Program Structure and Algorithms

Sid Nath

Lecture 13

Agenda

- Administrative
  - How is DP going?
- Lecture
  - Longest common sequence
  - Minimum edit distance
  - Optimal BST search cost
  - Matrix chain multiplication
  - Vertex cover in trees
  - Independent set in trees
- Quiz

Chain of Thought

- What are my subproblems?
- What are the decisions to solve each subproblem?
- Recursive formulation
  - Base case
- How many subproblems? What is the running time per subproblem?
- What is the overall running time?

DP: Common Subproblems

Finding the right subproblem takes creativity and experimentation. Some standard choices below

- Template #1: Input is $x_1, x_2, \dots, x_n, (or, x[1{:}n]$) and a subproblem is $x_1, x_2, \dots, x_i$ (or, $x[1{:}i])$
  - Number of subproblems is linear
- Template #2: Input is $x_1, x_2, \dots, x_n$, and a subproblem is $x_i, x_{i+1}, \dots, x_j$ (or, $x[i{:}j])$
  - #subproblems is quadratic $O(n^2)$
- Template #3: Inputs are $x_1, x_2, \dots, x_n$, and $y_1, y_2, \dots, y_m$; a subproblem is $x_1, x_2, \dots, x_i$ and $y_1, y_2, \dots, y_j$
  - #subproblems is quadratric $O(mn)$

DP in BioInformatics

- DNA is a string of bases: { ‘A’, ‘C’, ‘G’, ‘T’ }
- Given two DNA strands (== string), how “similar” are they? Are organisms closely related?
- Similarity
  - One strand is a substring of the other
  - If #changes needed to turn one strand into another is small (Min edit distance)
  - Find a third strand in which the bases appear in the same order as in the given strands (not necessarily consecutive). So, longer the third strand, the more similar are the input strands (longest common subsequence)

P10: Longest Common Subsequence (LCS)

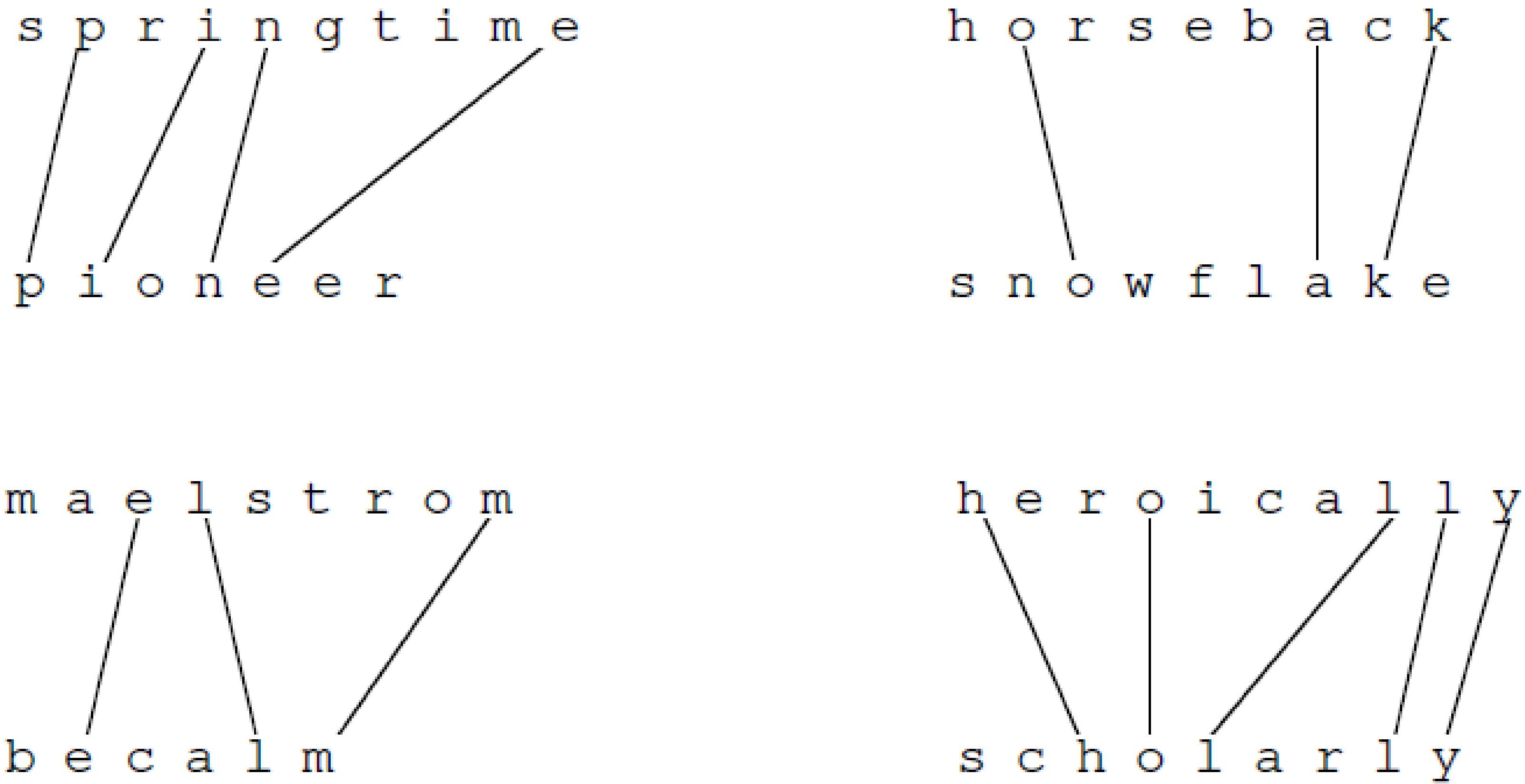
Given two sequences of length m and n, find a subsequence common to both whose length is the longest

