Problem 1, CS38 Set 2, Matt Lim

(a) So we begin with a set of positive integers $S = \{l_1, l_2, \cdots, l_k\}$ satisfying

$$\sum_{i=1}^{k} 2^{-l_i} \leq 1,$$

. We will let n be the maximum integer in S. Our algorithm will be as follows:

1. We will sort S in increasing order.
2. We will construct a full and complete binary tree of height n. Each node is initially colored white.
3. For each $l_i \in S$, in increasing order, we will traverse, from the root, l_i levels down, then delete all the nodes under the node we reach. So if we traverse all the way down to the bottom of the tree, nothing is deleted. We will color each traversed node black.
4. We will delete all unvisited (white) nodes

We will now prove that this algorithm works. It is clear to see that we delete all root-leaf paths whose lengths aren't in S (or whose lengths are repeats of other traversed paths). For example, if we have eight root-leaf paths of length three, but only one three in S, seven of those root length paths will be deleted. This is because we traverse one path of length l_i for each i, delete the subtree under that node (if it exists) then delete all other nodes. So we have established that we have a binary tree whose root-leaf paths have lengths $l_1, l_2, \ldots, l_n$. Now let us confirm that these lengths meet their specific requirement.

So, let us consider our set S. For any integer $l_i \in S$, the maximum number of times it may appear is 2^{l_i}. Now let us consider n, the maximum integer in S. We have that n can appear a maximum of 2^n times. And since n is the largest number in S, this means that S contains no more than 2^n "slots" (the size of S must be less than or equal to 2^n, and the number of slots taken up by all the l_is is at most 2^n). Now, consider the following. Each $l_i \in S$ takes up 2^{n-l_i} "slots" of S. This is easy to see when thinking of the representative binary tree; each level down in the binary tree doubles the number of available slots. So one level down gives us two slots (two lengths of 1/2), two levels down give us four slots, etc. So if we compare a leaf at a level l_i to a leaf at level n, we can clearly see that it takes up 2^{n-l_i} as many "slots" as the bottom level leaves (which take up one "slot"). Another way to look at it is that each leaf takes up as many slots as the number of leaves it would have produced in a full, complete binary tree, with the max root-leaf path length unchanged in that full, complete binary tree (height of the full, complete tree is equal to the height of the original tree). The one exception are the leaves at the very bottom of the original binary tree, which take up one slot. This all translates to our tree having at most 2^n leaf nodes/slots, with a leaf node with length l_i taking up 2^{n-l_i} slots of the tree. All this gives us the following:

$$\sum_{i=1}^{k} 2^{n-l_i} \leq 2^n$$

$$\sum_{i=1}^{k} \frac{2^{n-l_i}}{2^n} \leq 1$$

$$\sum_{i=i}^{k} 2^{-l_i} \leq 1$$

So our proof is complete.

(b) We have that for the Huffman code, the encoding lengths satisfy the equation

$$\sum_{i=1}^{n} 2^{-l_i} = 1$$

We can see this is true because we can represent the optimal prefix-free Huffman encoding as a binary tree, where the encoding lengths are the lengths of the root-leaf paths. So, given this equation, as well as the equation

$$\sum_{i=1}^{n} l_i p_i$$

we can use the Lagrange multiplier to get a bound for $\sum_{i=1}^{n} l_i p_i$. Let $f_i = l_i p_i$ and $g_i = 2^{-l_i}$. Then, we know that $\nabla f_i = \lambda \nabla g_i$. So, differentiating with respect to l, we get $p_i = -\lambda \cdot \ln(2) \cdot 2^{-l_i}$. We then proceed as follows to isolate λ:

$$2^{-l_i} = -\frac{p_i}{\lambda \cdot \ln(2)}$$

$$-\sum_{i=1}^{n} \frac{p_i}{\lambda \cdot \ln(2)} = 1$$

$$\frac{-\sum_{i=1}^{n} p_i}{\lambda \cdot \ln(2)} = 1$$

$$\frac{-\sum_{i=1}^{n} p_i}{\ln(2)} = \lambda$$

Plugging this lambda back into the first of the above equations yields the following:

