CS250 Homework #4

Daniel Chin *

November 9, 2024

1: P4.4.7 [12 pts]

A polygon is called **convex** if every line segment from one vertex to another lies entirely within the polygon. To **triangulate** a polygon, we take some of these line segments, which don't cross one another, and use them to divide the polygon into triangles. Prove, by strong induction for all naturals n with $n \geq 3$, that every convex polygon with n sides has a triangulation, and that every triangulation contains exactly $n - 2$ triangles. (**Hint**: When you divide an n-gon with a single line segment, you create an i-gon and a j-gon for some naturals i and j. What does your strong inductive hypothesis tell you about triangulations of these polygons?)

Solution:

Base Case: $n = 3$, polygon with 3 sides has a triangulation with $(n-2)$ triangles. A triangle contains 1 triangle.
IH: For all naturals n with $n \geq 3$, every convex polygon with n sides has a triangulation that contains exactly $n - 2$ triangles.
IS: Prove that an $n + 1$-gon will have $(n + 1) - 2$ triangulations.

Given that a line segment will split an n-gon into a i-gon and j-gon for naturals i and j, and that each triangulation contains $n-2$ triangles, the number of triangles in i and j can be represented as $(i - 2)$ and $(j - 2)$. We can rewrite the number of triangles in an n-gon to be $(n+1)-2 = ((i+1)-2)+((j+1)-2) = (i+j)-2$. However, if we add the sides of the i-gon and j-gon, we will count one side twice. We can rewrite $i + j$ as $n + 1$. Thus, we can calculate how many triangles are in a $(n + 1)$-gon by substituting $i + j$: $(n + 1) - 2 = n - 1$. Therefore, an $n + 1$-gon will have $n - 1$ triangulations by strong induction.

*Collaborated with Nobody.

2: P4.7.6 [12 pts]

(uses Java) We can define the **balanced parenthesis language** using recursion. This is the set of sequences of left and right parentheses that are balanced, in that every left paren has a matching right paren and the pairs are nested properly. We'll use "L" and "R" instead of "(" and ")" for readability.
We define the language Paren by the following four rules:

1. λ is in Paren.
2. If u is in Paren, then so is LuR.
3. If u and v are in Paren, then so is uv.
4. No other strings are in Paren.

Write a real-Java static method (or a Python function) `isBalanced` that takes a `String` argument and returns a `boolean` telling whether the input string is in Paren. A non-recursive method is simpler.

Solution:

```
// check if same amount of L and R
// check if nested properly
public static boolean isBalanced(String str) {
int count = 0;

if (str.length() == 0) {
    return true;
}
else {
    for (int i = 0; i < str.length(); i++) {
        if (str.charAt(i) == 'L' {
            count++;
        }
        else if (str.charAt(i) == 'R') {
            count--;
        }
        if (count < 0) {
            return false;
        }
    }
}
```

```
    return count == 0;
}
```

3: P4.9.2 [10 pts]

Prove that any directed cycle in the graph of a partial order must only involve one node. (**Hint:** If the cycle were to contain two distinct nodes x and y, what does transitivity tell you about arcs between x and y?)

Solution:

Prove: Any directed cycle in the graph of a partial order must only involve one node.

Assume a directed graph of a partial order has two distinct nodes x and y.
If x has a directed edge to y, and y has a directed edge to x, $x = y$ by a partial order's anti-symmetric property. Thus, the directed cycle only contains one node, which is $x = y$.
If x has a directed edge to y and y has a directed edge to z, then x has a directed edge to z by a partial order's transitive property. There can only be one node in the directed cycle because there is no path from any given node back to the node. Thus, the only directed cycle is one involving only one node.
Thus, any directed graph of a partial order cannot have two distinct nodes and must only involve one node.

4: P4.10.5 [10 pts]

Prove that if T is any rooted directed binary tree (where every internal node has out-degree exactly two), then the number of leaves in T is one greater than the number of internal nodes. (**Hint:** Use induction on the definition of such trees.)