$$X = \{x_1, x_2, \dots, x_m\}$$
$$Y = \{y_1, y_2, \dots, y_n\}$$

A subsequence does not have to be consecutive, but it must be in order

A subsequence of a character string $x_1 x_2 \dots x_m$ is a string of the form $x_{i1}\ x_{i2} \dots x_{ik}$, where $i_j < i_{j+1}$.

Examples


s p r i n g t i m e
p i o n e e r
h o r s e b a c k
s n o w f l a k e
m a e l s t r o m
b e c a l m
h e r o i c a l l y
s c h o l a r l y

A Brute-Force Algorithm

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

X has 2^m subsequences to check

Each subsequence takes $O(n)$ time to check in Y

Runtime: $O(n2^m)$

Towards a DP Formulations

Step 1: Characterize an LCS

$X_i = prefix\ \{x_1, \ldots, x_i\}$ // ending at i

$Y_j = prefix\ \{y_1, \ldots, y_j\}$ // ending at j

Idea: LCS of 2 sequences contains as a prefix an LCS of prefixes of the sequences

Let $Z = \{z_1, \ldots, z_k\}$ be any LCS of X and Y

Case 1: if $x_i = y_j, then\ z_k = x_i = y_j, and$

$$Z_{k-1} is\ an\ LCS\ of\ X_{i-1} and\ Y_{j-1}$$

Case 2a: if $x_i \neq y_j, and\ z_k \neq x_i, then$

$$Z\ is\ an\ LCS\ of\ X_{i-1} and\ Y_j$$

Case 2b: if $x_i \neq y_j, and\ z_k \neq y_j, then$

$$Z\ is\ an\ LCS\ of\ X_i\ and\ Y_{j-1}$$

Recursive Definition of an Optimal Solution

Step 2. Define the subproblem structure

c$[i, j]$ = length of LCS of X_i *and* Y_j; we want c$[m, n]$

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

Again, instead of recomputing subproblems, memoize!