$$2^{-l_i} = \frac{\ln(2) \cdot p_i}{\sum_{i=1}^{n} p_i \cdot \ln(2)}$$

$$2^{-l_i} = \frac{p_i}{\sum_{i=1}^{n} p_i}$$

$$\log_2(2^{-l_i}) = \log_2(\frac{p_i}{\sum_{i=1}^{n} p_i})$$

$$-l_i = \log_2(p_i) - \log_2(\sum_{i=1}^{n} p_i)$$

$$l_i = \log_2(\sum_{i=1}^{n} p_i) - \log_2(p_i)$$

Since $p_1, \ldots, p_n$ are the probabilities of n symbols, we have that their sum cannot exceed one. Therefore we have the following:

$$l_i = \log_2(\sum_{i=1}^{n} p_i) - \log_2(p_i) \leq \log_2(1) - \log_2(p_i)$$

$$l_i \leq -\log_2(p_i)$$

Then, since we have that all the l_i are integers, we can put a ceiling on this inequality so that it becomes

$$l_i \leq -\log_2(p_i) + 1$$

Plugging this back into our sum $\sum_{i=1}^{n} l_i p_i$ gives us the following:

$$\sum_{i=1}^{n} l_i p_i \leq \sum_{i=1}^{n} (-\log_2(p_i) + 1) \cdot p_i$$

$$\sum_{i=1}^{n} l_i p_i \leq \sum_{i=1}^{n} -\log_2(p_i) \cdot p_i + \sum_{i=1}^{n} p_i$$

$$\sum_{i=1}^{n} l_i p_i \leq \sum_{i=1}^{n} -\log_2(p_i) \cdot p_i + 1 \leq H(\mathbf{p}) + 1$$

Thus we have proved the desired inequality.

Problem 2, CS38 Set 2, Matt Lim

(a) We will first prove that *graphic matroids* are indeed matroids. So, given an undirected graph $G = (V, E)$, we will prove that the *graphic matroid* $M_G = (S_G, I_G)$ is a matroid. We will let S_G be defined as E, the set of edges of G. We will then define I_G in the following way: if A is a subset of E, then $A \in I_G$ if and only if A is acyclic. In other words, a set of edges A is independent if and only if the subgraph $G_A = (V, A)$ forms a forest. So we have that the independent sets of our *graphic matroid* are all subsets of the edges of G that are forests. Now, onto the proof.

We will first consider the first axiom. By definition, a forest includes a graph of a set of vertices that has no edges. Also, the empty set is a subset of any other set. So, I_G contains the empty set of edges. Next, the second axiom. We have that if you remove edges from a forest, you still have a forest, since removing edges from an acyclic set cannot create cycles. In other words, any subset of a forest is a forest. So we have that if $A \in I_G$ then every subset $B \subseteq A \in I_G$.

Now, onto the third axiom. Let $G_A = (V, A)$ and $G_B = (V, B)$ be forests of G with $|B| > |A|$, i.e. A and B are acyclic sets of edges, with B having more edges than A. We now claim that a forest $F = (V_F, E_F)$ has $|V_F| - |E_F|$ trees. Let us prove this claim. Suppose that F is made up of t trees, with the ith tree containing v_i vertices and e_i edges. Then we get the following (note that the second statement follows from Theorem B.2 on page 1174 of CLRS):

$$
\begin{aligned}
|E_F| &= \sum_{i=1}^{t} e_i \\
&= \sum_{i=1}^{t} (v_i - 1) \\
&= \sum_{i=1}^{t} v_i - t \\
&= |V_F| - t
\end{aligned}
$$

This then implies that $t = |V_F| - |E_F|$. Thus we have that G_A contains $|V| - |A|$ trees, and forest G_B contains $V| - |B|$ trees.

Since G_B has fewer trees than G_A, G_B has to contain some tree T whose vertices are in two different trees in G_A. Then, since T is connected, it must contain an edge (u, v) such that vertices u and v are in different trees in G_A. Since the edge (u, v) connects vertices in two different trees in G_A, adding the edge (u, v) to G_A does not create a cycle. Thus, M_G satisfies the third axiom, and we can conclude that M_G is a matroid.

We will now prove that *matric matroids* are indeed matroids. Let M be the matrix we are representing with our *matric matroid*. So now we have that I are all subsets of column vectors of M that are linearly independent and E is the set of all column vectors. Now we must prove that the three axioms hold.

