MINIMUM SPANNING TREE

1-the implementation of graph:-

1-Variables:

```
Node NodeList[];
int adjMat[][];
int numofnodes;
Stack dfs;
```

In this part , we defined these variables:

Node List : we defined this array to help us to storage all nodes in graph And make it easy to make operations on it(we will explain these operations soon)

adj matrix : help us to know if there are relations between nodes and each other (we put 1 if there is intersection between them else we put 0 or no thing)

num of nodes : we will need it later

dfs: we defined “dfs” stack to help in mst (we will explain that soon)

MINIMUM SPANNING TREE

2-“Graph” class and its constructor:

```
public Graph(int length) {
  NodeList = new Node[length];
  adjMat = new int[length][length];
  numofnodes = 0;
  dfs = new Stack(length);
}
```

-“Graph” constructor: we made this method to initialize length of all arrays that we need to storage(NodeList,adjMat) and make operations(dfs)

-Note:”dfs” isn,t array , it is stack implemented by array

MINIMUM SPANNING TREE

3-“Node” class and “Node” constructor

```
    private class Node  {
    public char Data;
    public boolean Visited;

        Node(char Data) {
        this.Data =Data;
        Visited = false;
    }
}
```

-in this part, we made this class and made it private to be non- adjustable

By user . ”Node” class contains “data” (name of Node) and “visited”(Boolean true or false)

-we made constructor to initialize the variables of this class (Node)

And we used it by making array of nodes to help us in this class

And add nodes to graph

MINIMUM SPANNING TREE

4-add Node in Graph

```
int addedNodes=0;
public void addNode(char label) {
    addedNodes=addedNodes+1;
   NodeList[numofnodes++] = new Node(label);
}
```

-in “Graph” class, we make this method as setter method and we made every node has a label(char) to help us to know the way of mst when we print it

-we storage this char in Node List as data of node and we made them unlimited

- we will need “addedNodes” variable in the step of mst and it increases by one when we add new node

---

5-get Node in graph

```
public void GetNode(int v) {
   System.out.print(NodeList[v].Data);
}
```

-this method is simple and it helps us to print data of node by index of this node in NodeList (array)

MINIMUM SPANNING TREE

6-intersection in adjMat

```
public int getAdjUnvisitedNode(int v) {
  for(int i = 0; i < numofnodes; i++) {
    if(adjMat[v][i] == 1 && NodeList[i].Visited == false) {
      return i;
    }
  }
  return -1;
}
```

-This method help us to know if there is a relation between v and all of I and if this node visited or in adj matrix by using for loop and if statement and when if true the method will return the index(I , beginning with 0)

IMPORTANT NOTE:-

“GetNode” method and “getAdjUnvisitedNode” method are not for user(we can make it private) but it help us in “mstanddfs” and make code very simple and encapsulated

MINIMUM SPANNING TREE

7- Add Edge

```
int addedEdges=0;
public void addEdge(int start, int end) {
    addedEdges=addedEdges+1;
   adjMat[start-1][end-1] = 1;
   adjMat[end-1][start-1] = 1;
}
```

-“addEdge” method is very important because it connects between nodes and make graph . this graph will be a function or relation we can benefit from it.

-we made two parameters for adjMat 2d array

-in the final 2 lines , we made the edge between 2 nodes by add 1 to this array because when we search for them we would know that these 2 nodes has a relation between them

-why do we put the parameter by -1?

because the user didn’t know that array is beginning by 0 and when user put 1,1 we put it in 0,0

-why do we put (end , start) and (start , end)?

Because it is undirected

(1,2)=(2,1)

MINIMUM SPANNING TREE

2-MST and DFS:-

```
public void mstanddfs() {
   if(addedEdges==addedNodes-1)
       System.out.println("0");
   else if(addedEdges>=addedNodes)
       System.out.println(addedEdges-addedNodes+1);
   else
       System.out.println("number of edges greater than number of nodes,try again------");
   NodeList[0].Visited = true;
   dfs.push(0);
   while(!dfs.isempty()) {
     int current = dfs.peek();
     int v = getAdjUnvisitedNode(current);
     if(v == -1) {
       dfs.pop();
     } else {

       NodeList[v].Visited = true;
       GetNode(current);
       GetNode(v);
       System.out.print(" ");
       dfs.push(v);

     }
   }
}
```

-in the first part, we wrote the code that give us number of minimum edges of graph (minimum spanning tree) depending on this equation:

Number of edges=number of nodes-1

-And every case was explained in the code.

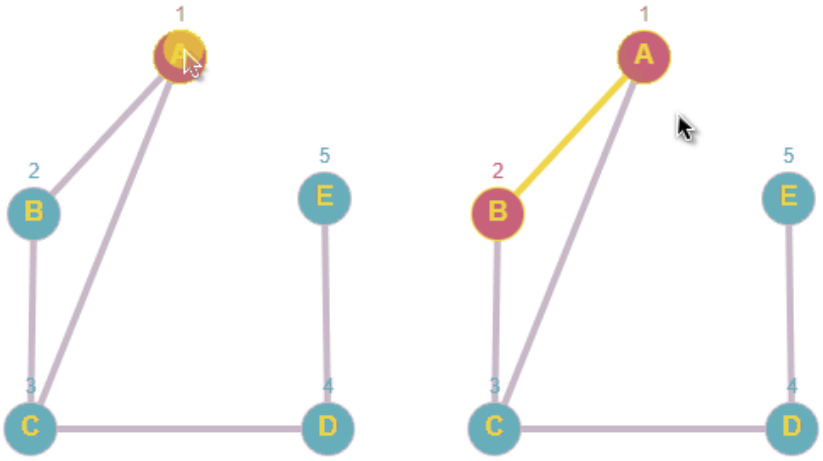
-We know that by subtracting num of edges in MST from edges in first graph and this give us number of edges that we don't need

