Recitation 12
(Additional Practice Problems)

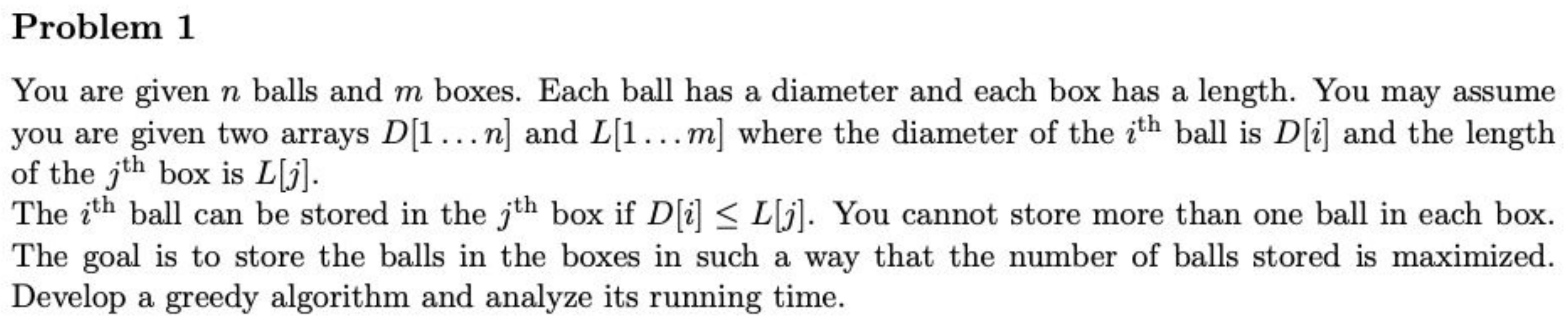
Online: Xinyi Zhao xz2833@nyu.edu

GCASL 461: Yifan Jin yj2063@nyu.edu

New York University

Basic Algorithms (CSCI-UA.0310-005)

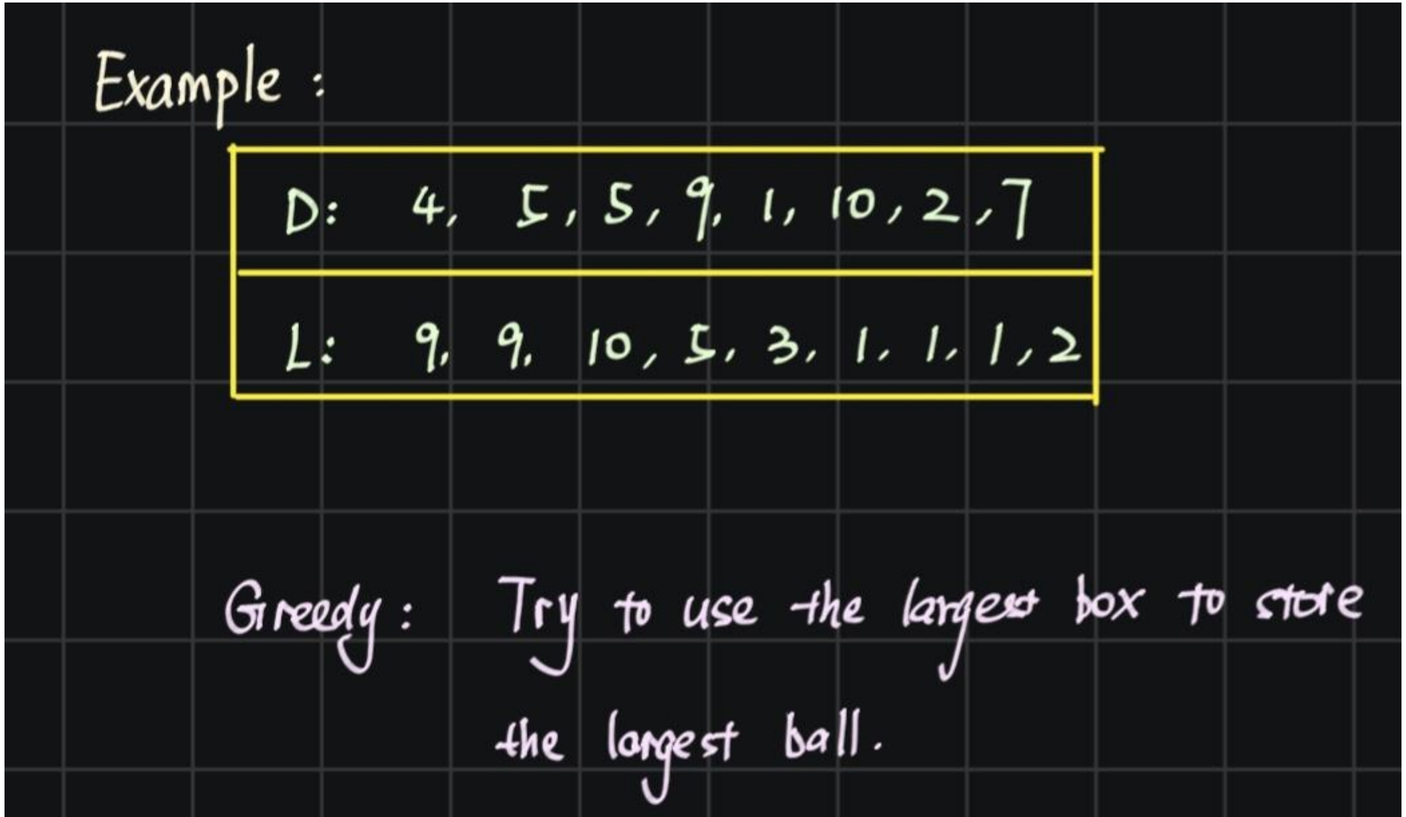
Problem 1



Problem 1

You are given n balls and m boxes. Each ball has a diameter and each box has a length. You may assume you are given two arrays $D[1 \dots n]$ and $L[1 \dots m]$ where the diameter of the i^{th} ball is $D[i]$ and the length of the j^{th} box is $L[j]$.
The i^{th} ball can be stored in the j^{th} box if $D[i] \leq L[j]$. You cannot store more than one ball in each box. The goal is to store the balls in the boxes in such a way that the number of balls stored is maximized. Develop a greedy algorithm and analyze its running time.

Problem 1


Example :

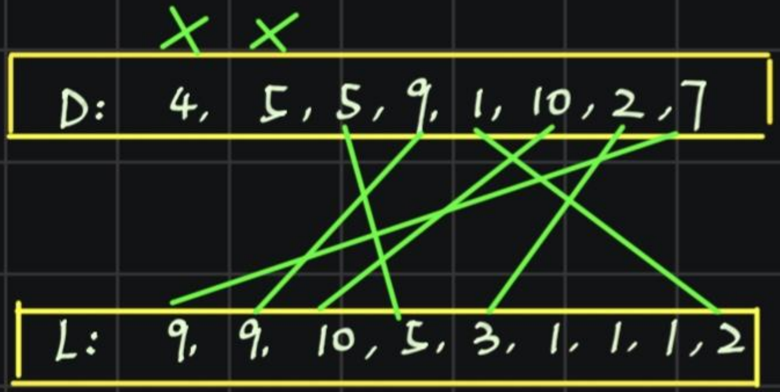
D:	4,	5,	5,	9,	1,	10,	2,	7	
L:	9,	9,	10,	5,	3,	1,	1,	1,	2

Greedy : Try to use the largest box to store the largest ball.

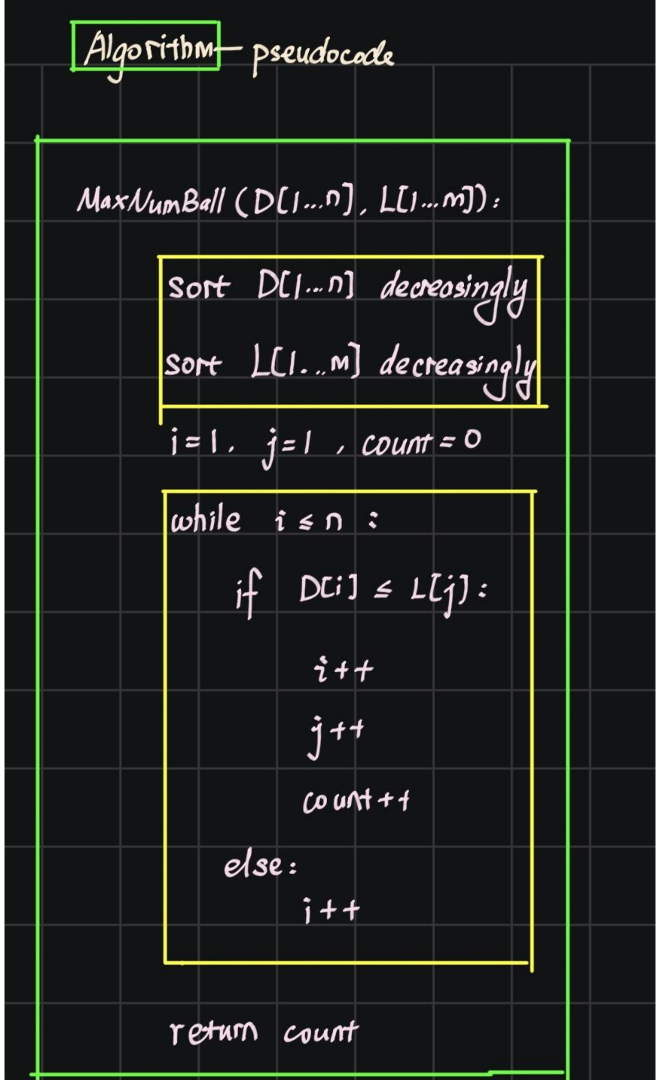
Problem 1

Greedy: Try to use the largest box to store the largest ball.

