Graph(weighted)

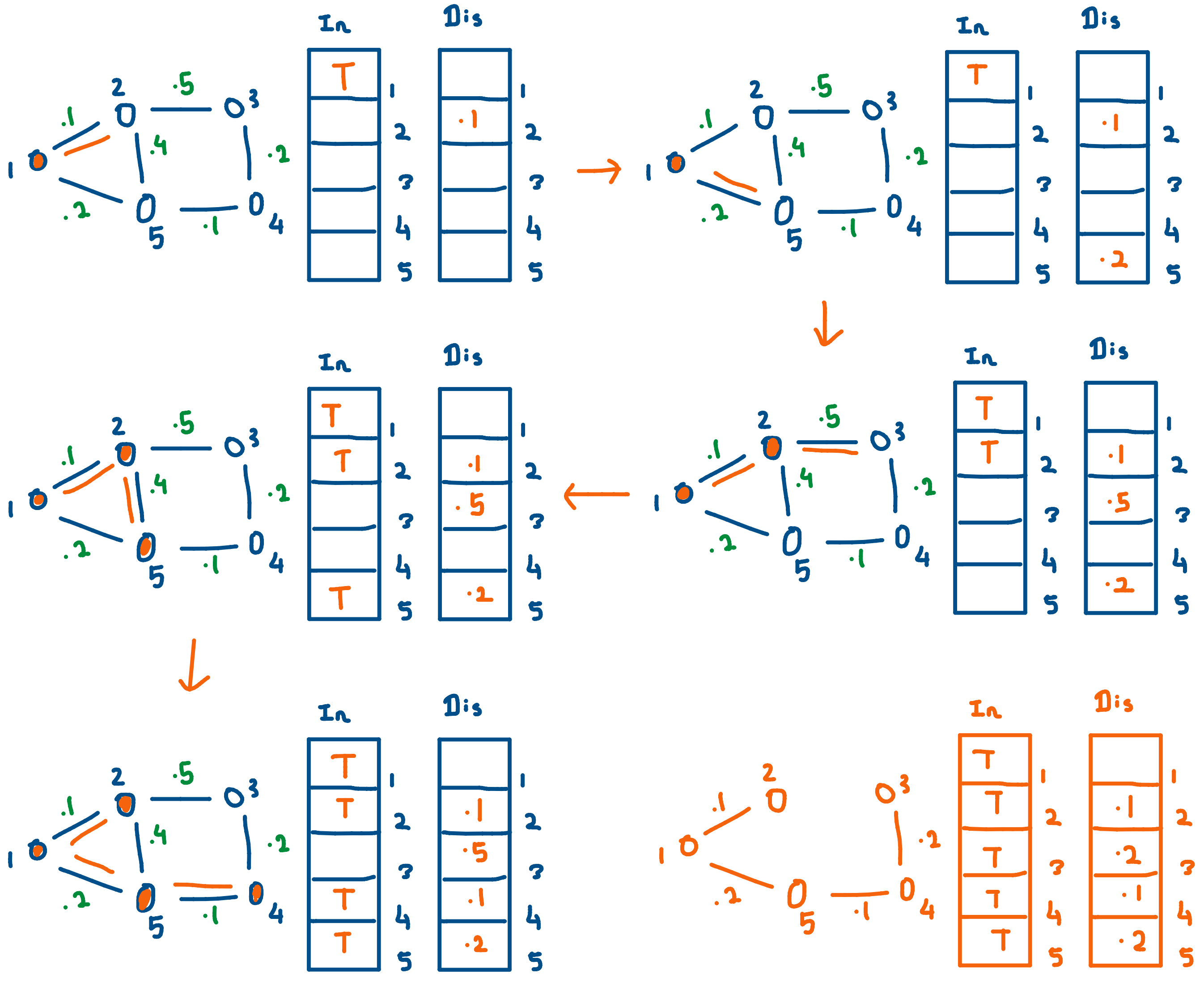
20 July 2021 03:00

Weighted graph algorithms involve finding out MST.
A minimum spanning tree minimizes the total length over all possible spanning trees. However, there can be more than one minimum spanning tree in a graph. Indeed, all spanning trees of an unweighted (or equally weighted) graph G are minimum spanning trees, since each contains exactly n − 1 equal-weight edges.