Pseudocode

```
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          // compute table entries in row-major order
      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
```

```
PRINT-LCS(b, X, i, j)
  if i == 0 or j = 0
      return              // the LCS has length 0
  if b[i, j] == "↖"
      PRINT-LCS(b, X, i − 1, j − 1)
      print x_i           // same as y_j
  elseif b[i, j] == "↑"
      PRINT-LCS(b, X, i − 1, j)
  else PRINT-LCS(b, X, i, j − 1)
```

Example

X = {s,p,a,n,k,i,n,g} Y = {a,m,p,u,t,a,t,i,o,n}

		a	m	p	u	t	a	t	i	o	n
	0	0	0	0	0	0	0	0	0	0	0
s	0	0	0	0	0	0	0	0	0	0	0
p	0	0	0	1	1	1	1	1	1	1	1
a	0	1	1	1	1	1	2	2	2	2	2
n	0	1	1	1	1	1	2	2	2	2	3
k	0	1	1	1	1	1	2	2	2	2	3
i	0	1	1	1	1	1	2	2	3	3	3
n	0	1	1	1	1	1	2	2	3	3	4
g	0	1	1	1	1	1	2	2	3	3	4
				p			a		i		n

Running Time and Space Complexity

$$\Theta(mn)$$

P11: Min Edit Distance

Given two strings $X[1:m]$ and $\mathrm{Y}[1:n]$ and editing moves as {insert, delete, substitute}, provide an efficient dynamic programming algorithm to determine the minimum number of edits to transform the first string into the second.

Min Edit Distance

S	–	N	O	W	Y		–	S	N	O	W	–	Y
S	U	N	N	–	Y		S	U	N	–	–	N	Y

Cost: 3 **Cost: 5**

Alignment cost: #columns where the letters differ. "-" is a gap

- Insert "U", substitute "O" with "N", delete "W"

- How many changes needed to go from X → Y?
  - X = STALL, Y = TABLE?

S T A LL

T A L L deletion

T A B L substitution

T A B L E insertion

Min Edit Distance (MED)

Subproblems: Look at prefixes $\mathrm{X_i} = X[1{:}\,i]$ and $Y_j = \mathrm{Y}[1{:}\,j]$.
$E(i,j) =$ min #edits to transform $X_i\ to\ Y_j$. We want $E(m,n)$

Decisions: Define $diff(i,j) = \begin{cases} 1, if\ x_i \neq y_j \\ 0, if\ x_i = y_j \end{cases}$

- Substitute $x_i \rightarrow y_j$: $diff(i,j) + E(i-1, j-1)$ // add 1 if they differ
- Delete x_i: $E(i-1,j) + 1$ (edit)
- Insert $x_i\ to\ make\ is\ same\ as\ y_j$: $E(i,j-1) + 1$ (edit)

Min Edit Distance (MED)

Decisions: Define $diff(i,j) = \begin{cases} 1, if\ x_i \neq y_j \\ 0, if\ x_i = y_j \end{cases}$

- Substitute $x_i \rightarrow y_j$: $diff(i,j) + E(i-1,j-1)$ // add 1 if they differ
- Delete x_i: $E(i-1,j)+1$ (edit)
- Insert $x_i\ to\ make\ is\ same\ as\ y_j$: $E(i,j-1)+1$ (edit)

- Recursion

$$E(i,j) = \min\{1+E(i-1,j), 1+E(i,j-1), diff(i,j)+E(i-1,j-1)\}$$

$Base\ cases$: $E(0,j) = j; E(i,0) = i$

Running time: $O(mn)$; O(mn) subproblems, each takes O(1) to solve

DP: Common Subproblems

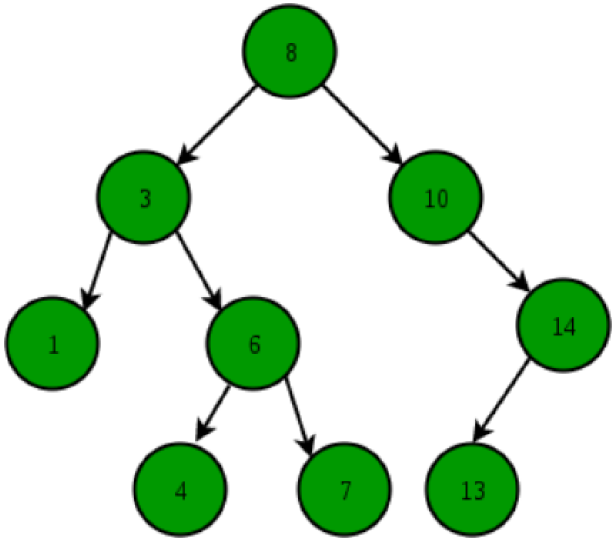
Finding the right subproblem takes creativity and experimentation. Some standard choices below

