CS 4150

OFFICE HOURS REVIEW #2

8/31/21

(1)

(Party Planning) You are planning a (post-COVID) party for your friends. Even though everyone on the potential invite list is friends with you, not all of your friends are friends with each other (but we assume you do know the friendships among your friends). You want to invite as many friends as possible to the party, *but* you want to make sure everyone invited is friends with at least 5 other people on the guest list (so no one is lonely).

Consider the following algorithm for determining the largest subset of your friends that you can invite without having to worry about lonely guests. The algorithm begins by assuming everyone will be invited, then iteratively "un-invites" anyone who doesn't have enough friends on the guest list.

Algorithm 1: Party Planning Algorithm

Input: List<Friendships> *friendships*
Result: Set<Friends>

invited ← `initialize`(*friendships*)
while $\exists\ p \in invited$ *with* < 5 *friends* $\in invited$ **do**
  `uninvite`(*p*)
end
return `convert_to_set`(*invited*)

(a) Prove this algorithm returns the largest possible invite list without any lonely friends (defined as those knowing less than 5 other people on the guest list).

Hint: what is the (formal) statement you're trying to prove?
And what is the (formal) negation of the statement?

Proof: Assume that alg. does not invite the largest poss. set w/ out lonely friends.

$\Rightarrow$ either invited a) contains a lonely friend or b) could have been larger.

a) $\exists p \in$ invited that's lonely. f the lonely friends. $f < 5$ friends in invited. $\Rightarrow\Leftarrow$ while loop

b) invited is not the largest list. Let S be the larger set.

b) could have been larger. $S' < S$ larger

b) invited is not the largest list. Let S be the larger set. f' the first in S that's uninvited.

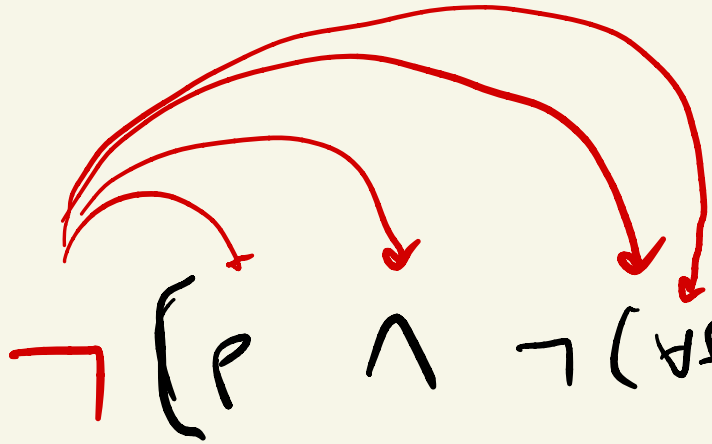
- Since f' is not lonely in S and no other friends from S have been uninvited,
- f' must still have 5 friends in invited

returns [largest possible list] **and** [no friends are lonely]

p q

$\equiv \forall f$ not lonely

$\equiv \neg(\forall f \text{ lonely})$



$\Rightarrow \neg p \vee \cancel{\neg} \exists f$ lonely (cancel)

Contradiction statement (english): assume that the algorithm does **not** return . . .

2. (Party Planning)

We prove the correctness of the Party Planning Algorithm by contradiction.

Proof. Assume that the Party Planning algorithm does not invite the largest possible set without lonely friends. This implies that either `invited` contains a lonely friend or could have been larger. Let's consider each case:

(a) **Case 1**: `invited` contains at least one lonely friend. Let f be a lonely friend. By definition, f has fewer than 5 friends who are invited. But then, f is in `invited` with fewer than 5 friends in `invited`, directly contradicting that the while loop terminated.

(b) **Case 2**: `invited` is not the largest set of friends that could have been invited. Let S be such a larger set of friends, and let f be the first friend in S which is uninvited by the Party Planning Algorithm. Since f is not lonely in S and no other friends from S have been uninvited, f must still have at least 5 friends in `invited`, namely their friends in S. However, this is a contradiction since the Party Planning Algorithm only uninvites a friend if they have fewer than 5 invited friends.

