Homework 4

Name: Jiyang Tang

NetID: jt304

# Written Part

## 22-2

a

Suppose the root of ​ is an articulation point. Then the removal of from would cause the graph to disconnect, so has at least children in .

If has only one child in ​​ then it must be the case that there is a path from to each of ’s other children. Since removing disconnects the graph, there must exist vertices and such that the only paths from to contain .

To reach from , the path must first reach one of ’s children. This child is connected to via a path which does not contain .

To reach , the path must also leave through one of its children, which is also reachable by . This implies that there is a path from to which does not contain , so it is a contradiction.

Suppose has at least two children and in . Then there is no path from to in which does not go through , since otherwise would be an ancestor of . Thus, removing disconnects the component containing and the component containing , so is an articulation point.

In conclusion, the root of is an articulation point of G if and only if it has at least two children in

b

Suppose that is a non-root vertex of and that has a child such that neither nor any of ’s descendants have back edges to a proper ancestor of . Let be an ancestor of , and remove from . Since the graph is undirected, the only edges in the graph are tree edges or back edges, which means that every edge incident with takes us to a descendant of , and no descendants have back edges, so at no point can we move up the tree by taking edges. Therefore is unreachable from , so the graph is disconnected and is an articulation point.

Suppose that for every child of there exists a descendant of that child which has a back edge to a proper ancestor of . Remove from . Every subtree of is a connected component. Within a given subtree, find the vertex which has a back edge to a proper ancestor of . Since the set of vertices which are not descendants of form a connected component, we have that every subtree of is connected to . Thus, the graph remains connected after the deletion of so is not an articulation point.

c

Since is discovered before all of its descendants, the only back edges which could affect are ones which go from a descendant of to a proper ancestor of . If we know for every child of , then we can compute easily since all the information is coded in its descendants.

Thus, the algorithm can be written as:

If is a leaf in ​ then is the minimum of and where is a back edge. If is not a leaf, is the minimum of , where is a back edge, and , where is a child of . Computing for a vertex is linear in its degree. The sum of the vertices' degrees gives twice the number of edges, so the total runtime is .

d

First we can apply the algorithm of part (c) in to compute for all . If if and only if no descendant of has a back edge to a proper ancestor of , if and only if is not an articulation point.

Thus, we need only check versus to decide in constant time whether is an articulation point, so the runtime is .

e

An edge lies on a simple cycle if and only if there exists at least one path from to which doesn't contain the edge , if and only if removing doesn't disconnect the graph, if and only if is not a bridge.

f

A edge lies on a simple cycle in an undirected graph if and only if either both of its endpoints are articulation points, or one of its endpoints is an articulation point and the other is a vertex of degree . There is also a special case where there is only one edge whose incident vertices are both degree . We can check this case in constant time. Since we can compute all articulation points in and we can decide whether or not a vertex has degree in constant time, we can run the algorithm in (d) and then decide whether each edge is a bridge in constant time, so we can find all bridges in time.

g

Obviously, every nonbridge edge is in some biconnected component, so we need to show that if and ​ are distinct biconnected components, then they contain no common edges.

Suppose to the contrary that is in both ​ and ​. And let be any edge in and be any edge in .

Then lies on a simple cycle with , consisting of the path

Similarly, lies on a simple cycle with consisting of the path

This means

is a simple cycle containing and . Therefore, it’s a contradiction.

Therefore, the biconnected components form a partition.

h

ALGORITHM label(G, u, k)  
 u.color = GRAY  
 for v ∈ G.neighbors[u]  
 (u, v).bcc = k  
 if v.color == WHITE  
 label(G, v, k)

## 23-3

a

Suppose that is a minimum spanning tree. Suppose there is some edge in it that has a weight that is greater than the weight of the bottleneck spanning tree. Then, let ​ be the subset of vertices of that are reachable from in , without going though . Define ​ symmetrically. Then, consider the cut that separates ​ from ​. The only edge that we could add across this cut is the one of minimum weight, so we know that there is no edge across this cut of weight less than .

