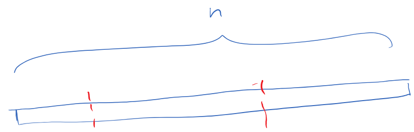
csi503-s21-lecture07

Tuesday, March 2, 2021

Dynamic Programming



example: Rod cutting

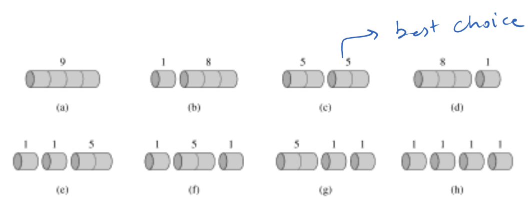
input: length n and prices p_i for $i = 1, \dots, n$

output: maximum revenue

length	0	1	2	3	4	5	6	7	8
price p_i		1	5	8	9	10	17	17	20

for size $n \longrightarrow 2^{n-1}$ different ways to cut it

cutting $n = 4$:



i	r_i (max revenue)	optimal solution
1	1	no cuts
2	5	"
3	8	"
4	10	2+2
5	13	2+3
6	17	no cuts (6)
7	18	1+6 or 2+2+3
8	22	2+6

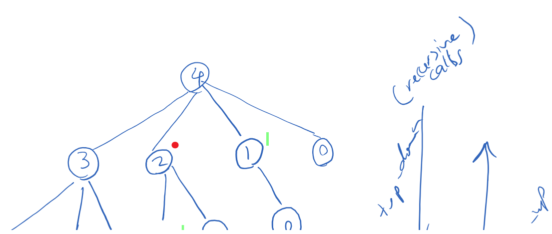
need to compare:
$$\begin{cases} p_n & : \text{no cuts} \\ r_1 + r_{n-1} \\ r_2 + r_{n-2} \\ \vdots \\ r_{n-1} + r_1 \end{cases}$$

$$r_n = \max(p_n, r_1 + r_{n-1}, \dots, r_{n-1} + r_1)$$

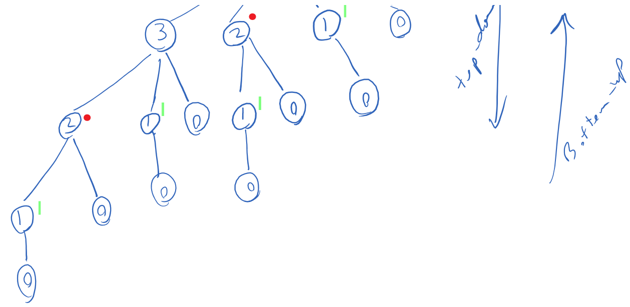
simplify this

$$r_n = \max_{1 \le i \le n} (p_i + r_{n-i})$$

CUT-ROD(p, n)
if $n == 0$
return 0



```
CUT-ROD(p, n)
  if n == 0
      return 0
  q = -∞
  for i = 1 to n
      q = max(q, p[i] + CUT-ROD(p, n - i))
  return q
```



inefficient : $T(n) = \begin{cases} 1 & n=0 \\ 1 + \sum_{j=0}^{n-1} T(j) & n>0 \end{cases} \Longrightarrow T(n) = 2^n$

avoid repetitions ⟶ dynamic programming

↳ rather than recomputing subproblems, store & reuse

```
MEMOIZED-CUT-ROD(p, n)
  let r[0..n] be a new array
  for i = 0 to n
      r[i] = -∞
  return MEMOIZED-CUT-ROD-AUX(p, n, r)
```

Top-down solution

```
MEMOIZED-CUT-ROD-AUX(p, n, r)
  if r[n] ≥ 0
      return r[n]          ← lookup step
  if n == 0
      q = 0
  else q = -∞
      for i = 1 to n
          q = max(q, p[i] + MEMOIZED-CUT-ROD-AUX(p, n - i, r))
  r[n] = q
  return q
```

