MA/CSSE 473 Day 29

Day30-Dynamic-Binomial-Warshall

Dynamic Programming

Binomial Coefficients

Warshall's algorithm

B-trees (Section 2 only)

- We will do a quick overview here.
- For the whole scoop on B-trees (Actually B+ trees), take CSSE 433, Advanced Databases.
- Nodes can contain multiple keys and pointers to other to subtrees

B-tree nodes

- Each node can represent a block of disk storage; pointers are disk addresses
- This way, when we look up a node (requiring a disk access), we can get a lot more information than if we used a binary tree
- In an n-node of a B-tree, there are n pointers to subtrees, and thus n-1 keys
- All keys in T_i are $\geq K_i$ and $< K_{i+1}$
  K_i is the smallest key that appears in T_i

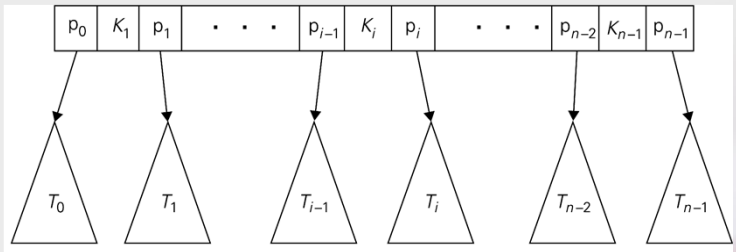


FIGURE 7.7 Parental node of a B-tree

B-tree nodes (tree of order m)

- All nodes have at most m-1 keys
- All keys and associated data are stored in special ***leaf*** nodes (that thus need no child pointers)
- The other (parent) nodes are ***index*** nodes
- All index nodes except the root have between $\lceil m/2 \rceil$ and m children
- root has between 2 and m children
- All leaves are at the same level
- The space-time tradeoff is because of duplicating some keys at multiple levels of the tree
- Especially useful for **data that is too big to fit in memory.** Why?
- Example on next slide

Example B-tree(order 4)

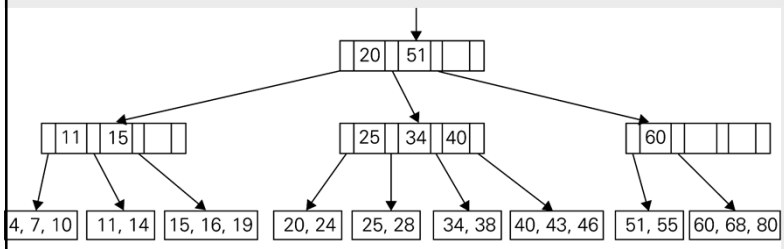


FIGURE 7.8 Example of a B-tree of order 4

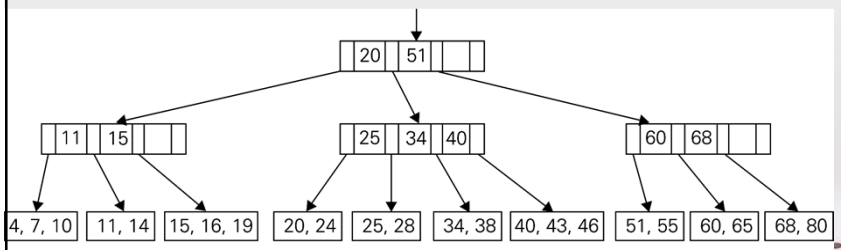
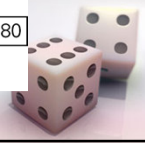


FIGURE 7.9 B-tree obtained after inserting 65 into the B-tree in Figure 7.8



B-tree Animation

- http://slady.net/java/bt/view.php?w=800&h=600

Search for an item

- Within each parent or leaf node, the items are sorted, so we can use binary search (log m), which is a constant with respect to n, the number of items in the table
- Thus the search time is proportional to the height of the tree
- Max height is approximately $\log_{\lceil m/2 \rceil} n$
- **Exercise for you:** Read and understand the straightforward analysis on pages 273-274
- Insert and delete are also proportional to height of the tree

Preview: Dynamic programming

- Used for problems with overlapping subproblems
- Typically, we save (memoize) solutions to the subproblems, to avoid recomputing them.

Dynamic Programming Example

- Binomal Coefficients:
- C(n,k) is the coefficient of x^k in the expansion of $(1+x)^n$
- C(n,0) =C(n,n) = 1.
- If 0 < k < n, C(n, k) = C(n-1, k) + C(n-1, k-1)
- Can show by induction that the "usual" factorial formula for C(n, k) follows from this definition.
  - Let's do it together
- If we don't cache values as we compute them, this can take a lot of time, because of duplicate (overlapping) computation.

Computing a binomial coefficient

Binomial coefficients are coefficients of the binomial formula:
$(a + b)^n = C(n,0)a^nb^0 + \ldots + C(n,k)a^{n-k}b^k + \ldots + C(n,n)a^0b^n$

Recurrence: $C(n,k) = C(n\text{-}1,k) + C(n\text{-}1,k\text{-}1)$ for $n > k > 0$
$C(n,0) = 1, \quad C(n,n) = 1$ for $n \geq 0$

Value of $C(n,k)$ can be computed by filling a table:

	0	1	2	. . .	k-1	k
0	1					
1	1	1				
.						
.						
.						
n-1				$C(n\text{-}1,k\text{-}1)$	$C(n\text{-}1,k)$	
n					$C(n,k)$	

Computing $C(n, k)$:

ALGORITHM *Binomial*(n, k)

```
//Computes C(n, k) by the dynamic programming algorithm
//Input: A pair of nonnegative integers n ≥ k ≥ 0
//Output: The value of C(n, k)
for i ← 0 to n do
    for j ← 0 to min(i, k) do
        if j = 0 or j = i
            C[i, j] ← 1
        else C[i, j] ← C[i − 1, j − 1] + C[i − 1, j]
return C[n, k]
```

Time efficiency: Θ(nk)

Space efficiency: Θ(nk)

Exercise 8.1.7 asks you to compare the efficiency of this approach with some other approaches

If we are computing C(n, k) for many different n and k values, we could cache the table between calls.

