

# MA/CSSE 473

## Day 31

Optimal BSTs



# MA/CSSE 473 Day 30

- HW 12 due Now
- Exam 2 is Tomorrow
  - Covers through Chapter 7 and HW12
  - Emphasis: Chapters 4-7
- **Student Questions**
- Warshall's Algorithm wrapup
- Optimal BSTs



# Dynamic programming

- Used for problems with overlapping subproblems
- Typically, we save (memoize) solutions to the subproblems, to avoid recomputing them.
- Example: Fibonacci numbers
  - Standard recursive formula is too expensive, because of repeated calculation of smaller fibonacci numbers.
  - Caching previously-computed values changes the algorithm from exponential to linear.



# Warshall's algorithm

- Similar to binomial coefficients algorithm
- Assumes that the vertices have been numbered 1, 2, ..., n
- Define the boolean matrix  $R^{(k)}$  as follows:
  - $R^{(k)}[i][j]$  is 1 iff there is a path in the directed graph  $i=v_0 \rightarrow v_1 \rightarrow \dots \rightarrow v_s=j$ , where
    - $s > 1$ , and
    - for all  $t = 1, \dots, s-1, v_t \leq k$   
i.e, none of the intermediate vertices are numbered higher than k.
- Note that T is  $R^{(n)}$



# $R^{(k)}$ example

- $R^{(k)}_{[i][j]}$  is 1 iff there is a path in the directed graph

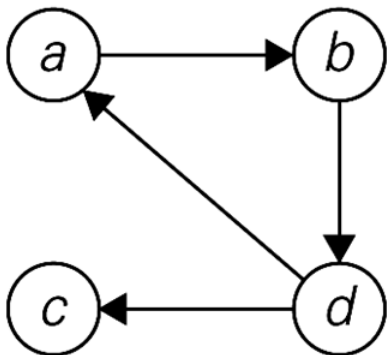
$i=v_0 \rightarrow v_1 \rightarrow \dots \rightarrow v_s=j$ , where

–  $s > 1$ , and

– for all  $t = 2, \dots, n-1, v_t \leq k$

- assuming that the nodes are numbered in alphabetical order, calculate  $R^{(0)}$  and  $R^{(1)}$

You can find a larger example in a book that is available at Safari Books on-line, through the Logan Library Web page (in the Databases section near the top of the page). The book is Sedgwick, *Algorithms Part 5*. See section 19-3



$$A = \begin{matrix} & \begin{matrix} a & b & c & d \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \end{matrix} & \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{matrix}$$



# Quickly Calculating $R^{(k)}$

- Back to the matrix multiplication approach:
  - How much time did it take to compute  $A^k[i][j]$ , once we have  $A^{k-1}$ ?
- Can we do better when calculating  $R^{(k)}[i][j]$  from  $R^{(k-1)}$ ?
- How can  $R^{(k)}[i][j]$  be 1?
  - either  $R^{(k-1)}[i][j]$  is 1, or
  - there is a path from  $i$  to  $k$  that uses no vertices higher than  $k-1$ , and a similar path from  $k$  to  $j$ .
- Thus  $R^{(k)}[i][j] = R^{(k-1)}[i][j]$  **or** (  $R^{(k-1)}[i][k]$  **and**  $R^{(k)}[k][j]$  )
- Note that this can be calculated in constant time
- Time for calculating  $R^{(k)}$  from  $R^{(k-1)}$ ?
- Total time for Warshall's algorithm?
- How does this time compare to using DFS?