$\Rightarrow$



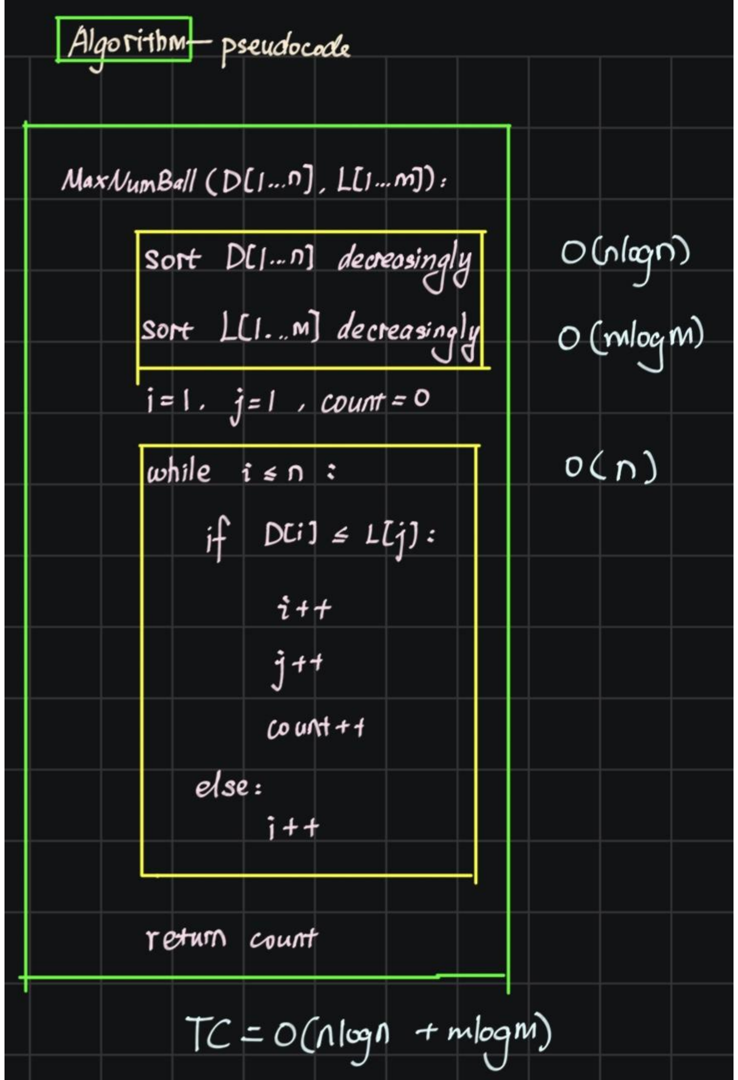
Problem 1



Algorithm – pseudocode

```
MaxNumBall(D[1...n], L[1...m]):
    sort D[1...n] decreasingly
    sort L[1...m] decreasingly
    i=1, j=1, count=0
    while i ≤ n:
        if D[i] ≤ L[j]:
            i++
            j++
            count++
        else:
            i++
    return count
```

Problem 1

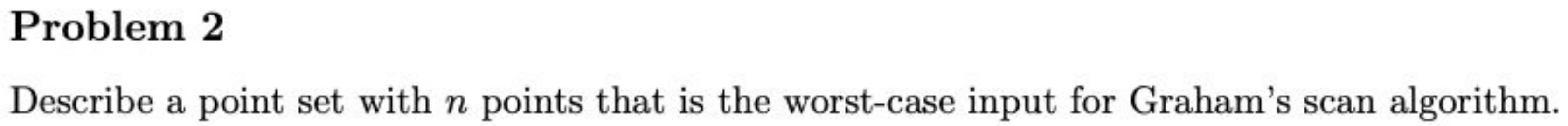


Algorithm — pseudocode

```
MaxNumBall(D[1...n], L[1...m]):
    sort D[1...n] decreasingly          O(nlogn)
    sort L[1...m] decreasingly          O(mlogm)
    i=1, j=1, count=0
    while i ≤ n :                       O(n)
        if D[i] ≤ L[j]:
            i++
            j++
            count++
        else:
            i++
    return count
```

$TC = O(n\log n + m\log m)$

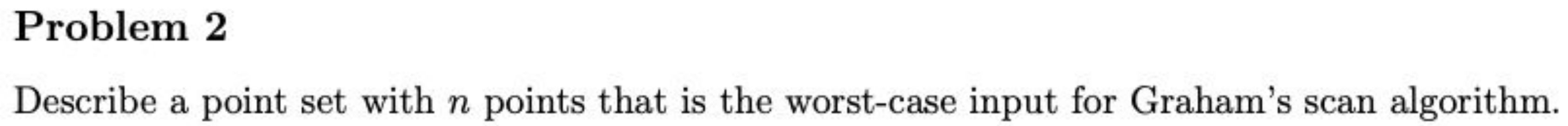
Problem 2



Problem 2

Describe a point set with n points that is the worst-case input for Graham's scan algorithm.

Problem 2



Problem 2

Describe a point set with n points that is the worst-case input for Graham's scan algorithm.

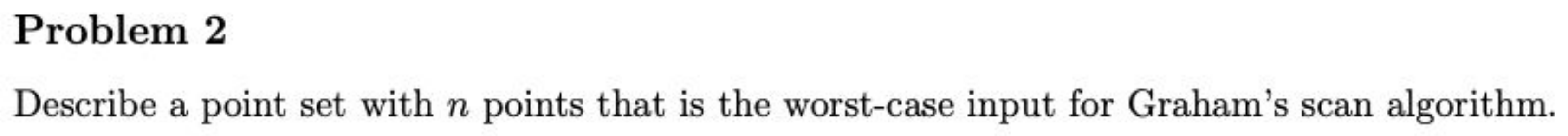
In the Graham's scan, there are different types of operations:

1- operations needed to sort out n-1 points

2- adding each point once to the stack

3- removing some of the points at most once from the stack

Problem 2



Problem 2

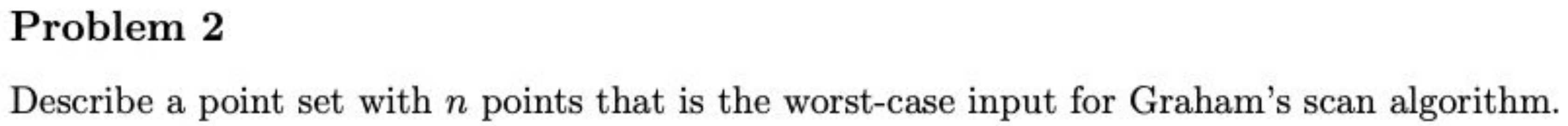
Describe a point set with n points that is the worst-case input for Graham's scan algorithm.

The number of operations for (1) becomes maximal depending on how the points are given as an input.

More precisely, if the input points are given in the reverse order as they appear in the sorted order in step (1).

This part only depends on how the input points are given to us.

Problem 2


Problem 2

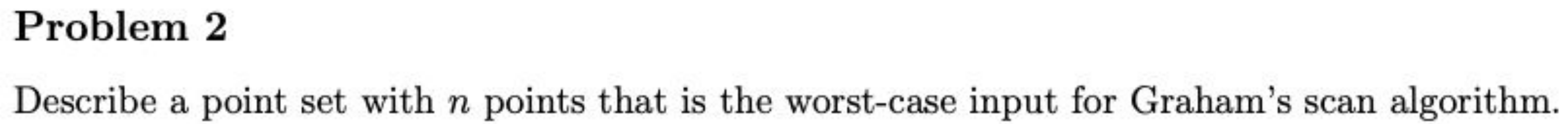
Describe a point set with n points that is the worst-case input for Graham's scan algorithm.

