Q.1 Show how to find the maximum spanning tree of a graph, which is the spanning tree of the largest total weight.

Answer:

def findMaxMST(G):

out = []

edges = sort the edges in decreasing order of weight

out.append(e1) // out will store all the edges with max weight in MST

while(out has n-1 edges):

if(edge.empty()):

print(“graph is disconnected”)

else:

e = edge.remove()

if(e does not form a cycle with edges in out):

out.append(e)

return out;

Q.2 (20 pts) Consider an undirected graph G = (V, E) with nonnegative edge weights we. Suppose that you have computed a minimum spanning tree of G, and that you have also computed shortest paths to all nodes from a particular node s. Now suppose each edge weight is increased by 1; the new weights are w’e = we + 1.

(a) Does the minimum spanning tree change? Give an example where it changes or prove it cannot change.

Answer:

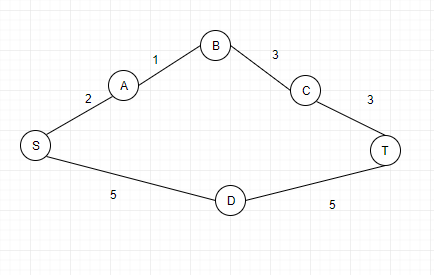
No. The MST would not change. In Kruskal’s algorithm to find MST, we sort all the edges by increasing order and build the graph with considering each edge with increasing order of weight. Since the weights are increased for each edge with the value of 1, the sorting order remains same and therefore, the MST will not change.

(b) Do the shortest paths change? Give an example where they change or prove they cannot change.

Answer:

Yes the shortest path may change. The length of the path depends on the number of edges on the path so with the increase or decrease in the weight of the edges shortest path may change

Example –



Here the shortest path from S to T is now is S-A-B-C-T.

After adding 1 to each edge, it now changes to S-D-T

Q. 3 Either prove that the output of the algorithm is a minimum-spanning tree or give a counter example when the algorithm does not output a minimum-spanning tree.

Answer:

The output of the given algorithm is a minimum spanning tree. The algorithm is parsing through the graph and removing the edges with max weights keeping the graph connected. So if a edge with high weight is removed and there is still path existing between the 2 vertices of the edge then this edge can be removed. But if the removal of some edge makes the graph disconnected, that node is important and cannot be removed.

Q.4 The diameter of a tree T=(V, E) is defined as the largest length of all shortest path (i.e., the path with the smallest number of edges) between pairs of vertices. Devise a O(|V| + |E|) algorithm to compute a diameter for a given tree.

Answer:

We need to find the longest path in a tree so, initially we need to know the 2 extreme ends and then the path between them can be the longest length path.

1. Start with any random node and run DFS to find the node with the longest depth which is one of the extreme point of the diameter.
2. From the farthest node, run again DFS to find the second extreme point of graph. This path is the diameter.

def DFS(Node root):

maxLength = 0

Node outNode;

fringe.push(root)

length = 0

while(!fringe.isempty()):

node = fringe.pop()

if node.left!=null:

fringe.push(node.left)

else if node.right!=null:

fringe.push(node.right)

if node.left==null and node.right==null:

fringe.push(node.left)

if(maxLength < length):

maxLength = length

outNode = node

length = 0

Q. 5 (20 pts) Let G be an arbitrary connected, undirected graph with a distinct positive weight on each edge. Let e=(u,v) and e’= (x, y) are two arbitrary edges connecting four distinct vertices u, v, x and y. Devise a polynomial time algorithm to find a cycle in G with the lowest total weight that contains both e and e’. What is the running time of your algorithm?

Answer:

To find the lowest total weight containing 2 edges, first remove the 2 edges and run the shortest path algorithm between the opposite vertices of 2 edges.

For example,

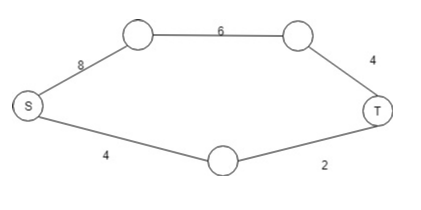
1. Remove edges e and e’
2. Find the shortest path from x to u and y to v
3. Find the shortest path from x to v and y to u
4. Consider the path with min weight and assign it to weight wpath
5. Add the edge weights of e and e’ to wpath. This gives the actual weight of shortest cycle of a graph having edges e and e’.

Time complexity: O((V+E))

Q. 6 (20 pts) Decide whether you think each of the following statements are true or false. Justify your answers.

a. Let G be an arbitrary flow network with a source s, a sink t and a positive integer capacity c(e) on each edge. If f is a maximum s-t flow in G, then f saturates every edge out of s with flow, i.e., for all edges e out of s, we have f(e)=c(e).

Answer:



No.