□

② Theorem: $\sum_{i=0}^{n} 3^i = \frac{3^{n+1}-1}{2}$, for every (∀) non-neg. integer n

Base case ($n=0$) $\sum_{i=0}^{0} 3^i = 3^0 = 1$, and $\frac{3^1-1}{2} = 1$ ✓

(IH) Assume that $\sum_{i=0}^{k} 3^i = \frac{3^{k+1}-1}{2}$ for $k \geq 1$

$$\sum_{i=0}^{k} 3^i = \frac{3^{k+1}-1}{2} \quad \text{for } k \geq 1 \quad \text{(IH)}$$

show $(k+1)$

$$\sum_{i=0}^{k+1} 3^i = \frac{3(k+1)+1}{2} - 1 = \frac{3^{k+2}-1}{2}$$

$$\Rightarrow \sum_{i=0}^{k} 3^i + 3^{k+1} \Rightarrow \frac{3^{k+1}-1}{2} + 3^{k+1}$$

$$\frac{3^{k+1}-1}{2} + 3^{k+1} \leftarrow \frac{2\left(3^{k+1}\right)}{2}$$

$$= \frac{\left(3^{k+1}\right) - 1 + 2\left(3^{k+1}\right)}{2} = \frac{\left(3^{k+1}\right)\overbrace{\left[1+2\right]}^{3} - 1}{2}$$

$$= \frac{\left(3^{k+1}\right)\cdot 3 - 1}{2} = \frac{3^{k+2}-1}{2} \quad \checkmark$$

Proof by induction: Let n be an arbitrary non-negative integer.

Assume inductively that $\sum_{i=0}^{k} 3^i = \frac{3^{k+1}-1}{2}$ for every non-negative integer $k < n$.

There are two cases to consider: Either $n = 0$ or $n \geq 1$.

- If $n = 0$, then $\sum_{i=0}^{n} 3^i = 3^0 = 1$, and $\frac{3^{n+1}-1}{2} = \frac{3^1-1}{2} = 1$.
- On the other hand, if $n \geq 1$, then

$$\begin{aligned}
\sum_{i=0}^{n} 3^i &= \sum_{i=0}^{n-1} 3^i + 3^n && \text{[definition of } \textstyle\sum\text{]} \\
&= \frac{3^n-1}{2} + 3^n && \text{[induction hypothesis, with } k = n-1\text{]} \\
&= \frac{3^{n+1}-1}{2} && \text{[algebra]}
\end{aligned}$$

In both cases, we conclude that $\sum_{i=0}^{n} 3^i = \frac{3^{n+1}-1}{2}$. □

③ **Theorem:** In every tree, the # of vertices is one more than the # of edges.

$n=1$

PROOF: The theorem is clearly true for the 1-node tree. So let T be an arbitrary tree with at least two nodes. Assume inductively that the number of vertices in T is one more than the number of edges in T. Suppose we add one more leaf to T to get a new tree T'. This new tree has one more vertex than T and one more edge than T. Thus, the number of vertices in T' is one more than the number of edges in T'. □

Q: Is this proof convincing?

This is not a proof. Every sentence is true, and the connecting logic is correct, but it does not imply the theorem, because it doesn't *explicitly* consider *all possible* trees. Why should the reader believe that their favorite tree can be recursively constructed by adding leaves to a 1-node tree? It's *true*, of course, but that argument doesn't *prove* it. Remember: ***There are only two ways to prove any universally quantified statement:*** Directly ("Let T be an arbitrary tree. . . ") or by contradiction ("Suppose some tree T doesn't. . . ").

Here is a *correct* inductive proof using the same underlying idea. In this proof, I don't have to prove that the proof considers arbitrary trees; it says so right there on the first line! As usual, the proof very strongly resembles a recursive algorithm, including a subroutine to find a leaf.

3 1/2 Can you give a correct inductive proof for the thm?

Theorem: In every tree, the # of vertices is one more than the # of edges.

This is not a proof. Every sentence is true, and the connecting logic is correct, but it does not imply the theorem, because it doesn't *explicitly* consider *all possible* trees. Why should the reader believe that their favorite tree can be recursively constructed by adding leaves to a 1-node tree? It's *true*, of course, but that argument doesn't *prove* it. Remember: ***There are only two ways to prove any universally quantified statement:*** Directly ("Let T be an arbitrary tree. . . ") or by contradiction ("Suppose some tree T doesn't. . . ").

