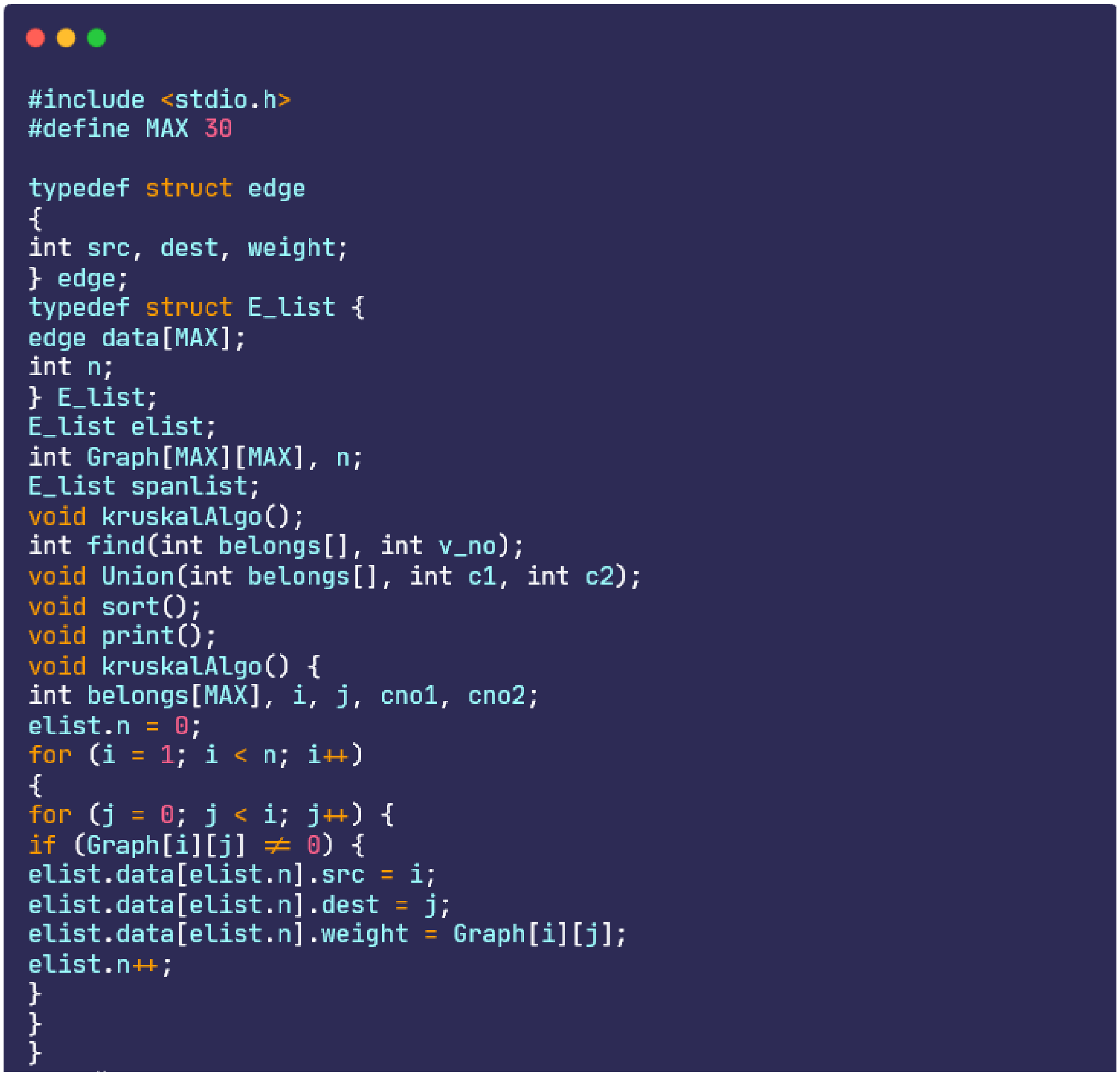
Danyl Fernandes
2020012004 (72)
23-04-2021