We will first consider the first axiom. We have that the empty set is a linearly independent set, and that it is a subset of any set. So axiom one holds. Now we will consider axiom two. This is fairly trivial; if you take away a column vector from set of independent column vectors, the new set will still be linearly independent (and thus in I). In other words, taking away column vectors from a set in I still results in a linearly independent set of column vectors. Thus, if $A \in I$, then every subset $B \subseteq A$ is also in I, and the second axiom holds. Finally, we will consider the third axiom. To prove that it holds, we will do a proof by contradiction. So consider two subsets of I, A and B, with $|B| > |A|$. Suppose that there does not exist some $x \in B \backslash A$ such that $A \cup \{x\}$ is in I. Then we have that there is no column vector in B that is independent with respect to the column vectors in A. But this means that every vector in B is dependent on A, which implies that every vector in B can be generated by some linear combination of vectors in A. But this is clearly a contradiction; a set of $n > k$ independent vectors cannot be generated by a set of k independent vectors. So we can conclude that there does exist $x \in B \backslash A$ such that $A \cup \{x\}$ is in I.

(b) Let $M = I$ be our matroid and E be our universe. Then we can give the following greedy algorithm:

GREEDY(I, E)

1. $A = \varnothing$
2. Sort the elements of E into monotonically decreasing order by their integer weights
3. **for** each $x \in E$, taken in monotonically decreasing order by integer weight
    4. **if** $A \cup \{x\} \in I$
        5. $A = A \cup \{x\}$
6. **return** A

We will now prove that this algorithm runs in time $O(|E|\log|E|)$ plus $O(|E|)$ calls to a procedure that determines whether or not a given set A is independent. Line 2, the sorting, takes $O(|E|\log|E|)$. Then we have that line 4 executes exactly $|E|$ times, which means that line 4, which checks to see if the set $A \cup \{x\}$ is independent, happens order $|E|$ times. Thus we have that this algorithm runs in time $O(|E|\log|E|)$ plus $O(|E|)$ calls to a procedure that determines whether or not a given set A is independent.

We must now prove that this algorithm returns an optimal, independent set. The independent part is easy; line 4 checks before adding each element that the resulting set is independent. So the set we return must be independent. This is an inductive argument, where the base case is the empty set, which is independent (A starts as empty). In other words, we start with an independent empty set, and the for-loop maintains the sets independent through each step. So our set is always independent by induction. Now we must only prove correctness. To do so, we will use some of the corollaries and lemmas in the book. All used corollaries and lemmas can be found on pages 441 and 442 of CLRS. Now, onto the proof.

By Corollary 16.9, the elements that our **GREEDY** algorithm skips initially (because they are not extensions of $\varnothing$) are no longer relevant at all, since they can never be useful (they will never be added to A). Lemma 16.7, which says that matroids exhibit the greedy-choice property, tells us that the algorithm never makes an error by adding x to A, since there exists an optimal subset containing x. This is an important part of the proof, so let's look into it a little more. Here is a proof of Lemma 16.7.

The first case to consider is that no such x exists. Then we have that the only independent subset is the empty set and the lemma is trivally true. The second case to consider is that B is a nonempty optimal subset. Let $x \notin B$ (otherwise we could just let $A = B$ and be done). We have that no $y \in B$ has a weight greater than $w(x)$. That is, for all $y \in B$, $w(y) \leq w(x)$. This is because $\{y\}$ is independent, since every subset of B is in I. This fact, along with how we defined x, gives us $w(y) \leq w(x)$ for any $y \in B$. Now we will construct the set A as follows. First we will have $A = \{x\}$. This set is independent because of how we choose x. Then we can use the exchange property to repeatedly find a new element of B that we can add to A until $|A| = |B|$, while keeping A independent. Then A and B are the same except A has x and B has some element y instead, i.e. $A = B - \{y\} \cup \{x\}$ for some $y \in B$. This then gives us that $w(A) = w(B) - w(y) + w(x) \implies w(A) \geq w(B)$. Thus we can conclude that because B is optimal, A must also be optimal. This concludes the proof of Lemma 16.7.

Lastly, we have that Lemma 16,10, which says that matroids exhibit the optimal-substructure property, tells us that the remaining problem (after adding x) is to find an obtimal subset in the matroid M', M' being the contraction of M by x. After our **GREEDY** algorithm sets $A = \{x\}$, we can view its remaining steps as acting in the matroid $M' = I'$, where

$$I' = \{B \subseteq E - \{x\} : B \cup \{x\} \in I\}$$

and the new universe is

$$E' = \{y \in E : \{x, y\} \in I\}$$

We can do this because B is independent in M' if and only if $B \cup \{x\}$ is independent in M, for all sets $B \in I'$. Thus, the next operation of our **GREEDY** algorithm will find a maximum-weight independent subset for M', and the overall algorithm will find a maximum-weight independent subset for M.