Here is a *correct* inductive proof using the same underlying idea. In this proof, I don't have to prove that the proof considers arbitrary trees; it says so right there on the first line! As usual, the proof very strongly resembles a recursive algorithm, including a subroutine to find a leaf.

Proof: Let T be an arbitrary tree. Assume that in any proper subtree of T, the number of vertices is one more than the number of edges. There are two cases to consider: Either T has one vertex, or T has more than one vertex.

- If T has one vertex, then it has no edges.
- Otherwise, T must have at least one vertex of degree 1, otherwise known as a leaf.

  Proof: Consider a walk through the graph T that starts at an arbitrary vertex and continues as long as possible without repeating any edge. The walk can never visit the same vertex more than once, because T is acyclic. Whenever the walk visits a vertex of degree at least 2, it can continue further, because that vertex has at least one unvisited edge. But the walk must eventually end, because T is finite. Thus, the walk must eventually reach a vertex of degree 1. □

  Let ℓ be an arbitrary leaf of T, and let T' be the tree obtained by deleting ℓ from T. Then we have the identity

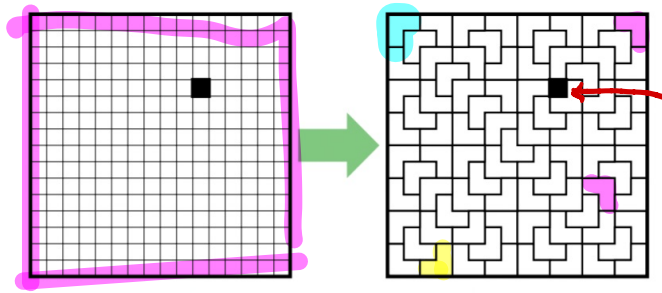
$$|E(T)| = |E(T')| + 1 = |V(T')| = |V(T)| - 1,$$

  where the first and third equalities follow from the definition of T', and the second equality follows from the inductive hypothesis.

In both cases, we conclude that the number of vertices in T is one more than the number of edges in T. □

③

$[2n \times 2n]$ grid



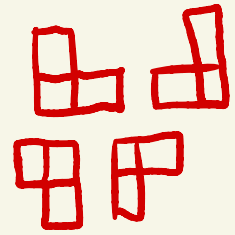
Almost tiling a 16 × 16 checkerboard with triominos.

* Can cover all squares but one.

∃ sqr | (∀ sqrs)

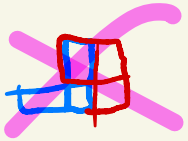
Theorem 5. *For any non-negative integer n, the $2^n \times 2^n$ checkerboard with any square removed can be tiled using L-shaped triominos.*

GOAL: want to cover as much of a $2^n \times 2^n$ grid w/ triominos.



(3-squares in an L-shape)

* no overlap

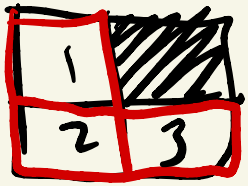


* all inside grids

Base case ($n=0$): $2^0 \times 2^0 = 1 \times 1$ grid

(IH) $k \leq n$ assume $2^{k-1} \times 2^{k-1}$ grid .....

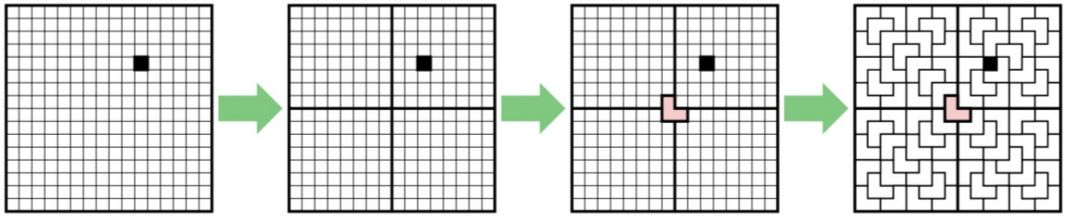
- $2^k \times 2^k$ grid, w.l.g



Proof by top-down induction: Let n be an arbitrary non-negative integer. Assume that for any non-negative integer $k < n$, the $2^k \times 2^k$ grid with any square removed can be tiled using triominos. There are two cases to consider: Either $n = 0$ or $n \geq 1$.