Transitive closure of a directed graph

- For each pair of vertices, (A,B), in the directed graph G, is there a path from A to B in G?
- Start with the boolean adjacency matrix A for the n-node graph G. A[i][j] is 1 if and only if G has a directed edge from node i to node j.
- The **transitive closure** of G is the boolean matrix T such that T[i][j] is 1 iff there is a nontrivial directed path from node i to node j in G.
- If we use boolean adjacency matrices, what does M^2 represent? M^3?
- In boolean matrix multiplication, + stands for **or**, and * stands for **and**

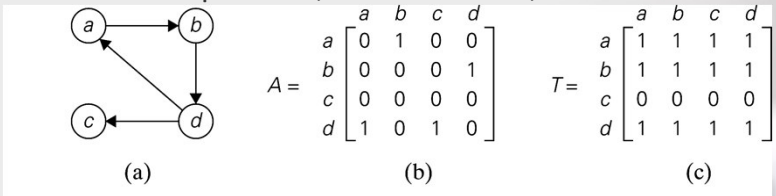


FIGURE 8.2 (a) Digraph. (b) Its adjacency matrix. (c) Its transitive closure.

Transitive closure *via* multiplication

- Again, using + for **or**, we get
  $T = M + M^2 + M^3 + \ldots$
- Can we limit it to a finite operation?
- We can stop at M^{n-1}.
  - How do we know this?
- Number of numeric multiplications for solving the whole problem?

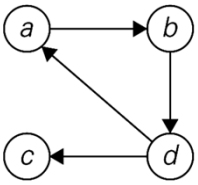
Warshall's algorithm

- 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 $i=v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_s=j$, where
    - s >=1, and
    - for all t = 1, …, 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
  $i=v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_s=j$, where
  - $s > 1$, and
  - for all $t = 2, \ldots, n-1$, $v_t \leq k$
- 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 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



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

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** $R^{(k-1)}[k][j]$)
- 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) ← A
for k ← 1 to n do
    for i ← 1 to n do
        for j ← 1 to n do
            R^(k)[i, j] ← R^(k−1)[i, j] or (R^(k−1)[i, k] and R^(k−1)[k, j])
return R^(n)
```

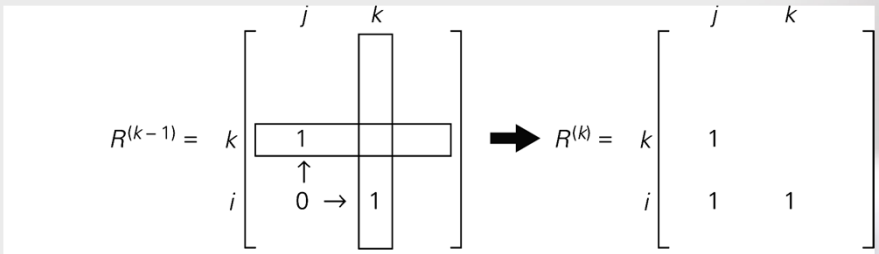


FIGURE 8.3 Rule for changing zeros in Warshall's algorithm

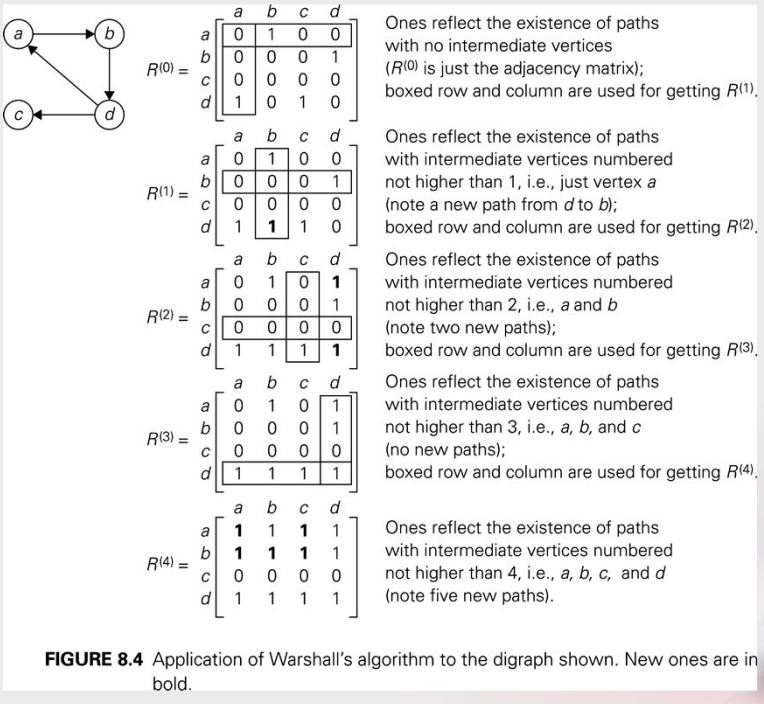


FIGURE 8.4 Application of Warshall's algorithm to the digraph shown. New ones are in bold.