1. For any positive integer n, the nth ***Fibonacci string*** F_n is defined recursively as follows, where $x \bullet y$ denotes the concatenation of strings x and y:

$$F_1 := \texttt{0}$$
$$F_2 := \texttt{1}$$
$$F_n := F_{n-1} \bullet F_{n-2} \quad \text{for all } n \geq 3$$

induction

For example, $F_3 = \texttt{10}$ and $F_4 = \texttt{101}$.

(a) What is F_8?

(b) ***Prove*** that every Fibonacci string except F_1 starts with `1`.

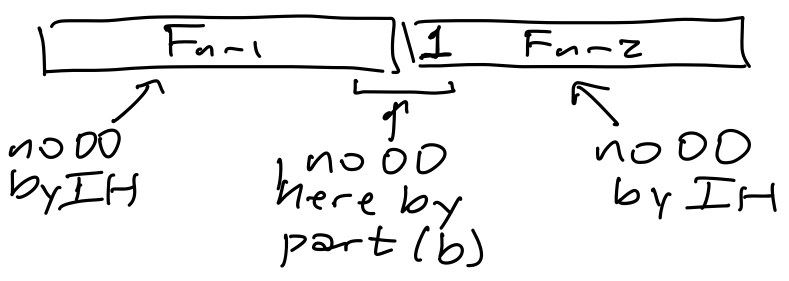
(c) ***Prove*** that no Fibonacci string contains the substring `00`.

(After review session)

(a) 3 : 10
4 : 101
5 : 10110
6 : 10110101
7 : $\underline{1011010110110}$
$\boxed{8 : 1011010110110\ 10110101}$

(b) Let n be any integer ≥ 2.
Assume for all $2 \leq m < n$ that F_m starts with 1
Two cases:
- $n = 2 \Rightarrow F_n = \textcircled{1}$ ✓
- $n > 2 \Rightarrow F_n = F_{n-1} \cdot F_{n-2} = \textcircled{1} \cdots \cdot F_{n-2}$ by IH ✓

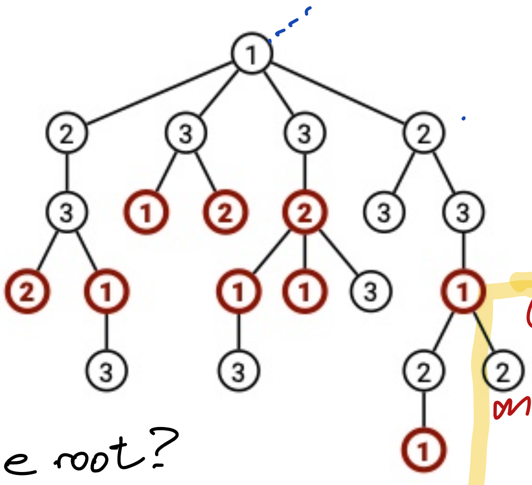
(c) Let n be any positive integer
Assume for all $m < n$ that F_m contains no 00
Two cases:
- $n \leq 8$ by inspection (see list above)
- $n > 8$

$F_n =$ 

(flavor text omitted)

More formally, suppose you are given a rooted tree T, representing the Twitbook company hierarchy. You need to label each vertex of T with an integer 1, 2, or 3, such that every node has a different label from its parent. The *cost* of a labeling is the number of vertices that have smaller labels than their parents. Describe and analyze an algorithm to compute the minimum cost of any labeling of the given tree T.

For example, the following figure shows a tree labeling with cost 9; the nine bold nodes have smaller labels than their parents. (This is *not* the optimal labeling for this tree.)



DP

Correct return value:

$$\min \begin{Bmatrix} \sum_{v \downarrow root} BS(v,1) \\ \sum_{v \downarrow root} BS(v,2) \\ \sum_{v \downarrow root} BS(v,3) \end{Bmatrix}$$

How do I label the root?

Subproblem:

BestScore(v,c) = Min score for subtree rooted at v if parent(v) is labeled c

We need... ~~$\min(BS(root,1), BS(root,2), BS(root,3))$~~

This is incorrect

$$BestScore(v,2) = \min \begin{Bmatrix} 1 + \sum_{w \downarrow v} BS(w,1) \\ 0 + \sum_{w \downarrow v} BS(w,3) \end{Bmatrix} \quad \begin{matrix} label(v)=1 \\ label(v)=3 \end{matrix}$$

$$BestScore(v,1) = \min \begin{Bmatrix} 0 + \sum_{w \downarrow v} BS(w,2) \\ 0 + \sum_{w \downarrow v} BS(w,3) \end{Bmatrix}$$

$$BestScore(v,3) = \min \begin{Bmatrix} 1 + \sum \text{——} 1) \\ 1 + \sum \text{——} 2) \end{Bmatrix}$$

off by one at root!