For part (2), we have exactly n operations which cannot be increased/decreased.

For part (3), the number of removals becomes maximized when almost all the points are removed.

Note that we need at least 3 points to remain in the stack since there are always at least three points on the boundary of any convex hull.

Problem 2



Problem 2

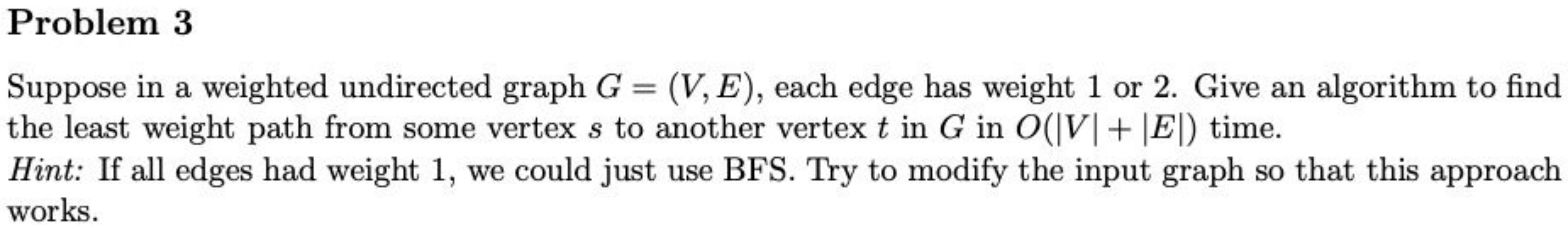
Describe a point set with n points that is the worst-case input for Graham's scan algorithm.

Thus, an example of a point set giving the worst-case for Graham's scan is:

A set of n points whose convex hull only consists of 3 points.

Any such point set gives the worst running time.

Problem 3



Problem 3

Suppose in a weighted undirected graph $G = (V, E)$, each edge has weight 1 or 2. Give an algorithm to find the least weight path from some vertex s to another vertex t in G in $O(|V| + |E|)$ time.
Hint: If all edges had weight 1, we could just use BFS. Try to modify the input graph so that this approach works.

Problem 3

If all edges had weight 1, we could just use BFS.

Why?

One important observation about BFS: the path used in BFS always has least number of edges between any two vertices. So if all edges are of same weight, we can use BFS to find the shortest path.

Running time: O(|V|+|E|)

Problem 3

For this problem, all edges are of weight either 1 or 2.

Can we transfer this graph to a new graph with all edges of weight 1? How?

G -> G’

Problem 3

We can modify the graph and split all edges of weight 2 into two edges of weight 1 each.

In the modified graph G’, we can use BFS(G’) to find the shortest path. Running time: O(|V|+|E|)

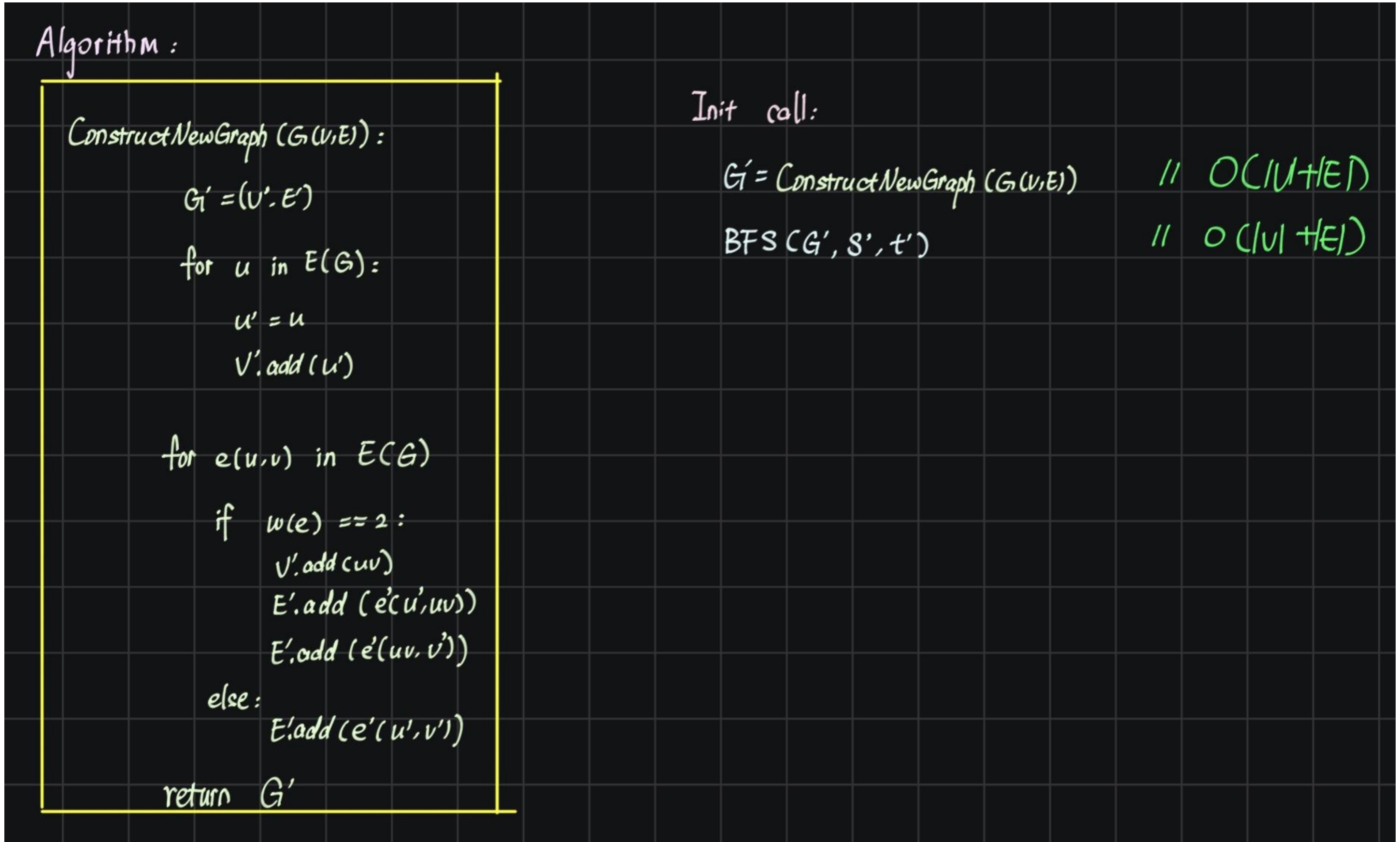
How to construct G’?

For each pair (u,v) whose edge(u,v) = 2:

Add intermediate vertex, uv;

Add one edge weighted 1 for (u, uv); add one edge weighted 1 for (uv, v).

Problem 3



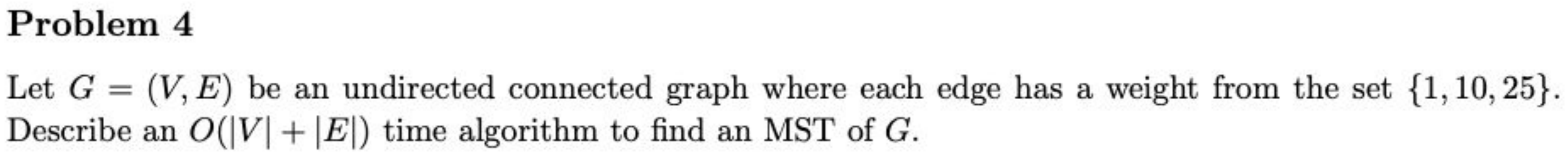
Algorithm:

```
ConstructNewGraph (G(V,E)):
        G' = (V', E')
        for u in E(G):
            u' = u
            V'.add(u')

        for e(u,v) in E(G)
            if w(e) == 2:
                V'.add(uv)
                E'.add(e'(u',uv))
                E'.add(e'(uv,v'))
            else:
                E'.add(e'(u',v'))
        return G'
```

Init call:

```
G' = ConstructNewGraph (G(V,E))    // O(|V|+|E|)
BFS(G', s', t')                    // O(|V|+|E|)
```

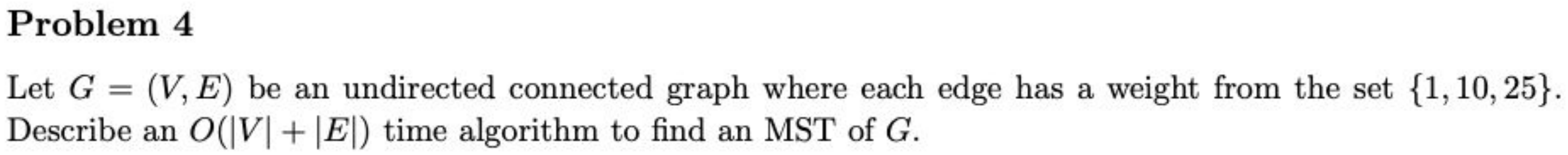
Problem 4



Problem 4

Let $G = (V, E)$ be an undirected connected graph where each edge has a weight from the set $\{1, 10, 25\}$. Describe an $O(|V| + |E|)$ time algorithm to find an MST of G.

