HW#7

* var.mod = initialize

1.) mod attribute of V

a.) * let .V = vertex, .color = color, .π = predecessor of v from source, u ∈ V and v ∈ V
assume that graph (G) is being passed as an adj list. let s = source vertex, let Q = queue STL structure

```
BFS_Components(G, S)
FOR each vertex u ∈ G.V-{s}
    u.color = white
    u.π = NULL
s.color = grey
s.π = NULL
Q = ∅
ENQUEUE(Q, s)
WHILE Q ≠ ∅
    u = DEQUEUE
    FOR each v ∈ G.adj[u]
        IF v.color == white
            v.color = grey
            v.π = u
            vertex++ edge++
            ENQUE(Q,v)
    u.color = black
    For w ∈ G.adj[v]
        IF w = grey
            list_comp[count] = comp[2]
            comp[0] = vertex
            list_comp[count][0] = vert
            comp[1] = edge
            list_comp[count][1] = edge
```

c.) This algorithm doesn't work b/c the algorithm can be mislead away from taking an initially heavier weight but faster in the long run (see ex_1, right)

cont...

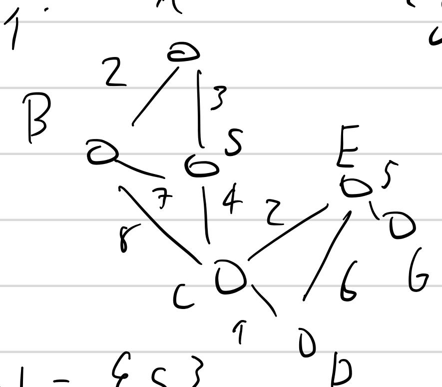
```
int #vertex
int #edge
Map<int, int*>
int component[2]
```

Runs not in Θ(V+E). Runs in Θ(V+VE)

```
vertex = 0
edge = 0
map<int, int*> list_comp
comp[2]     // vertex and edge #
count = 0   // # of component
            // key in map
```

* let S = Start let E = End (goal)

ex_1:



path = {S}
= {(S, A), (A, B), (B, C), (C, D), (D, E)}

2.)

a.) data structure: priority queue

Prim's Algorithm runs in $\Theta(E\log(V))$ (see aggregate analysis bellow)

```
MST_Prims(G, w, r)
  FOR each u ∈ G.V                          // 2V = Θ(V) oppe each
  [  u.key = ∞                              vertex
  [  u.π = NULL
  r.key = 0                                 // set central vertex (r) key Θ(1)
  Q = G.V                                   // Θ(V) enque everything
  WHILE Q ≠ ∅
  [  u = EXTRACT_MIN(Q)                     // Θ(log v) Θ(V) extract min
  [  FOR each v ∈ G.Adj[u]                  V times // Θ E, each edge
  [  [  IF v ∈ Q and w(u, v) < v.key
  [  [  [  v.π = u
  [  [  [  v.key = w(u,v)
```

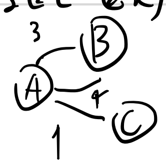
$\boxed{E\log(V)}$

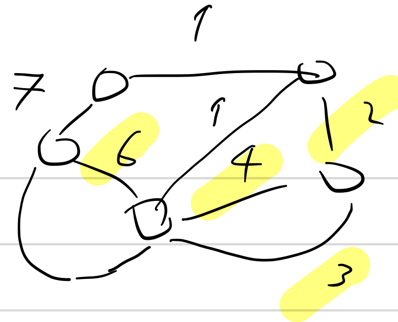
3.)

Tree: graph that is connected and acyclic.

a.) This is not a minimum because an edge can be added that is heavier to visit that vertex before the lighter one if theres arbitrary add edge order and no cycles in final graph (see ex): edge {A, C} and then {A, B} w/ value 4 could be added and then {A, B} val 3 wouldn't be added b/c create cycle (not MST)

(whole graph) →

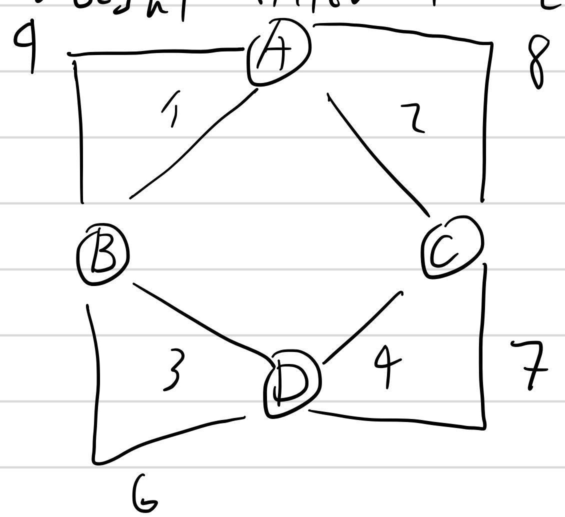




as a result would not be connected.
Therefore the graph isn't a tree b/c it's not connected and acyclic.
Also, not a MST

b.) This is not a minimum spanning tree because it could result in a cycle. A tree is an acyclic graph, therefore MST wouldn't be present (see ex bellow)
The graph doesn't filter for cycles, no tree is result.

(original graph)



(graph after algorithm)