- Template #1: Input is $x_1, x_2, \ldots, x_n$, $(or, x[1{:}n])$ and a subproblem is $x_1, x_2, \ldots, x_i$ (or, $x[1{:}i]$)
  - Number of subproblems is linear
- Template #2: Input is $x_1, x_2, \ldots, x_n$, and a subproblem is $x_i, x_{i+1}, \ldots, x_j$ (or, $x[i{:}j]$)
  - #subproblems is quadratic $O(n^2)$
- Template #3: Inputs are $x_1, x_2, \ldots, x_n$, and $y_1, y_2, \ldots, y_m$; a subproblem is $x_1, x_2, \ldots, x_i$ and $y_1, y_2, \ldots, y_j$
  - #subproblems is quadratic $O(mn)$

P12: Optimal BST Search Cost

BST is a binary tree data structure with the following properties

- Left subtree contains only nodes with key values lesser than the subtree's root's key value
- Right subtree contains only nodes with key values greater than the subtree's root's key value
- Left and right subtree also must each be a BST



Optimal BST

Given key sequence $K = \{k_1, k_2, \ldots, k_n\}$; distinct and sorted, and each key k_i has probability p_i that a search is for k_i

Goal: BST with <u>minimum expected</u> search cost

- Actual cost = #items examined

For key k_i cost = $depth_T(k_i) + 1$

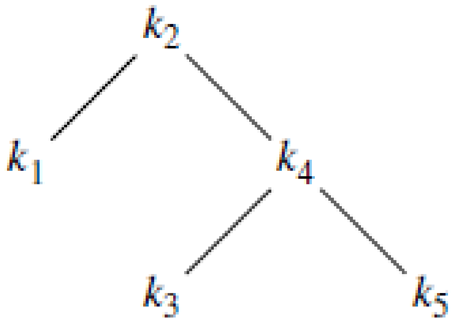
$depth_T(k_i)$ is the depth of k_i is BST T

$$E[search\ cost\ in\ T] = \sum_{i=1}^{n} (depth_T(k_i) + 1) \cdot p_i$$

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

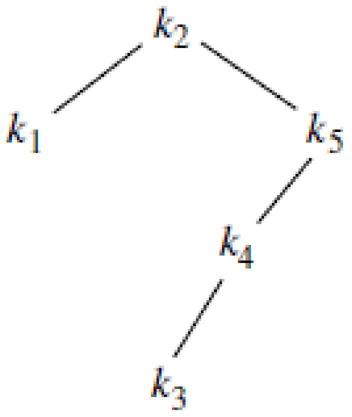
Examples

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



i	$\text{depth}_T(k_i)$	$\text{depth}_T(k_i) \cdot p_i$
1	1	.25
2	0	0
3	2	.1
4	1	.2
5	2	.6
		1.15

E[search cost] = 2.15

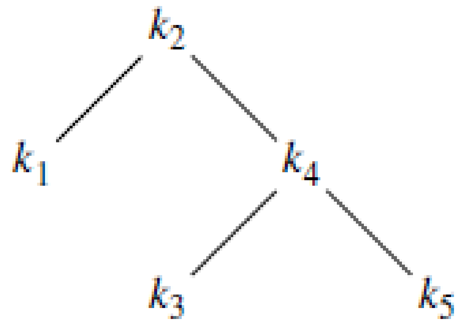


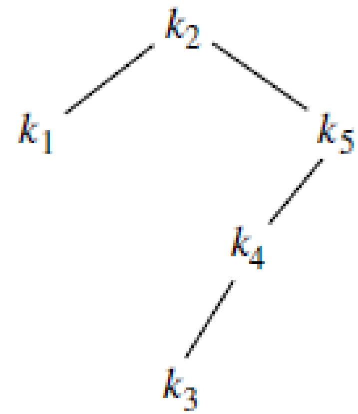
i	$\text{depth}_T(k_i)$	$\text{depth}_T(k_i) \cdot p_i$
1	1	.25
2	0	0
3	3	.15
4	2	.4
5	1	.3
		1.10

