CS 70 Discrete Mathematics and Probability Theory
Summer 2017 Hongling Lu, Vrettos Moulos, and Allen Tang

DIS 1C

1 Fibonacci for Home

Recall, the Fibonacci numbers, defined recursively as

$F_1 = 1$, $F_2 = 1$, and $F_n = F_{n-2} + F_{n-1}$.

Prove that every third Fibonacci number is even. For example, $F_3 = 2$ is even and $F_6 = 8$ is even.

Solution:

First, we should prove that all the Fibonacci numbers are integer by induction: $P(k)$ is "F_k is an integer". This follows from the fact that F_1 and F_2 are integer, and the induction step follows from $F_k = F_{k-1} + F_{k-2}$, the (strong) induction hypothesis that F_{k-1} and F_{k-2} are integers and the fact that the integers are closed under addition.

Now we prove that for all natural numbers $k \geq 1$, F_{3k} is even. The base case, $k = 1$, is that $F_3 = 2$ is even, which is clear.

For the induction step, we have that $F_{3k+3} = F_{3k+2} + F_{3k+1} = 2F_{3k+1} + F_{3k}$.

By the induction hypothesis $F_{3k} = 2q$ for some q, and we have that $F_{3k+3} = 2(F_{3k+1} + q)$, which implies that it is even. Thus, by induction we have that all F_{3k} are even.

2 Natural Induction on Inequality

Prove that if $n \in \mathbb{N}$ and $x > 0$, then $(1+x)^n \geq 1 + nx$.

Solution:

- *Base Case:* When $n = 0$, the claim holds since $(1+x)^0 \geq 1 + 0x$.
- *Inductive Hypothesis:* Assume that $(1+x)^k \geq 1 + kx$ for some value of $n = k$ where $k \in \mathbb{N}$.
- *Inductive Step:* For $n = k+1$, we can show the following:

$$\begin{aligned}(1+x)^{k+1} = (1+x)^k(1+x) &\geq (1+kx)(1+x) \\ &\geq 1 + kx + x + kx^2 \\ &\geq 1 + (k+1)x + kx^2 \geq 1 + (k+1)x\end{aligned}$$

By induction, we have shown that $\forall n \in \mathbb{N}, (1+x)^n \geq 1 + nx$.

3 A Tricky Game

(a) CS 70 course staff invite you to play a game: Suppose there are n^2 coins in a $n \times n$ grid ($n > 0$), each with their heads side up. In each move, you can pick one of the n rows or columns and flip over all of the coins in that row or column. However, you are not allowed to re-arrange them in any other way. You have an unlimited number of moves. If you happen to reach a configuration where there is exactly one coin with its tails side up, you will win the game. Are you able to win this game? Find all values of n for which you can win the game, and prove your statement. In other words, for each value of n that you listed, prove that you can win the game; then, prove that it is impossible to win the game for all other values of n.

(b) (Optional) Now, suppose we change the rules: If the number of "tails" is between 1 and $n-1$, you win. Are you able to win this game? Does that apply to all n? Prove your answer.

Solution:

(a) When $n = 1$, the answer is trivial. Let's then analyze the base case when $n = 2$. We will prove the following lemma.

Lemma: The 2×2 puzzle is unwinnable.

Proof: Let P be the property that the number of coins in a configuration with heads side up on the 2×2 grid is even. Note that P is true initially, and moreover it is not disturbed by flipping of any row or column. Hence is a P is an invariant of the configuration. By induction on the number of moves, P holds after any number of moves, and so we can only reach configurations where P is true.

> (In other words, we let $Q(k)$ denote the proposition that P holds after any sequence of k moves. The base case $Q(0)$ holds trivially. Also, $Q(k) \implies Q(k+1)$ holds for all $k \geq 0$, since every sequence of $k+1$ moves can be decomposed into a sequence of k moves followed by one more move, and if P holds before this last move, it holds after the last move, too, since P is not disturbed by flipping of any single row or column. Therefore by induction $Q(k)$ holds for all $k \geq 0$.)

But now the configuration with exactly one coins tails side up is incompatible with P and consequently is unreachable by any finite set of moves. Therefore the 2×2 puzzle is unwinnable.

Now, we will use the lemma to prove the following theorem.

Theorem: The $n \times n$ puzzle is winnable if and only if $n = 1$.

Proof: We show that the puzzle is unwinnable when $n \geq 3$. Suppose not, i.e., there is some winning sequence of moves that leaves just a single coin with "tails" side up at some location L. Consider any 2×2 sub-grid containing location L. Then we have found a sequence of moves which takes this 2×2 sub-grid from the initial all-heads position to a position containing 3 heads and 1 tail. But this is impossible, by the previous result. Hence our assumption that there exists a winning sequence of moves must have been impossible, which proves the theorem.

(b) Similarly, when $n = 1$, the answer is trivial. We will prove the following theorem:

Theorem: The game remains unwinnable for all $n \geq 2$.

Proof: Consider any pair of rows of coins. Let P be the property that the two rows are identically configured, and Q be the property that these two rows are exact inverses of each other (i.e., each head in the first row corresponds to a tail in the second row, and vice versa). Then (for every pair of rows) $P \vee Q$ is an invariant of the game, since it holds initially and is not disturbed by any move.

