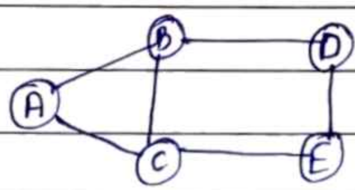
* Graph

- A graph G is a set of vertices (V) and set of edges (E).
- The set V is a finite, nonempty set of vertices. The set E is a set of pair of vertices representing edges.

$$G = (V, E)$$

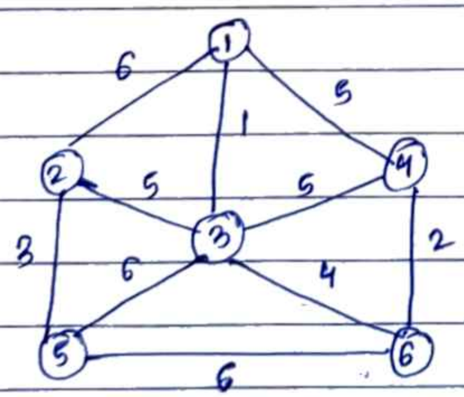
$V(G)$ = Vertices of graph G

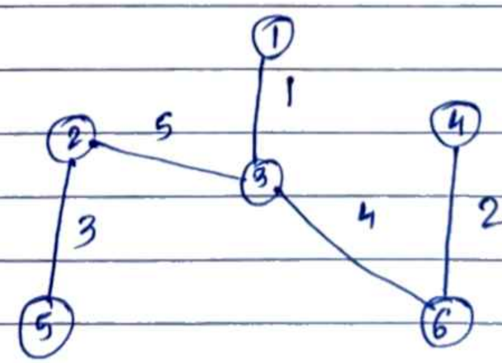
$E(G)$ = Edges of graph G



* Minimum spanning tree

- With applications of weighted graphs, it is often necessary to find a spanning tree for which the total weight of the edges in the tree is as small as possible.
- Such a spanning tree is called a minimal spanning tree or minimum cost spanning tree.





* Prim's Algorithm

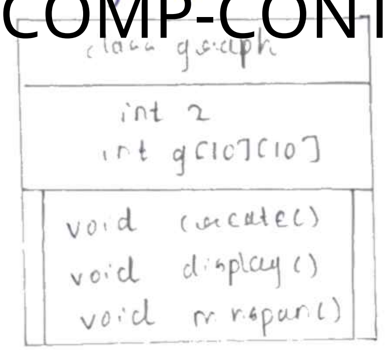
- Let the Graph $G = (V, E)$ has n vertices.

Step 1: Choose a vertex V_1 of G. Let $V_T = \{V_1\}$ and $E_T = \{\}$

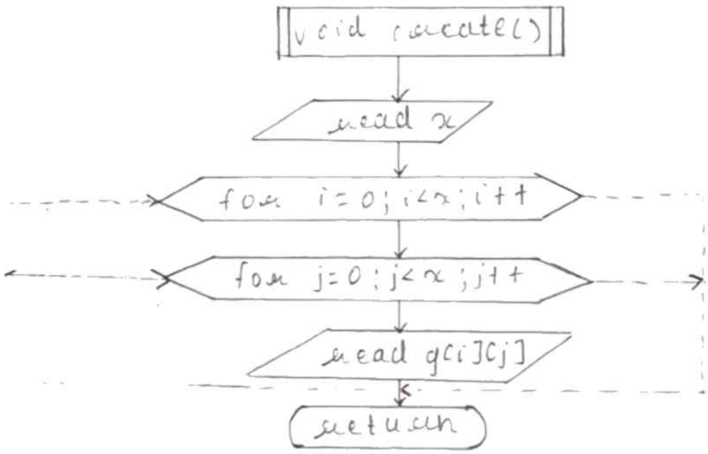
Step 2: Choose a nearest neighbour V_i of V_T that is adjacent to V_j, $V_j \in V_T$ and for which the edge (V_i, V_j) does not form a cycle with members of E_T. Add V_i to V_T and add (V_i, V_j) to E_T.

Step 3: Repeat step 2 until $|E_T| = n-1$. Then V_T contains all n vertices of G and E_T contains the edges of the minimum cost spanning tree of G.

