

- ① An edge is called happy if it is on a cycle, and unhappy otherwise.
 Input: An undirected graph G
 Task: Count the number of unhappy edges in G

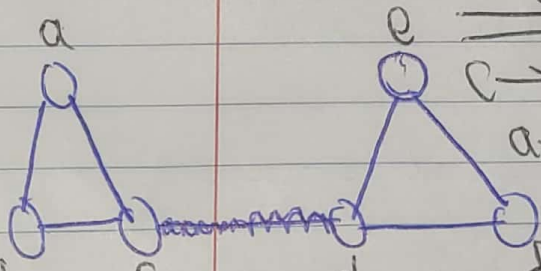
Soln

Set of Steps (Using DFS- depth First Search)

- Step I: Would need to tick/mark all the unvisited nodes
 Step II: Then, I would tick/mark all the edges as been unhappy edges
 Step III: Will run a depth first search and then record all the parent of the nodes visited. Then if the neighbour of a current node is visited and the parent node is not, will need to search and find a cycle in the graph.
 Step IV: Will then mark all the edges located in the cycle as been a happy edge
 Step V: Then, I would count the number of unhappy edges in the G .

Edge \rightarrow happy \rightarrow if it is on a cycle

Formal Proof of correctness


 \Rightarrow Will start the DFS on c. Will first start/visit c \rightarrow a and then mark a as a visited & set parent of a to be c.
 a \rightarrow b would set parent of b to be a; and b \rightarrow c, will find parent of b is a and c is visited.
 c would mark the edges in the cycle as been happy edges. c \rightarrow d, det is another cycle & would then finally count the unhappy edges which is 1.
 Running time: DFS $\rightarrow O(V + E)$; counting the edges $O(E)$
 $O(V + E) + O(E)$
 $\Rightarrow O(V + E)$
 The running overall time is $O(V + E)$
 num — unhappy edge.

2. Let T be a tree. Let a be the minimum weight and b be the maximum weight over all the edge weights in T . Then the strength of T is $(a^2 + b^2)$.

Input: - An undirected weighted graph G .

Task: - Give an efficient algorithm for computing a spanning tree of G that maximizes the strength of the tree.

Soln

Would use MST (maximized strength tree) in resolving the problem strength

— (min weight edge + max weight edge)

Would sort the edges in non-increasing order

Pseudocode: - Markset U for each vertex in G

for each edge (v, w) in sorted order

$M_u \rightarrow$ find the tree of u

$M_w \rightarrow$ find the tree of w

if $M_u \neq M_w$

Merge (M_u, M_w)

Take (v, w) as Maximized strength tree

Proof of Correctness

Will assume that a maximum strength tree will not include a maximum weight of the tree. Would let a be minimum weight and b be the maximum. The maximum weight strength of the tree is $d^2 + e^2 \Rightarrow a^2 + b^2 > d^2 + e^2$. e is less than b because the maximum weight edge is b and a is greater than d because the maximized strength tree takes the possible maximum edge in the sorted edges every time.

$\Rightarrow a^2 + b^2 > d^2 + e^2$, then the algorithm is correct

Running time: - The sets take $O(|V|)$ and the sorting edges $O(|E| \lg |E|)$. The loop is $O(|E|)$ and the merging is $O(\lg |V|)$

Total Time $\Rightarrow O(|E| \lg |E|) + O(|E|) + O(\lg |V|) = O(|E| \lg |V|)$

3.

Input: A directed graph G with n vertices
Task: If one can add at most one directed edge to G such that every vertex can reach all other vertices, print 'YES'. Otherwise, print 'Unsatisfiable'.