Problem 4



Problem 4

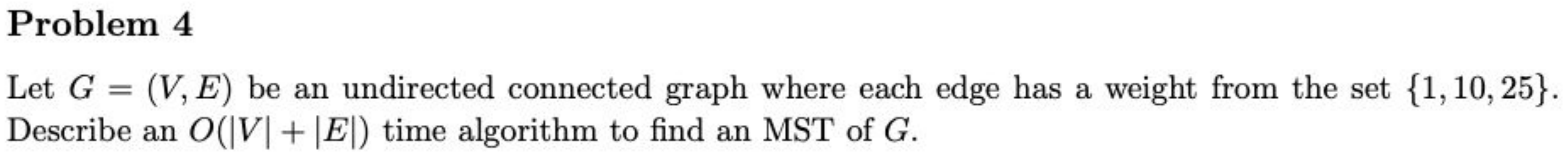
Let $G = (V, E)$ be an undirected connected graph where each edge has a weight from the set $\{1, 10, 25\}$. Describe an $O(|V| + |E|)$ time algorithm to find an MST of G.

The prim algorithm, running time **O(E log V)**.

O(log V) each operation

This is because we have to get the vertex with minimum value from a heap.

Problem 4



Problem 4

Let $G = (V, E)$ be an undirected connected graph where each edge has a weight from the set $\{1, 10, 25\}$. Describe an $O(|V| + |E|)$ time algorithm to find an MST of G.

The prim algorithm, running time **O(E log V)**.

O(log V) each operation

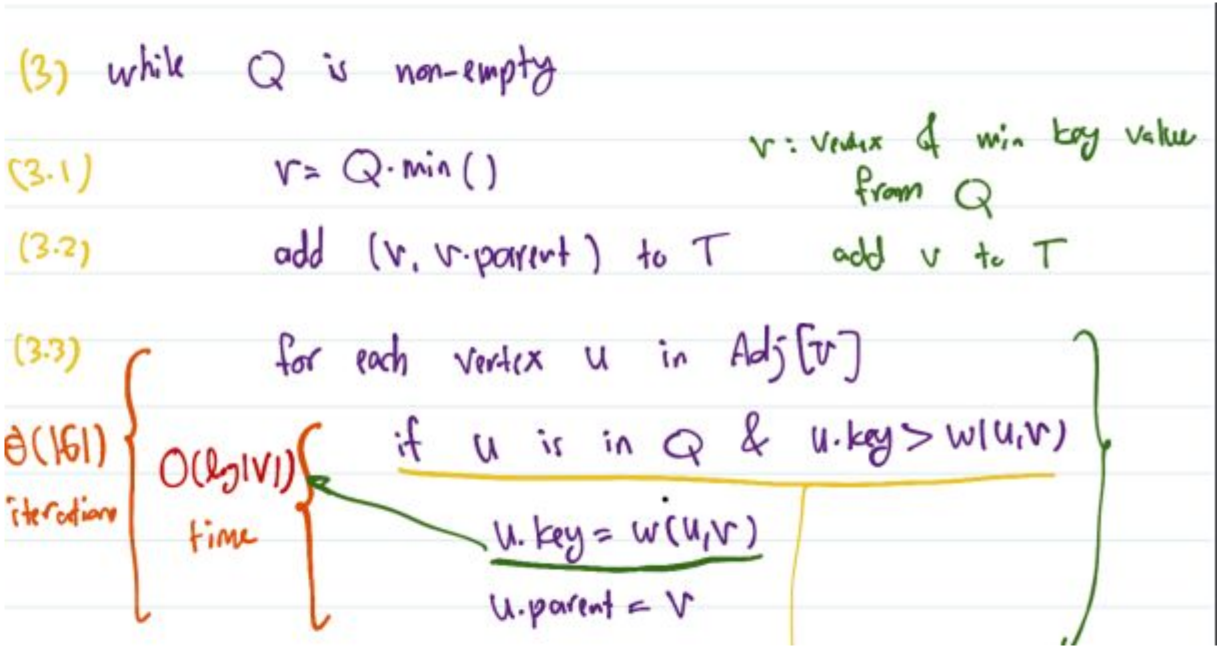
This is because we have to get the vertex with minimum value from a heap.

In this problem, the weight could only be 1/10/25, which are the only possible values.

Thus, instead of heap, we divide edges into 1-set, 10-set and 25-set. Time: **O(E)**

Problem 4

After that, repeat the following steps three times (as lectures notes):



```
(3)   while Q is non-empty
(3.1)         v = Q.min()                        v: vertex & min key value
                                                    from Q
(3.2)         add (v, v.parent) to T             add v to T
(3.3)         for each vertex u in Adj[v]
                  if u is in Q & u.key > w(u,v)
                      u.key = w(u,v)
                      u.parent = v
```

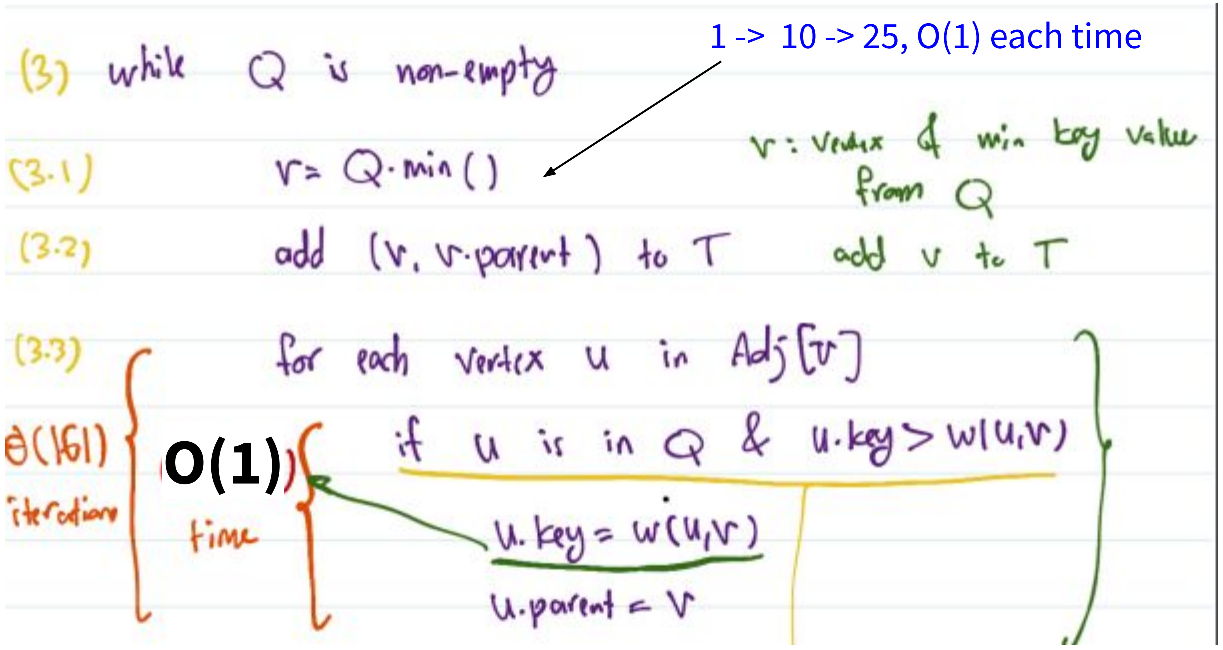
$\theta(|G|)$ iteration
$O(\log|V|)$ time

Problem 4

After that, repeat the following steps three times (as lectures notes):

1. Consider all edges in 1-set
2. Then consider all edges in 10-set (while keeping the result of previous one)
3. Consider all edges in 25-set (same as above)

