NAME: ____________________

SPIRE ID: ____________________

COMPSCI 250
Introduction to Computation
Second Midterm Fall 2024 – Solutions

D. A. M. Barrington and M. Golin 7 November 2024

DIRECTIONS:

- Answer the problems on the exam pages.
- There are five problems on pages 2-12, some with multiple parts, for 100 total points plus 10 extra credit. Final scale will be determined after the exam.
- Page 13 contains useful definitions and is given to you separately – do not put answers on it!
- If you need extra space use the back of a page – both sides are scanned.
- But, if you do write on the back, you must explicitly have a note on the front stating that you used the back page.
- No books, notes, calculators, or collaboration.
- In case of a numerical answer, an arithmetic expression like "$2^{17} - 4$" need not be reduced to a single integer.
- Your answers must be LEGIBLE, and not cramped. Write only short paragraphs with space between paragraphs

1	/10
2	/20+10
3	/10
4	/40
5	/20
Total	/100+10

Family Name: ______________________

Question 1 (10): (Induction 1)

For positive naturals n, let $S(n)$ be the sum $\frac{1}{2\cdot 3}+\frac{1}{3\cdot 4}+\frac{1}{4\cdot 5}+\cdots+\frac{1}{(n+1)(n+2)}$.

Formally, $S(n)=\sum_{i=1}^{n}\frac{1}{(i+1)(i+2)}$.

Examples: $S(1)=\frac{1}{2\cdot 3}=\frac{1}{6}$, $S(2)=\frac{1}{2\cdot 3}+\frac{1}{3\cdot 4}=\frac{3}{12}=\frac{1}{4}$, $S(3)=\frac{1}{2\cdot 3}+\frac{1}{3\cdot 4}+\frac{1}{4\cdot 5}=\frac{18}{60}=\frac{3}{10}$, etc..

Prove *by induction* that for all positive naturals n, $S(n)=\frac{n}{2(n+2)}$.

i) First write your induction hypothesis in the box below. This should be in the form $P(x)$, where you *must* explicitly explain what x is and write an unambiguous statement of $P(x)$.

Solution. $P(n)$ is the statement: "$S(n)=\frac{n}{2(n+2)}$."
It is defined over all naturals $n\geq 1$.

(ii) Next, write your base case(s) in the box below.

Solution. The base case is $n=1$
But $S(1)=\frac{1}{6}=\frac{1}{2\cdot 3}$, so $P(1)$ is correct.

(iii) Finally, provide your induction step. This step will be marked on how clear and mathematically precise your proof is. Ambiguous explanations will have points deducted.
Be sure to clearly describe your induction goal. If you run out of space, continue the proof on the back page (with a note stating that you are writing on the back).

Solution. Assume $P(n)$. Let $n\geq 2$. Proof will be by regular induction on the naturals. Assume $P(n)$.

$$\begin{aligned}
S(n+1) &= \sum_{i=1}^{n+1}\frac{1}{(i+1)(i+2)} \\
&= \left(\sum_{i=1}^{n}\frac{1}{(i+1)(i+2)}\right)+\frac{1}{(n+2)(n+3)} && \text{Def of summation} \\
&= \frac{n}{2(n+2)}+\frac{1}{(n+2)(n+3)} && \text{By the Induction Hypothesis} \\
&= \frac{n(n+3)+2}{2(n+2)(n+3)} \\
&= \frac{(n+2)(n+1)}{2(n+2)(n+3)} \\
&= \frac{(n+1)}{2((n+1)+2)}
\end{aligned}$$

Since $P(n)\rightarrow P(n+1)$, we have proven the complete inductive step.

Marking Notes for Q1:

Marking Note 1(i): $P(n)$ is not a predicate.
In induction, $P(n)$ must be a boolean predicate, i.e., it evaluates to true or false. In particular $P(n)$ cannot be a number, so a statement like $P(n) = S(n)$ or $P(n) = \frac{n}{2(n+2)}$ is immediately incorrect. Also, once $P(n)$ is not a predicate, the remainder of the induction proof cannot be correct either, so points being deducted for this are not just being deducted for part (i). Depending upon HOW well the rest of the proof was structured, more points might or might not be deducted later on.

Marking Note 1(ii): $P(n)$ Incorrect base case.
This was usually when a base case of $n = 0$ was used. The problem specifically states that $S(n)$ is only defined for*positive naturals*. So $S(0)$ is undefined and cannot be a base case.

The correct base case is $n = 1$.

Marking Note 1(iii): Does not explicitly identify where IH is used.
The problem statement was very clear that ”this step will be marked on how clear and mathematically precise your proof is. Ambiguous explanations will have points deducted.” If the IH is used without explicit acknowledgment of the fact, the proof is not precise and contains ambiguity. This was also made explicit in the class lecture notes (Lecture 23, page 4).

Marking Note 1(iv): LHS vs RHS issue.
This is the error in which the proof splits into a LHS and a RHS and shows that the two are equal but does not explicitly identify what the LHS and RHS are. This issue was discussed in Lecture 23, p 15, where it was pointed out that this is an error that had been seen in old exams and how to fix it. In some cases this was combined with the error from marking note 1(iii).

Family Name: ____________________

Question 2 (20+10): (Induction 2) A **rooted binary-ternary tree (RBTT)** is constructed as follows:

R0: It is either a single node, which is its root, or

R1: it is a new node, its root, which is connected to the roots of two other RBTT's or

R2: it is a new node, its root, which is connected to the roots of three other RBTT's.

Furthermore,

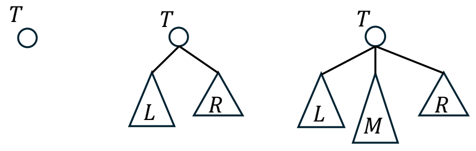
R3: The only RBTT's are those made by the first three rules.

Let $\mathcal{T}$ represent the set of all RBTT's. For tree $T \in \mathcal{T}$,

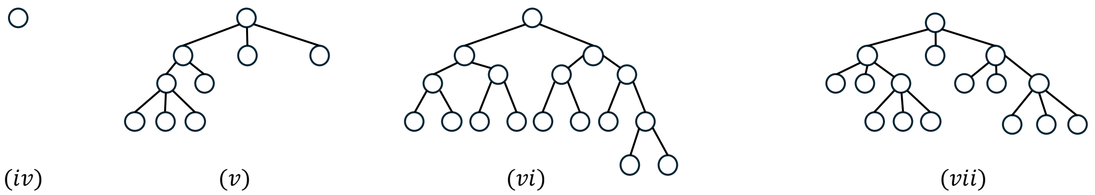
$$\begin{aligned} |T| &= \text{the number of nodes in } T, \\ L(T) &= \text{the number of leaves in } T, \\ I(T) &= \text{the number of internal nodes in } T, \\ d(T) &= \text{the depth of } T. \end{aligned}$$

Recall that the depth of T is the length of the longest path from the root to any leaf.