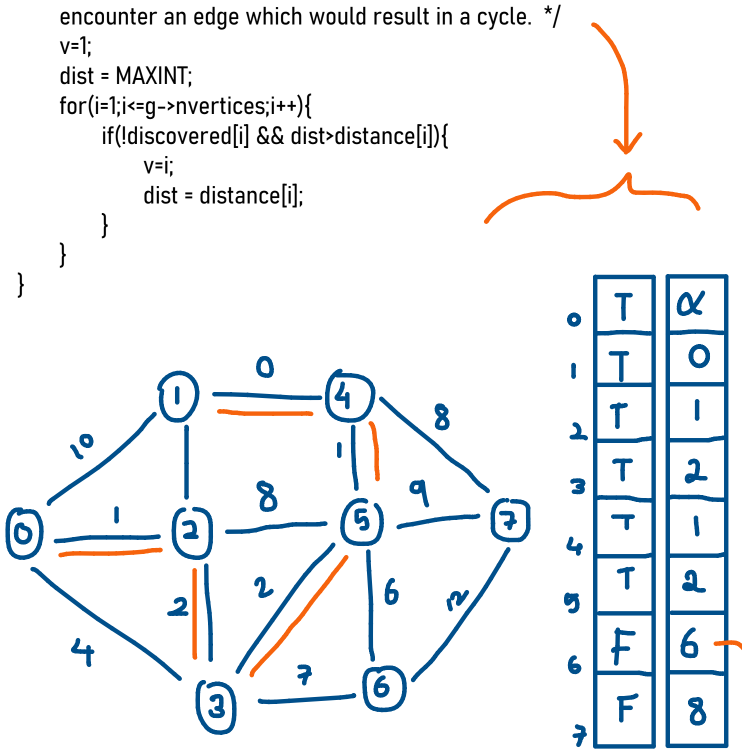
Prim's Algo :

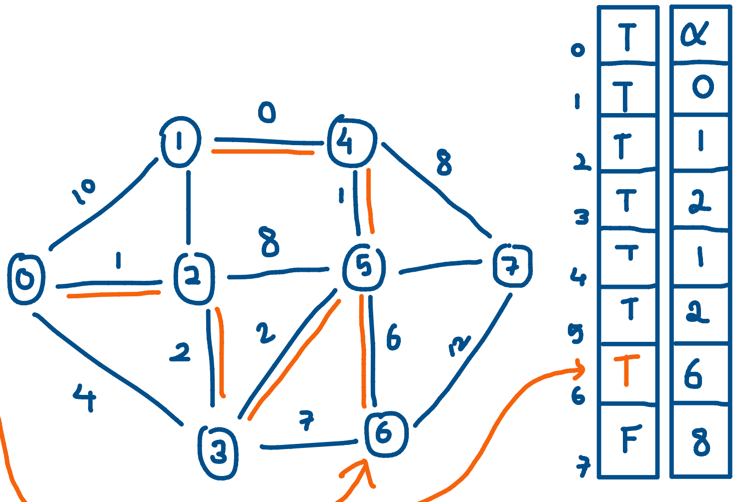
MST finding algorithm :

a. Greedy
b. Finds best local option



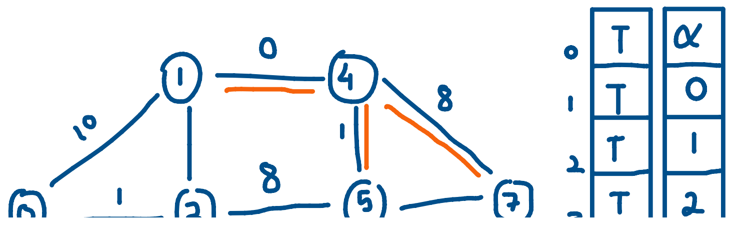
```
prim(graph *g,int start){
    int i;    /* counter */
    edgenode *p;    /* temporary pointer */
    bool intree[MAXV+1];    /* is the vertex in the tree yet? */
    int distance[MAXV+1];    /* cost of adding to tree */
    int v;    /* current vertex to process */
    int w;    /* candidate next vertex */
int weight;     /* edge weight */
int dist;     /* best current distance from start */

for (i=1; i<=g->nvertices; i++) {
      intree[i] = FALSE;
      distance[i] = MAXINT;
      parent[i] = -1;
}

distance[start] = 0;
v = start

while(!intree[v]){
      intree[v] = true;
      temp = g->edges[v];
      while(temp != NULL){
            y = temp->y;
            w = temp->weight;
            if(!intree[y] && distance[y]>w){
                  distance[y] = w;
                  parent[y] = v;
            }
            temp = temp->next;
      }
}

/* This rest of the code is when a branch arises in the tree or we
encounter an edge which would result in a cycle.  */
v=1;
dist = MAXINT;
for(i=1;i<=g->nvertices;i++){
      if(!discovered[i] && dist>distance[i]){
            v=i;
            dist = distance[i];
      }
}
}
```