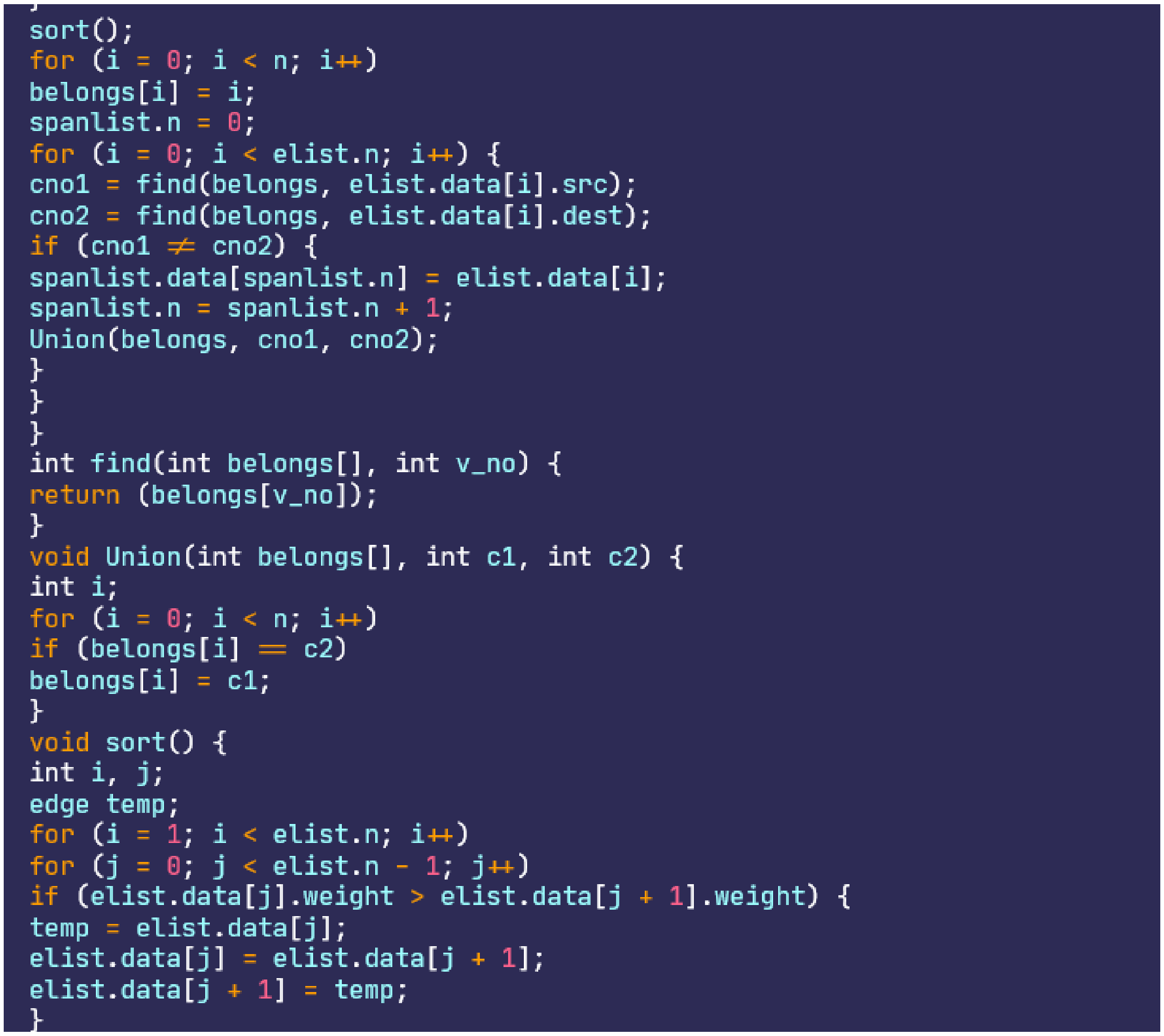
AOA Experiment 4

Aim:

To implement & analyze Minimum cost spanning tree - Kruskal's Algorithm:



