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 HW solutions out in 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 and 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?

at how to program?
Formalize:
The LIS of $A[1..n]$ is either:
- the LIS of $A[2..n]$

(or is it?)
Pseudo code

```
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
```
Correctness:
Dynamic Programming
-a fancy term for smarter recursion:
Memorization
-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

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

Directly get an algorithm:

\[
\text{FIB}(n) := \\
\begin{cases} 
\text{if } n \leq 2: \\
\quad \text{return } n \\
\text{else} \\
\quad \text{return } \text{FIB}(n-1) + \text{FIB}(n-2)
\end{cases}
\]

Runtime:
Applying memoization:

```
\textbf{MemFibo}(n):
  \textbf{if} (n < 2)
    \textbf{return} n
  \textbf{else}
    \textbf{if} F[n] \textbf{is undefined}
      F[n] \leftarrow \textbf{MemFibo}(n-1) + \textbf{MemFibo}(n-2)
    \textbf{return} F[n]
```

Better yet:

**ITERFIBO**(n):

\[
\begin{align*}
F[0] & \leftarrow 0 \\
F[1] & \leftarrow 1 \\
\text{for } i & \leftarrow 2 \text{ to } n \\
F[i] & \leftarrow F[i - 1] + F[i - 2] \\
\text{return } F[n]
\end{align*}
\]

Correctness:

Runtime and Space:
Even better!

**`ITERFIBO2(n):`**

```
prev ← 1
curr ← 0
for i ← 1 to n
    next ← curr + prev
    prev ← curr
    curr ← next
return curr
```

**Runtime/Space:**
Some notation:

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

Next:

$A : \overline{1 \ 2 \ 3 \ \ldots \ i \ \ldots \ j \ \ldots \ n}$

Then:

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

\[
\text{LIS}(A[1..n]):
\]
\[
\begin{align*}
A[0] & \leftarrow -\infty \quad \text{\{Add a sentinel\}} \\
\text{for } i & \leftarrow 0 \text{ to } n \quad \text{\{Base cases\}} \\
& \quad \text{ } \quad \text{LIS}[i, n + 1] \leftarrow 0 \\
\text{for } j & \leftarrow n \text{ downto } 1 \\
& \quad \text{for } i \leftarrow 0 \text{ to } j - 1 \\
& \quad \quad \text{if } A[i] \geq A[j] \\
& \quad \quad \quad \text{LIS}[i, j] \leftarrow \text{LIS}[i, j + 1] \\
& \quad \quad \text{else} \\
& \quad \quad \quad \text{LIS}[i, j] \leftarrow \max\{\text{LIS}[i, j + 1], 1 + \text{LIS}[j, j + 1]\} \\
\end{align*}
\]

return \text{LIS}[0, 1]

Runtime \& Space
How to improve space?