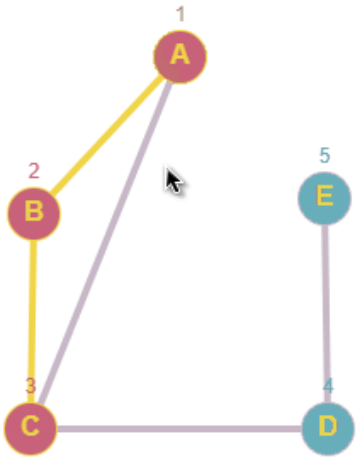
---

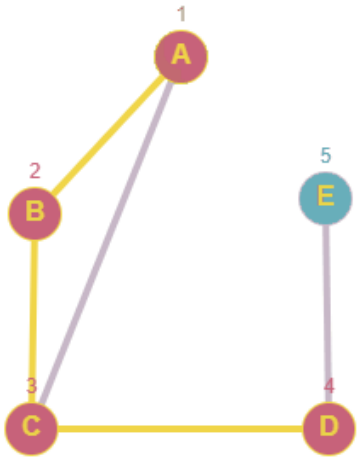
-in the final part, we wrote "DFS" Algorithm using "Stack" class (LIFO) I implemented (not all but method I needed) this class you can see in the project

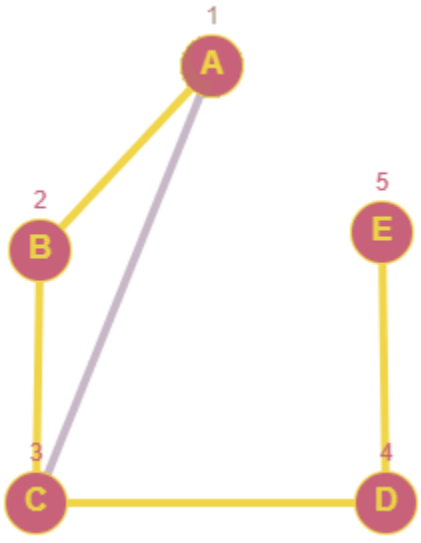
- and I print this graph after applying MST

MINIMUM SPANNING TREE









NOTE :- this diagram directed by me explaining Depth First Search (DST).

MINIMUM SPANNING TREE

-adjacency matrix

0	(1)	(2)	(3)	(4)	(5)
(1)	0	1(A)	1(E)	0	0
(2)	0	0	1(B)	0	0
(3)	0	0	0	1(C)	0
(4)	0	0	0	0	1(D)
(5)	0	0	0	0	0

Very Important Note:-

We apply this from example in project.

Input format:

First line will contain two integers n and m where n denotes the number of nodes and m denotes the number of edges in the graph respectively.

The next m lines contain two integers u_i and v_i denoting an undirected edge between the vertices u_i and v_i.

Output format:

Print the required total minimum cost of removing edges.

Sample input

5 5

1 2

2 3

3 4

4 5

1 3

Sample output

1

MINIMUM SPANNING TREE

-Stack and DFS

Order of visit : A B C D E

A

A B

A B C

A B C D

A B C D E

B : maximum branching factor of the search tree

M :maximum depth

Time Complexity of DFS:-

O(B^M)

Space Complexity of DFS:-

O(BM)

Simple run #1:-

-g.addNode(a);

- g.addNode(b);

-g.addNode(c);

-g.addEdge(1,2);

-g.addEdge(2,3)

- The number of ways that we do not need is 0

-AB||BC

MINIMUM SPANNING TREE

Simple run #2:-

-g.addNode(a);

- g.addNode(b);

-g.addNode(c);

-g.addNode(d);

-g.addEdge(1,2);

-g.addEdge(2,3);

-g.addEdge(3,4);

-g.addEdge(4,1);

-g.addEdge(1,3);

-g.addEdge(2,4);

-The number of ways that we dont need is 3

-ab||bc||cd

MINIMUM SPANNING TREE

-The main method:-

```
Scanner s=new Scanner(System.in);
System.out.println("Enter maximum number of Edges:");
int x=s.nextInt();
System.out.println("Enter maximum number of Nodes:");
int y=s.nextInt();
if(x>y){
Graph g=new Graph(x);
for(int i=0;i<y;i++){
System.out.println("Enter node"+" "+(i+1)+":");
char c=s.next().charAt(0);
g.addNode(c);
}
for(int i=0;i<x;i++){
System.out.println("Enter two number that make edge between two nodes :");
int start=s.nextInt();
int end=s.nextInt();
g.addEdge(start, end);
}
System.out.println("----------------------------------------");
g.mstanddfs();}

   else{
   Graph g=new Graph(y);
   for(int i=0;i<y;i++){
   System.out.println("Enter node"+" "+(i+1)+":");
   char c=s.next().charAt(0);
   g.addNode(c);
   }
   for(int i=0;i<x;i++){
   System.out.println("Enter two number that make edge between two nodes :");
   int start=s.nextInt();
   int end=s.nextInt();
   g.addEdge(start, end);
   }
   System.out.println("----------------------------------------");
   g.mstanddfs();}
```

-Space and time complexity:-

O(n)------->addNode and addEdge