Bottom-up Solution

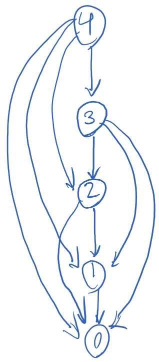
```
BOTTOM-UP-CUT-ROD(p, n)
  let r[0..n] be a new array
  r[0] = 0
  for j = 1 to n          → total length of the rod that is considered
      q = -∞
      for i = 1 to j      → left-most cut
          q = max(q, p[i] + r[j - i])
      r[j] = q
  return r[n]
```

⟶ $\Theta(n^2)$

subproblem graph : condensed version of recursion tree

↳ Directed Acyclic Graph

→ Directed Acyclic Graph



time complexity depends on each node + outgoing edges

say you have n nodes
m edges $\Longrightarrow \theta(n+m)$

for this problem $\Longrightarrow \theta(n^2)$

producing optimal solution:

```
EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
  let r[0..n] and s[1..n] be new arrays
  r[0] = 0
  for j = 1 to n
      q = -∞
      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
```

→ optimal cut for length j

```
PRINT-CUT-ROD-SOLUTION(p, n)
  (r, s) = EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
  while n > 0
      print s[n]
      n = n - s[n]
```

Problem: Longest Common Subsequence (LCS)

$X = \langle x_1, \ldots x_m \rangle$
$Y = \langle y_1, \ldots y_n \rangle$ $\Longrightarrow$ LCS: same sequence of characters but not consecutive necessarily

example: spring time
pioneer $\Longrightarrow$ LCS = pine

horse back
snow flake $\Longrightarrow$ LCS = oak

naive (brute force) solution: - all subsequences of X → 2^m

- check existence of subsequence in Y in $\theta(n)$

$\Longrightarrow$ total cost = $\theta(n\, 2^m)$

Form optimal substructure:

X_i = prefix $\langle x_1, \ldots x_i \rangle$, Y_i

Theorem: Let $Z = \langle z_1, \ldots, z_k \rangle$ be any LCS of X and Y.

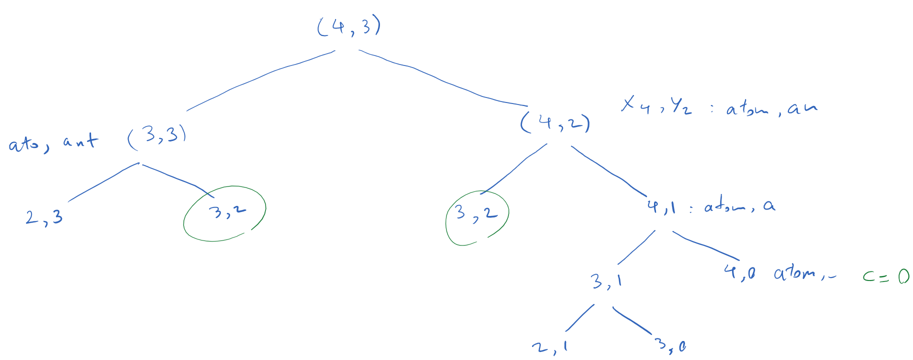
$$\begin{cases} \text{if } x_m = y_n \text{ then } z_k = x_m = y_n \text{ and } Z_{k-1} \text{ is in LCS of } X_{m-1} \text{ and } Y_{n-1} \\ \text{if } x_m \neq y_n \text{ then if } z_k \neq x_m \Rightarrow Z \text{ is an LCS of } X_{m-1} \text{ and } Y \\ \text{if } x_m \neq y_n \text{ then if } z_k \neq y_n \Rightarrow Z \text{ is an LCS of } X \text{ and } Y_{n-1} \end{cases}$$