Now there are only two cases: either there is some winning sequence of moves, or there is not. In the former case, in the winning final configuration some rows must have t tails, for some $1 \leq t \leq n-1$. This implies that every other row has either t or $n-t$ tails in it (according to whether it is P or Q that is true for this pair of rows), and since $t \geq 1$ and $n-t \geq 1$, this means that every row must have at least one coin with "tails" side up, for a total of at least n tails in the winning configuration. This is an absurdity. But this means the first case is impossible, hence the second case must always hold, and there can be no winning the puzzle when $n \geq 2$.

4 Binary Search

We have an array A, with sorted values from left to right. There are n values. Two arbitrary numbers are picked from 1 to n, denoting the start and end index of the search range, respectively. Given a value x, we want to use a binary search algorithm to see if this value is contained in the specified range. If x is found in this range, then the program should return the index of where it is in the array. However, if it is not found, it will return 0. An outline of the algorithm goes like this:

(1) Divide the search range in half and look at $A[m]$ where m is the middle index. If $A[m] = x$, we've found it and the program returns index m.

(2) If not, and $A[m] < x$, search the remaining right part of the array by setting the range from index $(m+1)$ to b. If $A[m] > x$, set the range from a to $(m-1)$.

(3) If the search range is empty, return 0. Otherwise, repeat from step (1).

(a) Write pseudocode (be as detailed as you can) of a recursive function that would follow the steps outlined above. No explicit `for` or `while` loops are allowed.

(b) We want to know if this pseudocode is doing what it's supposed to, returning a single index if x is in the search range or returning 0 if it's not. Try using induction to prove that your pseudocode is correct. Please outline all the steps in the proof.

Solution:

(a)
```
global array A
global variable x
```

```
function binary_search(x, start, end, A):
  if end < start
        return 0
  middle = (end + start) / 2
  if A[middle] = x
        return middle
  if A[middle] < x
        return binary_search(x, middle + 1, end, A)
  if A[middle] > x
        return binary_search(x, start, middle - 1, A)
```

start is the starting index of the search range, end is the ending index of the search range. If the search range has an even number of elements, we take $\lfloor (\texttt{start}+\texttt{end})/2 \rfloor$.

(b) We induct on the difference of start and end, $d = \texttt{end} - \texttt{start}$.
Base case: We have two base cases, $d = 0$ and $d = 1$.
For $d = 0$, we have a search range with 1 element. Obviously, $\texttt{start} = \texttt{end} = \texttt{middle}$ and if the value is equal to x, the program will return this index. If the value is not equal to x, there are two cases, $\texttt{x} < \texttt{A[middle]}$ and $\texttt{x} > \texttt{A[middle]}$.

- $\texttt{x} < \texttt{A[middle]}$: The ending index for the next step will be $\texttt{middle} - 1 = \texttt{start} - 1$, which is smaller than start, resulting in a return value of 0.
- $\texttt{x} > \texttt{A[middle]}$: Similarly, the starting index for the next step will be $\texttt{middle} + 1 = \texttt{end} + 1$, which is bigger than end, resulting in a return value of 0.

For $d = 1$, we have a 2-element array. Because we are taking the floor, $\texttt{middle} = \texttt{start}$. If $\texttt{x} = \texttt{A[middle]}$ then the program returns middle.

- $\texttt{x} < \texttt{A[middle]}$: Returns 0.
- $\texttt{x} > \texttt{A[middle]}$: Reduces to $d = 0$ case.

Induction hypothesis: We assume for all arrays with $d \leq n$, we can find the index of x or if not, the program returns a 0.

Inductive step: For an array with $d = n + 1$, we have two cases when d is even and d is odd. We denote d' as the new difference in the search range indices for the recursive call.

- d is odd: We consider $n = 2, 4, 6 \ldots$. The number of elements in the array is even, and $\texttt{middle} = (\texttt{start} + \texttt{end} - 1)/2$. If $\texttt{A[middle]} = \texttt{x}$, the program returns middle. Otherwise, if $\texttt{x} < \texttt{A[middle]}$, the recursive call gets a $d' = \texttt{middle} - 1 - \texttt{start} = (\texttt{start} + \texttt{end} - 1)/2 - 1 - \texttt{start} = (\texttt{end} - \texttt{start} - 3)/2 = n/2 - 1$, which is clearly $\leq n$, since $-2 \leq n \implies n/2 - 1 \leq n$.
  If $\texttt{x} > \texttt{A[middle]}$, $d' = \texttt{end} - (\texttt{middle} + 1) = \texttt{end} - (\texttt{start} + \texttt{end} + 1)/2 = (\texttt{end} - \texttt{start} + 1)/2 = n/2 + 1$, which is $\leq n$, since $2 \leq n \implies n/2 + 1 \leq n$.
- d is even: We consider $n = 1, 3, 5 \ldots$. The number of elements in the array is odd, and $\texttt{middle} = (\texttt{start} + \texttt{end})/2$. Similarly to the case above, the program will return

`middle` if `A[middle]` $=$ `x`. Otherwise, if `x` $<$ `A[middle]`, $d' = (n-1)/2$, and if `x` $>$ `A[middle]`, $d' = (n-1)/2$ (since symmetric). $d' \leq n$ since $-1 \leq n \implies (n-1)/2 \leq n$.

Thus, by strong induction, we have proved the proposition.

We want to note that in the above, we have written out all the steps in great detail where each step is a basic fact from high school algebra. In a solution, you don't need to write out the steps in that level of detail.