- The $2^0 \times 2^0$ grid has a single square, so removing one square leaves nothing, which we can tile with zero triominos.
- Suppose $n \geq 1$. In this case, the $2^n \times 2^n$ grid can be divided into four smaller $2^{n-1} \times 2^{n-1}$ grids. Without loss of generality, suppose the deleted square is in the upper right quarter. With a single L-shaped triomino at the center of the board, we can cover one square in each of the other three quadrants. The induction hypothesis implies that we can tile each of the quadrants, minus one square.

In both cases, we conclude that the $2^n \times 2^n$ grid with any square removed can be tiled with triominos. □



Top-down inductive proof of Theorem 4.

(4) Prove or Disprove the correctness of this algorithm.

Input: price $p \geq 8$.
Output: integers $n, m \geq 0$ so that $p = 5n + 3m$

PayWithThreeCentsAndFiveCents(p):

```
1   Let x = 8, n = 1, m = 1 (so that x = 5n + 3m).
2   While x < p:
3       x = x + 1
4       if n ≥ 1:
5           n := n − 2
6           m := m + 1
7       else
8           n := n + 2
9           m := m − 3
10      return (n, m)
```

(5) Prove or disprove that this algorithm is correct.

Input: price $p \geq 8$.
Output: integers $n, m \geq 0$ so that $p = 5n + 3m$

PayWithThreeCentsAndFiveCents(p):

```
1   Let x = 8, n = 1, m = 1 (so that x = 5n + 3m).
2   while x < p:
3       x = x + 1
4       if n ≥ 1:
5           n := n − 1
6           m := m + 2
7       else
8           n := n + 2
9           m := m − 3
10      return (n, m)
```

BASE : $p = 8$, $2+2+2+2$, $n = 0, m = 4$

Assume that $p = 5n + 3m$ where $n, m \geq 0$ are integers.
We need to show that $p + 1 = 5a + 3b$ for integers $a, b \geq 0$. Partition to cases:

- **Case 1:** $n \geq 1$. We have more than 1 5-cent piece.
  - In this case, $p + 1 = 5 \cdot (n - 1) + 3 \cdot (m + 2)$.
  - Remove one 5-cent piece, add 2 3-cent pieces.
- **Case 2:** $m \geq 3$. We have more than 3 3-cent pieces.
  - $p + 1 = 5 \cdot (n + 2) + 3 \cdot (m - 3)$.
  - Add 2 5-cent pieces, remove 3 3-cent pieces
- **Case 3:** $n = 0, m \leq 2$. We have no 5-cent pieces, and 2 or fewer 3-cent pieces.
  - $p = 5n + 3m \leq 6$, which is a contradiction to $p \geq 8$

6 Prove (via contradiction) the following claim.

A file compression algorithm attempts to reduce the size of files, by transforming each input file to an output file with fewer bits. A **lossless** algorithm allows you to reconstruct the original file exactly from its compressed version, whereas a **lossy** algorithm only allows you to reconstruct an approximation to the original file. Or, said another way, a lossless algorithm must convert input files to output files in a one-to-one-manner, so that two distinct input files are never compressed to the same output file.

Claim *A lossless compression algorithm that makes some files smaller must make some (other) files larger.*

Proof: Suppose not. That is, suppose that we had a lossless compression algorithm A that makes some files smaller and does not make any files larger.

Let x be the shortest file whose compressed size is smaller than its original size. (If there are two such files of the same length, pick either at random.) Suppose that the input size of x is m characters.

Suppose that S is the set of distinct files with fewer than m characters. Because x shrinks, A compresses x to a file in S. Because no files smaller than x shrink, each file in S compresses to a file (perhaps the same, perhaps different) in S.

Now we have a problem. A is supposed to be lossless, therefore one-to-one. But A maps a set containing at least $|S| + 1$ files to a set containing $|S|$ files, so the Pigeonhole Principle states that two input files must be mapped to the same output file. This is a contradiction.