Solution:

Let n = number of leaves
Let m = number of internal nodes
Base Case: T has one node, which is the root and a leaf. n = 1, m = 0
IH: Prove that n in T is one greater than m. n = m + 1
IS: (m + 1) + 1 = n + 1

When a node becomes an internal node by adding two children (an internal node must have out-degree of two), the number of leaves becomes n - 1 + 2 because two leaves get added, and the new internal node is not a leaf node because it now has children. Thus, one is added to the total number of leaves and one is added to the total number of internal nodes. Thus, n + 1 = (m + 1) + 1.

5: P4.11.1 [12 pts]

Show that a $3 \times n$ rectangle can be covered exactly with L-shaped tiles if and only if n is even. (**Hint:** For the negative result, use induction on all odd numbers and an indirect proof in the inductive step.)

Solution:

Prove that a $3 \times n$ rectangle can be covered exactly with L-shaped tiles if and only if n is even.
To do this, we prove that a $3 \times n$ rectangle can't be covered with L-shaped tiles if n is odd.

Assume that n is odd

Base Case: $n = 1$, smallest odd number. A 3×1 rectangle cannot be covered with L-shaped tiles. Thus the above statement holds.
IH: show that a $3 \times n$ rectangle cannot be covered exactly with L-shaped tiles.
IS: A $3 \times (n+2)$ rectangle can be broken up into a $3 \times (n+1)$ rectangle and 1 row, which is a 3×1 rectangle. A $3 \times (n+1)$ rectangle can be written as a combination of 3×2 rectangles because the number of rows would be even. Following from the base case where $n = 1$ and that a 3×1 rectangle cannot be covered using any L-shaped tiles, a $3 \times (n+2)$ rectangle cannot be covered using L-shaped tiles.
Thus, we have proved that a $3 \times n$ rectangle cannot be covered if n is odd by induction. Which also proves that it can only be covered iff n is even by indirect proof

6: P4.11.3 [12 pts]

Prove the claim at the end of the section about the Euclidean Algorithm and Fibonacci numbers. Specifically, prove that if positive naturals a and b are each at most $F(n)$, then the Euclidean Algorithm performs at most $n-2$ divisions. (You may assume that $n > 2$.) (It follows from this result that Fibonacci numbers are the worst case, but you may not use this fact to solve this problem!)

Solution:

Prove that if positive naturals a and b are each at most $F(n)$, then the Euclidean Algorithm performs at most $n-2$ divisions.
Base Case: $n = 3$, $F(3) = 2$, if a and b are at most 2, then there is also at most n - 2 divisions in the Euclidean Algorithm.
IH: Show that if positive naturals a and b are each at most $F(n)$, then the Euclidean Algorithm performs at most $n-2$ divisions.
IS: Since $F(n) < aF(n-1)$, $a > b$ takes $n-2$ divisions by the IH, Let $F(n) < a < F(n+1)$ and $a > b$. The remainder of $amodb$ is also at most F(n + 1), thus the EA repeats. Thus the EA will take $n-2+1$ divisions.

7: P9.1.5 [10 pts]

State and prove a theorem giving the maximum and minimum possible number of leaves in a rooted tree of depth d and degree k. Repeat for the maximum and minimum total number of nodes.

Solution:

Given depth d and degree k. Let T be a rooted tree

Theorem: The maximum amount of leaves in a rooted tree T is k^d

Proof: The maximum possible number of leaves occurs when each internal node has k children (degree k suggests that a node can have at most k children), and T has depth d. Which means that the most leaf nodes it can have is k^d.

Theorem: The minimum amount of leaves in T is 1 if d = 0. If d ¿ 0, the number of leaves = k.

Proof: The minimum possible number of leaves occurs when there is only one node in the tree. In this case, the node is considered both the root and a leaf because it has no children. If $d > 0$, then there would only be k leaves because k is defined as the maximum number of children that a node has. To minimize it, excluding the root, a node will have one child, and that child will have one child, repeating d times.s

Theorem: The minimum possible number of nodes in a tree is 1, when it is only the root of the tree (d = 0). If $d > 0$, the minimum number of nodes is d + k.

Proof: By the definition of a rooted tree, a single node with no edges, where the node is the root, is a rooted tree. Thus the minimum total number of nodes in a rooted tree is 1. In the $d > 0$ case, each node has up to k children, and depth d. The minimum possible nodes is when a node has one child, and then that child has one, then repeating d times.