However, we have that there is a bottleneck spanning tree with less than that weight. This is a contradiction because a bottleneck spanning tree, since it is a spanning tree, must have an edge across this cut.

b

First, we process the entire graph, and remove any edges that have weight greater than . If the remaining graph is connected, we can just arbitrarily select any tree in it, and it will be a bottleneck spanning tree of weight at most . Testing connectivity of a graph can be done in linear time by running a breadth first search and then making sure that no vertices remain white at the end.

c

First we can find the median of this list of numbers in time . Then, run the procedure from part b with this median value as input. Then there are two cases:

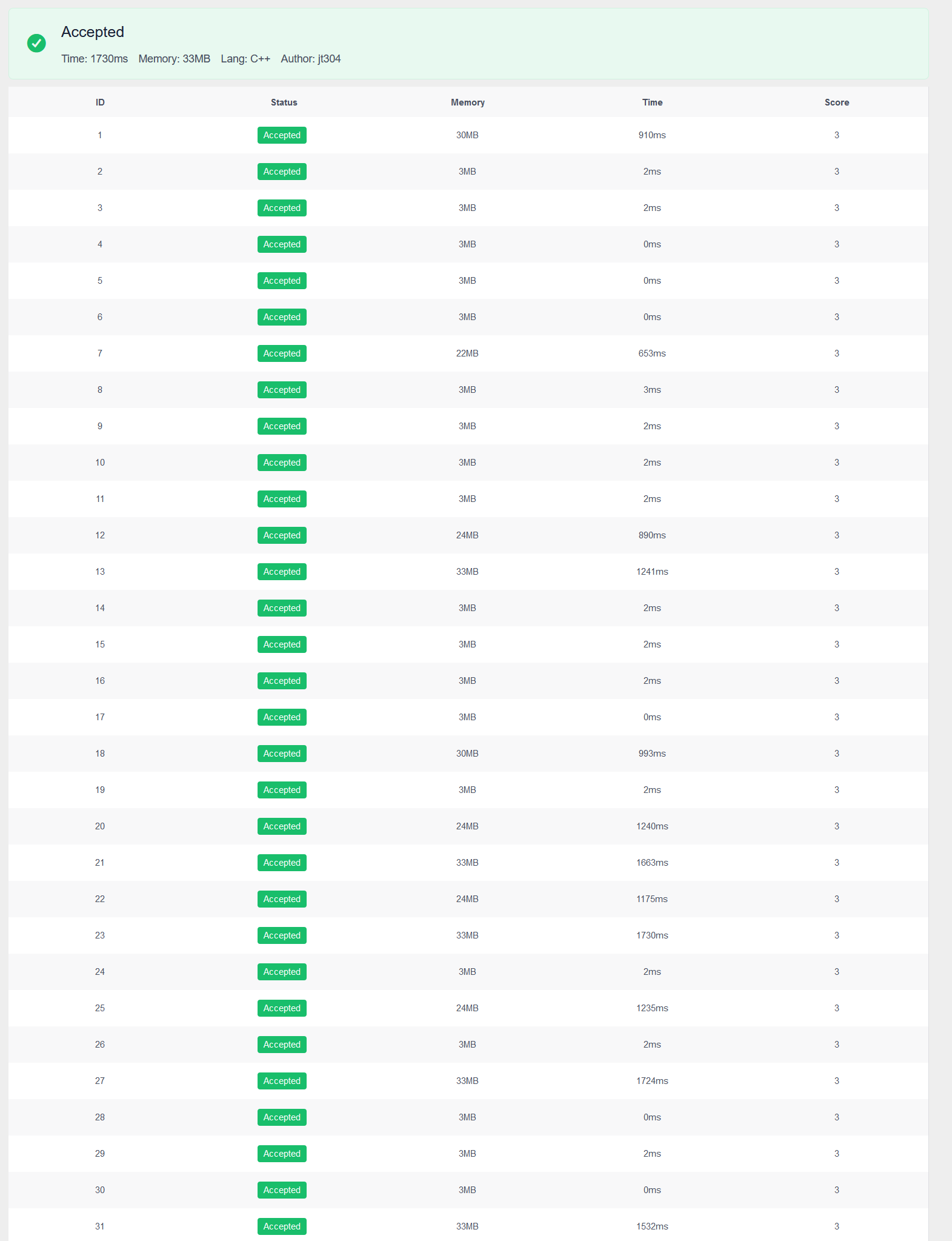
First, we could have that there is a bottleneck spanning tree with weight at most this median. Then just throw away the edges with weight more than the median and repeat the procedure on this new graph with half the edges.