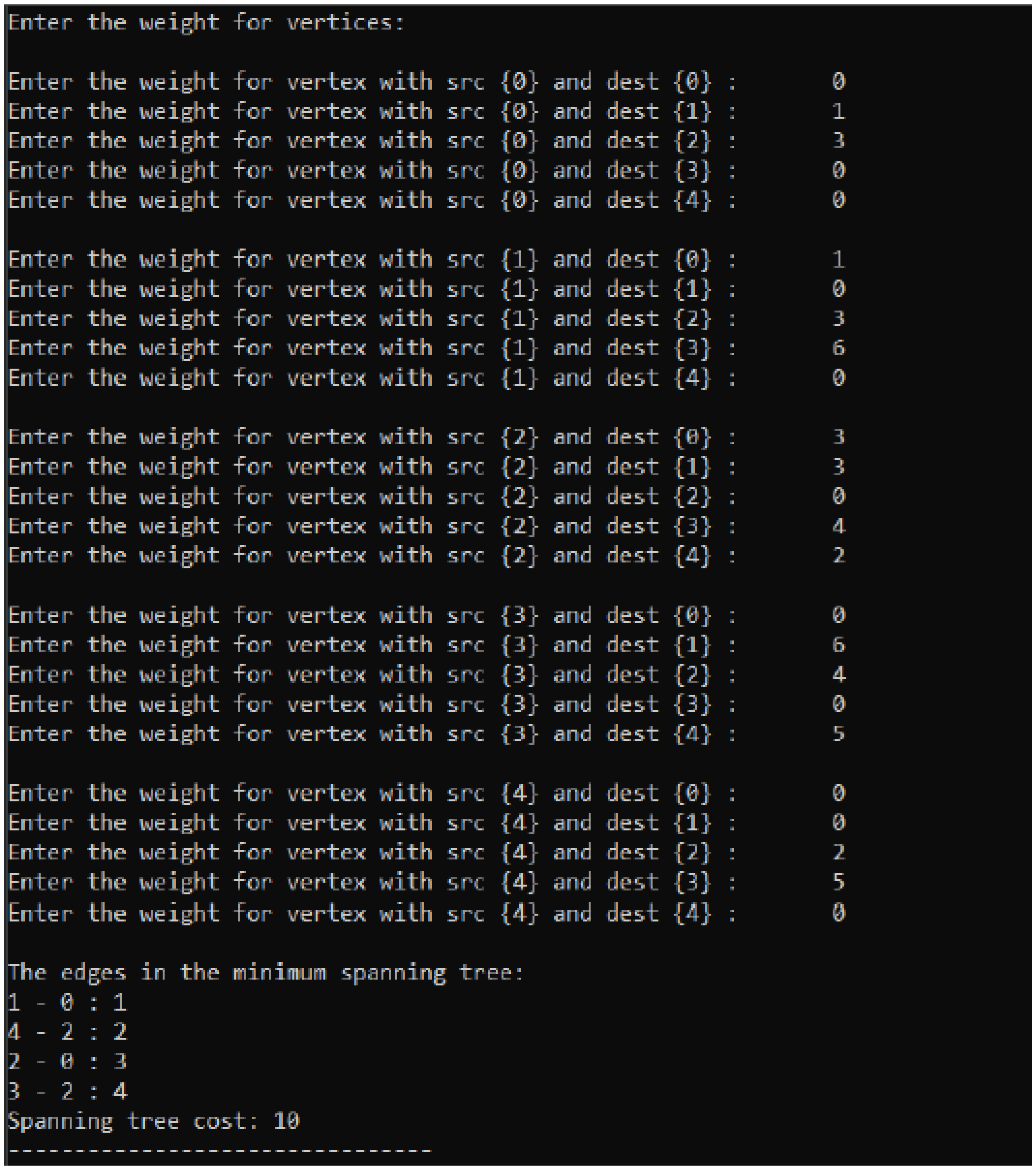
```c
#include <stdio.h>
#define MAX 30

typedef struct edge
{
int src, dest, weight;
} edge;
typedef struct E_list {
edge data[MAX];
int n;
} E_list;
E_list elist;
int Graph[MAX][MAX], n;
E_list spanlist;
void kruskalAlgo();
int find(int belongs[], int v_no);
void Union(int belongs[], int c1, int c2);
void sort();
void print();
void kruskalAlgo() {
int belongs[MAX], i, j, cno1, cno2;
elist.n = 0;
for (i = 1; i < n; i++)
{
for (j = 0; j < i; j++) {
if (Graph[i][j] != 0) {
elist.data[elist.n].src = i;
elist.data[elist.n].dest = j;
elist.data[elist.n].weight = Graph[i][j];
elist.n++;
}
}
}
```



```
}
sort();
for (i = 0; i < n; i++)
belongs[i] = i;
spanlist.n = 0;
for (i = 0; i < elist.n; i++) {
cno1 = find(belongs, elist.data[i].src);
cno2 = find(belongs, elist.data[i].dest);
if (cno1 != cno2) {
spanlist.data[spanlist.n] = elist.data[i];
spanlist.n = spanlist.n + 1;
Union(belongs, cno1, cno2);
}
}
}
int find(int belongs[], int v_no) {
return (belongs[v_no]);
}
void Union(int belongs[], int c1, int c2) {
int i;
for (i = 0; i < n; i++)
if (belongs[i] == c2)
belongs[i] = c1;
}
void sort() {
int i, j;
edge temp;
for (i = 1; i < elist.n; i++)
for (j = 0; j < elist.n - 1; j++)
if (elist.data[j].weight > elist.data[j + 1].weight) {
temp = elist.data[j];
elist.data[j] = elist.data[j + 1];
elist.data[j + 1] = temp;
}
```

```
}
void display() {
int i, cost = 0;
printf("The edges in the minimum spanning tree: ");
for (i = 0; i < spanlist.n; i++) {
printf("\n%d - %d : %d", spanlist.data[i].src, spanlist.data[i].dest,
spanlist
cost = cost + spanlist.data[i].weight;
}
printf("\nSpanning tree cost: %d", cost);
}
int main() {
int i, j, total_cost;
printf("Enter the total number of vertices: ");
scanf("%d",&n);
printf("Enter the weight for vertices: \n");
printf("\n");
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
printf("Enter the weight for vertex with src {%d} and dest {%d} : ",
scanf("%d",&Graph[i][j]);
}
printf("\n");
}
kruskalAlgo();
display();
}
```