Theorem: The maximum total number of nodes in T is $\sum_{i=0}^{d} k^i$.

Proof: The maximum total number of nodes in a tree is given by the sum of the amount of the nodes on each level up to depth d. Each node has k children, thus the maximum total number of nodes in T is $\sum_{i=0}^{d} k^i$.

8: P9.3.5 [10 pts]

Let T be a parse tree for an expression with only unary and binary operators, and let m be the number of primitive elements in the expression. Prove that if $m > 1$, then there exists a node x of T such that the subtree rooted at x contains at least $m/3$ of the primitive elements and at most $2m/3$ of the primitive elements.

Solution:

Prove that if $m > 1$, then there exists a node x of T such that the subtree rooted at x contains at least $m/3$ of the primitive elements and at most $2m/3$ of the primitive elements.

Base Case: $m = 2$, there are only two primitive elements. Satisfies the statement that thereis a node x of T such that a subtree is rooted at x and contains at least $m/3$ primitive elements and at more $2m/3$.
IH: Show that if $m > 1$, then there exists a node x of T such that the subtree rooted at x contains at least $m/3$ of the primitive elements and at most $2m/3$ of the primitive elements.
IS: Show that it is satisfied for m + 1 prim. elem.
Starting at the root of T, if it is unary operator, then we check its one child, if binary we check both. We keep checking and keep track of num of prim elem. Once we reach a prim elem, we can check to see if the num of total prim elem is $\leq m/3$. If it is more than m/3 and less than 2m/3, we can choose the current node as x.

9: P9.4.7 [12 pts]

Following Exercise 9.4.9, describe the state graph of the Towers of Hanoi puzzle for general n. Prove that the puzzle is always solvable, and find the number of moves in the shortest possible solution. (It will be useful to have a recursive definition of the state space.)

Solution:

Let n be the number of rings. Let Rod 1, 2, 3 be the corresponding rods.
Prove that the puzzle is always solvable:
Base Case: $n = 1$, there is one disk, which means that you can move the disk from rod 1 to rod 3.
IH: prove that you can always move n disks to rod 3
IS: you can move the $(n + 1)$th disk by removing the disks above it to rod 2, then moving the $(n + 1)$th to rod 3. Then you can move the rest of the disks to rod 3, thus solving the puzzle for $(n+1)$ disks. This proves that you can solve the puzzle for n disks by induction.

Find num of moves in shortest sol'n:
If $n = 1$ there is only one move needed to solve the puzzle.
For $n = 2$, there are 3 moves needed to solve the puzzle.
for $n = 3$, there are 7 moves needed to solve the puzzle
for $n = 4$, there are 15 moves needed
for $n = 5$, there are 31 moves needed
IH: prove that the number of moves shortest possible solution is $2^n - 1$
IS: Need to prove: the number of moves to solve a puzzle with $n+1$ disks is $2^{n+1}-1$. Moving the n disks above the $n + 1$ disk to rod 2 would be $2^n - 1$ moves. Then moving the $n + 1$ disk to rod 3 would be 1 move. Then moving the n disks back above $n + 1$ disk on rod 3 is another $2^n - 1$ moves. Thus, $2 * (2^n - 1) + 1$, which equals $2^{n+1} - 2 + 1 = 2^{n+1} - 1$. Therefore we have proved that it takes $2^n - 1$ moves in the shortest possible solution.

EC: P9.5.8 [10 pts]

In the knight's tour problem, we are looking for a path of knight's moves from a square of the $n \times n$ chessboard to itself, such that the path visits all n^2 squares of the board. How can we modify BFS or DFS to solve this problem? (The solution may not be efficient, but it is correct.)

Solution:

We can find a path of knight's moves from a square of an $n \times n$ chessboard to itself where it visits all squares of the board by:
Placing the knight on any square on the given $n \times n$ chessboard. Then we put the position and path into a stack. Check if we have visited all n^2 squares, stop if we have. If we haven't then we need to check all the possible moves from that possible that we haven't visited already. And each valid move we mark it and update position and path, then we push it onto the stack. If it is empty then backtrack and pick a new position.