CSCI 510 Models of Computation

HW5: Livin' on a Prayer

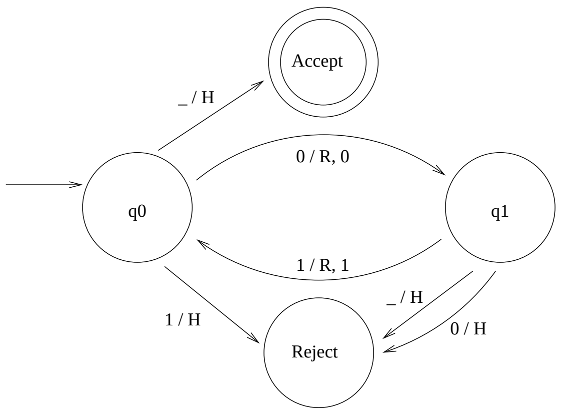
Solution Key

Include your full name*, CS login, and the problem number(s) on each piece of paper you hand in, and please staple your pages together before handing in.*

While collaboration is encouraged in this class, please remember not to take away notes from collaboration sessions other than your scheduled lab section.

Problem 1

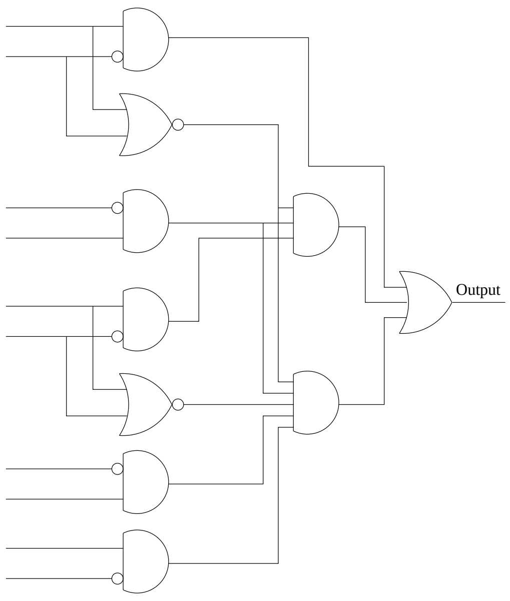
In this problem you will construct a circuit to compute the same function as a Turing Machine with a fixed-length tape. Consider the TM whose tape has 5 spaces and whose control unit is as follows:



Note that if the TM tries to move right off the end of the tape, it instead stays where it is (and, in this case, will likely reject soon after).

Construct a circuit that returns 1 *whenever the above TM would accept and* 0 *whenever it would reject. The circuit will always take input from 5 tape cells regardless of whether or not they are blank. For this problem each cell on the tape is represented by 2 bits of input to the circuit:* `00` *for* 0, `01` *for* 1, *and* `10` *for blank (*`11` *will never appear).*

The following circuit simulates the turing machine. Note that the first input is on top and each input is listed with the most significant bit on top.



The following questions are lab problems. *Please remember to prepare a solution for your assigned problem before going to your lab section.*

Note: *We will not ask you to reduce from an NP-complete problem you have not seen without telling you what to reduce from!*

Lab Problem 1

A riot has broken out at the a rock concert! Dream Theater played Panic Attack and people started panicking. Everyone started yelling at a bunch of people, only making those other people more panicked. The real problem occurred when they got into loops: one person screaming eventually caused someone else to scream at the original person. Eventually the noise became deafening and the band stopped playing. People who weren't being yelled at were able to calm down, but the loops just got worse.

Thankfully, the police arrived on the scene to stop the commotion. However, they only brought a few squad cars, and they only have enough space to take k people with them. They need to determine if they can find k people such that, if they take those people to the station, there won't be any loops and everyone will be able to calm down.

To model the problem, they represent the crowd as a **directed** *graph. Each* *node represents a person, and an edge from x_1 to x_2 means x_1 is yelling at x_2. A feedback loop is represented by a* cycle *in the graph. If the police can eliminate all cycles in the graph by removing k or fewer vertices, then the riots will stop, people will calm down, and the concert can continue. In other words, the police need to decide the language below, where $G = (V, E)$.*

$$\begin{aligned}\textsc{RiotControl} = \{\langle G, k\rangle \quad | \quad & \exists V' \subseteq V \textit{ such that} \\ & |V'| \leq k, \textit{ and} \\ & E' = \{(u,v) \in E \mid u \in V' \textit{ or } v \in V'\}, \textit{ and} \\ & G' = (V - V', E - E') \textit{ is acyclic}\}\end{aligned}$$