Second, we could have that there is no bottleneck spanning tree with at most that weight. Then, we should run a procedure to contract all the edges that have weight at most the weight of the median. This takes time and then we are left solving the problem on a graph that now has half the edges.

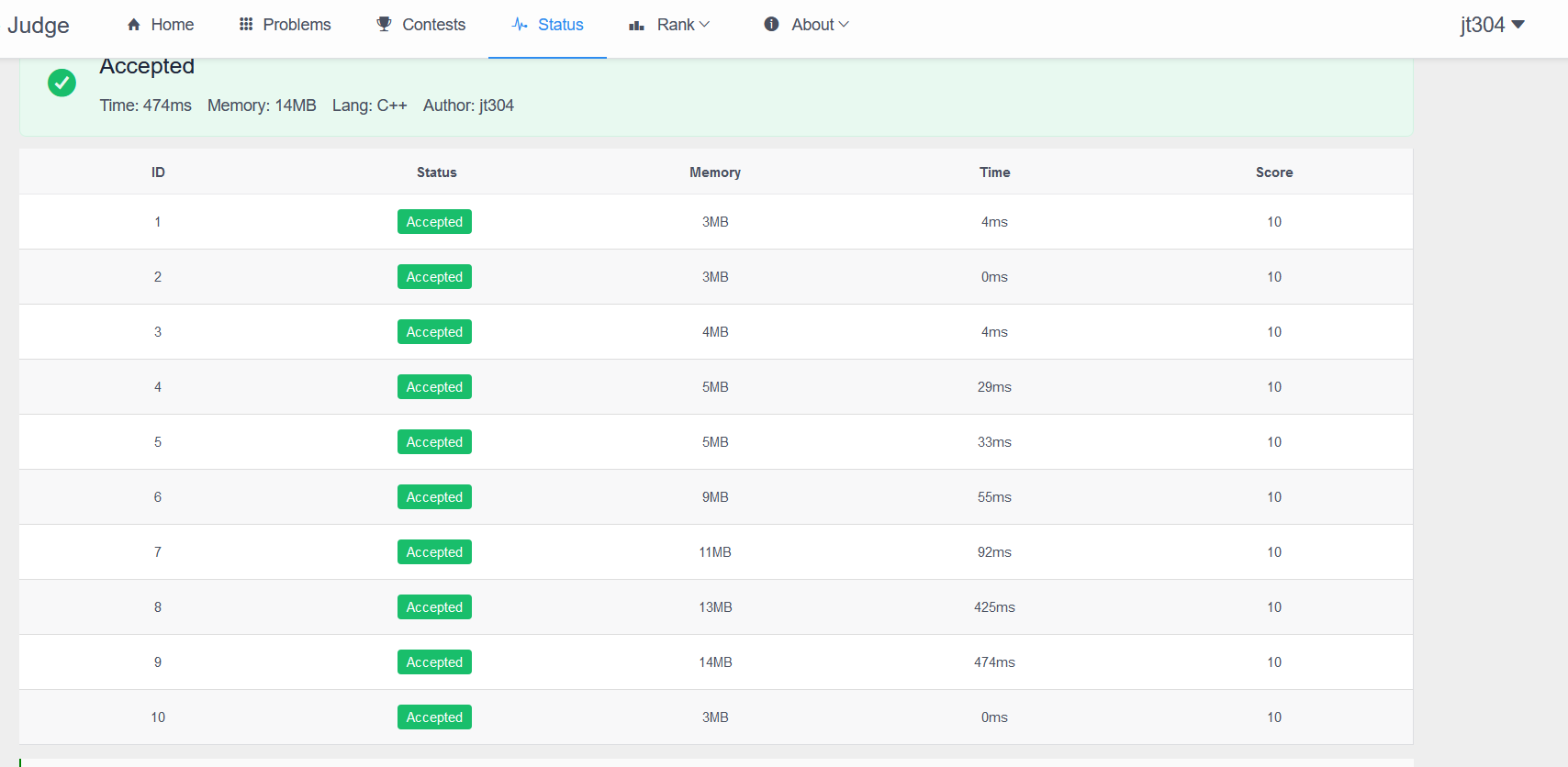
Observe that both cases are and each recursion reduces the problem size into half. The solution to this recurrence is therefore linear.

# DKUOJ

Problem 14



Problem 17



Problem 20