Soln

Pseudocode

Tarjan DFS(u)

Would modify Tarjan DFS(u) by adding a counter

$V.start \leftarrow clock + 1$

$V.low \leftarrow V.start$

push v on stack

for each neigh w of u

if w is unvisited

Tarjan DFS(w)

$V.low = \min(V.low, w.low)$

if $V.start = V.low$

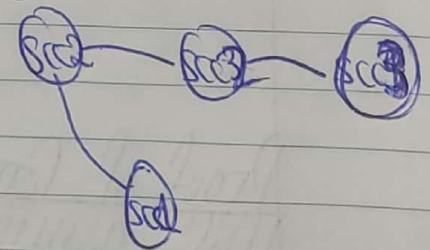
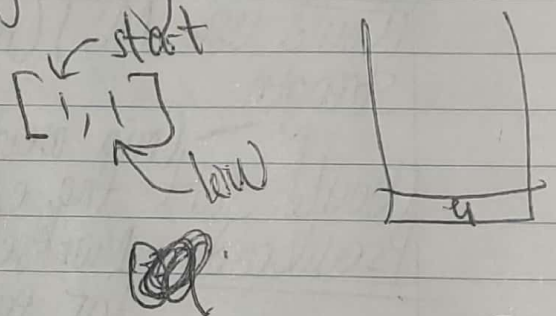
Remove everything about u (including v) from stack and report as an SCC

if counter < 3 , print YES

else

print 'Unsatisfiable'

Time $= O(V + E)$



Formal Proof of Correctness:

Tarjan DFS \rightarrow SCC in graph. If the graph is made of one SCC, then every vertex can reach all vertices without adding directed edge. But, if the graph has two SCC would add one directed edge to G then. But, if the SCC exceeds two, will then add more edges.

Running Time: The Tarjan DFS $(O(V + E))$ and the SCC is $(O(V))$. So therefore, the total is $O(V + E)$.

④ Let G be an undirected-edge weighted graph with V vertices and $O(V^2)$ edges. Describe an efficient algorithm to compute a minimum spanning tree that runs in $O(V^3)$ time. You are only allowed to use data structures such as arrays and priority queues. Do not use complex data structures such as Fibonacci heaps.

Soln

Using the set of steps

Step 1: - Would firstly create a list to keep track of all the vertices in the minimum spanning tree.

Step 2: - Would then initialize the value to be 0 and assign the first vertex to be 1.

Step 3: - Would use for loop to add vertices into the list. And if the list is not complete would need to select a vertex that has a minimum value and then add it to list.

Step 4: - Using a for loop, I would update all the values of vertices that are adjacent to the current vertex v . For v adjacent to vertex u if the weight of $v-u$ is $<$ the previous value of u . Would update the value weight of $v-u$.

Formal Proof of Correctness: - It has two set of vertices. The first one contains Minimum spanning tree and the second one contains vertices that are not minimum spanning tree. Adding the first vertex would form a MST.

Running Time: - The loop takes $O(V^2)$
 \Rightarrow The running time overall is $O(V^3)$

5.

Input: A set of n movies. A positive integer K . A positive integer q . A function f takes two movies X, Y and gives a similarity score $f(X, Y)$ in constant time.

Task: - Design an efficient algorithm that partitions the movies into exactly K disjoint sets such that movies from different sets have a similarity score at most q . If such groups cannot be made, then print "Unsatisfiable".

Soln

For example n movies $K=3$ $q=9$ Input
 $n=4$ \downarrow how many groups need to have \downarrow threshold

I would use a graph to find the highest score and then would find the similarity of both numbers. I use the edges in grouping it, would group the weighted edges according to the highest.

Steps: -

Step I: I would start by grouping the edges according to the weighted highest score.

Step II: I would use a for loop to iterate over the set of n . I would use two for loops.

Step III: Would use the second for loop to compare/check the current element n of my set with the other sets.

Step IV: - Then after that I would use a function to check the element currently in the set and also the other sets of remaining elements.

Step V: I would check if the function of the current next element is less than my q for the elements in n ; I would then return the set of the current set and the set of the remaining movies.

Step VI: I would also repeat this for the second set; and until the set is equal to K and it is not equal to K I would then print Unsatisfiable.

Formal Proof of Correctness:-

Have a set of n movies. I would assume all the movies have a good similarity and is less than q . So if it is 1, it would check the movies with the others. For example, movie 1, movie 3, movie 13, movie 1, movie 23, etc. provide $K=2$, then the algorithm would be correct.

And if the set of n movies would assume all the movies have a good similarity and is greater than q . Would check the first movie with the remaining ones; then it would be unsatisfiable.

Time Running:-

Having two for loop, the overall time complexity time is $O(n^2)$.