Prove that RiotControl *is NP-Complete.*

We will prove that this problem is NP-Complete by reducing from Vertex-Cover. Given an undirected graph G and integer k, we would like to create a directed graph, G' and integer k' such that $\langle G, k\rangle \in$ VertexCover if and only if $\langle G', k'\rangle \in$ RiotControl. G' and k' will be as follows

$$\begin{aligned} k' &= k \\ V(G') &= V(G) \\ E(G') &= \{(u,v) | \{u,v\} \in E(G)\}\end{aligned}$$

Thus we create G' by replacing every (undirected) edge in G with two directed edges, one going each way in G'.

To prove correctness, we first consider the case where $\langle G, k\rangle \in$ VertexCover. If this is true, then there exists a set, $S \subseteq V(G)$, consisting of k vertices, such that every edge in $E(G)$ is incident on at least one of the vertices in S. If we consider removing the vertices in S from G', then for every pair of edges, (u, v) and (v, u), either u or v must be removed since each of these pairs corresponds to an edge in the original graph. Thus, after removing the vertices in S from G', G' will contain no edges and therefore must be acyclic. Since there was a set of $k = k'$ vertices that could be removed to make G' acyclic, $\langle G', k'\rangle$ is in RiotControl.

Now we consider the case where $\langle G', k'\rangle$ is in RiotControl. This means that there are k' vertices which can be removed from G' to make it acyclic. Since all edges in G' come in pairs, (u, v) and (v, u), the only way to remove all cycles is to remove one of the incident vertices for each pair. Since each of these pairs corresponds to an edge from the original graph, this means that these k' vertices cover all of the edges in G. Thus there is a set of $k = k'$

vertices in G such that every edge is incident on one or more of the vertices, therefore $\langle G, k\rangle$ is in VertexCover.

Thus our reduction is correct because $\langle G, k\rangle \in$ VertexCover $\leftrightarrow \langle G', k'\rangle \in$ RiotControl. The reduction takes polynomial time because all it requires is visiting each edge in the original graph and converting it into two corresponding directed edges. Finally, RiotControl is in NP because a non-deterministic Turing Machine could decide whether $\langle G, k\rangle$ is in RiotControl by the following:

1. If k is greater than $|V(G)|$, reject.
2. Non-deterministically choose k vertices to remove from G.
3. For each pair of remaining vertices, u, v, if PATH(u, v) and PATH(v, u), then reject.
4. Accept.

This algorithm is poly-time since there are a polynomial number of pairs of vertices and $PATH$ is solvable in polynomial time. It is correct because if there are k vertices which can be removed from G to make it acyclic, then those k will be chosen. If not, then there will still exist a cycle and thus for some u, v, there will be a path from u to v, and from v to u. Thus, RiotControl is in NP and is polynomial time reducible from VertexCover, which is NP-Complete. Therefore RiotControl is NP-Complete.

Lab Problem 2

Your three-person band is just starting to make it big. You've already been on a small tour and signed a deal for two records, but you are finally starting accumulate a large number of fans. As a result, your label has decided to start marketing band merchandise. They've made band shirts, bags, guitar cases, decals, you name it. Since they are for your band, the label has decided to send you some of the merchanise for free to distribute amoung your three band members. In the end, the label sends you n pieces of merchanise. For each item i, $1 \leq i \leq n$, the item has a retail price of v_i, which is a positive integer written in binary. Everyone in the band wants to be fair, so you decide to split the merchanise such that each band member gets the same value of gear.

One can formally model the problem by the following language:

MERCHANDISE $= \{\langle v_1, \dots, v_n \rangle \mid$ *each* v_i *is a binary integer and there exists a way to split the set of integers into disjoint sets* S_1*,* S_2 *and* S_3 *such that the sum of the elements within each set* S_i *is equal.*$\}$

Prove that MERCHANDISE *is NP-complete.*

Firstly, MERCHANDISE $\in$ NP because we can non-deterministically partition the items into three groups and check (simply by adding the values) if the three partitions sum to the same thing. If they do, we accept, otherwise reject.