Output:



```
Enter the weight for vertices:

Enter the weight for vertex with src {0} and dest {0} :       0
Enter the weight for vertex with src {0} and dest {1} :       1
Enter the weight for vertex with src {0} and dest {2} :       3
Enter the weight for vertex with src {0} and dest {3} :       0
Enter the weight for vertex with src {0} and dest {4} :       0

Enter the weight for vertex with src {1} and dest {0} :       1
Enter the weight for vertex with src {1} and dest {1} :       0
Enter the weight for vertex with src {1} and dest {2} :       3
Enter the weight for vertex with src {1} and dest {3} :       6
Enter the weight for vertex with src {1} and dest {4} :       0

Enter the weight for vertex with src {2} and dest {0} :       3
Enter the weight for vertex with src {2} and dest {1} :       3
Enter the weight for vertex with src {2} and dest {2} :       0
Enter the weight for vertex with src {2} and dest {3} :       4
Enter the weight for vertex with src {2} and dest {4} :       2

Enter the weight for vertex with src {3} and dest {0} :       0
Enter the weight for vertex with src {3} and dest {1} :       6
Enter the weight for vertex with src {3} and dest {2} :       4
Enter the weight for vertex with src {3} and dest {3} :       0
Enter the weight for vertex with src {3} and dest {4} :       5

Enter the weight for vertex with src {4} and dest {0} :       0
Enter the weight for vertex with src {4} and dest {1} :       0
Enter the weight for vertex with src {4} and dest {2} :       2
Enter the weight for vertex with src {4} and dest {3} :       5
Enter the weight for vertex with src {4} and dest {4} :       0

The edges in the minimum spanning tree:
1 - 0 : 1
4 - 2 : 2
2 - 0 : 3
3 - 2 : 4
Spanning tree cost: 10
--------------------------------
```

Daryl Fernandes
2020012004 (72)

Exp 04 :

Theory :

- Kruskal's algorithm develops the Minimum spanning tree of a given graph by constructing a forest of minimum weighted edges that can be completed into a spanning tree if it is acyclic

Algorithm :

```
MST Kruskal (G, W)
  A ≠ O
  for each vertex v ∈ G.V
        Make-set(V)
  // sort the edges G.E into
     //    non decreasing order by weig
   for each edge (u,v) ∈ G.E,
 // taken in non-decreasing order
    if find-set (V) ≠ findset (v)
          A = A ∪ {(U,V)}
                UNION (U,V)
   return A.
```

Analysis :

- G = (V,E) be a given graph where V is a set of vertices and E is a set of edges. Kruskal's algorithm uses a priority queue (a min-heap) data structure where priorities are assigned to weights of all the edges of a graph.