(c) I' is defined as all subsets of the complements of bases of I, I being a matroid. We will now prove that I' itself is a matroid. We will first consider the first axiom. We have that the empty set is a subset of any set. So if I contains the empty set, we have that I' contains the complement of the empty and all its subsets. And since the empty set is a subset of any set, then I' contains the empty set. If I contains more than the empty set, it must contain some basis with order greater than zero. Then we have that I' contains the complement of that basis, and by extension the empty set (a subset of the complement of that basis). So we have that the first axiom holds. We will next consider the second axiom. Showing this holds is made fairly trivial by the definition of a dual matroid. A dual matroid is said to contain the complements of the bases of I, and all subsets of these sets. So the second axiom holds by definition.

Finally, we will consider the third axiom. So, we will consider A and B which are subsets in I' with $|B| > |A|$. We will then pick two bases in I and call them X and Y. We have that the cardinality of these two bases are the same due to the definition of bases given in the problem. That is, $|X| = |Y|$. Note that we can pick X and Y such that

$$X \cap B = \varnothing$$

$$Y \cap A = \varnothing$$

since A is a subset of the complement of some basis Y and B is a subset of the complement of some basis X.

Then we have the following:

$$X \backslash A \subseteq Y$$

Then we can pick a basis C disjoint from A and apply the exchange theorem to keep adding elements $x \in C$ into $X \backslash A$ until we create Y (so $X \backslash A$ eventually becomes Y after enough exchanges).

Now we end up with two cases. The first case is when $B \backslash A$ contains an element not in Y. That is, $\exists x \in B \backslash A \notin Y$. Then we can pick that element, put it in A, and we are done. That is, $A \cup \{x\} \in I'$, since $(A \cup \{x\}) \cap Y = \varnothing$. This is because we picked Y such that A is the subset of the complement of some basis Y, and it can't be the complement itself because $|B| > |A|$. So the third axiom holds.

The second case is when $B \backslash A$ contains no element that is not in Y. We can prove that this case never occurs using a proof by contradiction. So, assume that this second case occurs. Then we get the following inequalities:

$$|X| = |X \backslash A| + |X \cap A| \leq |A \backslash B| + |X \backslash A| < |B \backslash A| + |X \backslash A| \leq |Y|$$

The first inequality comes from the fact that $X \cap B = \varnothing$ and the last inequality comes from the fact that we have $B \backslash A \subset Y$. But this is a contradiction because this means that we have $|X| < |Y|$, when $|X|$ should be equal to $|Y|$. So we have that this second case can never happen. So only the first case can happen. And in that case the third axiom is satisfied. Thus we can conclude that the third axiom is always satisfied and the proof is complete.

(d) To prove that

$$I_k = \{A : A \in I \text{ and } |A| \leq k\}$$

is a matroid for each positive integer k, we will simply prove it for some arbitrary positive integer k. So, let's begin.

We will first consider the first axiom. This is fairly trivial. I is a matroid, so $\varnothing \in I$. We also have that $|\varnothing| = 0 \leq k$, where k is any positive integer. Thus $\varnothing \in I_k$. Now onto the second axiom. Let us consider an arbitrary subset $A \in I_k$. By definition of I_k, $A \in I$. And since I is a matroid, all subsets of A are in I. And since taking a subset of A cannot increase its order, and $|A| \leq k$ (since $A \in I_k$), then all subsets of A have order less than or equal to k. By this fact, and the fact that all such subsets are in I, we can see that all subsets of A are also in I_k. Finally, we must verify that the third axiom holds. So consider the two subsets B and A, both in I_k, with $|B| > |A|$. Since $|B| > |A|$, there must exist $x \in B \backslash A$. Then we have that $|B| \leq k \implies |A| < k$. Thus $|A \cup \{x\}| \leq k$. And since A and B are both in I, a matroid, then $A \cup \{x\}$ is also in I. So we can conclude that $A \cup \{x\} \in I_k$.