1 -> 10 -> 25, O(1) each time

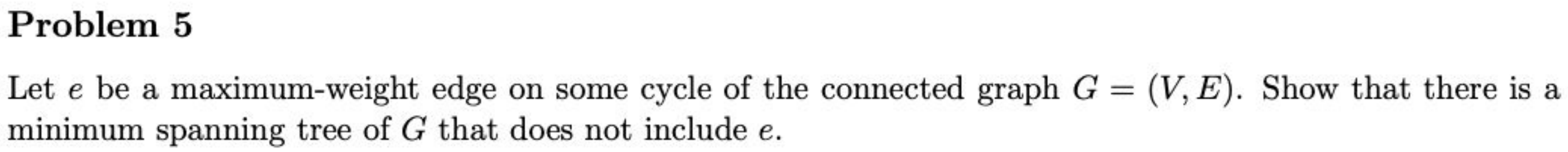


```
(3)   while Q is non-empty
(3.1)       v = Q.min()                       v : vertex of min key value
                                                  from Q
(3.2)       add (v, v.parent) to T            add v to T
(3.3)       for each vertex u in Adj[v]
                if u is in Q & u.key > w(u,v)
                    u.key = w(u,v)
                    u.parent = v
```

$\theta(161)$ iteration

O(1) time

Problem 5



Problem 5

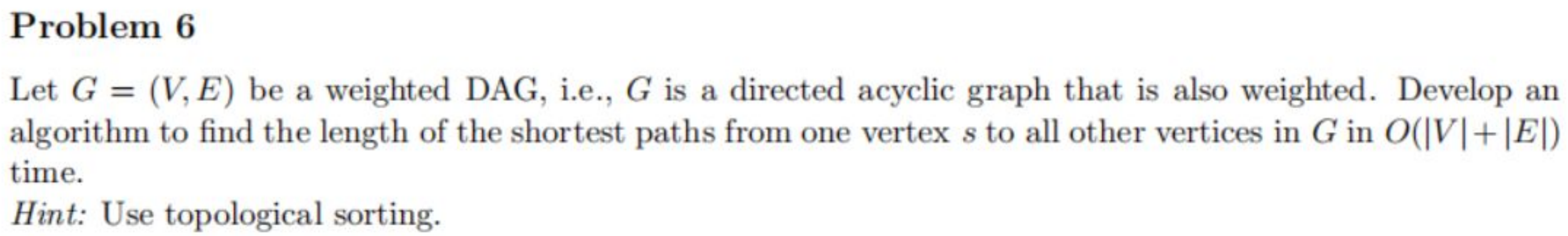
Let e be a maximum-weight edge on some cycle of the connected graph $G = (V, E)$. Show that there is a minimum spanning tree of G that does not include e.

Problem 5

Proof:

Assume that e belongs to an MST T1. Then deleting e will break T1 into two subtrees with the two ends of e in different subtrees. The remainder of G reconnects the subtrees, hence there is an edge f of G with ends in different subtrees, i.e., it reconnects the subtrees into a tree T2 with weight less than that of T1, because weight(f) <= weight(e). Then T2 is a MST which doesn't include edge e.

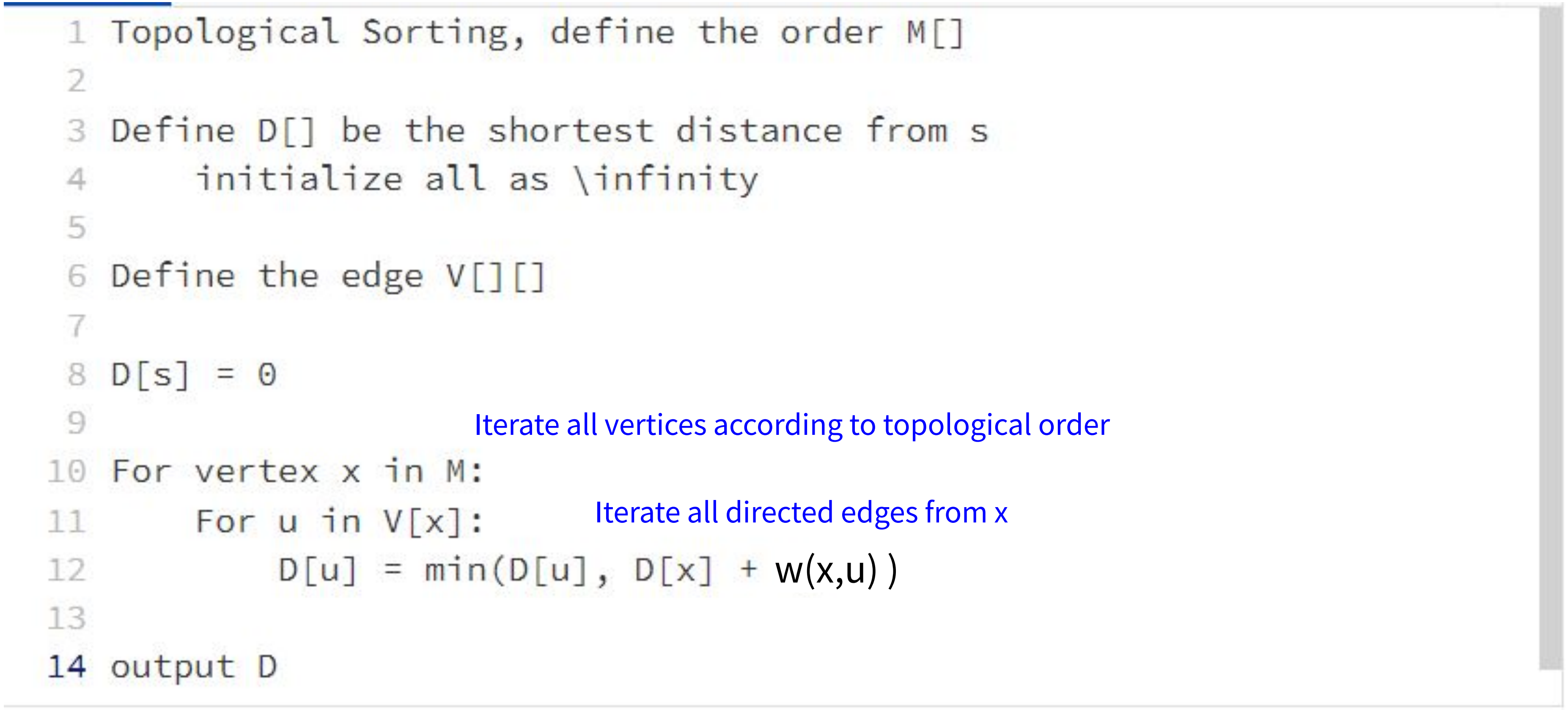
Problem 6



Problem 6

Let $G = (V, E)$ be a weighted DAG, i.e., G is a directed acyclic graph that is also weighted. Develop an algorithm to find the length of the shortest paths from one vertex s to all other vertices in G in $O(|V|+|E|)$ time.
Hint: Use topological sorting.

Problem 6

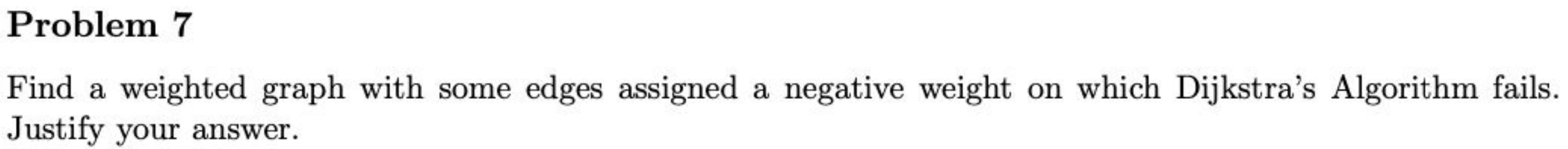


```
Topological Sorting, define the order M[]

Define D[] be the shortest distance from s
    initialize all as \infinity

Define the edge V[][]

D[s] = 0

For vertex x in M:
    For u in V[x]:
        D[u] = min(D[u], D[x] + w(x,u) )

output D
```