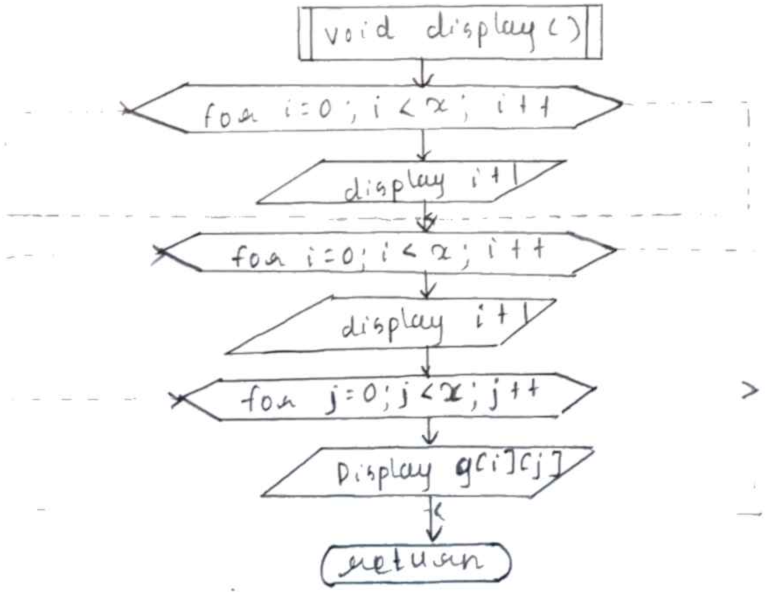
Flowchart for class graph



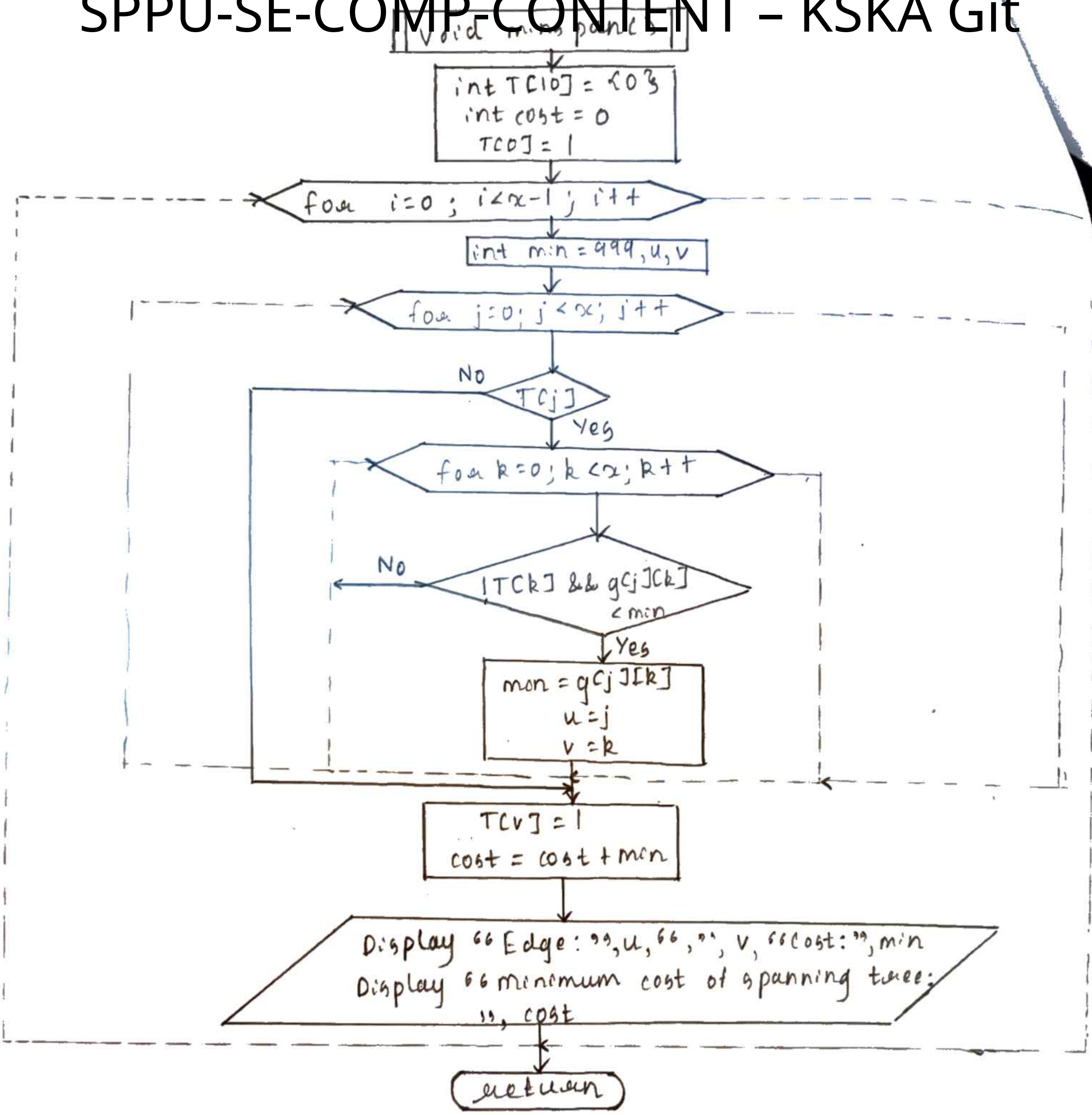
→ Flowchart for void create()