**Code and example on next slides**



## ALGORITHM *Warshall*( $A[1..n, 1..n]$ )

//Implements Warshall's algorithm for computing the transitive closure

//Input: The adjacency matrix  $A$  of a digraph with  $n$  vertices

//Output: The transitive closure of the digraph

$R^{(0)} \leftarrow A$

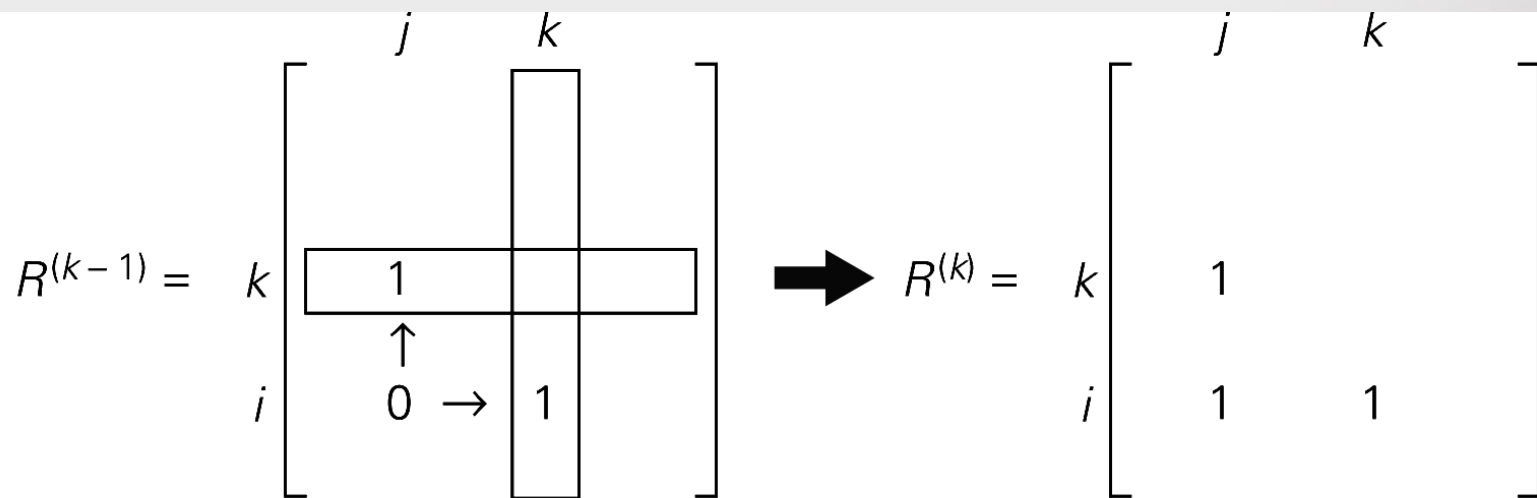
**for**  $k \leftarrow 1$  **to**  $n$  **do**

**for**  $i \leftarrow 1$  **to**  $n$  **do**

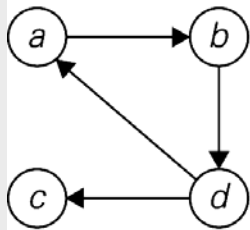
**for**  $j \leftarrow 1$  **to**  $n$  **do**

$R^{(k)}[i, j] \leftarrow R^{(k-1)}[i, j]$  **or** ( $R^{(k-1)}[i, k]$  **and**  $R^{(k-1)}[k, j]$ )

**return**  $R^{(n)}$



**FIGURE 8.3** Rule for changing zeros in Warshall's algorithm



$$R^{(0)} = \begin{matrix} & \begin{matrix} a & b & c & d \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \end{matrix} & \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{matrix}$$

Ones reflect the existence of paths with no intermediate vertices ( $R^{(0)}$  is just the adjacency matrix); boxed row and column are used for getting  $R^{(1)}$ .

$$R^{(1)} = \begin{matrix} & \begin{matrix} a & b & c & d \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \end{matrix} & \begin{bmatrix} 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & \mathbf{1} & 1 & 0 \end{bmatrix} \end{matrix}$$

Ones reflect the existence of paths with intermediate vertices numbered not higher than 1, i.e., just vertex  $a$  (note a new path from  $d$  to  $b$ ); boxed row and column are used for getting  $R^{(2)}$ .

$$R^{(2)} = \begin{matrix} & \begin{matrix} a & b & c & d \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \end{matrix} & \begin{bmatrix} 0 & 1 & \mathbf{0} & \mathbf{1} \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & \mathbf{1} & \mathbf{1} \end{bmatrix} \end{matrix}$$

Ones reflect the existence of paths with intermediate vertices numbered not higher than 2, i.e.,  $a$  and  $b$  (note two new paths); boxed row and column are used for getting  $R^{(3)}$ .

$$R^{(3)} = \begin{matrix} & \begin{matrix} a & b & c & d \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \end{matrix} & \begin{bmatrix} 0 & 1 & 0 & \mathbf{1} \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & \mathbf{1} \end{bmatrix} \end{matrix}$$

Ones reflect the existence of paths with intermediate vertices numbered not higher than 3, i.e.,  $a$ ,  $b$ , and  $c$  (no new paths); boxed row and column are used for getting  $R^{(4)}$ .

$$R^{(4)} = \begin{matrix} & \begin{matrix} a & b & c & d \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \end{matrix} & \begin{bmatrix} \mathbf{1} & 1 & \mathbf{1} & 1 \\ \mathbf{1} & \mathbf{1} & \mathbf{1} & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{matrix}$$

Ones reflect the existence of paths with intermediate vertices numbered not higher than 4, i.e.,  $a$ ,  $b$ ,  $c$ , and  $d$  (note five new paths).