Diagrams (i-iii) are illustrations of the rules. Diagrams (iv)-(vii) are examples of some RBTT's and their associated values.



(i) (ii) (ii)



(iv)	(v)	(vi)	(vii)
$L(T) = 1$	$L(T) = 6$	$L(T) = 9$	$L(T) = 11$
$I(T) = 0$	$I(T) = 3$	$I(T) = 8$	$I(T) = 5$
$d(T) = 0$	$d(T) = 3$	$d(T) = 4$	$d(T) = 3$
$\lvert T\rvert = 1$	$\lvert T\rvert = 9$	$\lvert T\rvert = 17$	$\lvert T\rvert = 16$

Parts (A)-(C) of this problem are on the following pages ((C) is for extra credit). When writing the solutions to parts (A)-(C), you must use the the mathematical notation we provided above.

If your proof has multiple pieces, place each piece in a separate paragraph with space between the paragraphs. Be sure to clearly describe your induction goal.

If you run out of space, continue the proof on the back page of the problem (with a note stating that you are writing on the back).

Marking Notes for Q2:

All three of (A), (B) and (C) could be solved using either structural induction or induction over the naturals. The parameter in induction over the naturals could be $d(T)$, $|T|$, $L(T)$, $I(T)$ or even number of steps required to construct T.

For (A), we provide three example solutions; one each using structural induction , induction on $|T|$ and induction on $d(T)$.

For (B), we provide a structual induction proof and a proof by induction on $d(T)$.

For (C), we only provide a structural induction solution.

There were many different errors that when combined, caused even more errors. In those cases, it was impossible to assign points to each individual error. That would have deducted all the points from many solutions. Instead, some of the rubrics *identified* error types, which are described below without point deductions being identified with those rubrics. The deductions then came from other rubrics which assigned the severity of the (combined) errors.

Note that many of the errors described in the rubrics overlap with each other. So, points deducted did not correspond to any specific marking note but rather to how wrong a proof subjectively was.

Marking Note 2(i): Type Mismatch.
This occurs if the type used in the definition of the IH doesn't match the type used in the Base case or the IS.

For example, if doing structural induction, the IH is defined by $P(T)$ where T is a RBBT. The base case should be T_0, where T_0 is the one node RBBT. Writing that the base case is $P(1)$ or $P(0)$ would then be wrong. A base case of $P(0)$ or $P(1)$ would be used, e.g., in induction on the size of the RBBT or on the depth of the RBBT

Type mismatches could make the entire proof wrong. As an example, in structural induction, writing that the Induction Step is to prove that $P(T) \to P(T+1)$ makes no sense. A RBBT doesn't have a successor, so "$T + 1$" makes no sense.

Marking Note 2(ii): Expanding a RBBT.
Any proof that said that the inductive step worked by adding leaves or nodes in some way to another base RBBT was categorically wrong and had at least half the problem points deducted.

This is exactly the "expansion" or "extending" error that was explicitly described in Lecture 23 on pages 71-72 for RBTs and also on p 41. See the explanation there for why it is wrong (short version is that the only way that you can access RBBTs is through their definition, which builds them by adding RBBTs (not nodes) as the left and right child of a root).

This error also includes the error of starting by saying $P(T)$ is true for some T and that the process adds another tree (and maybe the root) to the first tree. RBBTs are NOT built by adding a tree to another tree. By definition (and induction must follow the definition) they are built by starting with a node and connecting that root to 2 or 3 subtrees.

Marking Note 2(iii): Not Explicitly stating where induction was used and/or why it is allowed to be used.
In order to get full points it was required to identify where the IH was used. For example, if a RBBT was built from 3 subtrees, it was necessary to point out that each of those subtrees satisfied the IH.

IN non-structural induction proofs, it was also necessary to explicitly point out *why it was valid to apply the IH.* For example, in (A), if induction was done on size (as in the second solution for part (A),) it was necessary to say that, if $|T| = n + 1$, then each of its 2 (or 3) subtree children T_c (T_c being T_L, T_R and, if needed, T_M,) has size $|T_c| \le n$, and, therefore, from the IH, each of the subtrees satisfies $d(T_c) \le L(T_c) - 1$. If this was just used without being explicitly explained, points were deducted.

Marking Note 2(iv): Starting the IS by *assuming* that the IG is correct or by not defining what the IS is working on.

As an example of the first, some solutions started part (c) by saying "If we assume $P(T)$ is true then we can assume $P(T_L)$ and $T(R)$ are true" or used a similar logic somewhere in the internals of their proof.

The GOAL of the Inductive Step is to prove that $P(T)$ is true. The proof cannot start by *assuming* that $P(T)$ is true. Even if the remainder of the solution was correct, points were deducted because the logic was wrong

The second issue was related but different.

In the structural case, many solutions *started* with T_L (T_M) and T_R without defining what they were. That is wrong.

A structural proof MUST start by introducing the T for which it will prove $P(T)$. T_L (T_M) and T_R are the *specific* subtrees from which T is built using rules R_1 (and R_2). They're not arbitrary trees that are being connected together. Proofs that did not follow this structure were inherently incorrect. (On a logical level, proofs that did not start by introducing T first would be making a error related to the expansion error described in marking note 2(ii)).

A natural based proof has to do the above but with one extra step. For example if the induction is being done on depth, the IS would try to prove that EVERY RBBT T of depth $d+1$ satisfies the rule. So, such a proof must start with an intro like "Let T be *any* RBBT with $d(T) = d+1$. Let T_L and T_R be the subtrees through which it is built via rule R_2. Since $D(T_L) \le d$ and $d(T_R) \le d$ we know, by the IH, that they satisfy the property,...."

If the proof is missing the fact that is is specializing to pick an arbitrary RBBT of depth $d+1$, it is missing important info. Likewise, if it doesn't identify T_L and T_R as being specialized to be the subtrees used to construct that specific T it is also missing important info.

Marking Note 2(v): Doing structural induction but writing $P(x) \to P(x+1)$ or by increasing some RBBT parameter

The formula $P(x) \to P(x+1)$ is only true for induction on the naturals. It cannot be applied in the case of structural induction. In this case, the structural induction was on RBBTs. The notation $T + 1$ where T is an RBBT makes no sense.

A structural induction proof must start with some T for which we want to prove $P(T)$. After starting with T we say that, by the IH, the T_L, T_R (and, if needed, T_M,) from which T is built all satisfy the IH.

The use of the notation $T+1$ not only makes no sense but signifies an expansion or bottoms-up type proof with similarities to the error in marking note 2(ii). This is totally opposite to the top-down approach of structural induction.