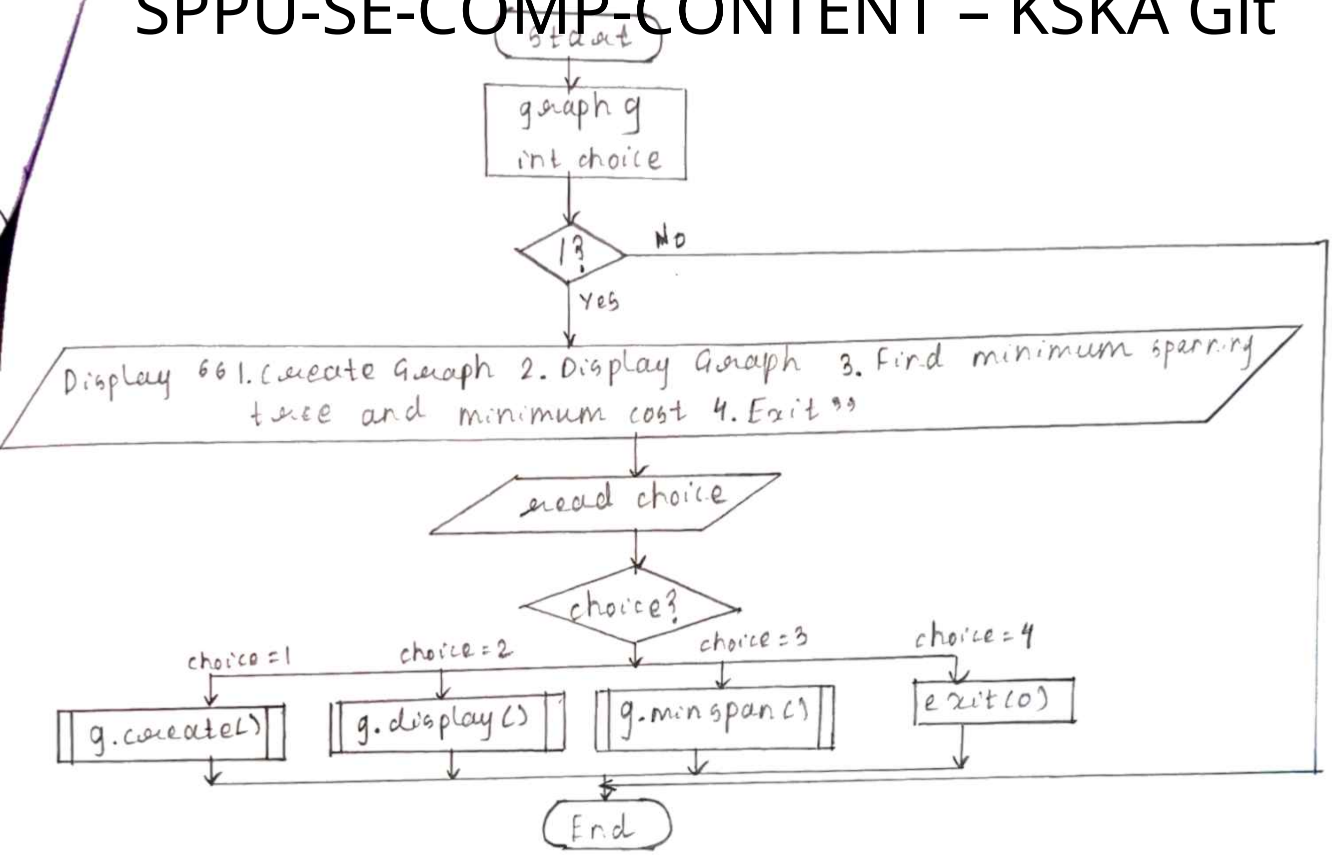
→ Flowchart for void display()



