http://www.cs.unh.edu/~ruml/cs758

Rod Cutting

The Problem

Given table of profits p_i for each possible integer length i, find the best way to cut a rod of length n. Cuts are free, but must be of integer length.

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

$\approx 2^{n-1}$ possible solutions! How to solve in $O(n^2)$ time?

Optimal Value

Step 1: write down value of optimal solution

$$\begin{aligned} best(n) &= \text{best profit achievable for length } n \\ best(n) &= \max_{first=1}^{n} \left(p_{first} + best(n - first)\right) \\ best(0) &= 0 \end{aligned}$$

What is the complexity of the naive recursive algorithm?
How to make this efficient?

Step 2: compute optimal value (top-down or bottom-up)

1. best[0] $\leftarrow 0$
2. for len from 1 to n
3. best[len] $\leftarrow \max_{\text{first}=1}^{\text{len}} (p_{\text{first}} + \text{best}[\text{len} - \text{first}])$
4. best[n]

Will this access uninitialized data?
What is the complexity?

Solution Recovery


```
best[0] ← 0
cut[0] ← 0
for len from 1 to n
    best[len] ← −∞
    for first from 1 to len
        this ← p_first + best[len − first])
        if this > best[len]
            best[len] ← this
            cut[len] ← first
print best[n]
while n > 0
    print cut[n]
    n ← n − cut[n]
```

Properties

- optimal substructure: global optimum uses optimal solutions of subproblems
- ordering over subproblems: solve 'smallest' first, build 'larger' from them
- 'overlapping' subproblems: polynomial number of subproblems, each possibly used multiple times
- independent subproblems: optimal solution of one subproblem doesn't affect optimality of another

Optimal Substructure

shortest path

- path to any intermediate vertex along optimal path must be optimal path to that vertex. otherwise, could be shorter.

longest simple path

- path to an intermediate vertex along optimal path may not use vertices used elsewhere: subproblems are not independent.

Break

- asst 4
- asst 5

Two-Dimensional Dynamic Progamming

Longest Common Subsequence

Given two strings, x of length m and y of length n, find a common (non-contiguous) subsequence that is as long as possible.

$x = \texttt{ABCBDAB}$
$y = \texttt{BDCABA}$

Longest Common Subsequence

Given two strings, x of length m and y of length n, find a common (non-contiguous) subsequence that is as long as possible.

$x =$ `ABCBDAB`
$y =$ `BDCABA`

LCS $=$ `BCBA` or `BCAB`

$x' =$ `AB-C-BDAB`
$y' =$ `-BDCAB-A-`

What is the complexity of the naive algorithm?
How to make this efficient?

Recursive Approach

$LCS(i, j)$ means length of LCS considering only up to x_i and y_j

$LCS(i,j)$ means length of LCS considering only up to x_i and y_j

$$LCS(i,j) = \begin{cases} 0 & \text{if } i \text{ or } j = 0 \\ LCS(i-1,j-1)+1 & \text{if } x_i = y_j \\ \max(LCS(i-1,j), \\ \qquad LCS(i,j-1)) & \text{otherwise} \end{cases}$$

Optimal Substructure

global optimum uses optimal solutions of subproblems

Proof by contradiction: What if subsolution were not optimal?

Let z be an $LCS(i, j)$ of length k.

1. If $x_i = y_j$, then $z_k = x_i = y_j$ and $LCS(i-1, j-1) = z_0..z_{k-1}$.
   Not including z_k makes LCS suboptimal: contradiction!
   If $z_0..z_{k-1}$ were not LCS, z could be longer, hence not optimal: contradiction!
2. If $x_i \neq y_j$ and $z_k \neq x_i$, then z is $LCS(i-1, j)$.
   If longer exists, z would not be an LCS: contradiction!
3. If $x_i \neq y_j$ and $z_k \neq y_j$, then z is $LCS(i, j-1)$
   Similar to 2.

Summary of Dynamic Programming

1. optimal substructure: global optimum uses optimal solutions of subproblems
2. ordering over subproblems: solve 'smallest' first, build 'larger' from them
3. 'overlapping' subproblems: polynomial number of subproblems, each possibly used multiple times
4. independent subproblems: optimal solution of one subproblem doesn't affect optimality of another

- top-down: memoization
- bottom-up: compute table, then recover solution

EOLQs

For example:

- What's still confusing?
- What question didn't you get to ask today?
- What would you like to hear more about?

Please write down your most pressing question about algorithms and put it in the box on your way out.
Thanks!