A related issue is using structural induction but saying at the beginning of the IS that "the depth of T is being increased by 1" or "the size of T is being increased by 1", or "the number of leaves in T is being increased by 1". all of those suffer from the same issue . All of those would be wrong.

With structural induction you are not increasing some parameter (that's what induction on the naturals would do).

A structural induction proof must start with some T for which we want to prove $P(T)$. After starting with T we say that, by the IH, the T_L, T_R and, if needed, T_M, from which T is built all satisfy the IH.

Marking Note 2(vi): Not using IH properly.
This often combined with the other errors above.

The main issue was, that, independent of the type of induction used, when trying to show in (A), that $d(T) \leq L(T) + 1$, it was absolutely necessary that the solution uses the fact that, from the IH,
$d(T_L) \leq L(T_L) + 1$, $d(T_R) \leq L(T_R) + 1$, and, if there is a T_m, $d(T_M) \leq L(T_M) + 1$ as well.

It was also necessary to explain WHY those statements were correct.

If structural induction was used, they are obviously correct. If induction on $|T|$ used, then because $|T_L| < |T|$, $|T_R| < |T|$ and $|T_M| < T$. If induction on $d(T)$ used, then because $d(T_L) < d(T)$, $d(T_R) < d(T)$ and $d(T_M) < d(T)$. Similar statements are needed for induction on $L(T)$ and $I(T)$.

This same error occurred in parts (B) and (C).

Marking Note 2(vii): Incorrectly using induction in (B)(c) .
A correct answer in B(c) MUST have the following steps (but see Caveat at end):

1. Starts with introduction of current RBBT T.
If not structural induction, it precedes this with increment of induction parameter n, i.e., n being the depth, number of nodes, number of leaves, etc. Then it introduces T as aany RBBT with parameter value $n+1$

2. It then states that T is built as a root with subchildren T_L, T_R (and, if R_2, T_M).

3. It then states that, from the induction hypothesis,
$L(T_L) \leq 3^{d(T_L)}$, $L(T_R) \leq 3^{d(T_R)}$ (and $L(T_M) \leq 3^{d(T_M)}$).

4. Using the result of the 2 or 3 applications of the IH it then shows
$L(T) = L(T_1) + L(T_R) \leq 3^{d(T_L)} + 3^{d(T_R)}$ or
$L(T) = L(T_1) + L(T_R) \leq 3^{d(T_L)} + 3^{d(T_R)} + 3^{d(T_M)}$.

5. It must explicitly state and use the fact that
$d(T) = \max(d(T_L), d(T_R)) + 1$ or $d(T) = \max(d(T_L), d(T_R), d(T_M)) + 1$

6. Combining steps 4 and 5 it then gets the required result that
$L(T) \leq 2^3 \cdot 3^{\max(d(T_L),d(T_R))} = 3 \cdot d(T) = 1 = 3^{d(T)}$ or
$L(T) \leq 2^3 \cdot 3^{\max(d(T_L),d(T_R),d(T_M))} = 3 \cdot d(T) = 1 = 3^{d(T)}$.

It is impossible to prove B(c) *by induction* without using all of these steps.
Step 1 is needed because that is how induction works. It's a top-down process, not a bottom-up one.
Step 2 is needed because that is the only way to introduce the subtrees (since induction is top-down). This property is strongly emphasized in the notes. See, e.g., Lesson 23, pp 40-41 and 97.
Step 3 is needed because that is the ONLY use of induction available. The statement on the smaller subtrees that are children of the root.

Step 4 (or equvalent) is needed because that is the only way to combine the individual induction statements to get an upper bound on $L(T)$.
Step 5 (or equivalent) is needed because that is the only way combine the individual induction statements to relate the upper bounds of the smaller induction statements to an upper bound using $d(T)$.
Step 6 is needed to combine the previous steps.

Some of these steps, especially 4-6 can be combined in various ways BUT, all of their pieces must be present in the proof. Any proof that is missing some component is wrong in some fashion.

Caveat: There is one type of proof that did not need the explicit version of steps 3-6. That is if the IH was $P(d)$: "every tree with depth $d(T) \leq d$ satisfies $L(T) \leq 3^d$.".
Note that this replaces all of the $d(T_c)$ values by 3^d and is slightly simpler to write.

Some solutions talked about finding a "worst-case" solution or "maximal" solution. That is an invalid approach since you have no way of knowing what the worst-case is. The only entrée into RBBTs is the given recursive definition. Any other properties, e.g., worst case, would have to be proven.

For example, some solutions discussed "adding" a node to one of the subtrees and looking at how the depth changed. That is invalid because there is no way of knowing how things change without proving the facts via induction, Also see marking note 2(ii)

Marking Note 2(viii): Did not identify either R1 or R2 as providing tight bounds. This was for part (C).

In parts (A) and (B) we marked leniently for solutions that were missing one of the R1 or R2 case, if they showed how the proof worked. The marking for (C) was much stricter. This is because the two cases are not the same. (Some solutions claimed they were the same, but this is incorrect.) R1 provides a tight lower bound and R2 provides a tight upper bound. Points were also deducted if both R1 and R2 were done but the proof did not somehow indicate the diffeernce.

Note: 'Tight' means using $\leq$ or $\geq$. Loose means using $<$ or $>$.

(A) Prove by induction that every $T \in \mathcal{T}$ satisfies

$$d(T) \leq L(T) - 1.$$

a) First write your induction hypothesis in the box below. This should be in the form $P(x)$, where you *must* explicitly explain what x is and write an unambiguous statement of $P(x)$.

Solution. $P(T)$ is the statement: "$d(T) \leq L(T) - 1$".
$P(T)$ is defined over all trees $T \in \mathcal{T}$.

(b) Next, write your base case(s) in the box below.

Solution. The base case is $T = T_0$ where T_0 is the one node tree.
Since $d(T_0) = 0$ and $L(T_0) = 1$, $P(T_0)$ is correct

(c) Finally, provide your induction step. This step will be marked on how clear and mathematically precise your proof is. Ambiguous explanations or explanations missing details will have points deducted.