Danyl Fernandes
2020012004 (72)

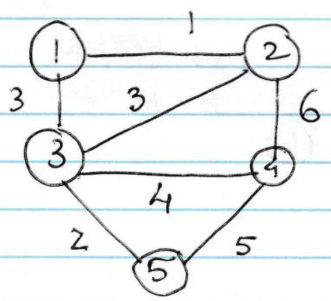
Using this the edges are arranged in ascending order of their weights in time O(|E| log |E|)

- In each step, Kruskal's algorithm connects two partial subtrees in a forest by including the minimum weighted edge without forming a cycle. To do this Kruskals algorithm performs |E| number of delete operations on a priority queue.

- Since a single delete operation a priority queue takes O(log |E|), the total time of deleting |E| edges will take O(|E| log |E|) time.

- The total time complexity of Kruskal's algorithm including construction of a priority queue of edges and deletion of edges from it is given as O(|E| log |E|) + O(|E| log |E|) = O(|E| log |E|)

Example:

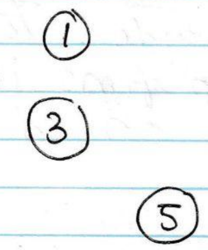


1) We arrange all the edges of a given

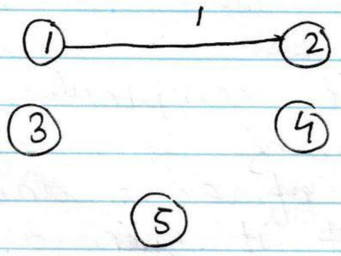
Danyl Fernandes
2020012004 (72)

graph is acendeing order of their weights as below <1,2>, (3,5), <1,3>, <2,3>, <3,4>, <4,5>, <2,4>

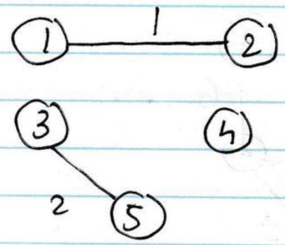
2) Construct a forest of partial subtrees each of them, containing a single node



3) Add the first minimum weighted edge <1,2> to connect nodes 1 & 2



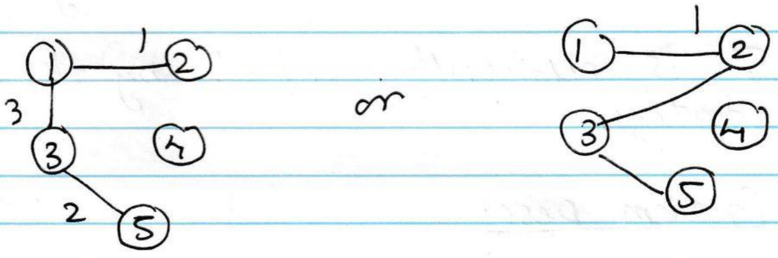
4) Include the next minimum weighted edge <3,5> to connect nodes 3 & 5 in a forest



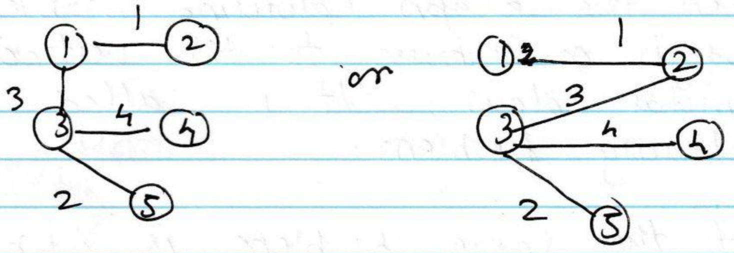
5) The next minimum weighted edges are

Danyl Fernandes
2020012004 (72)

<1,3> & <~~2~~3,3>. By adding either to connect two partial subtrees
i) a subtree containing nodes 1 & 2
ii) a subtree with nodes 3, 5 we get following two varities of a forest.
We cannot as they form a cycle in a forest



6) Add the next minimum edge to a forest to connect a subtree with nodes 1,2,3,5, to node 4.



7) We cannot add the remaining edges (4,5) & (2,4) as they from a cycle in a forest. So the tree obtained in the above step(6) is the final MST of given graph with minum cost = 10 units

= <u>Conclusion</u>: We successfuly implemented Kruskal's algorithm & analyzed it.