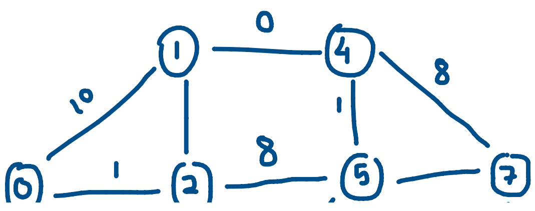


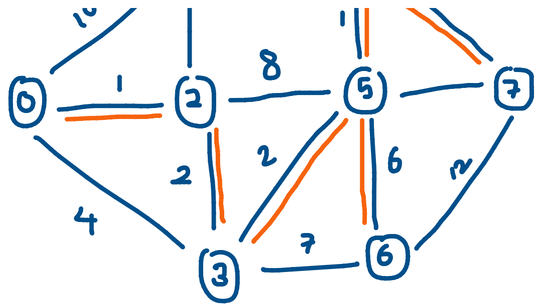


Here 6 is the first non tree vertex that has least weight value so we put it into the MST

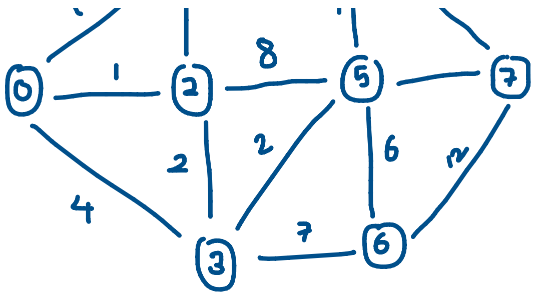
In this position in prim when we reach 1, the only available edge is ready in the tree and connection would cause a cycle. So we have to fall back to a non tree edge which has the least value in the distance array







2	T	1
3	T	2
4	T	1
5	T	2
6	T	6
7	T	8



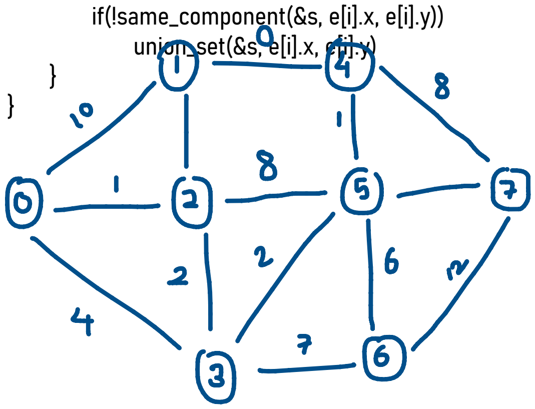
In this position in prim after inserting the 6 node we find only edge left is 7 so we get it in the tree with distance already present.

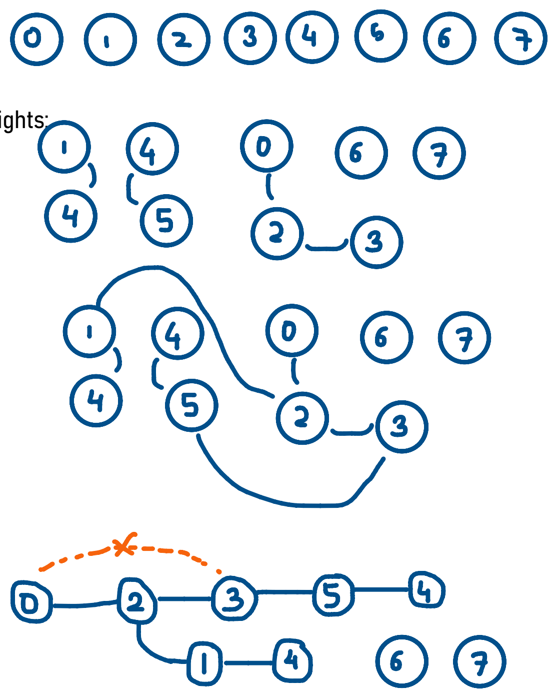
Kruskal's Algo :

It is a MST finding algorithm which uses the union set data structure where different edges are grouped into different categories and then classified and union into one single group
Steps in Kruskal's algo:

i. Make array of edges
ii. Sort that array
iii. Insert sorted edges from array into set union
iv. Categorize one edge into a color group
v. If another edge has a common vertex with any other group, categorize the new edge to the previous one's group if the previous one had more vertices in them.

```
kruskal(graph *g){
    int i;    /* counter */
    set_union s;    /* set union data structure */
    edge_pair e[MAXV+1];    /* array of edges data structure */
    bool weight_compare();    /* Function to compare weights b/w edges */

    set_union_init(&s, g->nvertices);
    to_edge_array(g, e); /* sort edges by increasing cost */
    qsort(&e,g->nedges,sizeof(edge_pair),weight_compare);

    for(i=0;i<g->nedges;i++){
        if(!same_component(&s, e[i].x, e[i].y))
            union_set(&s, e[i].x, e[i].y)
    }
}
```