Recursive formulation: $c[i,j]$: length of LCS of X_i & Y_j

→ $c[m,n]$ result wanted

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

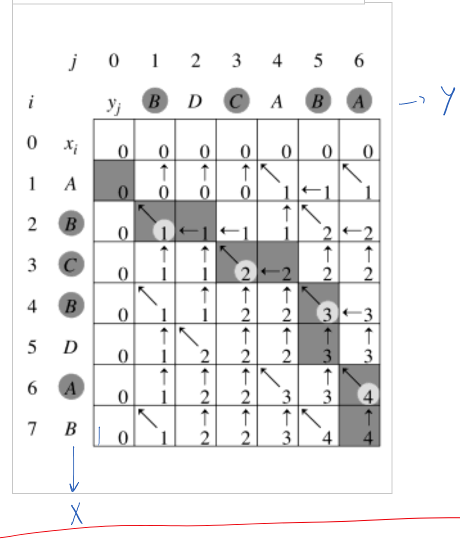
example: X = atom $m=4$
Y = ant $n=3$



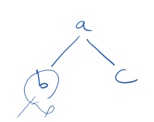

```
LCS-LENGTH(X, Y, m, n)
  let b[1..m, 1..n] and c[0..m, 0..n] be new tables
  for i = 1 to m
      c[i,0] = 0
  for j = 0 to n
      c[0,j] = 0
  for i = 1 to m
      for j = 1 to n
          if x_i == y_j
              c[i,j] = c[i-1, j-1] + 1
              b[i,j] = "↖"
          else if c[i-1, j] ≥ 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
```

$\Theta(mn)$

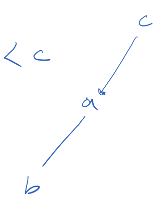
```
PRINT-LCS(b, X, i, j)
  if i == 0 or j == 0
      return
  if b[i, j] == "↖"
      PRINT-LCS(b, X, i − 1, j − 1)
      print x_i
  elseif b[i, j] == "↑"
      PRINT-LCS(b, X, i − 1, j)
  else PRINT-LCS(b, X, i, j − 1)
```



Problem: Optimal Binary Search Tree



$\longrightarrow b < a < c$



Set of search keys: n keys $\langle k_1, k_2, \dots, k_n \rangle$

sorted

probability of each key being searched is different, p_i

Goal: expected search cost is minimum

- for key k_i : cost of search: $depth(k_i) + 1$

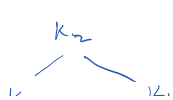
- $E[\text{cost of search}] = \sum_{i=1}^{n} cost_i \cdot p_i$

$$= \sum_{i=1}^{n} (depth(k_i) + 1) \cdot p_i$$

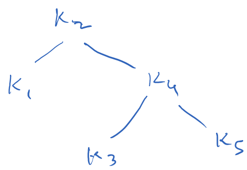
$$= \sum_{i=1}^{n} depth(k_i) \cdot p_i + \underbrace{\sum_{i=1}^{n} p_i}_{=1}$$

$$= 1 + \sum_{i=1}^{n} depth(k_i) \cdot p_i$$

i	1	2	3	4	5
p_i	.25	.2	.05	.2	.3

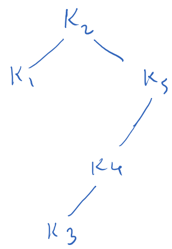


i	depth(k_i)	depth(k_i) . p_i
1	1	.25
2	0	0



1	1	.25
2	0	0
3	2	.1
4	1	.2
5	2	.6
		1.15

$1.15 \longrightarrow E = 2.15$

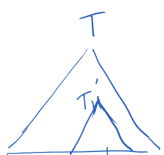


i	depth (k_i)	depth (k_i) $\cdot p_i$
1	1	.25
2	0	0
3	3	.15
4	2	.4
5	1	.3
		1.1

$1.1 \longrightarrow E = 2.1$

exhaustive search for optimal solution $\longrightarrow \Omega(4^n / n^{3/2})$ BSTs

Optimal substructure:

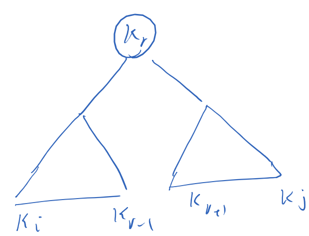


if T is optimal BST, subtree T' should be also optimal

$\langle k_i, \ldots, k_j \rangle \longrightarrow$ Prove by cut-and-paste strategy