**FIGURE 8.4** Application of Warshall's algorithm to the digraph shown. New ones are in bold.



# Warmup: Optimal linked list order

- Suppose we have  $n$  distinct data items  $x_1, x_2, \dots, x_n$  in a linked list.
- We know the probabilities  $p_1, p_2, \dots, p_n$  that each of the items is the one we'll be searching for.
- What is the expected number of probes before a successful search completes?
- How can we minimize this number?
- What about an unsuccessful search?



# Examples

- $p_i = 1/n$  for each  $i$ .
  - What is the expected number of probes?
- $p_1 = 1/2, p_2 = 1/4, \dots, p_{n-1} = 1/2^{n-1}, p_n = 1/2^{n-1}$ 
  - expected number of probes:

$$\sum_{i=1}^{n-1} \frac{i}{2^i} + \frac{n}{2^{n-1}} = 2 - \frac{1}{2^{n-1}} < 2$$

- What if the same items are placed into the list in the opposite order?

$$\sum_{i=2}^n \frac{i}{2^{n+1-i}} + \frac{1}{2^{n-1}} = n - 1 + \frac{1}{2^{n-1}}$$

- A Maple Worksheet contains the last two calculations.  
Good practice? prove them by induction



# What if we don't know the probabilities?

- Sort the list to at least improve the average time for unsuccessful search
- Self-organizing table:
  - Elements accessed more frequently move toward the front of the table; elements accessed less frequently toward the back. Strategies:
    - Move Ahead One Position (interchange with previous element)
    - Interchange with first element
    - Move to Front (only efficient if the list is a linked list)



# Optimal Binary Search Trees

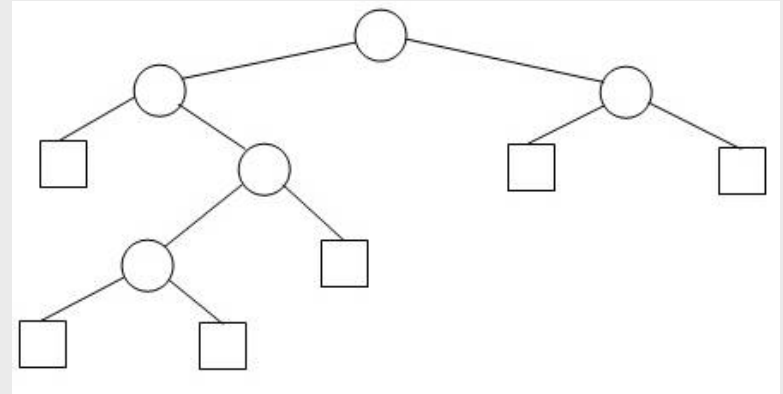
- Suppose we have  $n$  distinct data items  $x_1, x_2, \dots, x_n$  (arranged into increasing order) that we wish to arrange into a Binary Search Tree.
- This time the expected number of probes for an unsuccessful search depends on the shape of the tree and where the search ends up.
- Let  $y$  be the value we are searching for
- Let  $p_i$  be the probability that  $y$  is item  $x_i$
- For  $i = 1, \dots, n-1$  let  $q_i$  be the probability that  $x_i < y < x_{i+1}$
- Similarly, let  $q_0$  be the probability that  $y < x_1$ , and  $q_n$  the probability that  $y > x_n$
- Note that 
$$\sum_{i=1}^n p_i + \sum_{i=0}^n q_i = 1$$

but we can also just use frequencies when finding the optimal tree (and divide by their sum to get probabilities)



# Extended binary search tree

- It's simplest to describe this problem in terms of an **extended binary search tree (EBST)**: a BST enhanced by drawing "external nodes" in place of all of the null pointers in the original tree



- Formally, an Extended Binary Tree (EBT) is either
  - an external node, or
  - an (internal) root node and two EBTs  $T_L$  and  $T_R$
- In diagram, Circles = internal nodes, squares = external nodes
- It's an alternative way of viewing a binary tree
- The external nodes stand for places where an unsuccessful search can end or where an element can be inserted
- An EBT with  $n$  internal nodes has \_\_\_\_ external nodes (Prove this by induction— a review from 230)



# How many possible BST's

- Given distinct items  $x_1 < x_2 < \dots < x_n$ , how many different Binary Search Trees can be constructed?
- Figure it out for  $n=2, 3, 4$
- Write the recurrence relation
- Solution is the **Catalan number**  $c(n)$

$$c(n) = \binom{2n}{n} \frac{1}{n+1} = \frac{(2n)!}{n!(n+1)!} \approx \frac{4^n}{n^{3/2} \sqrt{\pi}}$$

- Verify for  $n = 2, 3, 4, 5$