Iterate all vertices according to topological order

Iterate all directed edges from x

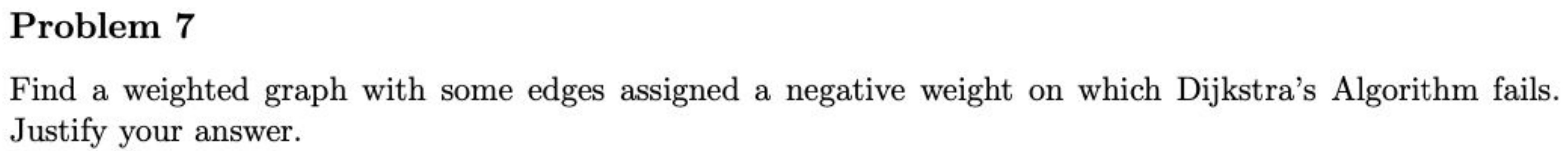
Problem 7



Problem 7

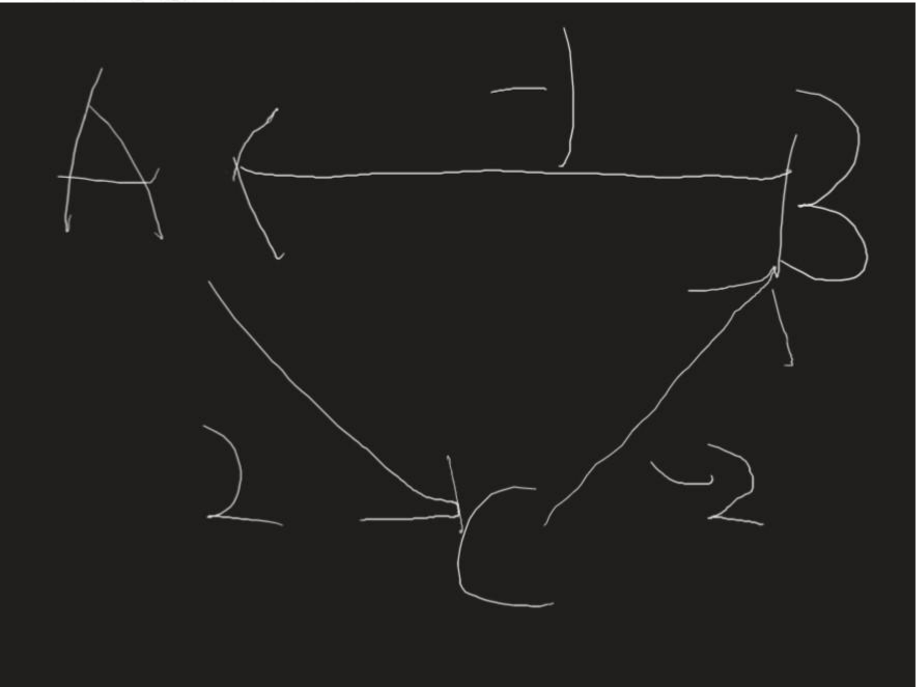
Find a weighted graph with some edges assigned a negative weight on which Dijkstra's Algorithm fails. Justify your answer.

Problem 7



Problem 7

Find a weighted graph with some edges assigned a negative weight on which Dijkstra's Algorithm fails. Justify your answer.



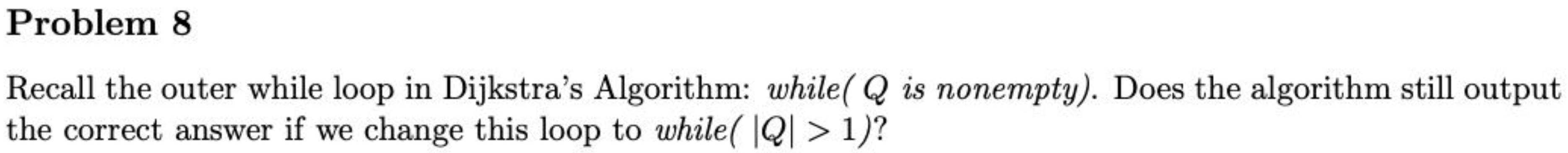
Starting from A:

Initialize Dis[A] =0
update Dis[C] = 2
Choose C
Update Dis[B] = 0

The final Dis result is [0,0,2].

However, since there is a negative loop (A->C->B->A has the weight of -1), The distance from A to B,C (and even A itself) could be -infinity. (travel along the negative loop infinite times)

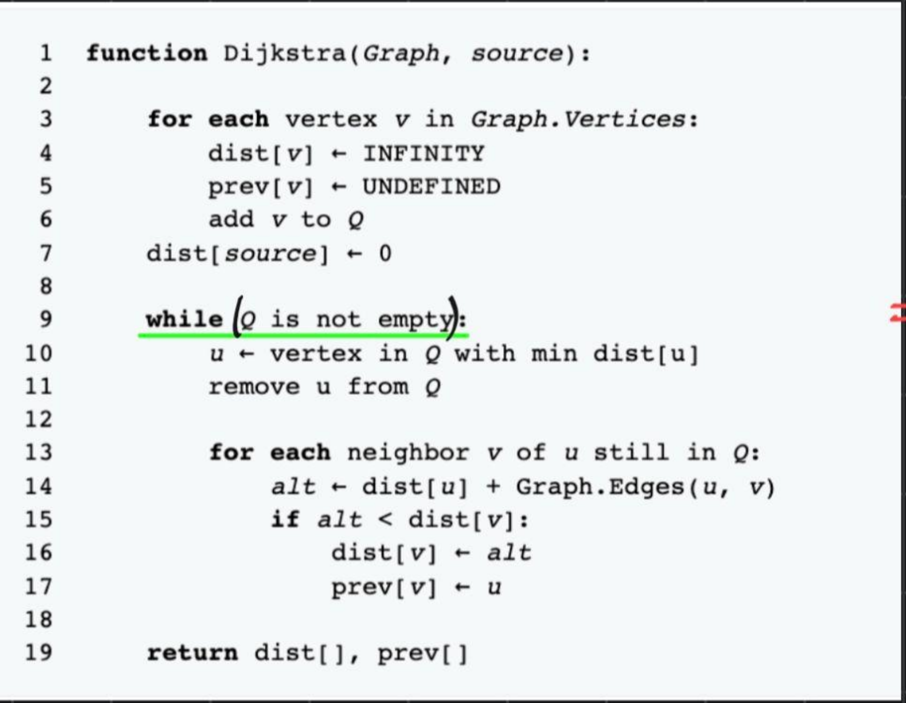
Problem 8



Problem 8

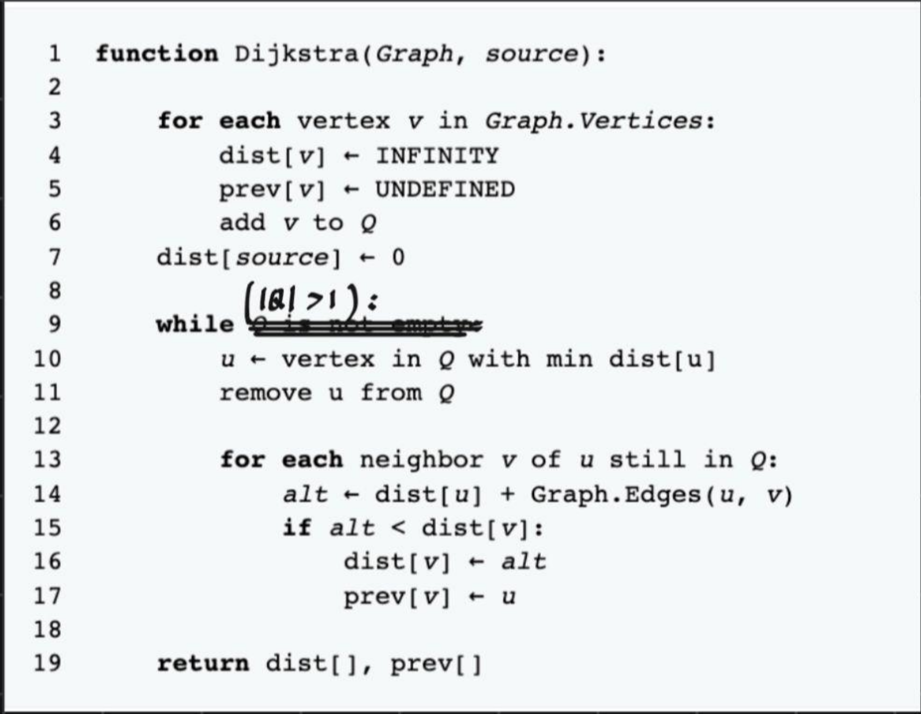
Recall the outer while loop in Dijkstra's Algorithm: *while(Q is nonempty)*. Does the algorithm still output the correct answer if we change this loop to *while($|Q| > 1$)*?