Given a SUBSETSUM input of $S = \langle x_1, \dots, x_m, t \rangle$, first compute

$$a = \sum_{i=1}^{m} x_i.$$

If $t > a$, reject. Otherwise construct the set $W = \langle x_1, \dots, x_m, a+1-t, t+1, a+1 \rangle$. We claim that $W \in$ MERCHANDISE if and only if $S \in$ SUBSETSUM. This is true because, if $S \in$ SUBSETSUM, then there is some subset $Q \subseteq \{x_1, \dots, x_m\}$ such that $\sum_{x \in Q} x = t$. In this case, if we partition $Q \cup \{a+1-t\}$, $\{a+1\}$, and $(\{x_1, \dots, x_m\} - Q) \cup \{t+1\}$, we see that each subset in the partition sums to $a+1$ and $W \in$ MERCHANDISE.

Now, if $W \in$ MERCHANDISE, we note that the sum of all elements in W is $3a+3$, so each partition must add up to $a+1$. Moreover, $t \leq a$, so $a+1$, $a+1-t$, and $t+1$ must all be in different groups since the sum of any two is larger than $a+1$. Also, $a+1$ must be in a group of its own. Therefore, there must be some set $Q \subseteq \{x_1, \dots, x_m\}$ such that the sum of $Q \cup \{a+1-t\}$ is $a+1$. This means that the elements of Q must sum to t. Therefore, we have found a subset of the original elements that sums to t, so $S \in$ SUBSETSUM. Hence we see that $W \in$ MERCHANDISE if and only if $S \in$ SUBSETSUM.

Finally, we note that the only computation is summing the elmenets $x_1, \dots, x_m$, which is polynomial in the length of the input, and then computing $a+1$, $a+1-t$, and $t+1$, which are all polynomial in a and t, and hence in the length of the input. Therefore we see that the reduction is polynomial time and hence we have proven that MERCHANDISE is NP-complete.

Lab Problem 3

An instance of family tree *is a directed graph (***digraph***). Each vertex represents a person. An edge (u, v) means that u is a (biological) parent of v (equivalently, that v is a child of u). Each vertex has either two parents, or none. The graph must be acyclic, because one cannot be one's own ancestor. Finally, although the vertices of a family tree are not labelled with the biological sexes of the people they represent, people nevertheless have biological sexes, so there exists some assignment of biological sexes to the vertices such that every child has parents of distinct biological sexes.*

In this problem, we define the language of Yes instances FAMILY-TREE(g) *that consists of digraphs representing valid family trees with g biological sexes.*

(a) FAMILY-TREE(2) *is either NP-complete or is in P. If it is NP-complete, give a reduction from k-COLOR. If it is in P, give an algorithm that decides it.*

(b) For $g \geq 3$, FAMILY-TREE(g) *is either NP-complete or is in P. If it is NP-complete, give a reduction from k-COLOR. If it is in P, give an algorithm that decides it.*

Remember: *even though there are g biological sexes, each child has* exactly *two parents of different biological sexes, and the input doesn't specify the biological sex of any vertices.*

(a) FAMILY-TREE(2) is in P. We can decide the problem with the following algorithm:

1. For each node, search the edge set for all parents of that node. Count the number of parents. If any node has either only one parent, or more than two parents, reject.
2. For each node, generate the set of all descendents of that node by first generating the set of all descendents of depth 1 or less, then using that set to generate the set of all descendents of depth 2 or less, and so on. If any descendent set contains the original node, then we have a cycle, so we reject.
3. Pick an arbitrary unlabelled node and label it female.
4. Any time you label a node some biological sex x, find all of the children of that particular node, and then find the set of all of the