Sorted edge array by weights:

a. 1,4-0
b. 5,4-1

c. 0,2-1
d. 2,3-2
e. 3,5-2

f. 1,2-3

g. 0,3-4
h. 5,6-6
i. 3,6-7

j. 4,7-8
k. 5,7-9
l. 0,1-10
m. 6,7-12

In case of option "g" 0 and 3 are both in the same group
so that edge is ignored

Set Union DS:

It is a data structure used in Kruskal MST algo. The steps are of this are:

a. Parent array contains parent of elements, initially the parent of each element is the element itself.
b. Find function to find the parent of a certain group vertex
c. Union of edges and grouping

```
typedef struct{
      int parent[SETSIZE+1];
      int nsize[SETSIZE+1];
      int n;
}set_union;

set_union_init(set_union *s,int n){
      int i;
      for(i=0;i<n;i++){
            s->parent[i] = i;
            s->nsize[i] = 1;
      }
}

set_find(set_union *s, int x){
      if(s->parent[x]==x)
            return x;
      else
            return set_find(s,s->parent[x]);
}
```

```
union_set(set_union *s, int x, int y){
    int r1,r2;
    if(s->parent[x] == s->parent[y])
        return
    r1 = parent(s,x);
    r2 = parent(s,y);
    else if(s->nsize[r2] >= s->nsize[r1]){
        s->nsize[r2] += s->nsize[r1];
        s->parent[r1] = r2;
    }
    else if(s->nsize[r2] < s->nsize[r1]){
        s->nsize[r1] += s->nsize[r2];
        s->parent[r2] = r1;
    }
}
```

Types of MSTs:

Different types of MSTs and their designs are:

a. Maximum Spanning Trees: Make all the weights negative and find
b. Minimum Product STs: Log of all weights
c. Maximum Bottleneck STs: Bottleneck of flow as weights

Dijkstra's Shortest Path Algo:

A shortest path find algo where dynamic programming is used and is very similar to prim's algorithm and only 3 lines from prim's is changed.

```
prim(graph *g,int start){
    int i;    /* counter */
    edgenode *p;    /* temporary pointer */
    bool intree[MAXV+1];    /* is the vertex in the tree yet? */
    int distance[MAXV+1];    /* cost of adding to tree */
    int v;    /* current vertex to process */
    int w;    /* candidate next vertex */
    int weight;    /* edge weight */
    int dist;    /* best current distance from start */

    for (i=1; i<=g->nvertices; i++) {
        intree[i] = FALSE;
        distance[i] = MAXINT;
        parent[i] = -1;
    }

    distance[start] = 0;
    v = start

    while(!intree[v]){
        intree[v] = true;                  ]
        temp = g->edges[v];                ] → changed
        while(temp != NULL){               ]
            y = temp->y;
            w = temp->weight;
            if(distance[y] > distance[v] + w){
                distance[y] = distance[v] + w;
                parent[y] = v;
            }
            temp = temp->next;
        }
    }

    /* This rest of the code is when a branch arises in the tree or we
```

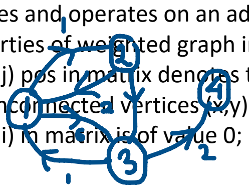
```
        encounter an edge which would result in a cycle.  */
        v=1;
        dist = MAXINT;
        for(i=1;i<=g->nvertices;i++){
            if(!discovered[i] && dist>distance[i]){
                v=i;
                dist = distance[i];
            }
        }
}
```

Floyd Warshall Algo:

An all pairs shortest path algorithm, finds best path between any pair of vertices and operates on an adjacency matrix.

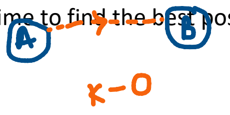
Properties of weighted graph in adjacency matrix:

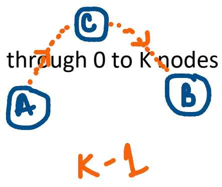
a. (i,j) pos in matrix denotes the weight b/w that edge
b. unconnected vertices (x,y) has weight infinity
c. (i,i) in matrix is of value 0;

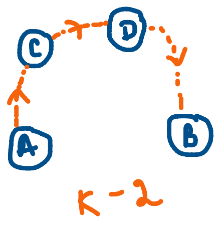


	1	2	3	4
1	0	1	6	∝
2	2	0	2	∝
3	1	∝	0	2
4	∝	∝	∝	0

Algorithm checks the path between A and B through 0 to K nodes one at a time to find the best possible path.







This goes on to K = K via all the intermediate edges between A and B to find the best possible path.

```
floyd_warshall(adjacency_matrix *g){
        int i,j,k;
        int through_k;

        for(k=1;k<=g->nvertices;k++)
            for(i=1;i<=g->nvertices;i++)
                for(j=1;j<=nvertices;j++){
                    through_k = g->weights[i][k] + g->weights[k][j];
                    if(g->weights[i][j] > through_k)
                        g->weights[i][j] = through_k;
                }
}
```