Problem 8



```
function Dijkstra(Graph, source):

    for each vertex v in Graph.Vertices:
        dist[v] ← INFINITY
        prev[v] ← UNDEFINED
        add v to Q
    dist[source] ← 0

    while (Q is not empty):
        u ← vertex in Q with min dist[u]
        remove u from Q

        for each neighbor v of u still in Q:
            alt ← dist[u] + Graph.Edges(u, v)
            if alt < dist[v]:
                dist[v] ← alt
                prev[v] ← u

    return dist[], prev[]
```

$\Rightarrow$

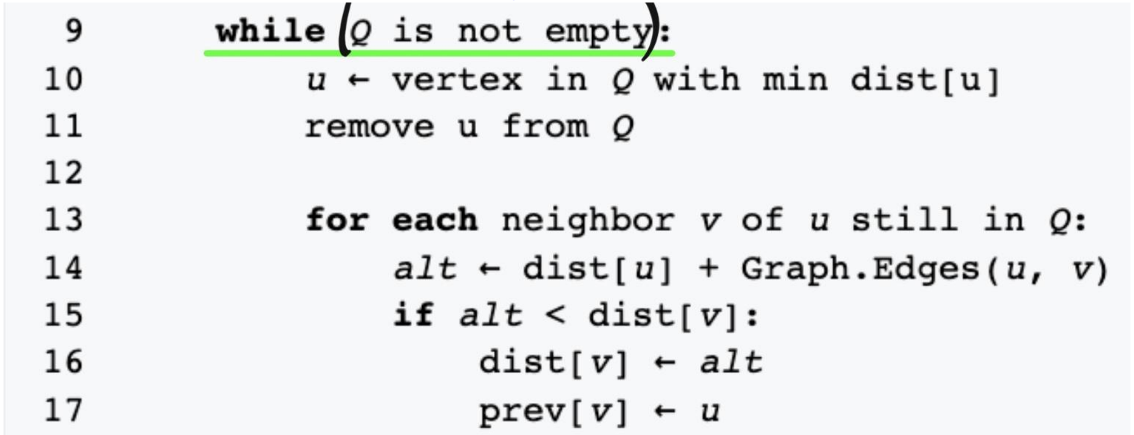


```
function Dijkstra(Graph, source):

    for each vertex v in Graph.Vertices:
        dist[v] ← INFINITY
        prev[v] ← UNDEFINED
        add v to Q
    dist[source] ← 0

    while ~~Q is not empty:~~ (|Q| > 1):
        u ← vertex in Q with min dist[u]
        remove u from Q

        for each neighbor v of u still in Q:
            alt ← dist[u] + Graph.Edges(u, v)
            if alt < dist[v]:
                dist[v] ← alt
                prev[v] ← u

    return dist[], prev[]
```

Problem 8

The difference between the original version and the modified version is that when there is only one vertex inside of Q, it can enter the while loop in the original version but cannot enter the while loop in the modified version.

Let's consider there is only one vertex left in the Q, in the original version,



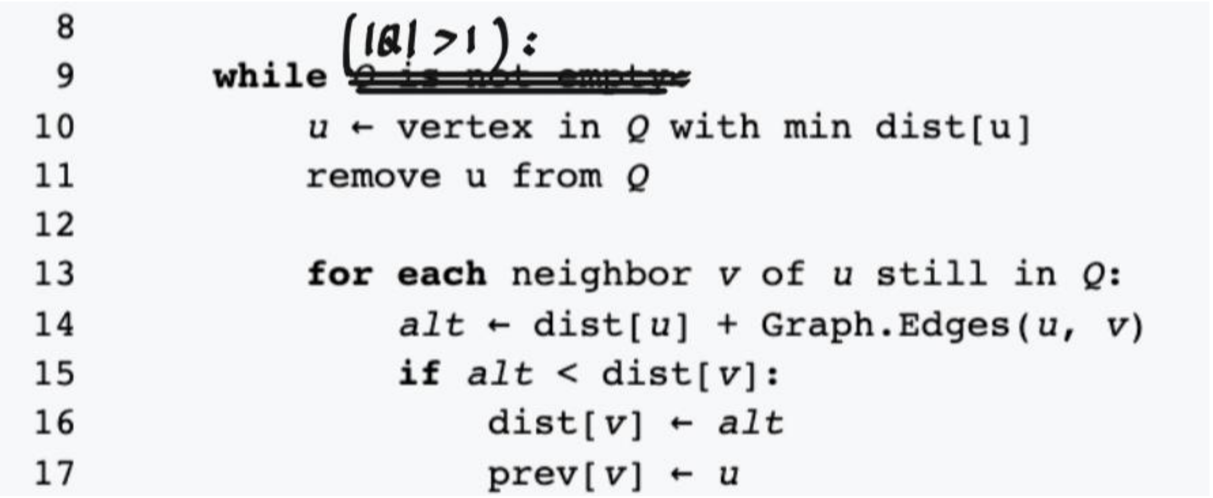
```
 9      while (Q is not empty):
10          u ← vertex in Q with min dist[u]
11          remove u from Q
12
13          for each neighbor v of u still in Q:
14              alt ← dist[u] + Graph.Edges(u, v)
15              if alt < dist[v]:
16                  dist[v] ← alt
17                  prev[v] ← u
```

Line 10 and 11 will be executed, the vertex u will be popped out from Q.

Q is now empty, so it cannot enter line 13, which means dist and prev remains the same as before.

Problem 8

Now let’s consider there is only one vertex left in the Q, in the modified version,



```
    (|Q| > 1):
while ~~Q is not empty:~~
    u ← vertex in Q with min dist[u]
    remove u from Q

    for each neighbor v of u still in Q:
        alt ← dist[u] + Graph.Edges(u, v)
        if alt < dist[v]:
            dist[v] ← alt
            prev[v] ← u
```

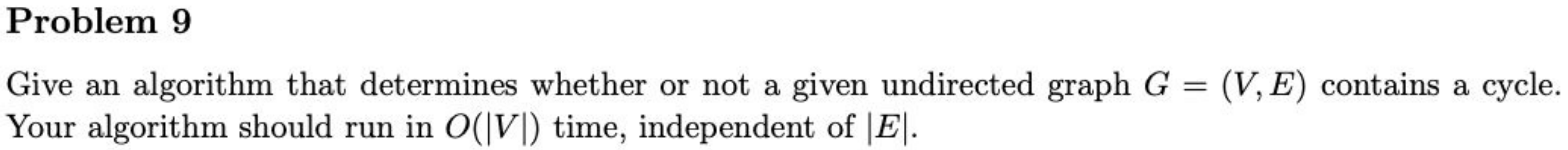
It cannot enter the while loop at all, which means dist and prev remains the same as before.

Problem 8

In conclusion, changing the outer while loop to while(|Q|>1) will still output the correct answer.

The main reason is that the last vertex in the Q has already updated before the last loop and its neighbors have been visited and updated also through previous loops.

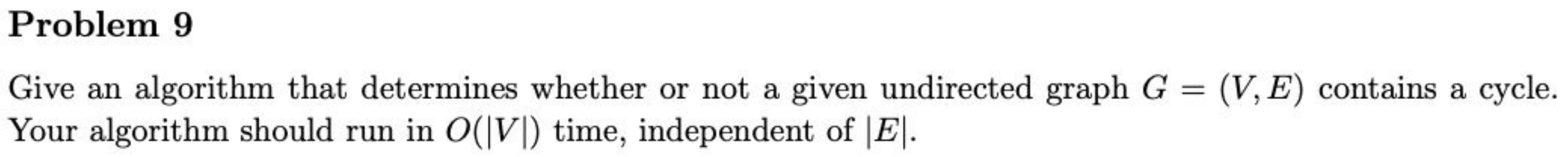
Problem 9



Problem 9

Give an algorithm that determines whether or not a given undirected graph $G = (V, E)$ contains a cycle. Your algorithm should run in $O(|V|)$ time, independent of $|E|$.

Problem 9



Problem 9

Give an algorithm that determines whether or not a given undirected graph $G = (V, E)$ contains a cycle. Your algorithm should run in $O(|V|)$ time, independent of $|E|$.

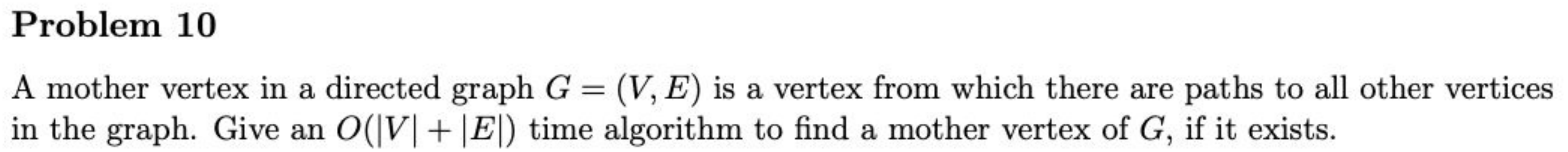
1. If |E| >= |V|, there must exists a cycle.