Consider a graph with nodes S, A, B, C and T

Edge and weights are:

S -> A has weight 8

A -> B has weight 6

S -> C has weight 4

C -> T has weight 2

B -> T has weight 4

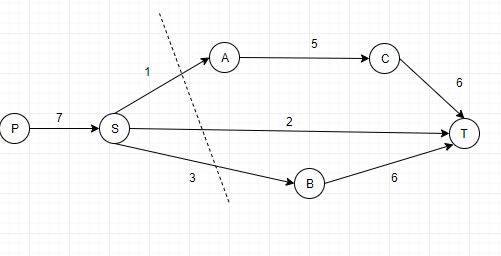
Then the maximum flow is 6 but the nodes from S are not saturated. Hence the given statement is false.

b. Let G be an arbitrary flow network with a source s, a sink t and a positive integer capacity c(e) on each edge. Let (A, B) be a minimum s-t cut w.r.t. these capacities. Now suppose we add 1 to every capacity, then (A,B) is still a minimum s-t cut w.r.t. these new capacities, i.e., c’(e)=c(e)+1 for every edge e.

Answer:

After adding the edge with weight 1, the min cut might change.

Consider a graph below from P to T flow graph.

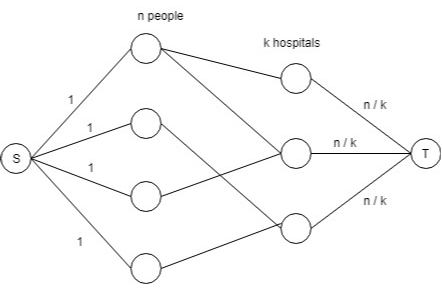


From P to T with the initial capacity as given, maximum flow is 6 and minimum cut is as described in the figure that is SA ST and SB. But after increasing each edge weight by 1, the minimum cut becomes PS.

Hence, the minimum cut can change after changing the weights of edges.

Q.7

Answer:



Consider a directed flow graph with source and a sink node, and connect the source node to all the patients. There are n patients and k hospitals. Each hospital can occupy n/k patients.

Since there are n patients, we will assign 1 value for each edge from source to people. There are n such people nodes. We will assign n/k weight for each edge from k hospitals to terminal node.

We can then use Edmonds-Karp algorithm to evaluate the maximum flow. If the maximum flow is equal to the number of patients then it is possible to allocate all the patients with the given [n/k] condition, else it is not. The runtime of this algorithm will be polynomial as: Runtime to build graph: O(nk) {n is number of patients and k is number of hospitals} Runtime of Edmonds-Karp algorithm : O(v\*e^2) {v is number of vertices and e is number of edges}

Q.8

Here’s a problem that occurs in automatic program analysis. For a set of variables x1, …, xn, you are given some equality constraints, of the form “xi=xj” and some disequality constraints, of the form “xi¹xj”. Is it possible to satisfy all of them? For instance, the constraints x1=x2, x2=x3, x3=x4, and x1¹x4 cannot be satisfied. Devise an efficient algorithm that takes m constraints over n variables and decides whether the constraints can be satisfied.

Answer:

This can be solved with the approach of BFS along with bipartite sets. Say we have set1 and set2 where set 1 ahas all elements with equal values and set 2 has all elements with value not equal to any variable from set 1.

Let all the constraints are denoted by the vertices as the variables and edges as the weight with value 1 for equality and -1 for inequality.

Starting with one node and applying BFS, we will check all the neighbors and assign same set for equal variables and opposite sets for unequal variables. If some variable has already assigned a set which is not following the constraint then return false else return True

Example:

For example in question,

We will start with node x1 and assign it to set1 and it to queue.

While queue not empty:

Take the element in queue which is x1, and check all its neighbors.

Since x1=x2 and x2.set = null, x2.set = x1.set that is both x1 and x2 are assigned to set 1

Then x1 != x4 and x4.set ==null so, x4 is assigned to set opposite to x1 that is set2.

There are no more neighbors of x1 so start with next variable in queue that is x2.

X2 has neighbor x3 with equality constraint and x3 has not assigned any set. So, set of x3 will be same as x2. X2 has no more neighbors so we will move with next element in queue that is x3.

X3 has only one neighbor x3=x4 but x4 is assigned to set2 which is not same as the set of x3. Hence the constraints are violated.

Def detectVioltion():  
 startNode.set = 1

Fringe.append(startNode) // queue

While(!Fringe.isempty()):

Node = fringe.remove(0)

For s in node.neighbors():

set = node.set

If s.set == null:

If edgeWeight(s,node) == 1:

s.set = node.set

else:

s.set = -1\*node.set

else:

If edgeWeight(s,node) == 1 and s.set != node.set:

Return False

Else If edgeWeight(s,node) == -1 and s.set == node.set:

Return False

Fringe.append(s)

Return True