Memoize into T v.BS1 v.BS2 v.BS3

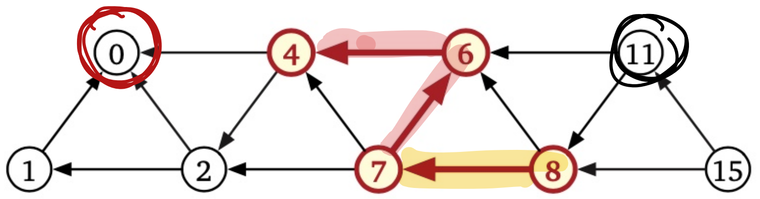
Evaluate postorder

$O(n)$ time

2. Let G be a ***directed*** acyclic! graph, where every vertex v has an associated height $h(v)$, and for every edge $u\to v$ we have the inequality $h(u) > h(v)$. Assume all heights are distinct. The *span* of a path from u to v is the height difference $h(u) - h(v)$.

Describe and analyze an algorithm to find the ***minimum span*** of a path in G with ~~at least~~ k edges. Your input consists of the graph G, the vertex heights $h(\cdot)$, and the integer k. Report the running time of your algorithm as a function of V, E, and k.

For example, given the following labeled graph and the integer $k = 3$ as input, your algorithm should return the integer 4, which is the span of the path $8\to7\to6\to4$.



DAG
DP

Define $MS(v,\ell) =$ min. span of a path starting at v with ~~$\geq$~~ ℓ edges

We need $\min_v MS(v,k)$

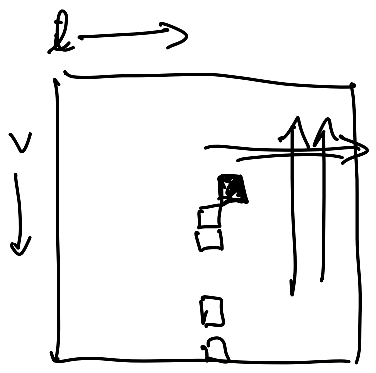
$$MS(v,\ell) = \begin{cases} 0 & \ell = 0 \\ \min\limits_{v\to w} \{h(v) - h(w) + MS(w,\ell-1)\} & \end{cases}$$

$(\min \emptyset = \infty)$

what's the first edge?

Memoize into array $v.MS[0..k]$ at every node v

```
Evaluate:   for all v in rev. top. order  ← postorder
              for ℓ ← 0 to k
                 for all edges v → w
```



Time: $O((V+E)k)$

Okay to assume G is weakly connected, so $V = O(E)$

Specifically: $\sum_v \sum_{\ell=0}^{k} O(\deg(v))$
$= k \cdot \sum_v O(\deg(v)) = O(Ek)$

1. For any two sets X and Y of integers, the Minkowski sum $X + Y$ is the set of all pairwise sums $\{x + y \mid x \in X, y \in Y\}$.

   (a) Describe an analyze an algorithm to compute the number of elements in $X + Y$ in $O(n^2 \log n)$ time, where $n = |X| + |Y|$. *[Hint: The answer is **not** always $|X| \cdot |Y|$.]*

   (b) Describe and analyze an algorithm to compute the number of elements in $X + Y$ in $O(M \log M)$ time, where M is the largest absolute value of any element of $X \cup Y$.

$-M \le x \le M$

FFT

(a)

```
S ← ∅
for x in X
    for y in Y
        add x+y to S
```

$O(n^2)$

$\underbrace{\text{sort } S}_{\boxed{O(n^2 \log n)}}$ (remove dupes) $\underbrace{\text{and count}}_{O(n^2)}$

(b)

Rep X as poly $\sum_{x \in X} z^{x+M} = P$ $\quad P[-M..M] = \begin{cases} 0 & x \notin X \\ 1 & x \in X \end{cases}$