Solution. The proof will be using structural induction. (Induction on size was also possible.) There are two cases

- **T is a root connected to the roots of two other RBBTs T_L and T_R:**
  By the IH, $d(T_L) \leq L(T_L) - 1$ and $d(T_R) \leq L(T_R) - 1$.
  By the definition of depth, $d(T) = 1 + \max(d(T_L), d(T_R))$.
  By the construction, $L(T) = L(T_L) + L(T_R)$.
  So

$$\begin{aligned} d(T) &= 1 + \max(d(T_L), d(T_R)) \\ &\leq 1 + d(T_L) + d(T_R) \\ &\leq 1 + (L(T_L) - 1) + (L(T_R) - 1) \qquad \text{(IH)} \\ &= 1 + L(T) - 2 \qquad (\text{Since } L(T) = L(T_L) + L(T_R)) \\ &= L(T) - 1. \end{aligned}$$

- **T is a root connected to the roots of three other RBBTs T_L, T_M and T_R:**
  By the IH, $d(T_L) \leq L(T_L) - 1$, $d(T_M) \leq L(T_M) - 1$, $d(T_R) \leq L(T_R) - 1$.
  By the definition of depth, $d(T) = 1 + \max(d(T_L), d(T_M), d(T_R))$.
  By the construction,$L(T) = L(T_L) + L(T_M) + L(T_R)$.
  So

$$\begin{aligned} d(T) &= 1 + \max(d(T_L), d(T_M), d(T_R)) \\ &\leq 1 + d(T_L) + d(T_M) + d(T_R) \\ &\leq 1 + (L(T_L) - 1) + (L(T_M) - 1) + (L(T_R) - 1) \qquad \text{(IH)} \\ &= 1 + L(T) - 3 \qquad (\text{Since } L(T) = L(T_L) + L(T_M) + L(T_R)) \\ &< L(T) - 1. \end{aligned}$$

Since R1 and R2 are the only ways of building a RBBT and we have shown that in both cases, $d(T) \leq L(T) - 1$, the proof is completed.

Family Name: ____________________

We now show a SECOND Proof of (A), that every $T \in \mathcal{T}$ satisfies $d(T) \leq L(T)-1$,, this time doing induction on the number of nodes in the RBBT.

a) First write your induction hypothesis in the box below. This should be in the form $P(x)$, where you *must* explicitly explain what x is and write an unambiguous statement of $P(x)$.

Solution. $P(n)$ is the statement: " For all $T \in \mathcal{T}$ satisfying $|T| \leq n$, $d(T) \leq L(T) - 1$".
$P(n)$ is defined over all positive naturals n.

(b) Next, write your base case(s) in the box below.

Solution. The base case is $n = 1$. There is only one tree T with $|T| \leq 1$.
This is the one node tree T_0.
Since $d(T_0) = 0$ and $L(T_0) = 1$, $P(1)$ is correct

(c) Finally, provide your induction step. This step will be marked on how clear and mathematically precise your proof is. Ambiguous explanations or explanations missing details will have points deducted.

Solution. The proof will be by induction on the size of the RBBT.
Assume $P(n)$. To prove $P(n+1)$ we must prove that every RBBT T, with $|T| = n+1$, satisfies $d(T) \leq L(T) - 1$. There are two cases

- **T is a root connected to the roots of two other RBBTs T_L and T_R:**
  By construction $|T_L| \leq n$ and $|T_R| \leq n$. Therefore we can apply the IH to know $d(T_L) \leq L(T_L) - 1$ and $d(T_R) \leq L(T_R) - 1$.
  By the definition of depth, $d(T) = 1 + \max(d(T_L), d(T_R))$.
  By the construction, $L(T) = L(T_L) + L(T_R)$. So

$$\begin{aligned} d(T) &= 1 + \max(d(T_L), d(T_R)) \\ &\leq 1 + d(T_L) + d(T_R) \\ &\leq 1 + (L(T_L) - 1) + (L(T_R) - 1) \qquad \text{(IH)} \\ &= 1 + L(T) - 2 \qquad \text{(Since } L(T) = L(T_L) + L(T_R)) \\ &= L(T) - 1. \end{aligned}$$

- **T is a root connected to the roots of three other RBBTs T_L, T_M and T_R:**
  By construction $|T_L| \leq n$, $|T_M| \leq n$ and $|T_R| \leq n$. Therefore, we can apply the IH to know $d(T_L) \leq L(T_L) - 1$, $d(T_M) \leq L(T_M) - 1$, $d(T_R) \leq L(T_R) - 1$.
  By the definition of depth, $d(T) = 1 + \max(d(T_L), d(T_M), d(T_R))$.
  By the construction,$L(T) = L(T_L) + L(T_M) + L(T_R)$. So

$$\begin{aligned} d(T) &= 1 + \max(d(T_L), d(T_M), d(T_R)) \\ &\leq 1 + d(T_L) + d(T_M) + d(T_R) \\ &\leq 1 + (L(T_L) - 1) + (L(T_M) - 1) + (L(T_R) - 1) \qquad \text{(IH)} \\ &= 1 + L(T) - 3 \qquad \text{(Since } L(T) = L(T_L) + L(T_M) + L(T_R)) \\ &< L(T) - 1. \end{aligned}$$

Since R1 and R2 are the only ways of building a RBBT and we have shown that in both cases, $d(T) \leq L(T) - 1$, the proof is completed.

We now show a THIRD Proof of (A), that every $T \in \mathcal{T}$ satisfies $d(T) \leq L(T) - 1$,, this time doing induction on the depth of RBBT. It would also be possible to prove this doing induction on the number of leaves or internal nodes in the RBBT, or even the number of construction steps required to build the RBBT.

a) First write your induction hypothesis in the box below. This should be in the form $P(x)$, where you *must* explicitly explain what x is and write an unambiguous statement of $P(x)$.

Solution. $P(d)$ is the statement: " For all $T \in \mathcal{T}$ satisfying $d(T) \leq d$, $d(T) \leq L(T) - 1$". $P(d)$ is defined over all naturals d.

(b) Next, write your base case(s) in the box below.

Solution. The base case is $d = 0$. There is only one tree with T with $d(T) \leq 0$.
This is the one node tree T_0.
Since $d(T_0) = 0$ and $L(T_0) = 1$, $P(0)$ is correct

(c) Finally, provide your induction step. This step will be marked on how clear and mathematically precise your proof is. Ambiguous explanations or explanations missing details will have points deducted.

