**2.1 Task 1**

Impement Kruskal’s algorithm.

## Procedure: Task 1

#include*<*bits /stdc++.h*>* using namespace std ;

// Creating shortcut for an integer pair typedef pair*<*int , int*>* iPair ;

// Structure to represent a graph struct Graph

{

int V, E; vector*<* pair*<*int , iPair*>>* edges ;

// Constructor

Graph( int V, int E)

{

this−*>*V = V; this−*>*E = E;

}

// Utility function to add an edge void addEdge( int u, int v , int w)

{

edges . push back ({w, {u, v}});

}

// Function to find MST using Kruskal ’ s

// MST algorithm int kruskalMST ();

};

// To represent Disjoint Sets struct DisjointSets

{

int ∗parent , ∗rnk ; int n;

// Constructor .

DisjointSets ( int n) {

// Allocate memory this−*>*n = n; parent = new int [n+1]; rnk = new int [n+1];

// Initially , all vertices are in // different sets and have rank 0. for ( int i = 0; i *<*= n; i++)

{

rnk [ i ] = 0;

//every element is parent of i t s e l f parent [ i ] = i ;

}

}

// Find the parent of a node ’u’ // Path Compression int find ( int u)

{

/∗ Make the parent of the nodes in the path from u−−*>* parent [u] point to parent [u] ∗/

i f (u != parent [u ]) parent [u] = find ( parent [u ] ) ;

return parent [u ] ;

}

// Union by rank void merge( int x , int y)

{

x = find (x) , y = find (y );

/∗ Make tree with smaller height a subtree of the other tree ∗/

i f (rnk [ x ] *>* rnk [ y ]) parent [ y ] = x ;

else // If rnk [ x ] *<*= rnk [ y ] parent [ x ] = y ;

i f (rnk [ x ] == rnk [ y ]) rnk [ y]++;

}

};

/∗ Functions returns weight of the MST∗/

int Graph : : kruskalMST ()

{

int mst wt = 0; // I n i t i a l i z e result

// Sort edges in increasing order on basis of cost sort ( edges . begin () , edges . end ());

// Create disjoint sets

DisjointSets ds(V);

// Iterate through all sorted edges vector*<* pair*<*int , iPair*> >*:: iterator it ; for ( it=edges . begin (); it !=edges . end (); it++)

{

int u = it−*>*second . f i r s t ; int v = it−*>*second . second ;

int setu = ds . find (u ); int setv = ds . find (v );

// Check i f the selected edge is creating

// a cycle or not ( Cycle is created i f u // and v belong to same set ) i f ( set u != set v )

{

// Current edge will be in the MST

// so print it cout *<<* u *<<* ” − ” *<<* v *<<* endl ;

// Update MST weight mst wt += it−*>*f i r s t ;

// Merge two sets ds . merge( set u , set v );

}

}

return mst wt ;

}

// Driver program to test above functions int main()

{

/∗ Let us create above shown weighted and unidrected graph ∗/ int V = 9 , E = 14;

Graph g(V, E);

// making above shown graph g . addEdge(0 , 1 , 4);

g . addEdge(0 , 7 , 8);

g . addEdge(1 , 2 , 8);

g . addEdge(1 , 8 ,5 );

|  |  |
| --- | --- |
| g . addEdge(1 , 6 , | 10); |
| g . addEdge(2 , 6 , | 4); |
| g . addEdge(2 , 3 , | 4); |
| g . addEdge(2 , 8 , | 4); |
| g . addEdge(2 , 5 , | 4); |
| g . addEdge(2 , 1 , | 8); |
| g . addEdge(3 , 6 , | 3); |
| g . addEdge(3 , 2 , | 4); |
| g . addEdge(3 , 4 , | 3); |
| g . addEdge(4 , 3 , | 3); |
| g . addEdge(4 , 6 , | 6); |
| g . addEdge(4 , 5 , | 1); |
| g . addEdge(4 , 7 , | 2); |
| g . addEdge(5 , 2 , | 4); |
| g . addEdge(5 , 7 , | 3); |
| g . addEdge(5 , 4 , | 1); |
| g . addEdge(6 , 1 , | 10); |
| g . addEdge(6 , 2 , | 4); |
| g . addEdge(6 , 3 , | 3); |
| g . addEdge(6 , 4 , | 6); |
| g . addEdge(7 , 4 , | 2); |
| g . addEdge(7 , 5 , | 3); |

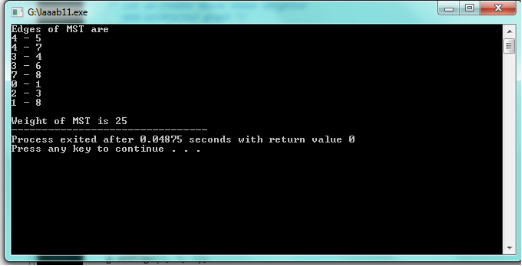


Figure 1: output

g . addEdge(7 , 8 , 3);

g . addEdge(8 , 1 , 5);

g . addEdge(8 , 2 , 4);

g . addEdge(8 , 5 , 3); cout *<<* ”Edges of MST are \n”; int mst wt = g . kruskalMST (); cout *<<* ”\nWeight of MST is ” *<<* mst wt ;

return 0;

}