- a subtree $\langle k_i, \ldots, k_j \rangle$
- we need to pick one key as root, k_r

$\Rightarrow \begin{cases} \text{left subtree of } k_r, & \langle k_i, \ldots, k_{r-1} \rangle \\ \text{right subtree of } k_r, & \langle k_{r+1}, \ldots, k_j \rangle \end{cases}$



recursive solution:

$\begin{cases} \text{find optimal BST for } \langle k_i, \ldots, k_j \rangle & : j \geq i \\ \text{base case: empty tree} & : j = i-1 \end{cases}$

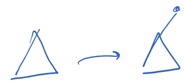
$e[i,j]$ = expected cost of searching optimal BST for $k_i, \ldots, k_j$

if $j = i-1 \longrightarrow e[i,j] = 0$

if $j \geq i \longrightarrow$ select root k_r

i

when a subtree becomes a child/subtree of a nodes



depth of each node in subtree increased by 1

$\hookrightarrow$ expected cost increase

$$w(i,j) = \sum_{l=i}^{j} p_l$$

Let's choose k_r as root for $k_i, \ldots k_j$:

$$\hookrightarrow e[i,j] = p_r + (e[i,r-1] + w(i,r-1]) + (e[r+1,j] + w[r+1,j])$$

$$w[i,j] = w[i,r-1] + p_r + w(r+1,j]$$

$$e[i,j) = e[i,r-1] + e[r+1,j] + w[i,j]$$

try all k_r candidates and choose the best:

$$e[i,j] = \begin{cases} 0 & \text{if } j = i-1 \\ \min\limits_{i \leq r \leq j} \{ e[i,r-1] + e[r+1,j] + w(i,j) \} & \text{if } j \geq i \end{cases}$$

```
OPTIMAL-BST(p, q, n)
  let e[1..n+1, 0..n], w[1..n+1, 0..n], and root[1..n, 1..n] be new tables
  for i = 1 to n+1
      e[i, i-1] = 0
      w[i, i-1] = 0
  for l = 1 to n
      for i = 1 to n - l + 1
          j = i + l - 1
          e[i, j] = ∞
          w[i, j] = w[i, j-1] + p_j
          for r = i to j
              t = e[i, r-1] + e[r+1, j] + w[i, j]
              if t < e[i, j]
                  e[i, j] = t
                  root[i, j] = r
  return e and root
```

subtree with l keys

key i,...,j

finding optimal root r

$\Longrightarrow \Theta(n^3)$

```
CONSTRUCT-OPTIMAL-BST(root)
  r = root[1, n]
  print "k"_r, "is the root"
  CONSTRUCT-OPT-SUBTREE(1, r-1, r, "left", root)
  CONSTRUCT-OPT-SUBTREE(r+1, n, r, "right", root)

CONSTRUCT-OPT-SUBTREE(i, j, r, dir, root)
  if i ≤ j
      t = root[i, j]
      print "k"_t, "is" dir "child of k"_r
      CONSTRUCT-OPT-SUBTREE(i, t-1, t, "left", root)
      CONSTRUCT-OPT-SUBTREE(t+1, j, t, "right", root)
```

e \ j	0	1	2	3	4	5
i = 1	0	.25	.65	.8	1.25	2.10
2		0	.2	.3	.75	1.35
3			0	.05	.3	.85
4				0	.2	.7
5					0	.3
6						0

p_i

w \ j	0	1	2	3	4	5
i = 1	0	.25	.45	.5	.7	1.0
2		0	.2	.25	.45	.75
3			0	.05	.25	.55
4				0	.2	.5
5					0	.3
6						0

$root$ \ j	1	2	3	4	5
i = 1	1	1	1	2	2
2		2	2	2	4
3			3	4	5
4				4	5
5					5