E[search cost] = 2.10

Observations

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





- Optimal BST might not have the smallest height

- Optimal BST might not have the highest-probability key at the root

DP: Structure of an optimal solution

Consider any substree of a BST. It contains in a contiguous range $k_i, \dots, k_j$ for some $1 \le i \le j \le n$

If T is an optimal BST and T contains subtree T' with keys $k_i, \dots, k_j$, then T' must be an optimal BST for keys $k_i, \dots, k_j$

One key k_r must be the root, where $i \le r \le j$

Left subtree contains $k_i, \dots, k_{r-1}$ and right subtree contains $k_{r+1}, \dots, k_j$

If we examine all candidate roots k_r then we're guaranteed to find an optimal BST for $k_i, \dots, k_j$

DP: A Recursive Solution

Find an optimal BST for $k_i, \dots, k_j$, where $i \geq 1, j \leq n, j \geq i - 1$; when $j = i - 1$, the tree is empty

Let $e[i,j]$ be the min expected search cost of optimal BST for $k_i, \dots, k_j$

Case 1: If $j = i - 1,$ then $e[i,j] = 0$

Case 2: If $j > i$, select a root k_r and make optimal BSTs for $k_i, \dots, k_{r-1}$ and $k_{r+1}, \dots, k_j$ for left and right subtrees, respectively

If $r = i$ → left subtree is empty; if $r = j$ → right subtree is empty

When a subtree becomes the subtree of a node

- Depth of every node in the subtree increases by 1
- Expected search cost increases by $w(i,j) = \sum_{l=i}^{j} p_l$

DP: A Recursive Solution

If k_r is the root of an optimal BST for $k_i, \ldots, k_j$

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

Also, $w(i,j) = w(i,r-1) + p_r + w(r+1,j)$

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

Try all candidates for k_r

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

DP: Pseudocode (Tabular, bottom-up)

```
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          // base cases
      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          // try all possible roots r
              t = e[i, r - 1] + e[r + 1, j] + w[i, j]
              if t < e[i, j]      // new minimum?
                  e[i, j] = t
                  root[i, j] = r
  return e and root
```

Runtime: $\Theta(n^3)$: $n^2 subproblems \times \Theta(n) per\ subproblem$

Construct an Optimal BST: Print Solution

```
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)
```

P13: Matrix Chain Multiplication

Given a sequence (chain) $\{A_1, A_2, \ldots, A_n\}$ matrices, compute the product $A_1 \cdot A_2 \cdot \ldots \cdot A_n$ using standard matrix multiplication (not Strassen's) to *minimize* the number of scalar multiplications

Matrix multiplication

- Is not commutative, i.e., $A \times B \neq B \times A$
- Is associative, i.e., $A \times (B \times C) = (A \times B) \times C$

Matrices must be compatible: number of columns of A equals number of rows of B.

$A\ is\ p \times q; B\ is\ q \times r; C = A \cdot B = p \times r$; Takes pqr multiplications

Example

Given A_1 is 10×100; A_2 is 100×5; A_3 is 5×50

Compute: $A_1 \cdot A_2 \cdot A_3$

Case 1: Parenthesize by $(A_1 \cdot A_2) \cdot A_3$

First, $10 \times 100 \times 5 = 5000$ multiplications

Then, $10 \times 5 \times 50 = 2500$ multiplications

Total: 7500 multiplications

Case 2: Parenthesize by $A_1 \cdot (A_2 \cdot A_3)$

First, $100 \times 5 \times 50 = 25000$ multiplications

Then, $10 \times 100 \times 50 = 50000$ multiplications

Total: 75000 multiplications

DP: Structure of an optimal solution

Let $A_{i:j}$ be the matrix product $A_i \cdot A_{i+1} \cdot A_j$

If $i < j$, then split between A_k and A_{k+1} for some $i \leq k < j \rightarrow A_{i:k}$ *and* $A_{k+1:j}$; then multiply together

Cost consists of cost of computing $A_{i:k}$ + $A_{k+1:j}$ + cost of multiplying them together

Optimal substructure: Suppose opt parenthesization of $A_{i:j}$ splits between $A_{i:k}$ *and* $A_{k+1:j} \rightarrow A_{i:k}$ must be optimal AND $A_{k+1:j}$ is optimal