Solution. The proof will be by induction on the size of the RBBT.
Assume $P(d)$. To prove $P(d+1)$ we must prove that every RBBT T, with $d(T) = d+1$ satisfies $d+1 = d(T) \leq L(T) - 1$. There are two cases

- **T is a root connected to the roots of two other RBBTs T_L and T_R:**
  By the definition of depth, $d(T) = 1 + \max(d(T_L), d(T_R))$.
  So $d(T_L) \leq d$ **and** $d(T_R) \leq d$
  Therefore we can apply the IH to know $d(T_L) \leq L(T_L) - 1$ and $d(T_R) \leq L(T_R) - 1$.
  By the construction, $L(T) = L(T_L) + L(T_R)$. So

$$\begin{aligned} d(T) &= 1 + \max(d(T_L), d(T_R)) \\ &\leq 1 + d(T_L) + d(T_R) \\ &\leq 1 + (L(T_L) - 1) + (L(T_R) - 1) \qquad \text{(IH)} \\ &= 1 + L(T) - 2 \qquad \text{(Since } L(T) = L(T_L) + L(T_R)\text{)} \\ &= L(T) - 1. \end{aligned}$$

- **T is a root connected to the roots of three other RBBTs T_L, T_M and T_R:**
  By the definition of depth, $d(T) = 1 + \max(d(T_L), d(T_M), d(T_R))$.
  So $d(T_L) \leq d$**,** $d(T_M) \leq d$ **and** $d(T_R) \leq d$ **Therefore, we can apply the IH to know** $d(T_L) \leq L(T_L) - 1$, $d(T_M) \leq L(T_M) - 1$, $d(T_R) \leq L(T_R) - 1$.
  By the construction,$L(T) = L(T_L) + L(T_M) + L(T_R)$. So

$$\begin{aligned} d(T) &= 1 + \max(d(T_L), d(T_M), d(T_R)) \\ &\leq 1 + d(T_L) + d(T_M) + d(T_R) \\ &\leq 1 + (L(T_L) - 1) + (L(T_M) - 1) + (L(T_R) - 1) \qquad \text{(IH)} \\ &= 1 + L(T) - 3 \qquad \text{(Since } L(T) = L(T_L) + L(T_M) + L(T_R)\text{)} \\ &< L(T) - 1. \end{aligned}$$

Since R1 and R2 are the only ways of building a RBBT and we have shown that in both cases, $d(T) \leq L(T) - 1$, the proof is completed.

(B) Prove by induction that every $T \in \mathcal{T}$ satisfies

$$L(T) \leq 3^{d(T)}.$$

a) First write your induction hypothesis in the box below. This should be in the form $P(x)$, where you *must* explicitly explain what x is and write an unambiguous statement of $P(x)$.

Solution. $P(T)$ is the statement "$L(T) \leq 3^{d(T)}$".
$P(T)$ is defined over all trees $T \in \mathcal{T}$.

(b) Next, write your base case(s) in the box below.

Solution. The base case is $T = T_0$ where T_0 is the one node tree.
Since $d(T_0) = 0$, $L(T_0) = 1$, and $0 \leq 1 = 3^0$, $P(T_0)$ is correct

(c) Finally, provide your induction step. This step will be marked on how clear and mathematically precise your proof is. Ambiguous explanations or explanations missing details will have points deducted.

Solution. The proof will be using structural induction. (Induction on size was also possible.) There are two cases

- **T is built by having a root connected to the roots of two other RBBTs T_L and T_R:**
  By the IH, $L(T_L) \leq 3^{d(T_L)}$ and $L(T_R) \leq 3^{d(T_R)}$.
  By the definition of depth, $d(T) = 1 + \max(d(T_L), d(T_R))$, so $d(T_L) \leq d(T) - 1$ and $d(T_R) \leq d(T) - 1$.
  By the construction $L(T) = L(T_L) + L(T_R)$. So

$$\begin{array}{rcll} L(T) & = & L(T_L) + L(T_R) & \\ & \leq & 3^{d(T_L)} + 3^{d(T_R)} & \text{(IH)} \\ & \leq & 3^{d(T)-1} + 3^{d(T)-1} & \\ & = & 2 \cdot 3^{d(T)-1} & \\ & \leq & 3 \cdot 3^{d(T)-1} = 3^{d(T)}. & \end{array}$$

- **T is built by having a root connected to the roots of three other RBBTs T_L, T_M and T_R:**
  By the IH, $L(T_L) \leq 3^{d(T_L)}$, $L(T_M) \leq 3^{d(T_M)}$ and $L(T_R) \leq 3^{d(T_R)}$.
  By the definition of depth, $d(T) = 1 + \max(d(T_L), d(T_M), d(T_R))$, so $d(T_L) \leq d(T) - 1$, $d(T_M) \leq d(T) - 1$, and $d(T_R) \leq d(T) - 1$.
  By the construction $L(T) = L(T_L) + L(T_M) + L(T_R)$. So

$$\begin{array}{rcll} L(T) & = & L(T_L) + L(T_M) + L(T_R) & \\ & \leq & 3^{d(T_L)} + 3^{d(T_M)} + 3^{d(T_R)} & \text{(IH)} \\ & \leq & 3^{d(T)-1} + 3^{d(T)-1} + 3^{d(T)-1} & \\ & = & 3^{d(T)}. & \end{array}$$

Since R1 and R2 are the only ways of building a RBBT and we have shown that in both cases, $L(T) \leq 3^{d(T)}$ the proof is completed.

Family Name: ____________________

We now show a SECOND proof of (B), that every $T \in \mathcal{T}$ satisfies $L(T) \leq 3^{d(T)}$., doing induction on the depth of the RBBT.

(B) Prove by induction that every $T \in \mathcal{T}$ satisfies

$$L(T) \leq 3^{d(T)}.$$

a) First write your induction hypothesis in the box below. This should be in the form $P(x)$, where you *must* explicitly explain what x is and write an unambiguous statement of $P(x)$.

Solution. $P(d)$ is the statement "$L(T) \leq 3^d$ for all $T \in \mathcal{T}$ satisfying $d(T) \leq d$ ".
$P(d)$ is defined over all naturals d.

(b) Next, write your base case(s) in the box below.

Solution. The base case is $d = 0$. The only RBBT with $d \leq 0$ is T_0, the one node tree. Since $d(T_0) = 0$, $L(T_0) = 1$, and $0 \leq 1 = 3^0$, $P(T_0)$ is correct

(c) Finally, provide your induction step. This step will be marked on how clear and mathematically precise your proof is. Ambiguous explanations or explanations missing details will have points deducted.

Solution. The proof will be on depth d. Assume $P(d)$. We want to prove $P(d+1)$. This means that we need to prove for every $T \in \mathcal{T}$ satisfying $d(T) = d+1$, $L(T) \leq 3^{d+1}$. Let T be an arbitrary such RBBT.

