CS 4800: Algorithms & Data

Lecture 7

January 30, 2018

Dynamic Programming

Log cutter

- Cut big piece of wood into boards
- i-inch board is worth p_i
- Want to make most money from n-inch thick raw material

An example

- n=4
- p_1=1, p_2=5, p_3=8, p_4=7
- Greedy!
  - Greedy1: Cut the board that is worth the most. Repeat.
    - Total value = ?
  - Greedy2: Cut the board with best ratio money/material.
    - Total value = ?
- What is optimal solution?

Observation

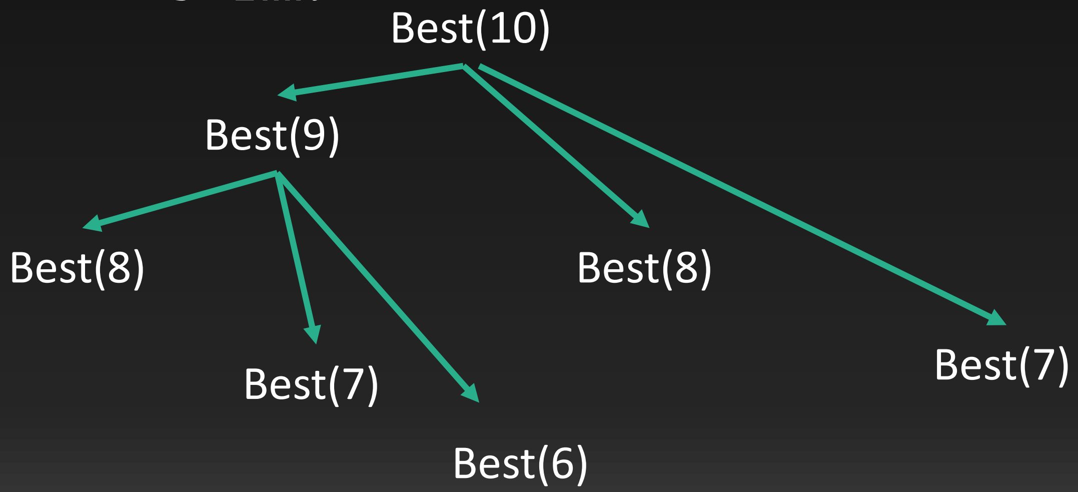
- Consider optimal solution
- Say the first board to cut is of size s
- n-s units remain
- Claim. The rest of the solution is optimal for n-s
- Proof. If not, substitute in the best solution for size n-s and get a better solution for size n

Optimal substructure!

- Choice to make: pick s from 1,2,…,n

Recursive solution

- Best(n):
  - If n = 0, return 0
  - Return $\max_{s=1 \ldots n} (p_s + Best(n-s))$



Memoization

- Initialize $Best[0] \leftarrow 0$
- ComputeBest(i):
  - If Best[i] is calculated, return Best[i]
  - Else
    - $Best[i] \leftarrow \max_{s=1\ldots i}\left(p_s + ComputeBest(i-s)\right)$
    - Return Best[i]

Bottom-up style

- Best[0] = 0
- For i from 1 to n
  - $Best[i] \leftarrow \max_{s=1\ldots i}(p_s + Best[i-s])$

Implementation in python

```
def logcutter(p, n):
    best = [-1] * (n+1)    #-1 means not computed
    best[0] = 0

    def compute_best(i):
        if best[i] == -1:
            for j in range(1, i+1):
                tmp = p[j-1] + compute_best(i-j)
                if tmp > best[i]:
                    best[i] = tmp
        return best[i]

    return compute_best(n)
```

Caution: this is not a recommended style unless you know how to set your stack size. This style can run into stack overflow!

Implementation in python

```
def logcutter(p, n):
    best = [-1] * (n+1)
    best[0] = 0

    for i in range(1, n+1):
        for j in range(1, i+1):
            tmp = p[j-1] + best[i-j]
            if tmp > best[i]:
                best[i] = tmp

    return best[n]
```

Retrace whole solution

```
def logcutter1(p, n):
    best = [-1] * (n+1)    #-1 means not computed
    best[0] = 0
    choice = [0] * (n+1)

    for i in range(1, n+1):
        for j in range(1, i+1):
            tmp = p[j-1] + best[i-j]
            if tmp > best[i]:
                best[i] = tmp
                choice[i] = j

    i = n
    while i > 0:
        print('cut a board of thickness %d'%(choice[i]))
        i -= choice[i]
```

Dynamic Programming

- Optimal substructure: reduce large problem to small problems
- Memoization

Coin change

- Coin of denominations d_1, d_2, …, d_k
- Wants to make change for n cents using as few coins as possible

Example

- $k=3$, $d_1=1$, $d_2=15$, $d_3=25$
- Wants to make change for 30 cents
- Greedy!
  - Pick coin of maximal value not exceeding the remaining change. Repeat.
  - How many coins?
- What is optimal solution?

Memoization

- Initialize $Best[0] \leftarrow 0$
- $ComputeBest(v)$:
  - If $Best[v]$ is calculated, return $Best[v]$
  - Else
    - $Best[v] \leftarrow 1 + \min_{i=1\ldots k, d_i \leq v} ComputeBest(v - d_i)$
    - Return $Best[v]$

Bottom-up style

- $Best[0] \leftarrow 0$
- For v from 1 to n
  - $Best[v] \leftarrow 1 + \min_{i=1\ldots k, d_i \le v} Best[v - d_i]$