Need to consider all possible splits, i.e., values of k

DP: A Recursive Solution

Let $m[i,j]$ be the min #scalar mults to compute $A_{i:j}$, and we want $m[1,n]$

Case 1: if $i=j$,

$m[i,i]=0\ \forall i=1,2,\dots,n$

Case 2: if $i<j$,

$m[i,j]=m[i,k]+m[k+1,j]+p_{i-1}p_kp_j;\ i\le k<j$

We have to try all possible values of k and pick the min

$$m[i,j]=\begin{cases}0, & if\ i=j\\ \min\{m[i,k]+m[k+1,j]+p_{i-1}p_kp_j\}:\ i\le k<j, & if\ i<j\end{cases}$$

Define $s[i,j]$ to be a value of k to split $A_{i:j}$

DP: Pseudocode (Tabular, bottom-up)

```
MATRIX-CHAIN-ORDER(p, n)
  let m[1 : n, 1 : n] and s[1 : n − 1, 2 : n] be new tables
  for i = 1 to n                        // chain length 1
      m[i, i] = 0
  for l = 2 to n                        // l is the chain length
      for i = 1 to n − l + 1            // chain begins at A_i
          j = i + l − 1                 // chain ends at A_j
          m[i, j] = ∞
          m[i, j] = ∞
          for k = i to j − 1
              q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
              if q < m[i, j]
                  m[i, j] = q           // remember this cost
                  s[i, j] = k           // remember this index
  return m and s
```

```
PRINT-OPTIMAL-PARENS(s, i, j)
  if i == j
      print "A"_i
  else print "("
      PRINT-OPTIMAL-PARENS(s, i, s[i, j])
      PRINT-OPTIMAL-PARENS(s, s[i, j] + 1, j)
      print ")"
```

Runtime: $\Theta(n^3)$ as we have $O(n^2)$ subproblems, each takes $O(n)$ time to solve

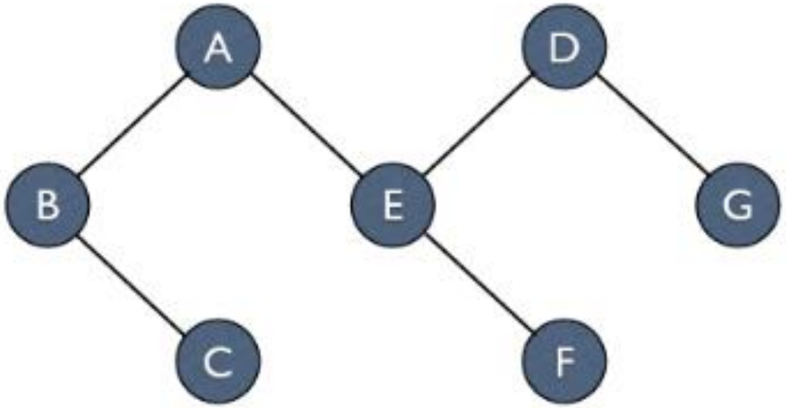
P14: Vertex Cover in Trees

A vertex cover of a graph $G = (V, E)$ is a subset of vertices $S \subseteq V$ that includes at least one endpoint of every edge in E.

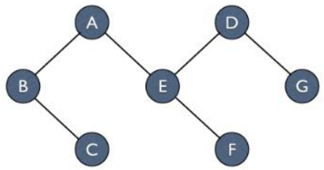
This is NP-Complete.

Provide an efficient algorithm to find the smallest vertex cover of T, given undirected tree $T = (V, E)$.

For example, in the following tree, possible vertex covers are {A, B, C, D, E, F, G}, {A, C, D, F}, but not {C, E, F}. The smallest is {B, E, G} and has size 3.



Vertex Cover in Trees



Subproblems: For each node $u \in T$, $V(u)$ = size of min cover (vertex cover) for the subtree rooted at u. We need $V(r)$, where r is the tree root

Decisions and recursion:

(i) If VC does not use node u, then it must use all children of u, along with VC of subtrees rooted at u's grandchildren.