Problem 3, CS38 Set 2, Matt Lim

(a) Here is a way to formulate the minimum cost spanning tree problem as the problem of finding a maximum weight independent set in a matroid. Note that we will assume our graph is connected, because we are going for a tree. We would have our representative matroid M be a graphic matroid, as defined in $2(a)$. Then, as in $2(b)$, we would give nonnegative integer weights for each element of the universe E, which would correspond to the edge weights of the graph. That is, each element in the matroid would represent an edge in the graph, and would be given a positive integer weight. However, these edge weights would not be the same as the edge weights of the graph. Instead, we would "flip" the weights. That is, we would take the maximum edge weight from the graph, add one to it, and subtract the weight of each edge to find the new weight of each edge. So, if the max edge weight is w_{max}, and we have an edge with weight w in the graph, the corresponding weight in the matroid is $w_{matroidedge} = w_{max} + 1 - w$. We can rewrite this as $w_{matroidedge} = w_0 - w$, where $w_0 = w_{max} + 1$. Then, finding the maximum weight independent set in this matroid would give the edges in the minimum cost spanning tree for its corresponding graph. To see a short proof of this, consider the following. Each maximal independent subset A corresponds to a spanning tree with $|V| - 1$ edges. Then we have the following (where $w'(e)$ represents the weight of the edge e in the matroid, $w(e)$ represents the weight of the edge e in the graph, $w'(A)$ represents the sum of the matroid edge weights, and $w(A)$ represents the sum of the graph edge weights):

$$\begin{aligned} w'(A) &= \sum_{e \in A} w'(e) \\ &= \sum_{e \in A} (w_0 - w(e)) \\ &= (|V| - 1)w_0 - \sum_{e \in A} w(e) \\ &= (|V| - 1)w_0 - w(A) \end{aligned}$$

Clearly, maximizing $w'(A)$ must minimize $w(A)$.

(b) To do this problem, we will prove that a variation of the graphic matroids from $2(a)$, the only difference being that the independent sets are all subsets of the edges of G that are pseudo-forests, are indeed matroids. So, let us begin (let I_G be our graph matroid for pseudo-forests).

We will first consider the first axiom. By definition, a pseudo-forest includes a graph of a set of vertices that has no edges. Also, the empty set is a subset of any other set. So, I_G contains the empty set. Next, the second axiom. We have that if you remove an edges from a pseudo-forest, you still have a pseudo-forest, since removing edges from an pseudo-forest cannot create another cycle. In other words, any subset of a set of pseudo-forest edges is still a set of pseudo-forest edges. So we have that if $A \in I_G$ then every subset $B \subseteq A \in I_G$.

Finally, we will consider the third axiom. Let A and B be subsets of I_G, with $|B| > |A|$. That is, B is a pseudo-forest with more edges than the pseudo-forest A. Here we have three cases, as presented below.

- The first case is when A has 0 cycles. In this case, we can have that B has either 0 or 1 cycles. In either of these two cases, we still have that B contains an edge that A doesn't, since $|B| > |A|$. And since adding an edge to A can create at most one cycle, we have that $A \cup \{x\}$, where $x \in B \backslash A$, will still be in I_G. So the axiom holds.
- The second case is when A has a cycle, and there exists an edge e that is a part of that cycle that is not a part of B. Then, we can do the following. First, we remove e from A. Then, we also take out an edge from B (if B has a cycle, take out the edge from the cycle, else, take out any edge). Now we have forests A' and B' with $|B'| > |A'|$. Since they are forests, we have from the proof of $2(a)$ that there is an edge x in $B' \backslash A'$ such that $A' \cup \{x\}$ is a forest. So we can add x to A', then add e back to A', which can add at most one cycle (we can also add the edge that was removed from B back). So we end up with $A \cup \{x\} \in I_G$, and the third axiom holds.

- The third case is when A has a cycle which is completely in B. In this case, we can do the following. We will remove the same edge from the cycle in A and the cycle B. We will call this edge e. So, we end up with A' and B', which are both forests (since we got rid of the cycles). Now, from the proof of $2(a)$, we have that there exists an edge x in $B' \backslash A'$ such that $A' \cup \{x\}$ is a forest. So we can add x to A', then add e back to A', which can add at most one cycle (we can also add e back to B'). So we end up with $A \cup \{x\} \in I_G$, and the third axiom holds.