There are two cases:

- **T is built by having a root connected to the roots of two other RBBTs T_L and T_R:**
  By the definition of depth, $d + 1 = d(T) = 1 + \max(d(T_L), d(T_R))$, so $d(T_L) \leq d$ and $d(T_R) \leq d$. So, **By the IH,** $L(T_L) \leq 3^d$ and $L(T_R) \leq 3^d$.
  By the construction $L(T) = L(T_L) + L(T_R)$. So

$$\begin{aligned} L(T) &= L(T_L) + L(T_R) \\ &\leq 3^d + 3^d \qquad \text{(IH)} \\ &< 3 \cdot 3^d = 3^{d+1}. \end{aligned}$$

- **T is built by having a root connected to the roots of three other RBBTs T_L, T_M and T_R:**
  By the definition of depth, $d+1 = d(T) = 1 + \max(d(T_L), d(T_M), d(T_R))$. So $d(T_L) \leq d$, $d(T_M) \leq d$, and $d(T_R) \leq d$. So **By the IH,** $L(T_L) \leq 3^d$, $L(T_M) \leq 3^d$ and $L(T_R) \leq 3^d$.
  By the construction $L(T) = L(T_L) + L(T_M) + L(T_R)$. So

$$\begin{aligned} L(T) &= L(T_L) + L(T_M) + L(T_R) \\ &\leq 3^d + 3^d + 3^d \qquad \text{(IH)} \\ &\leq 3 \cdot 3^d = 3^{d+1}. \end{aligned}$$

Since R1 and R2 are the only ways of building a RBBT and we have shown that in both cases, $L(T) \leq 3^{d+1}$ the proof is completed.

Family Name: ____________________

(C) Extra Credit Problem. Prove by induction that every $T \in \mathcal{T}$ satisfies

$$I(T) + 1 \leq L(T) \leq 2I(T) + 1.$$

a) First write your induction hypothesis in the box below. This should be in the form $P(x)$, where you *must* explicitly explain what x is and write an unambiguous statement of $P(x)$.

Solution. $P(T)$ is the statement: " $I(T) + 1 \leq L(T) \leq 2I(T) + 1$ ".
$P(T)$ is defined over all trees $T \in \mathcal{T}$.

(b) Next, write your base case(s) in the box below.

Solution. The base case is $T = T_0$ where T_0 is the one node tree.
Since $L(T_0) = 1$ and $I(T_0) = 0$, and $0 + 1 \leq 1 \leq 2 \cdot 0 + 1$, $P(T_0)$ is true

(c) Finally, provide your induction step. This step will be marked on how clear and mathematically precise your proof is. Ambiguous explanations or explanations missing details will have points deducted.

Solution. The proof will be using structural induction. (Induction on size, depth, nummber of leaves or number of nodes was also possible.) There are two cases

- **T is built by having a root connected to the roots of two other RBBTs T_L and T_R:**
  By the IH, $I(T_L) + 1 \leq L(T_L) \leq 2I(T_L) + 1$ and $I(T_R) + 1 \leq L(T_R) \leq 2I(T_R) + 1$.
  By the construction $I(T) = 1 + I(T_L) + I(T_R)$ and $L(T) = L(T_L) + L(T_R)$.
  So

$$\begin{aligned} I(T) + 1 &= 1 + I(T_L) + I(T_R) + 1 \\ &\leq L(T_L) + L(T_R) \qquad \text{(IH)} \\ &= L(T) \end{aligned}$$

  and

$$\begin{aligned} L(T) &= L(T_L) + L(T_R) \\ &\leq (2I(T_L) + 1) + (2I(T_R) + 1) \qquad \text{(IH)} \\ &= 2(I(T_L) + I(T_R) + 1) \\ &= 2I(T) \\ &< 2I(T) + 1. \end{aligned}$$

- **T is built by having a root connected to the roots of three other RBBTs T_L, T_M and T_R:**
  By the IH, $I(T_L) + 1 \leq L(T_L) \leq 2I(T_L) + 1$, $I(T_M) + 1 \leq L(T_M) \leq 2I(T_M) + 1$ and $I(T_R) + 1 \leq L(T_R) \leq 2I(T_R) + 1$.
  By the construction $I(T) = 1 + I(T_L) + +I(T_M) + I(T_R)$ and $L(T) = L(T_L) + L(T_M) + L(T_R)$.

So

$$\begin{aligned}
I(T)+1 &= \Big(1+I(T_L)+I(T_M)+I(T_R)\Big)+1 \\
&= (1+I(T_L))+(1+I(T_M))+(1+I(T_R))-1 \\
&\leq L(T_L)+L(T_M)+L(T_R)-1 \qquad \text{(IH)} \\
&= L(T)-1 < L(T)
\end{aligned}$$

and

$$\begin{aligned}
L(T) &= L(T_L)+L(T_M)+L(T_R) \\
&\leq (2I(T_L)+1)+(2I(T_M)+1)+(2I(T_R)+1) \qquad \text{(IH)} \\
&= 2(I(T_L)+I(T_M)+I(T_R)+1)+1 \\
&= 2I(T)+1.
\end{aligned}$$

Since R1 and R2 are the only ways of building a RBBT and we have shown that in both cases, $I(T)+1 \leq L(T) \leq 2I(T)+1$, the proof is completed.

Family Name: ____________________

Question 3 (10) (Expression Trees)

In this problem we deal with **boolean expression trees** as in Discussion #7, where we have a unary operator $\neg$, two binary operators $\wedge$ and $\vee$, and constants 0 and 1. Recall the definitions of the **infix**, **prefix**, and **postfix** strings for any expression.

- (a, 2) Write the prefix string for the boolean expression with infix string (no justification needed):
  $\neg((1 \vee (1 \wedge (1 \vee 0))) \wedge (\neg(1 \wedge 0) \wedge (1 \wedge \neg 1)))$

  The parentheses are only provided to make the expression unambiguous. They should *not* appear in the prefix string you write out or in the postfix string in part (b).

  Solution.
  $$\neg \wedge \vee 1 \wedge 1 \vee 10 \wedge \neg \wedge 10 \wedge 1 \neg 1$$

- (b, 2) Write the postfix string for the same boolean expression (no justification needed):

  Solution.
  $$1110 \vee \wedge \vee 10 \wedge \neg 11 \neg \wedge \wedge \neg$$

- (c, 6) Argue convincingly that the prefix string of *any* boolean expression ends with a constant and (unless it has only one letter) begins with an operator. [Hint: Consider the recursive definition of the prefix string.]

  Solution. A prefix string for an expression tree is defined recursively, following the definition for expression trees themselves.

  – A single-node tree has a one-letter prefix string, its constant.
  – A new tree, made from a $\neg$ operator applied to an existing tree T with prefix string u, has a prefix $\neg u$.
  – A new tree, made from a binary operator applied to two existing trees T and T', with prefix strings u and u', has a prefix string $\wedge uu'$ or $\vee uu'$.

  Let the predicate $P(T)$ mean "the prefix string w of T ends with a constant, and either w has a single letter or w begins with an operator". We prove by induction on all binary expression trees that $P(T)$ is true.

  For the base case, if T is a single-node tree, w has a single letter that is a constant. So w ends in a constant and $P(w)$ is true.

  If S is a tree formed by applying $\neg$ to a tree T with prefix string w, the prefix string of S is $\neg w$. It clearly begins with an operator, and by the IH $P(w)$, w ends with constant and so does $\neg w$.