   The running time is O(1)

2. Otherwise, |E| < |V|, so run DFS to find whether there is a cycle

   The running time is O(|V|+|E|) = O(|V|)

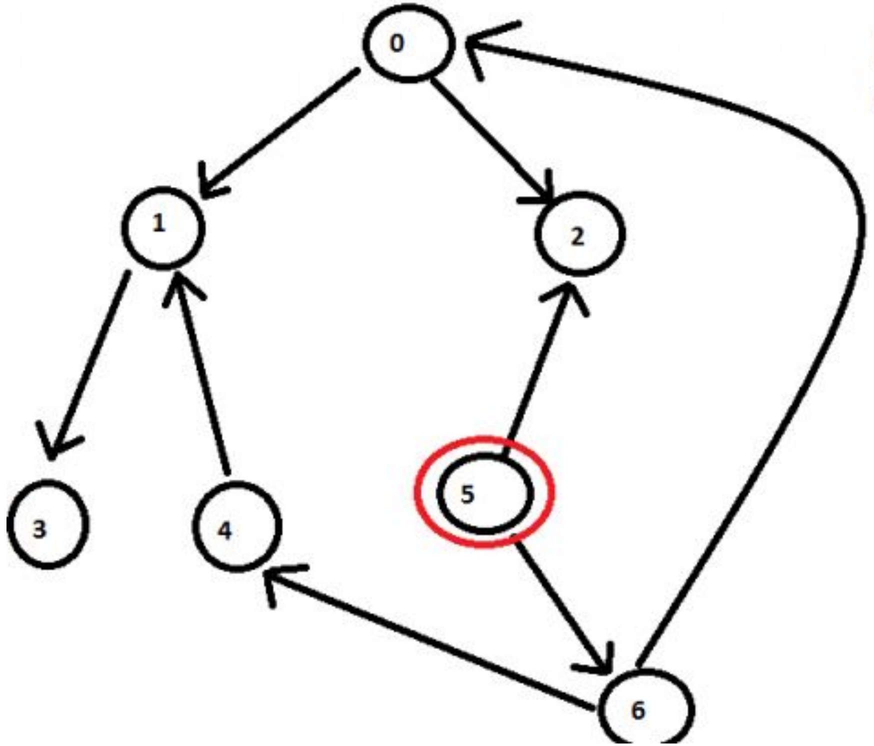
Problem 10



Problem 10

A mother vertex in a directed graph $G = (V, E)$ is a vertex from which there are paths to all other vertices in the graph. Give an $O(|V| + |E|)$ time algorithm to find a mother vertex of G, if it exists.

Problem 10



In this graph the mother vertex is- '5'(circled red) as we can reach any node from - '5' through a directed path

To reach 0-

5->6->0

To reach 1-

5->6->0->1

To reach 2-

5->2

To reach 3-

5->6->0->1->3

To reach 4-

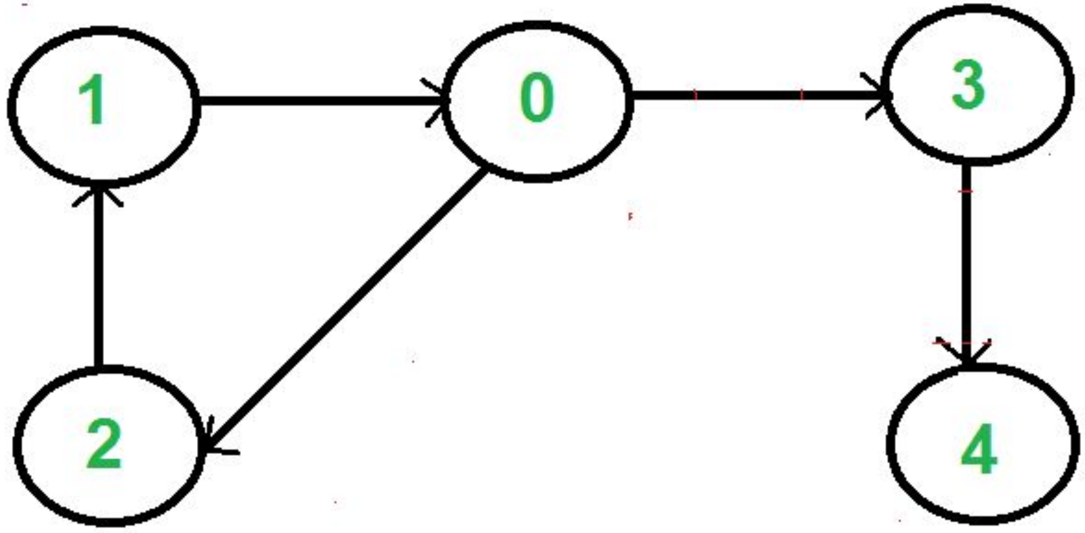
5->6->4

To reach 6-

5->6

Problem 10

There can be more than one mother vertices in a graph. We need to output anyone of them. For example, in the below graph, vertices 0, 1, and 2 are mother vertices.



Mother Vertices : 0, 1 and 2

Problem 10

How to find mother vertex?

Case 1:- **Undirected Connected Graph**: In this case, all the vertices are mother vertices as we can reach to all the other nodes in the graph.

Case 2:- **Undirected Disconnected Graph**: In this case, there is no mother vertices as we cannot reach to all the other nodes in the graph.

Case 3:- **Directed Connected/Disconnected Graph**: In this case, we have to find a vertex -v in the graph such that we can reach to all the other nodes in the graph through a directed path.

Problem 10

A Naive approach :

A trivial approach will be to perform a DFS/BFS on all the vertices and find whether we can reach all the vertices from that vertex. This approach takes O(V*(E+V)) time, which is very inefficient for large graphs.

Can we do better? Can we find a mother vertex in O(V+E) time?

The idea is based on Strongly Connected Component Algorithm. In a graph of strongly connected components, mother vertices are always vertices of source component in component graph. The idea is based on below fact.

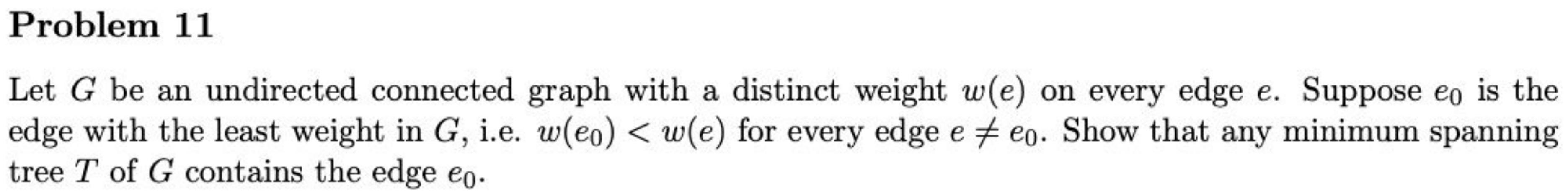
If there exist mother vertex (or vertices), then one of the mother vertices is the last finished vertex in DFS. (Or a mother vertex has the maximum finish time in DFS traversal).

Problem 10

Algorithm(**Directed Connected/Disconnected Graph**):

1. Construct the component graph G^{SCC} of G.
2. Do DFS traversal of the component graph G^{SCC}. While doing traversal keep track of last finished vertex ‘v’. This step takes O(V+E) time.
3. If there exist mother vertex (or vertices), then v must be one (or one of them). Check if v is a mother vertex by doing DFS/BFS from v. This step also takes O(V+E) time.

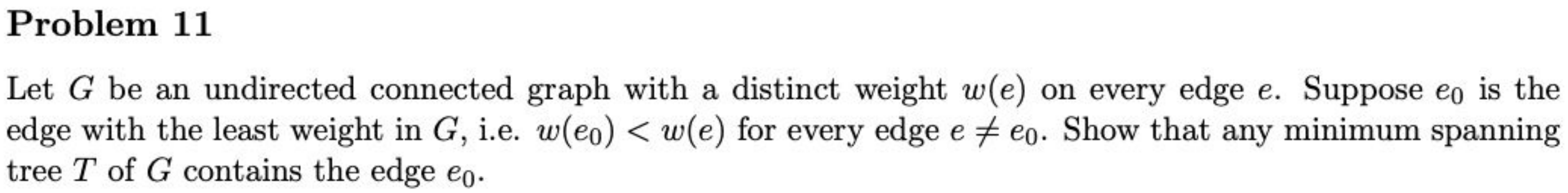
Problem 11



Problem 11

Let G be an undirected connected graph with a distinct weight $w(e)$ on every edge e. Suppose e_0 is the edge with the least weight in G, i.e. $w(e_0) < w(e)$ for every edge $e \neq e_0$. Show that any minimum spanning tree T of G contains the edge e_0.

Problem 11



Problem 11

Let G be an undirected connected graph with a distinct weight $w(e)$ on every edge e. Suppose e_0 is the edge with the least weight in G, i.e. $w(e_0) < w(e)$ for every edge $e \neq e_0$. Show that any minimum spanning tree T of G contains the edge e_0.

If there exists a MST which does not contain edge e0, assume e0 = <u0, v0>.

Then in this MST, there must exists another edge e’ which connects the component containing u0 and that containing v0.

Also, w(e0) < w(e’).

Contradiction to that it is a MST

Thus, replace e’ with e0, it is still a spanning tree, and the tree weight decreases.

Q & A

Thank you