Now that we have that I_G (graphic matroid for pseudo-forests) is a matroid, the problem of finding a maximum weight pseudo-forest in an undirected graph with non-negative edge weights is the same as finding a maximum weight independent set in I_G, where the edges in I_G have weights corresponding to the edge weights in the graph.

(c) For this problem, we will start with a graphic matroid representing the network of links. The elements of the universe E of our graphic matroid I_G will represent the links as edges. We will have a weight function for these edges, $w(e)$. The weight of each edge will be the same as the cost of its corresponding link. So, we have our graphic matroid. Now we will take the dual (as defined in $2(c)$) of I_G. We will call the resulting matroid I_{DG}. Then, with k as defined in the problem, we will make a new matroid, I_{KDG}, where

$$I_{KDG} = \{A : A \in I_{DG} \text{ and } |A| \leq k\}$$

Then, finding the maximum weight independent set of I_{KDG} will give you the k links to retire so as to maximize the yearly savings while still retaining connectivity. Now for an explanation. We take the dual because this gives us the subsets of edges that we can remove without affecting connectivity. This is by the definition of the dual. We restrict the order of k so that the greedy algorithm (from $2(b)$) that gets the maximum set only retrieves a subset of k edges/links (because that is the number we want to remove). The greedy algorithm takes care of maximizing the savings, as it chooses the most expensive k links to remove (while still retaining connectivity).

Problem 4, CS38 Set 2, Matt Lim

Let P be the set of m pairs of vertices. Let **make-set**(x), **union**(x, y), **link**(x, y) (used in **union**), and **find-set**(x) be as defined on page 571 of CLRS. Basically, we are using link-by-rank with path compression as shown in lecture. The pseudocode for this algorithm can be given as follows:

LCA(vertex u)

1. **make-set**(u)
2. **find-set**$(u).ancestor = u$
3. **for** each child v of u in T
    4. **LCA**(v)
    5. **union**(u, v)
    6. **find-set**$(u).ancestor = u$
7. $u.color = black$
8. **for** each node v such that $\{u, v\} \in P$
    9. **if** $v.color == black$
        10. output "The least common ancestor of u and v is **find-set**$(v).ancestor$"

We will now prove that this algorithm runs in time $O((n+m)\log^* n)$. The algorithm is basically just building a Union-Find data structure. From the lecture slides, we have that "Starting from an empty data structure, link-by-rank with path compression performs any intermixed sequence of $m \geq n$ **find-set** and $n-1$ **union** operations in $O(m\log^* n)$ time. But this is basically exactly what our algorithm does. It does order $n+m \geq n$ number **find-set** operations; $n-1$ occur in line 6 and $k \leq m$ occur in line 10. We can see that the $n-1$ **union** operations occur in line 5, as every node except for the root node has **union** done on it. Thus we can conclude that the overall running time is $O((n+m)\log^* n)$.

Now for the proof of correctness. We will do this by induction on the number of levels in the tree, n. First comes the base case, $n = 1$. So we have one root node with an arbitrary number of children. The reason our algorithm works for this case is because of the correctness of the **find-set** operation. For the first child, **union** sets that child as the representative node of the union-find data structure. Thus the **find-set** operation finds that first child, and sets its ancestor to the root node. So that ancestor is correct. Then we can see that due to the **union** operation, that first child is always the representative node in the union-find data structure. So whenever we do **find-set**, we find that node. And since when we output the ancestor we do **find-set**, for all the other children of the root node, it outputs the ancestor of the first child, which is the ancestor of all the children. So the least common ancestor of all the children is true. Then, when we do pairs that involve the root node itself, **find-set** still finds the first child, whose ancestor is the root node, so we get the right answer for pairs involving the root node. Now for the inductive assumption. We will assume that this algorithm outputs the correct least common ancestors for all trees of level $\leq n$. Now we must only prove that it outputs the correct least common ancestors for all trees of level $n+1$. WLOG we can just consider adding one node m on the $n+1$st level. Note that the least common ancestors of the parent of m with every other node (including with the parent itself) are the same as the least common ancestors of m with every other node. And by our inductive assumption, we have outputted the correct ancestor of m with every other node (such that m and that other node are in the pairings). So if we want to get the correct least common ancestor for m and every other node, we can just use the correct least common ancestors for m's parent with every other node. So we can just change the output pairings for m's parent and replace m's parent with m. Thus we get the correct ancestor for the $n+1$st level, and by induction we can conclude that this algorithm works for trees of any level.