  If S is a tree formed by applying a binary operator to trees T and T' with prefix strings u and u', then the prefix string of S is either $\wedge uu'$ or $\vee uu'$. Clearly that string begins with an operator, and the IH (which says that $P(T)$ and $P(T')$ are both true) says that u' ends with a constant, and so does the prefix string of S.

Marking Notes for Question 3:

I'm sorry I (Dave) went overboard in making this expression so long. A large number of people misparsed the infix expression, and so got an incorrect tree. I gave 3/4 for parts (a) and (b) if I was fairly confident that you gave correct postfix and prefix expressions for *some* tree, even if it was the wrong one.

For part (c), this was difficult to grade as so many people gave imprecise arguments about what should be a precise fact about the prefix strings. You didn't need to use induction as I did in the solution. A shorter correct argument might be "If the expression has more than one letter, it was made by applying an operator, and the first letter of the prefix string must have been the operator. And any of the operations, including the base case, result in the last letter being a constant.

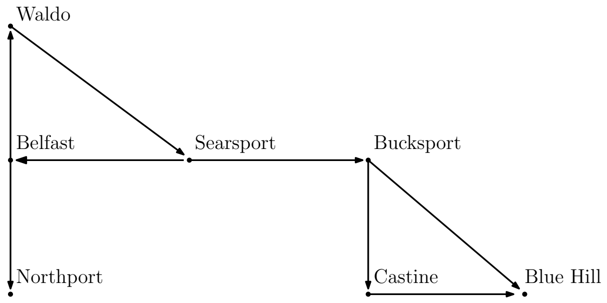
I didn't pose the question very clearly either, for which I apologize.

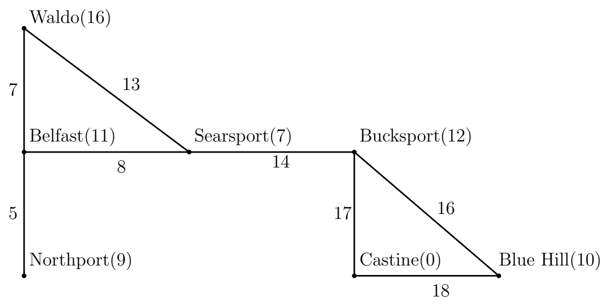
Family Name: ____________________

Question 4 (40) (Search Algorithms)

This problem deals with the road network connecting seven cities in Maine. Our first diagram D, the one on the left, is a *directed* graph with seven nodes and eight directed edges. We will also use the *undirected* graph U obtained from D by making all the edges two-way.

The second diagram L, the one on the right, is a *labeled undirected* graph on the same cities where the label on each edge is the driving distance (in miles) from one city to the other. The *node* labels, in parentheses to the right of the city name, represent the crow-flies distance (in miles) between that city and Castine.





(A) Undirected Breadth-First Search

Carry out a breadth-first search of the undirected graph U, with Belfast as the start node and with no goal node. If two or more nodes go on to the queue at the same time, they come off in alphabetical order with respect to the city name. Indicate the order in which the nodes are placed on the queue.

Draw a BFS tree, indicating which are tree and which are non-tree edges. Include all the edges of the graph in your drawing.

Solution:

We begin by putting Belfast on the queue.

We then put Belfast's neighbors Northport, Searsport, and Waldo on the queue.

Northport comes off, and nothing else goes on at that point.

Searsport comes off, and Bucksport and a second node for Waldo go on.

Waldo comes off, and no new node goes on because Searsport is on the closed list.

Bucksport comes off, and Blue Hill and Castine go on.

The second node for Waldo is discarded, as Waldo is on the closed list.

Blue Hill comes off, and a second node for Castine is put on.

Castine comes off, and finally the second node for Castine is discarded.

In the tree, the tree edges are (Be, N), (Be, S), (Be, W), (S, Bu), (Bu, BH), and (Bu, C). There are two non-tree edges (W, S) and (BH, C).

Marking notes: Most of these submissions were pretty good, with some errors converting a correct sequence of events into a BFS tree. Lots of people tried to assign edge types, but in a BFS all non-tree edges are just non-tree edges. I didn't take off for this error.

Family Name: ____________________

(B) Directed Depth-First Search

Carry out a depth-first search of the directed graph D, with Belfast as the start node and with no goal node. If two or more nodes go on to the stack at the same time, they come off in alphabetical order with respect to the city name. Indicate the order in which the nodes are placed on the stack.

Draw a DFS tree, and indicate the type (tree, back, cross, or forward) for each edge. Include all the edges of the graph in your drawing.

Solution:

We begin with Belfast on the stack.

Belfast comes off, and Northport, and Waldo go on.

Northport comes off, and nothing goes on.

Waldo comes off, and Searsport goes on.

Searsport comes off, we find a back edge to Belfast, and Bucksport goes on.

Bucksport comes off, and Blue Hill and Castine go on.

Blue Hill comes off, and nothing goes on.

Castine comes off, and we discover the edge from Castine to Blue Hill as a cross edge.

In the tree, the tree edges are (Be, N), (Be, W), (W, S), (S, Bu), (Bu, BH), and (Bu, C). There is one back edge (S, Be) and one cross edge (Ca, BH).

Marking notes: Most of these also went pretty well. There was some confusion around the bottom of the tree, and some didn't pay attention to the direction on the edges.

(C) Uniform-Cost Search

Carry out a uniform-cost search for the labeled graph L, with Belfast as the start node and Castine the goal node. Indicate the order in which the nodes are placed on the priority queue. Use a closed list, so that an entry for a node should not be placed onto the priority queue if an entry for that node has already been *removed* from the priority queue. But note that there may be multiple nodes on the priority queue for the same city.

Solution:

We begin with $(Be, 0)$ on the PQ.

We take $(Be, 0)$ and put on $(N, 5)$, $(W, 7)$ and $(S, 8)$.

We take $(N, 5)$ off and put no new nodes on.

We take $(W, 7)$ off and put $(S, 20)$ on. Note that the algorithm is unaware of the better path to S that is now on the PQ.

We take $(S, 8)$ off and put $(Bu, 22)$ on. We do not put a second node for W on, because Waldo is on the closed list at this point.

We take $(S, 20)$ off and discard it as Searsport is on the closed list.

We take $(Bu, 22)$ off and put $(BH, 38)$ and $(C, 39)$ on.

We take $(BH, 38)$ off and put $(C, 56)$ on.

We take $(C, 39)$ off. Since this is a goal node, we stop and conclude that the best path from Belfast to Castine has a length of 39 miles.

