# Rod Cutting

## Reconstructing the optimal solution

```
Extended-Bottom-Up-Cut-Rod(p, n)
let r[0..n] and s[0..n] be new arrays
r[0] = 0
for j = 1 to n
    q = -infinity
    for i = 1 to j
        if q < p[i] + r[j - i]
            q = p[i] + r[j - i]
            s[j] = i
    r[j] = q
return r and s
```

`s[j] = 1` tracks the best cut location so far for length $j$.

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | |
|---|---|---|---|---|---|---|---|---|---|---|
| $r[i]$ | 0 | 1 | 5 | 8 | 10 | 13 | 17 | 18 | 22 | Max Revenue |
| $s[i]$ | 0 | 1 | 2 | 3 | 2 | 2 | 6 | 1 | 2 | $1^{st}$ cut position |

## Optimal Substructre

- Create an optimal solution to a problem using optimal solutions to subproblems.
- We can't use DP if the optimal solution to a problem might not require subproblem solutions to be optimal
  $\Rightarrow$ This often happens when the subproblemsn are **not independent** of each other.

## Overlapping Subproblems

- For DP to be useful, the recursive algorihtm should require us to compute optimal solutions to the **same subproblems** over and over again.
- In total, there should be a small number of distinct subproblems (i.e. polynomial in the input size).

DP can both be **top-down** or **bottom-up**.

# 2-Dimensional Example: Longest Common Subsequence

Problem: Give 2 sequences, $X =< x_1, x_2, \ldots, x_m >$ and $Y = y_1, y_2, \ldots, y_n$, find a common subsequence whose lengith is maximum. Subsequence need not be

consecutive, but *must be in order*.

| X | Y | X ∩ Y |
|---|---|---|
| springtime | printing | printi |
| ncaa tournament | north carlina | ncarna |
| basketball | snoeyink | se |
| ABCBDAB | BDCABA | BCBA |

## Naive Appraoch

For every subsequence of $X$, check whether it's a subsequence of $Y$.

Suppose $m$ is the length of $X$ and $n$ is the length of $Y$. There are $2^m$ subsequences of $X$, so the time complexity is $O(2^m)$.

$$\text{Runtime} = \Theta(n \cdot 2^m)$$

## Optimal substructure in LCS

- The last elements of $X$ and $Y$ are equal. Then they must both be part of the LCS.
- So we can chop both elements off the ends of the subsequence (adding them to a common subsequence) and find the longest common subsequence of the smaller sequences.

$$\begin{aligned} \text{If } X = ABCBDAB \quad Y = BDCABAB \qquad Z =< z_1, \ldots, B > \\ X = ABCBDA \quad Y = BDCABA \qquad Z =< z_1, \ldots, A, B > \end{aligned}$$

**Subproblem 1** $X = ABCBD$ and $Y = BDCA$

**Subproblem 2** $X = ABCB$ and $Y = BDCAB$

**Theorem 1** *Let $Z =< z_1, \ldots, z_k >$ be any LCS of $X$ and $Y$.*

1. *If $x_m = y_n$, then $z_k = x_m = y_n$ and $Z_{k-1}$*

Define $c[i, j]$ = length of LCS of $X_i$ and $Y_j$. We want $c[m, n]$.

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j, \\ \max(c[i, j-1], c[i-1, j]) & \text{if } i, j > 0 \text{ and } x_i \neq y_j \end{cases}$$

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j, \\ \max(c[i, j-1], c[i-1, j]) & \text{if } i, j > 0 \text{ and } x_i \neq y_j \end{cases}$$

$$LCS[i,j] = \Big\{ 1 + LCS[i-1, j-1] \quad \text{if } x_i = y_j,$$

## Recursive Solution (for getting length)

| | j | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|---|
| i | | $y_j$ | B | D | C | A | B | A |
| 0 | $x_i$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | A | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 2 | B | 0 | 1 | 1 | 1 | 1 | 2 | 2 |
| 3 | C | 0 | 1 | 1 | 2 | 2 | 2 | 2 |
| 4 | B | 0 | 1 | 1 | 2 | 2 | 3 | 3 |
| 5 | D | 0 | 1 | 2 | 2 | 2 | 3 | 3 |
| 6 | A | 0 | 1 | 2 | 2 | 3 | 3 | 4 |
| 7 | B | 0 | 1 | 2 | 2 | 3 | 4 | 4 |

```
LCS-Length(X, Y)
    m <- length[X]
    n <- length[Y]
    c[0, 0] <- 0
    for i <- 1 to m
        c[i, 0] <- 0
    for j <- 1 to n
        c[0, j] <- 0
    for i <- 1 to m
        for j <- 1 to n
            if x[i] = y[j]
                c[i, j] <- c[i-1, j-1] + 1
            elseif c[i-1, j] >= c[i, j-1]
                c[i, j] <- c[i-1, j]
            else
                c[i, j] <- c[i, j-1]
    return c[m, n]
```

This returns the length. how do we get the actual LCS?

```
LCS-Length(X, Y, m, n)
    let b[1..m, 1..n] and c[0..m, 0..n] be new tables
    for i = 1 to m
        c[i, 0] = 0
    for j = 1 to n
        c[0, j] = 0
    for i = 1 to m
        for j = 1 to n
            if x[i] = y[j]
                c[i, j] = c[i-1, j-1] + 1
                b[i, j] = "north-west arrow"
            elseif c[i-1, j] >= c[i, j-1]
                c[i, j] = c[i-1, j]
                b[i, j] = "↑"
```

```
        else
            c[i, j] = c[i, j-1]
            b[i, j] = "←"
return c and b
```