Rod Cutting

Given integers $P[1..n]$.
How to cut up rod into integer length pieces of max total cost?

Cost of piece of length i is $P[i]$.

Backtracking:

For each possible first length, check total possible using recursion to compute prodit from remains.

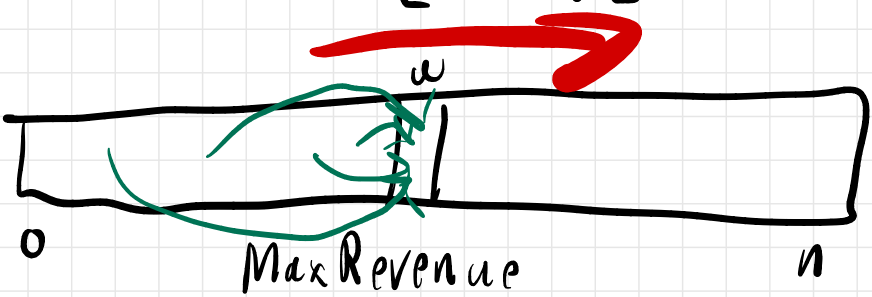
MaxRevenue(i): total we can earn cutting up rod of length i.

Final answer is MaxRevenue(n).

$$\text{MaxRevenue}(i) = \begin{cases} 0 & \text{if } i = 0 \\ \max\limits_{1 \le j \le i} \{ p[j] + \text{MaxRevenue}(i-j) \} & \text{o.w.} \end{cases}$$

Memoize in array

MaxRevenue[0..n]



So fill from left to right

($i \leftarrow 0$ to n)

```
FastRodCutting(n, P[1 .. n]):
    MaxRevenue[0] ← 0
    for i ← 1 to n
        MaxRevenue[i] ← −∞
        for j ← 1 to i
            if P[j] + MaxRevenue[i − j] > MaxRevenue[i]
                MaxRevenue[i] ← P[j] + MaxRevenue[i − j]
    return MaxRevenue[n]
```

$O(n^2)$ time

$O(n)$ space

Could return lists of cuts by remembering best j for each i.

Dynamic Programming

<u>recursion</u> without repetition

Erickson Section 3.4

1) Formulate problem recursively

a) <u>Specification</u>: Give a precise definition of the recursive subproblems. Also, what is the real answer to original problem?

b) Solution: Recursive alg

or recurrence.

recommended

2) Build solutions bottom up using some appropriate data structure.

a) Identify subproblems.

RecFibo + MaxRevenue used

$i \in \{0, \ldots, n\}$.

b) Choose a memoization data structure. (an array?)

c) Find an evaluation order.

d) Analze space & time.

space: # subproblems

time: at most

subproblems ·

time per subproblem

e) Write the algorithm.

for loops for eval order

copy-paste the recurrence

to fill in table

WARNING:

Don't be greedy!

(yet)

Longest Increasing Subsequence

Given a sequence S, a <u>subsequence</u> of S comes from deleting some elements but not reordering.

<u>substring</u>: subsequence but elements are contiguous.

Given a sequence $A[1..n]$ of integers. Find (length of) longest subsequence of A s.t. elements are increasing.

In other words, want max length $1 \le i_1 < i_2 < \dots < i_\ell \le n$ s.t. $A[i_k] < A[i_{k+1}]$ for all k.

3 1 4 1 5 9 2 6 5 3 | $5^?$ 8 9 7 9 3 2 3 8 4 6 2 6

take the 5?

no!

3 1 4 1 5 9 2 6 5 3 5 | $8^?$ 9 7 9 3 2 3 8 4 6 2 6

take the 8?

check both options with recursion!

Subproblems based on:

- first index of the suffix
- largest (previous) element that came before

(try to use as few subproblem

parameters as possible)

LISbigger(i,j): length of LIS of $A[j \ldots n]$ s.t. all elements greater than $A[i]$ (the "previous" element)

Let $A[0] := -\infty$.

Want to return LISbigger$(0,1)$

LISbigger(i,j) =

$$\begin{cases} 0 & \text{if } j > n \\ \text{LISbigger}(i, j+1) & \text{if } A[i] \geq A[j] \\ \max\{\text{LISbigger}(i, j+1), \; 1 + \text{LISbigger}(j, j+1)\} & \text{o.w.} \end{cases}$$

Subproblems: $0 \le i \le n$

$1 \le j \le n+1$

Data structure:

2D array

$LISbigger[0..n, 1..n+1]$

Dependencies + eval order:

Space: $O(n^2)$

Time: $O(n^2) \cdot O(1) = O(n^2)$

```
FastLIS(A[1 .. n]):
  A[0] ← −∞                          ⟨⟨Add a sentinel⟩⟩
  for i ← 0 to n                     ⟨⟨Base cases⟩⟩
      LISbigger[i, n+1] ← 0
  for j ← n down to 1
      for i ← 0 to j − 1             ⟨⟨...or whatever⟩⟩
          keep ← 1 + LISbigger[j, j+1]
          skip ← LISbigger[i, j+1]
          if A[i] ≥ A[j]
              LISbigger[i, j] ← skip
          else
              LISbigger[i, j] ← max{keep, skip}
  return LISbigger[0, 1]
```