Y $\sum_{y \in Y} z^{y+M} = Q$

Compute $P \cdot Q = R$ $\leftarrow$ $\boxed{O(M \log m)}$ time via FFT

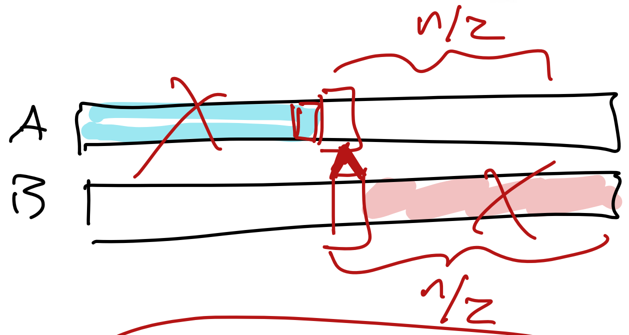
$\rightarrow$ Return # non-zero coeffs

$$R = \sum_{i=-2M}^{2M} a_i z^i \qquad a_i = \#\{x, y \mid x + y = i\}$$

3. Suppose you are given two sorted arrays $A[1..n]$ and $B[1..n]$ containing distinct integers. Describe a fast algorithm to find the median (meaning the nth smallest element) of the union $A \cup B$. For example, given the input

$$A[1..8] = [0, 1, 6, 9, 12, 13, 18, 20] \qquad B[1..8] = [2, 4, 5, 8, 17, 19, 21, 23]$$

your algorithm should return the integer 9. *[Hint: What can you learn by comparing one element of A with one element of B?]*



Mergesort
Binary search
Recursion
D+C

```
if n < 10^100!
    brute force
```

~~if A[n/2] = B[n/2] return A[n/2]~~

Beware OBOEs!

```
if A[n/2] > B[n/2]
    return median(A[1..n/2], B[n/2..n])
else
    return median(A[n/2..n], B[1..n/2])
```

$T(n) = T(n/2) + O(1)$

$O(\log n)$ time

To avoid off by one errors, we need to remove exactly the same number of elements from A and B

```
if A[⌊n/2⌋] > B[⌊n/2⌋]
    return median(A[1..⌈n/2⌉],
                  B[⌊n/2⌋+1..n])
else
    ....
```

Assume distinct values

2. You and your eight-year-old nephew Elmo decide to play a simple card game. At the beginning of the game, the cards are dealt face up in a long row. Each card is worth a different number of points. After all the cards are dealt, you and Elmo take turns removing either the leftmost or rightmost card from the row, until all the cards are gone. At each turn, you can decide which of the two cards to take. The winner of the game is the player that has collected the most points when the game ends.

   Having never taken an algorithms class, Elmo follows the obvious greedy strategy—when it's his turn, Elmo *always* takes the card with the higher point value. Your task is to find a strategy that will beat Elmo whenever possible. (It might seem mean to beat up on a little kid like this, but Elmo absolutely *hates* it when grown-ups let him win.)

   (a) Prove that you should not also use the greedy strategy. That is, show that there is a game that you can win, but only if you do *not* follow the same greedy strategy as Elmo.

   (b) Describe and analyze an algorithm to determine, given the initial sequence of cards, the maximum number of points that you can collect playing against Elmo.

DP

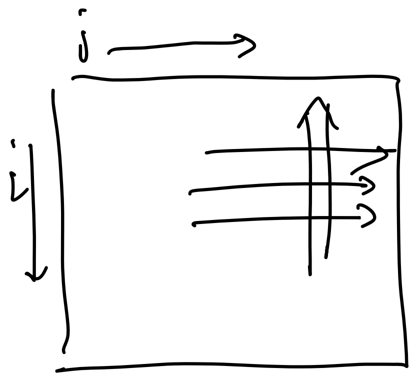
(After review session)

ⓐ Assuming I move first: $1\ 2\ 3\ \infty\ 5$

ⓑ Spose card values are $C[1..n]$

Let $MaxPts(i,j)$ = max # pts I can win from $C[i..j]$ assuming I move first

$$MaxPts(i,j) = \begin{cases} 0 & \text{if } i > j \\ C[i] & \text{if } i = j \\ \max \begin{cases} C[i] + MaxPts(i+2, j) & \text{if } C[i+1] > C[j] \\ C[i] + MaxPts(i+1, j-1) & \text{if } C[i+1] < C[j] \\ C[j] + MaxPts(i+1, j-1) & \text{if } C[i] > C[j-1] \\ C[j] + MaxPts(i, j-2) & \text{if } C[i] < C[j-1] \end{cases} \end{cases}$$



$O(n^2)$ time