CSCI 3100

- Backtracking (+ Recursion)
- Dynamic Programming

Announcements

- A bit late to office hours tomorrow
- Here 1-2pm today (ish)
- HW due on Friday
  - written, by start of class
  - in general, I'll often give out HW solutions ~1 class later, so on time is appreciated!

Today: Recap of recursion/backtracking (in Lecture Notes - 3)

- Find a small choice that reduces the problem size
- For each answer to the choice, choose answer + recurse

  (while considering only subsolutions consistent with that choice)

Last time: subset sum

Also in notes: - queens
- NFAs

Longest Increasing Subsequence or (LIS)

Given: List of integers $A[1..n]$

Goal: Find longest subsequence whose elements are strictly increasing

Formally: $A[1..n]$

Example:

[12, 5, 1, 3, 4, 13, 6, 11, 2, 20]

Best?

+ how to program?

Formalize:

The LIS of $A[1..n]$ is either:

- the LIS of $A[2..n]$
- $A[1]$ followed by LIS of $A[2..n]$

(or is it?)

Pseudocode

```
FILTER(A[1..n], x):
  j ← 1
  for i ← 1 to n
      if A[i] > x
          B[j] ← A[i];  j ← j + 1
  return B[1..j]
```

```
LIS(A[1..n]):
  if n = 0
      return 0
  else
      max ← LIS(prev, A[2..n])
      L ← 1 + LIS(A[1], FILTER(A[2..n], A[1]))
      if L > max
          max ← L
      return max
```

Runtime:

Correctness:

Dynamic Programming

- a fancy term for smarter recursion:

memoization

- Developed by Richard Bellman in mid-1950s

("programming" here actually means planning or scheduling)

Key: When recursing, if many recursive calls to overlapping subcases, remember prior results and don't do extra work!

Simple example:

Fibonacci Numbers

$F_0 = 0$, $F_1 = 1$, $F_n = F_{n-1} + F_{n-2}$ $\forall n \geq 2$

Directly get an algorithm:

```
FIB(n):
if n < 2:
    return n
else
    return FIB(n-1) + FIB(n-2)
```

Runtime:

Applying memoization:

```
MemFibo(n):
    if (n < 2)
        return n
    else
        if F[n] is undefined
            F[n] ← MemFibo(n − 1) + MemFibo(n − 2)
        return F[n]
```

Better yet:

```
ITERFIBO(n):
    F[0] ← 0
    F[1] ← 1
    for i ← 2 to n
        F[i] ← F[i−1] + F[i−2]
    return F[n]
```

Correctness:

Run time & space:

Even better!

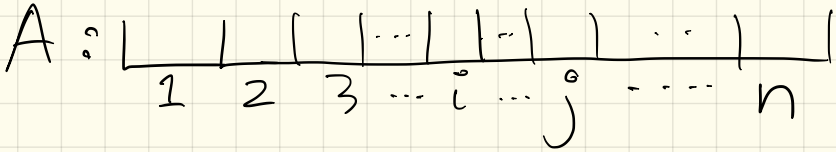
```
IterFibo2(n):
    prev ← 1
    curr ← 0
    for i ← 1 to n
        next ← curr + prev
        prev ← curr
        curr ← next
    return curr
```

Run time/space:

Back to LIS:

Some notation:

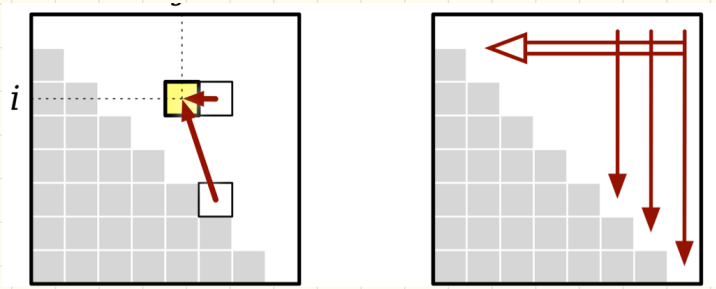
Let $LIS(i,j)$:= length of longest subsequence of $A[j..n]$ with elements $> A[i]$



Then:

$$LIS(i,j) = \begin{cases} 0 & \text{if } j > n \\ LIS(i, j+1) & \text{if } A[i] \geq A[j] \\ \max\{LIS(i, j+1),\ 1 + LIS(j, j+1)\} & \text{otherwise} \end{cases}$$

So, build a solution:



$$LIS(i,j) = \begin{cases} 0 & \text{if } j > n \\ LIS(i, j+1) & \text{if } A[i] \geq A[j] \\ \max\{LIS(i, j+1),\ 1 + LIS(j, j+1)\} & \text{otherwise} \end{cases}$$

Algorithm:

```
LIS(A[1..n]):
  A[0] ← −∞                          ⟨⟨Add a sentinel⟩⟩
  for i ← 0 to n                     ⟨⟨Base cases⟩⟩
      LIS[i, n+1] ← 0

  for j ← n downto 1
      for i ← 0 to j − 1
          if A[i] ≥ A[j]
              LIS[i, j] ← LIS[i, j+1]
          else
              LIS[i, j] ← max{LIS[i, j+1], 1 + LIS[j, j+1]}

  return LIS[0, 1]
```

Runtime or Space:

How to improve space?