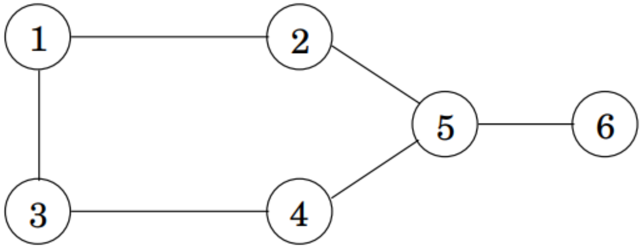
(ii) If VC includes u, then we need VC of subtrees rooted at children of u union u

$$V(u) = \min\{\sum_{j \in children(u)} (1 + \sum_{k \in children(j)} V(k)), \sum_{j \in children(u)} V(j) + 1\}$$

Base case: $V(u) = 0$ if u is a leaf.

Running time: $O(|V| + 2 \cdot |E|)$ *subproblems, each takes* $O(1)$ *to solve*, so $O(|V| + |E|)$

P15: Independent Set in Trees

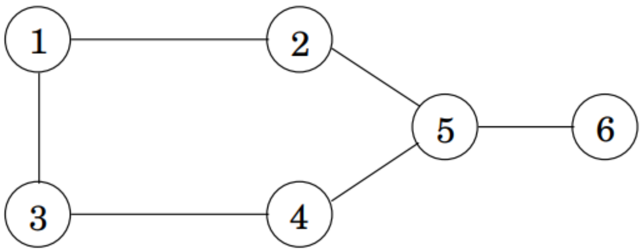


IS = {1, 5}, {2, 3, 6}
But not, {1, 4, 5}

An independent set (IS) of a graph $G = (V, E)$ is a subset of vertices $S \subseteq V$ if there are no edges between them. This is NP-Complete.

Provide an efficient algorithm to find the largest independent set of T, given undirected tree $T = (V, E)$.

Independent Set in Trees



IS = {1, 5}, {2, 3, 6}
But not, {1, 4, 5}

An independent set (IS) of a graph $G = (V, E)$ is a subset of vertices $S \subseteq V$ if there are no edges between them. Provide an efficient algorithm to find the largest independent set of T, given undirected tree $T = (V, E)$.

Subproblems: $I(u)$ = size of largest IS of subtree rooted at u, and we want to find $I(r)$, where r is the root of the tree.

Decisions and recursion: Include u (1 + cannot include children) OR do not include u.

$$I(u) = \max\{1 + \sum_{w \in grandchildren(u)} I(w), \sum_{w \in \text{children}(u)} I(w)\}$$

Base case: $\mathrm{I}(u) = 1$ if u is a leaf.

Running time: $\mathrm{O}(|\mathrm{V}| + 2 \cdot |E|)$ *subproblems, each takes* $O(1)$ *to solve*, so $\mathrm{O}(|\mathrm{V}| + |E|)$

DP Problems So Far…

Prob #	Definition	Opt Type	Template	Running Time (only DP part)
1	Shortest paths in DAGs	Min	#1	O(|V|+|E|) x O(1)
2	Bellman Ford	Min	#1	O(|V|) x O(|E|)
3	Floyd Warshall	Min	#2	$O(n^2) \times O(n)$
4	Transitive Closure of Graph	Min	#2	$O(n^2) \times O(n)$
5	Rod Cutting	Max	#1	O(n) x O(n)
6	Knapsack w/ repetition	Max	#1	O(W) x O(n)
7	Knapsack w/o repetition	Max	#1	O(nW) x O(1)
8	Weighted Interval Scheduling	Max	#1	O(n) x O(1)
9	Share trading	Max	#1	O(n) x O(1)
10	Longest common subsequence	Max	#3	O(mn) x O(1)
11	Min edit distance	Min	#3	O(mn) x O(1)
12	Opt BST search cost	Min	#2	$O(n^2) \times O(n)$
13	Matrix chain mult	Min	#2	$O(n^2) \times O(n)$
14	Vertex cover in trees	Min	#1	O(|V|+|E|) x O(1)
15	Independent set in trees	Max	#1	O(|V|+|E|) x O(1)

Lecture 13 summary

- Problems
  - P10: Longest common subsequence
  - P11: Min edit distance
  - P12: Opt BST search cost
  - P13: Matrix chain mult
  - P14: Vertex cover in trees
  - P15: Independent set in trees

- Please practice coding these up based on pseudocode provided in class