DP Example 2: Rod cutting

Module 4: techniques

The problem: We are given a long steel rod and we need to cut it into shorter rods which we then sell. Making each cut is free and all rod lengths are always integers. Assume we are given, for each $i = 1, 2, 3, ...$, the price p_i (in dollars) that we can sell a rod of length i. Goal: Given a rod of length n inches and a table of prices p_i for $i = 1, 2, 3, ..., n$, determine the maximal revenue obtainable by cutting up the rod and selling the pieces.

Example: Find the maximal revenue for a rod of length $n = 10$ obtainable with the prices below.

length	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	19	17	17	20	24	30

First steps, towards understanding the problem: Draw all possible ways a rod of length n can be cut, for $n = 1, 2, 3, 4$. Write down the revenue of the cut in each case.

$n = 1$

$n = 2$

$n = 3$

$n = 4$

Question: How many different cuts for a rod of length n?

Answer: You have the choice of $n-1$ cuts: you can cut at distance $1, 2, ..., n-1$ from the beginning of the rod. Can view each cut as a binary variable, with values 0 (no cut) or 1 (cut). There are 2^{n-1} different combinations. Each one corresponds to a different way to cut the rod (but note that different cuts might result in the same set of rods, and thus have the same cost).

Choice of subproblem. This problem has one parameter, which is the size of the rod. When we make a cut, we get segments of smaller sizes. We denote by r_n or $r(n)$ the maximal revenue obtainable by cutting up a rod of length n.

Optimal substructure: Assume someone told us that the first (left-most) cut in the optimal solution was at distance i from the beginning; thus the first piece has length i.
Claim: Then it has to be that $r(n) = p_i + r(n-i)$. In other words, the optimal revenue consists of the price of that first piece p_i plus the optimal revenue obtainable for the remaining rod. Basically this says that if we want maximal revenue for a rod of length n, once we determined a cut, we want maximal revenue for the remaining piece.
Proof: By contradiction.. Assume $r(n) = p_i + x$. If $x < r(n-i)$ then we found a better revenue for a rod of length $n-i$ which is not possible, because $r(n-i)$ represents the optimal cost obtainable from a rod of length $n-i$.

Recursive formulation: Once we established that the problem has optimal substructure (the optimal solution consists of optimal solutions to sub-problems), we can use it in the following way:

If we knew where the first (leftmost) cut was, we'd recurse from there. But we don't know where the first cut is, so, so we have to consider all options: the first cut can be at distance 1 from the start, or at distance 2, or 3, ..., or n (in this case, there is no cut). The maximal revenue is the largest revenue obtainable by one of these choices. Thus we get

$$r_n = \max\{p_1 + r(n-1), p_2 + r(n-2), ..., p_i + r(n-i), ..., p_{n-1} + r(1), p_n\}$$

```
//Note: the price array p[1..n] is a global variable
//Output: returns the maximal revenue obtainable by cutting up a rod of length x.
maxrev(x)

    if (x ≤ 0): return 0

    For i = 1 to x: compute p[i] + maxrev(x − i) and keep track of max

    RETURN this max
```

Correctness: It tries *all* possibilities for first cut and recurses on the rest (correct bec. of optimal substructure).

Analysis: We can write a recurrence for the running time $T(n)$ of the recursive algorithm $maxrev(n)$ and show that it is exponential $T(n) = \Omega(2^{n/2})$.

(to do)

Overlapping subproblems: Why is $maxrev(n)$ inneficient? Draw the recursion tree for $n = 4$. How many different sub-problems are there in total, and can a problem be solved multiple times?

Rod cutting: DP solution with memo-ized recursion

We create a table of size $[0..n]$, where $table[i]$ will store the result of $maxrev(i)$. We initialize all entries in the table as 0. We call $maxrevDP(n)$ and return the result.

//Note: the prices $p[1..n]$ and the table $table[1..n]$ are global variables
//Output: returns the maximal revenue obtainable by cutting up a rod of length x.
maxrevDP(x)

if $(x \leq 0)$: return 0

IF $table[x] \neq 0$: RETURN $table[x]$

For i = 1 to x: compute $p[i] +$ `maxrevDP`$(x - i)$ and keep track of max

$table[x] =$ max

RETURN $table[x]$

Running time for $maxrevDP(n) : \Theta(n^2)$

Rod cutting: iterative DP solution

// Input: the prices $p[1..n]$
// Output: returns the maximal revenue obtainable by cutting up a rod of length n.
maxrevDP_iterative()

create $table[0..n]$ and initialize $table[i] = 0$ for all i

for $(k = 1; k \leq n; k++)$

for $(i = 1; i \leq k; i++)$

set $table[k] =$ `max`$\{table[k], p[i] + table[k - i]\}$

RETURN $table[n]$

Running time for $maxrevDP\_iterative(n) : \Theta(n^2)$

From optimal revenue to the cuts

- Computing full solution (without storing additional information while filling the table):

Input: The table $table[0..n]$ as computed above, where $table[i]$ stores the maxrev obtainable from a rod of length i. And the prices $p[1..n]$.
Output: the set of cuts corresponding to $table[n]$

$curLength = n$
while ($curLength > 1$) do:

    for ($i = 1; i \leq curLength; i++$)

        //is the value $table[curLength]$ achieved via a first cut of length i ?
        if $table[curLength] == (p[i] + table[curLength - i])$:
            output that a cut of length i was made
            $curLength = curLength - i$

Running time: $\Theta(n^2)$, no extra space

- Computing full solution (with storing additional information while filling the table):

In addition to $table[0...n]$ we use an array $firstcut[0..n]$ where $firstcut[i]$ will store the first cut in $maxrev(i)$. We can extend the algorithm for computing maxrevDP(x) (either recursive or iterative) to also compute $firstcut[x]$: basically if the maximum revenue for x is achieved with the first cut being of length k, we'll store that $firstcut[x] = k$.

Input: The table $table[0..n]$ as computed above, where $table[i]$ stores the maxrev obtainable from a rod of length i. And $firstcut[0..n]$ where $firstcut[i]$ will store the first cut in $maxrev(i)$.
Output: the set of cuts corresponding to $table[n]$

$curLength = n$
while ($curLength > 1$) do:

    output a cut of length $firstcut[curLength]$

    $curLength = curLength - firstcut[curLength]$

Running time: $\Theta(n)$, with $\Theta(n)$ extra space for $firstcut[0..n]$