→ Flowchart for void mingpan()



flowchart for int main()



→ Pseudocode for class graph

1. Declare int x
   int g[10][10]
2. Declare function create()
   function display()
   function minspan()

→ Pseudocode for ~~class~~ void create()

1. Display "Enter number of nodes:"
2. read x
3. for i=0; i<x; i++ do
   begin
   for j=0; j<x; j++ do
   begin
   Display "Enter cost of graph:"
   read g[i][j]
   end
   end
4. return

→ Pseudocode for void display()

1. for i=0; i<x; i++ do
   begin
   display i+1
   end
2. for i=0; i<x; i++ do
   begin
   display i+1
   for j=0; j<x; j++ do
   begin
   display g[i][j]
   end
   end

```
3 return

→ Pseudocode for void minspan()
1. declare int T[10] = {0}
2. initialize cost = 0
3. initialize T[0] = 1
4. for i=0 ; i<x-1; i++ do
begin
    initialize min = 999 ,u, v
    for j=0; j<x; j++ do
    begin
            if T[j] then
                for k=0; k<x ;k++ do
                begin
                        if !T[k] and g[j][k] < min then
                                initialize min= g[j][k]
                                        u=j
                                        v=k

                end
    end
    initialize T[v]=1
    cost = cost + min
    Display "Edge: (", u, ", ", v, ") cost: ", min
    Display "Minimum cost of spanning tree: ", cost
end
5 return

→ Pseudocode for int main()
1. Start
2 create object g
3 Declare int choice
4. while (1) do
```

begin
Display "1. Create Graph 2. Display Graph 3. Find minimum spanning tree and minimum cost 4. Exit"
read choice
switch (choice)
case1:
call function g.create()
break
case 2:
call function g.display()
break
case 3:
call function g.minspan()
break
case 4:
exit(0)
break
default:
Display "Enter valid choice"
break
end
5 End

Q1. Suppose we have an undirected graph with weights that can be either positive or negative. Do Prim's and Kruskal's algorithm produce an MST for such a graph?

Ans. Yes, negative edge weights are no problem for Prim's and Kruskal's algorithm

- Both of these algorithms are 'greedy' algorithms and the reason why the greedy approaches to finding the MST work is that you can always get a better ST if there is an unused edge that has a lower weight than any edge on the cycle that would be created by adding it on the ST.
- This principle holds for both positive and negative edge weights.

Q2. Can a graph have more than one spanning trees

Ans. A connected graph can have more than one spanning tree.

- All spanning trees must contain the same number of vertices as of graph, and the number of edges must be equal to |V| - 1.
- The spanning tree must not contain any cycle.
- A spanning tree is a subgraph of a graph in which all the vertices are connected and it does not have any loops.
- Therefore, a graph can have multiple spanning trees

Q3. State the difference between Prims and Kruskals algorithm.

Soln:

	Kruskals	Prims
1.	Select the shortest edge in a network.	1. Select any vertex
2.	Select the next shortest edge which does not create a cycle.	2. Select the shortest edge connected to the vertex
3.	If graph is created using adjacency matrix, time complexity → $O(n^2)$	3. If graph is created using adjacency matrix, time complexity → $O(n^2)$
4	If graph is created using list, Time complexity → $O(n^2) + O(E.\log n) = O(n^2)$	4. If graph is created using list, Time complexity → $O(E.\log n)$
5.	Kruskals algorithm performs better for sparse graph	5. Prims algorithm is comparatively less efficient for a sparse graph.