데이터 구조 과제

학번: 12170584

이름: 이진호

**Kruskal algorithm**

**문제정의:**

최소비용 신장트리(minimum spanning tree)를 구하는 알고리즘의 하나인 크루스칼(kruskal)알고리즘이다. stdin에서 input graph 데이타를 입력받으며, 프로그램 수행 후 적절한 답을 stdout에 출력.

**크루스칼(kruskal) 알고리즘:**

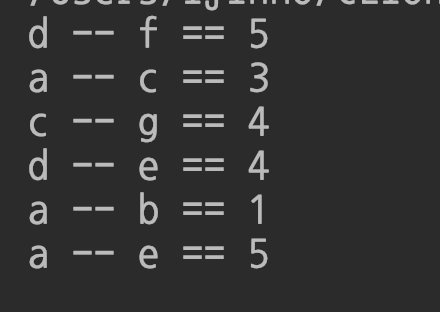
최소 비용으로 모든 vertices -1 = edge’s number로 이루어진 그래프를 말한다.

* 가장 낮은 가중치를 가진 edge를 순서대로 선택하여 저장한다.
* 만약 cycle을 이루게 된다면 배제시킨다.
* Vertices – 1 의 edge가 저장된다면 종료한다.

**코드 설명:**

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
enum {*a*,*b*,*c*,*d*,*e*,*f*,*g*};  
  
struct Edge  
{  
 int src, dest, weight;  
};  
  
struct Graph  
{  
 int V, E;  
  
 struct Edge\* edge;  
};  
  
//create graph  
struct Graph\* createGraph(int V, int E)  
{  
 struct Graph\* graph = new Graph;  
 graph->V = V;  
 graph->E = E;  
  
 graph->edge = new Edge[E];  
  
 return graph;  
}  
  
//create subset  
struct subset  
{  
 int parent;  
 int rank;  
};  
  
//find function  
int find(struct subset subsets[], int i)  
{  
 // find root  
 if (subsets[i].parent != i)  
 subsets[i].parent = find(subsets, subsets[i].parent);  
  
 //return root  
 return subsets[i].parent;  
}  
  
//union function  
void Union(struct subset subsets[], int x, int y)  
{  
 int xroot = find(subsets, x);  
 int yroot = find(subsets, y);  
  
 //if x root is lower than yroot -> union yroot to x parent  
 if (subsets[xroot].rank < subsets[yroot].rank)  
 subsets[xroot].parent = yroot;  
 //if y root is lower than xroot -> union xroot to y parent  
 else if (subsets[xroot].rank > subsets[yroot].rank)  
 subsets[yroot].parent = xroot;  
  
 //if rank is same  
 else  
 {  
 subsets[yroot].parent = xroot;  
 subsets[xroot].rank++;  
 }  
}  
  
//compare function  
int comp(const void\* a, const void\* b)  
{  
 struct Edge\* a1 = (struct Edge\*)a;  
 struct Edge\* b1 = (struct Edge\*)b;  
  
 return a1->weight > b1->weight;  
}  
  
//kruskal function  
void Kruskal(struct Graph\* graph)  
{  
 int V = graph->V;  
 struct Edge result[V];  
 int e = 0;  
 int i = 0;  
  
 //sorting weight  
 qsort(graph->edge, graph->E, sizeof(graph->edge[0]), comp);  
  
 struct subset \*subsets =  
 (struct subset\*) malloc( V \* sizeof(struct subset) );  
  
 // Create V subsets with single elements   
 for (int v = 0; v < V; ++v)  
 {  
 subsets[v].parent = v;  
 subsets[v].rank = 0;  
 }  
  
 //Do progress until V-1  
 while (e < V - 1)  
 {  
 //choose smallest edge  
 struct Edge next\_edge = graph->edge[i++];  
  
 int x = find(subsets, next\_edge.src);  
 int y = find(subsets, next\_edge.dest);  
  
 //if edge don't make cycle  
 if (x != y)  
 {  
 result[e++] = next\_edge;  
 Union(subsets, x, y);  
 }  
 // Else discard the next\_edge   
 }  
  
 //print the result  
 for (i = 0; i < e; ++i)  
 printf("%c -- %c == %d\n", result[i].src+97, result[i].dest+97,  
 result[i].weight);  
 return;  
}  
  
//processing function  
void processing(int V,int E){  
  
 struct Graph\* graph = createGraph(V, E);  
  
 graph->edge[0].src = *a*;  
 graph->edge[0].dest = *b*;  
 graph->edge[0].weight = 1;  
  
 graph->edge[1].src = *a*;  
 graph->edge[1].dest = *c*;  
 graph->edge[1].weight = 3;  
  
 graph->edge[2].src = *a*;  
 graph->edge[2].dest = *d*;  
 graph->edge[2].weight = 6;  
  
 graph->edge[3].src = *a*;  
 graph->edge[3].dest = *e*;  
 graph->edge[3].weight = 5;  
  
 graph->edge[4].src = *a*;  
 graph->edge[4].dest = *g*;  
 graph->edge[4].weight = 7;  
  
 graph->edge[5].src = *b*;  
 graph->edge[5].dest = *e*;  
 graph->edge[5].weight = 2;  
  
 graph->edge[6].src = *b*;  
 graph->edge[6].dest = *g*;  
 graph->edge[6].weight = 5;  
  
 graph->edge[7].src = *c*;  
 graph->edge[7].dest = *d*;  
 graph->edge[7].weight = 2;  
  
 graph->edge[8].src = *c*;  
 graph->edge[8].dest = *f*;  
 graph->edge[8].weight = 6;  
  
 graph->edge[9].src = *c*;  
 graph->edge[9].dest = *g*;  
 graph->edge[9].weight = 4;  
  
 graph->edge[10].src = *d*;  
 graph->edge[10].dest = *e*;  
 graph->edge[10].weight = 4;  
  
 graph->edge[11].src = *d*;  
 graph->edge[11].dest = *f*;  
 graph->edge[11].weight = 5;  
  
 Kruskal(graph);  
}  
  
//main function  
int main()  
{  
 int V = 7; //Vertices  
 int E = 12; //edges  
  
 processing(V,E);  
  
 return 0;  
}

**출력 결과**

****

**결론 –** 크루스칼 알고리즘을 통해 최단 경로를 구하려고 하였으나 입력에 따른 출력을 구현 하여야 하는데, 입력을 받은 후 출력이 제대로 되지않는 결과 때문에 원하는 결과를 이뤄낼 수 없었다. 그래서 각각 edge를 추가하고 weight를 넣어서 출력을 했을때 목표하는 결과를 낼 수 있었다.