Marking notes: The most common errors were small (one point each). When you discover the node (S, 20) from Waldo, Searsport is already on the closed list, so you should not process this node. On the other hand, when you process Blue Hill, you should create a node (C, 56) even though (C, 39) is currently sitting in the PQ – the algorithm cannot see that at this point.

(D) A^* Search

Carry out an A^* search for the labeled graphs, with Belfast as the start node and Castine as the goal node, using the heuristic value $h(x)$ for each node x given by the node labels. Indicate the order in which the nodes are placed on the priority queue, with the priority for each node. Use a closed list as described in part (c). But note that there may be multiple nodes on the priority queue for the same city.

Solution:

We begin with $(Be, 0 + 11)$ on the PQ.

We take $(Be, 0 + 11)$ off and put on $(N, 5 + 9)$, $(W, 7 + 16)$, and $(S, 8 + 7)$.

We take $(N, 5 + 9)$ off and put no new nodes on.

We take $(S, 8 + 7)$ off and put $(Bu, 22 + 12)$ on **and (W, 21+16)**.

We take $(W, 7 + 16)$ and add no new nodes because Searsport is already on the closed list.

We take $(Bu, 21 + 12)$ off and put on $(C, 39 + 0)$ and $(BH, 38 + 10)$.

We discard $(W, 21 + 16)$ **because** w **is on the closed list.**

We take $(C, 39 + 0)$ and end the search, concluding that the best path from Belfast to Castine has a length of 39 miles.

This heuristic allowed us to avoid exploring Blue Hill. A better heuristic, better aligned with driving distances, might have let us avoid exploring Northport and Waldo.

Marking notes: There were a lot of good answers, but a lot of bad ones as well. I nearly always deducted at least three points for getting the wrong answer of 39 for Castine. (One exception was for people who missed that this labeled graph is undirected, so they missed the edge from Belfast to Searsport. If they got 58 for Castine on *both* searches, I only took off one total point.) There is always a "runaway numbers" error in these problems, when you wind up adding a heuristic two or more times into the path total. A single node's priority value should only involve its own heuristic value, not those of other nodes.

Probably the most common error, a minor one, was to not visit Waldo a second time. When you process Searsport, Waldo is not yet on the closed list, so you should make a second node for it. (Again, the algorithm does not know that there is a better path already in the PQ.) **Apologies, in that this second visit was not on the first version of these solutions.** Because that new node has a priority less than (C, 39), you have to take it off the PQ and discard it before you can find (C, 39) and end the search.

Question 5 (20): The following are ten true/false questions, with no explanation needed or wanted, no partial credit for wrong answers, and no penalty for guessing.

After reading the questions, write the correct answer, either T (for true) or F (for false), in the corresponding column. Be sure that your "T" and "F" characters are consistent and distinct.

(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)	(j)
F	F	T	T	F	F	F	F	T	T

- (a) Let k be any positive natural, and let X be the set $\{0, 1, \ldots, k-1\}$. Define the operations of successor, addition, and multiplication using arithmetic modulo k. Then the resulting system of arithmetic satisfies the Peano Axioms for naturals.

  FALSE (45% wrong). This system has number whose successor is 0.

- (b) Let $P(w)$ be a predicate on binary strings. If we prove $P(\lambda)$ and $\forall w : P(w) \rightarrow (P(w0) \wedge P(w10) \wedge P(w11))$, we may conclude $\forall w : P(w)$.

  FALSE (27% wrong). These proofs do not force $P(1)$ to be true.

- (c) Let $P(n)$ be a predicate on the naturals. If we prove $\forall n : P(n) \rightarrow P(n+3)$, we can complete the proof with three base cases $P(0)$, $P(1)$, and $P(2)$.

  TRUE (25% wrong). For any natural n, $P(n)$ can be derived from one of the three base cases using the inductive step.

- (d) Let D be a directed graph with three nodes and no self-loops. If D has at least five directed edges, it is strongly connected, but if it has only four edges it may fail to be strongly connected.

  TRUE (47% wrong). If five of the six possible edges are there, the sixth can be replaced by a two-step path. If the four edges were (x, y), (x, z), (y, x), and (y, z), for example, there is no path from z to x, and so the graph is not strongly connected.

- (e) Let G be a connected undirected graph that has exactly one simple cycle. Then if we remove any one of the edges in G, the resulting graph will be a tree.

  FALSE (52% wrong – it's unusually for the majority to get it wrong). This is a somewhat nasty question, I'll admit. It doesn't say that you're removing one of the edges in the *cycle*, just a node in the graph.

- (f) Consider tiling a $2 \times n$ rectangle with L-shaped tiles as in Lecture 20. Then it is not the case that this can be done if and only if n is divisible by 3.

  FALSE (28% wrong). Sorry about the double negative. Mordecai asked me (Dave) to get rid of it but I forgot to do it. It *is* true that you can do it if and only if 3 divides n – if it does you do it with 2×3 rectangles, and if it doesn't you can't possibly do it because the number of total squares isn't a multiple of 3.

- (g) Consider breadth-first and depth-first searches of a directed acyclic graph G, where there is no closed list or other means to detect visited nodes. Assume that there is some path from the start to the goal node. Then the BFS is guaranteed to find the goal node, but the DFS is not.

FALSE (28% wrong). If the graph is infinite, the statement is false because both searches terminate eventually and actually take about the same amount of time, since they search each path from the root to any leaf. But if the graph could be infinite, the BFS will find it while the DFS might not, so the statement is true. I said late in my exam session that the graph was finite, since that's what we originally meant, but Mordecai didn't because it was late and the exam was running too long, so we decided to give full credit for both answers.

- (h) In a breadth-first search tree for a directed graph, every non-tree edge must go from a node to one of its ancestors in the tree.

  FALSE (27% wrong). Such an edge could to any node that has already been seen. In a directed graph, we have no reason to think that this node is an ancestor – it could have been processed entirely when all nodes reachable *from it* have been found.

- (i) Let H be a labeled directed graph where every directed edge has cost 3. Let x and y be two nodes of H such that there is at least one directed path from x to y. Then a breadth-first search from x can be used to find the minimum-cost path from x to y.

  TRUE (32% wrong). Since all the nodes have the same cost, the minimum-cost path is the one with the fewest edges. We know that BFS finds paths with fewer edges before it finds paths with more edges.

- (j) Let G be a two-player deterministic game defined by a finite game tree, where each leaf is labeled either as a White win, a draw, or a Black win. Then either White has a winning strategy for G or Black has a strategy that guarantees White will not win.

  TRUE (39% wrong). The Determinacy Theorem says that there is some value v such each player has a strategy that will ensure getting either v or some better value for them. This value has to be a leaf value. If v is a White win, then White has a winning strategy. If not, Black has a strategy to achieve at least v, which is not a win for White.