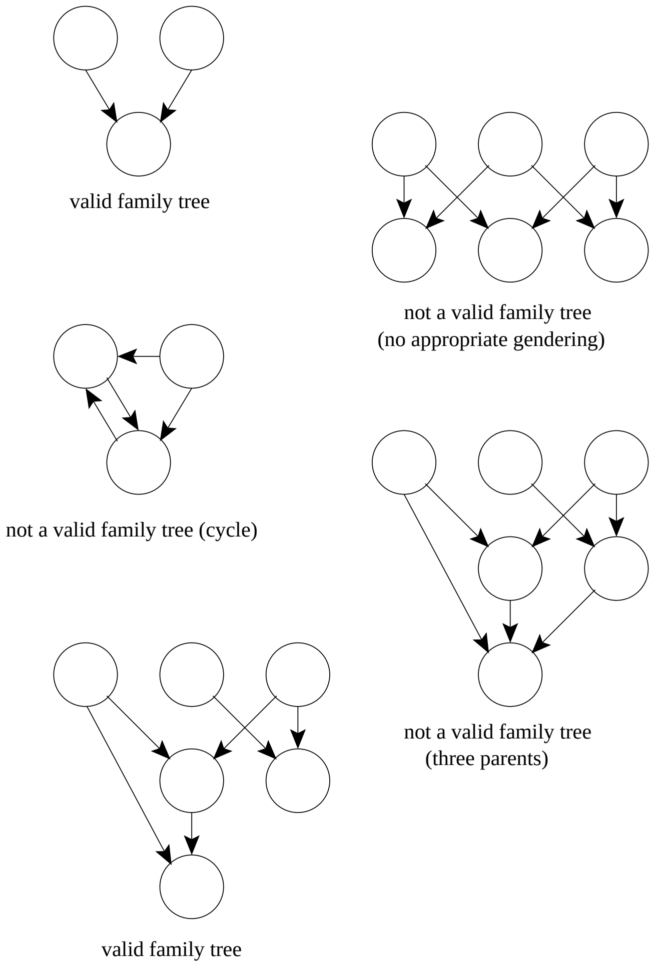


Figure 1: Examples of members and non-members of Family-Tree(2).

other parents of those children. If any of those nodes is already labelled with biological sex x, reject. Otherwise, label all of the unlabelled nodes to be the opposite biological sex to x, and recur.

5. Repeat steps 3-4 until all nodes of the graph have been labelled.

This algorithm takes polynomial time. Step 1 takes $O(|E|)$ time, because you need only look at each edge once. Step 2 takes $O(|V|)$ time, because it's just breadth-first search. Step 3 takes $O(1)$ time, and step 4 takes $O(|E|^2)$ time, since for each edge, you might need to examine each edge in the graph. Steps 3 and 4 are repeated $O(|V|)$ times, so overall it takes $O(|V||E|^2)$ time, which is polynomial in the size of the input.

Does this algorithm correctly decide if a family tree is valid? Well, note that in step 4, we ensure that when we've labelled one node, we've made sure that each of its children has parents of different biological sexes, and made sure that each of those parents' children has parents of different biological sexes, and so on. So essentially, we have labelled all nodes whose biological sexes might in some way conflict with a node that we've already labelled. So the biological sexes of the unlabelled nodes are independent of the nodes that we've already labelled, and so we can freely choose an arbitrary biological sex (female, in the above algorithm) for the next node that we wish to label. Hence, the algorithm above will generate a gendering for the family tree if and only if such a gendering is possible. Therefore, FAMILY-TREE(2) $\in P$.

(b) FAMILY-TREE(g) is NP-complete. We will show that this is true by reduction from k-COLOR. Given some graph $G = (V, E)$, we want to find a coloring for the graph using k colors. We construct a directed graph $G' = (V', E')$ to pass into FAMILY-TREE(k) as follows:

1. For each vertex $v \in V$, add a family member m_v to V'.
2. For each edge $(u, v) \in E$, add one new family member $e_{u,v}$ to V', and add two directed edges $(m_u, e_{u,v})$ and $(m_v, e_{u,v})$ to the directed edge set E'.

If G can be colored using k colors, then we can construct a valid gendering for G' by choosing arbitrary biological sexes for all vertices $e_{u,v}$, and then gendering each of the nodes m_v based on the color of the node v. All of the vertices of the form $e_{u,v}$ have no children, and so their biological sexes are inconsequential. Because we have a valid coloring with k colors, we know that each edge is incident upon two nodes of distinct colors; thus, any child node $e_{u,v}$ has parents m_u and m_v with distinct biological sexes, and therefore we have a valid gendering.

If G' has a valid gendering using k colors, we can construct a valid coloring for G by using the biological sexes of each of the nodes m_v to determine the color of the node v. We know that for each edge (u, v) in E, we constructed a node $e_{u,v}$ with parents m_u and m_v, so that means that m_u and m_v must have distinct biological sexes, and so u and v will have distinct colors. Hence, any time there is a valid gendering, there is also a valid coloring.

Therefore, the above algorithm is a correct reduction. It's also poly-time, because step 1 takes time polynomial in the number of vertices in the original graph, and step 2 takes time polynomial in the number of edges in the original graph, thus making the whole algorithm polynomial in the input size. Therefore, k-COLOR is polynomial-time reducible to Family-Tree(g).

Note: The question as stated did not require you to show that Family-Tree(g) is in NP (and therefore NP-complete, not just NP-hard) , but it's easy enough to show that it is by nondeterministically selecting one of the g biological sexes for each node, and then checking to make sure that all children have parents of distinct biological sexes.