1. Explain what’s wrong with the Fibonacci heap in the figure below. (NOTE: Gray nodes represent marked nodes, and the min pointer has been omitted for simplicity. Children of each node are intentionally drawn as undirected edges, although they would typically be implemented as a circular linked list with a pointer from the parent.)

**ANSWER:** As proven in class, if a node in a Fibonacci heap has \( k \) children (for \( k > 0 \)), then one of its children must have at least \( k - 2 \) children. (This is because the last of its children to be added must originally have had a rank of at least \( k - 1 \) in order to be linked to a node that already had \( k - 1 \) children. Since then, it could have lost at most 1 additional child, leaving a total of \( k - 2 \) children.) The “0” node in the root list of the Fibonacci heap has 4 children, but has no child with 2 or more children. If additional children were deleted from the 1, 4, or 8 node, then that node should have been moved to the root list via a cascading cut.

2. Explain what’s wrong with the link/cut tree in the figure below. (This figure represents the “actual tree,” not the virtual tree used to implement the actual tree.)

**ANSWER:** The “a” node has two incoming solid edges. In a link/cut tree, each node may have at most one incoming solid edge – all others must be dashed.
3. Consider the flow problem below, with source $s$, sink $t$, and edge capacities as shown.

(a) Find a maximum flow. Is it unique?
(b) Find a minimum cut. Is it unique?
(c) Explain why Dinic’s algorithm is guaranteed to terminate in one iteration (that is, after finding a single blocking flow).

\begin{center}
\begin{tikzpicture}[node distance = 2cm, line width=1pt]
  \node (s) at (0,0) {$s$};
  \node (b) at (-1,-1) {$b$};
  \node (e) at (0,-1) {$e$};
  \node (d) at (1,-1) {$d$};
  \node (h) at (0,-2) {$h$};
  \node (t) at (0,-3) {$t$};

  \path[->] (s) edge node[above] {4} (b);
  \path[->] (s) edge node[above] {5} (d);
  \path[->] (b) edge node[right] {6} (e);
  \path[->] (b) edge node[right] {7} (h);
  \path[->] (d) edge node[above] {1} (t);
  \path[->] (e) edge node[above] {3} (t);
  \path[->] (h) edge node[above] {3} (t);
\end{tikzpicture}
\end{center}

**ANSWER:**

(a) A maximum flow is 3 units along the $s$-$b$-$h$-$t$ path and 3 units along the $s$-$d$-$e$-$t$ path. (Specifically: $f(s,b) = 3$, $f(b,h) = 3$, $f(h,t) = 3$; $f(s,d) = 3$, $f(d,e) = 3$, $f(e,t) = 3$.) This maximum flow is not unique. We can construct alternate maximum flows by sending more flow along the $b$-$e$ and $d$-$h$ edges and less along the $b$-$h$ and $d$-$e$ edges.

(b) A minimum cut is $X = \{s, b, d, e, h\}$ and $\overline{X} = \{t\}$. (The edges across the cut are $e$-$t$ and $h$-$t$, with a total capacity of 6.) This minimum cut is unique.

(c) Dinic’s algorithm will terminate in one iteration because all paths from $s$ to $t$ are shortest paths. In the operation of Dinic’s algorithm, the length of the shortest path from $s$ to $t$ increases by one or more in each iteration until no paths of that length exist. Since no paths of length more than 3 exist, Dinic’s algorithm will terminate after adding a blocking flow from paths of length 3.

4. Using the potential function from class, $\Phi = \sum_x r(x) = \sum_x \lceil \lg(\text{size}(x)) \rceil$, prove that the potential of a splay tree with $n$ nodes must always be less than or equal to $\lg(n!)$, where $n! = n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1$. (Hint: Recall that $\lg(ab) = \lg(a) + \lg(b)$.)

**ANSWER:** Consider a chain of nodes. The root has size $n$; its child has size $n-1$; and so on, until reaching the leaf with size 1. The total potential of this is $\lg(n) + \lg(n-1) + \ldots + \lg(1) = \lg(n!)$. To prove that this is indeed a maximum, consider a total ordering of the nodes, such that node $n_i$ can only have nodes $n_j$ as its descendants when $j \geq i$. (We can construct such an ordering for any tree by taking the topological partial ordering and arbitrarily extending it into a total ordering.) The first node in the ordering can have size at most $n$ (since all nodes
can be its descendants), the second can have size at most \(n - 1\) (since we must exclude the first node), the third can have size at most \(n - 2\), and so on. Since \(\log\) is a monotonic function, we obtain the maximum potential by setting each node to its maximum possible size. Since such an ordering exists for any tree, any tree must have \(\log(n!)\) as its maximum potential.

5. Describe an algorithm for converting all solid edges into dashed edges in a link/cut tree, using only expose operations. (In other words, do not directly call the link or cut operations.)

**ANSWER:** Call expose on each node in turn, in a bottom-up manner starting from the leaves. The effect of each expose is to remove all incoming solid edges and make them dashed. Expose only adds solid edges between the node and the root, so if each node is only exposed before its ancestors, then those solid edges will be removed when its ancestors are exposed later.

The following pseudocode illustrates how this can be done by a simple recursive function:

```pseudo
make-dashed(n):
    for each child c of n:
        make-dashed(c)
    expose(n)
```

Calling `make-dashed` on the root will thus convert all solid edges to dashed (and in \(O(n)\) calls to expose).

6. Describe how to implement the function `add-single-cost(v, c)`, which adds \(c\) to the cost of a single vertex \(v\) in a link/cut tree and runs in amortized \(O(\log(n))\) time. (Note that this is different from `addcost(v, c)`, which adds the cost \(c\) to every node from \(v\) to the root.) You may use path operations and other tree operations as subroutines.

**ANSWER:** This can be done by the following pseudocode:

```pseudo
p = successor(v)
cut(v)
addcost(v, c)
link(v, p)
```

By cutting \(v\), it becomes the root of its own tree. `addcost` then can be used to add the cost \(c\) to all nodes from \(v\) to the root, which in this case is only \(v\). Finally, we relink \(v\) to the rest of the tree, which we saved before cutting \(v\).

7. Let \(f\) be a maximum flow on \(G\), and let \((u, v)\) be an edge saturated by \(f\). Show by counterexample that increasing \(\text{cap}(u, v)\) does not necessarily increase the value of a maximum flow on \(G\). In other words, show that increasing the capacity of a single saturated edge may not increase the amount of flow that can be pushed through the flow graph.

**ANSWER:** Consider the flow graph \(s \rightarrow v \rightarrow t\), where the capacities and flow values are all 1. Both edges are clearly saturated. However, increasing the capacity of \((s, v)\) to 2 does not increase the maximum possible flow, since the edge \((v, t)\) is still saturated and \(X = \{s, v\}\) remains a minimum cut with capacity 1.
8. Let $f$ be a maximum flow on $G$, and let $(u, v)$ be an edge not saturated by $f$. Use the max-flow/min-cut theorem to prove that increasing $\text{cap}(u, v)$ never increases the value of a maximum flow on $G$.

**Answer:** By the max-flow/min-cut theorem, if $f$ is a maximum flow, then its value $|f|$ is equal to the capacity of some (minimum) cut, $\text{cap}(X, \overline{X})$. This means that every edge from $v \in X$ to $w \in \overline{X}$ must be saturated. Increasing the capacity of some unsaturated edge will therefore have no effect on any edge in this cut. Since the capacity of the minimum cut is unchanged, the maximum flow value must also be unchanged, since the two are equal.