MA/CSSE 473 Day 28

Dynamic Programming

Binomial Coefficients

Warshall's algorithm

Student questions?

No handout today

- Announcements:
- HW 12 is now available.
- Due next Thursday.
- Exam Tuesday, Nov 4
- In my office today: Hours 6, 9, first half of 10.

B-trees

- We will do a quick overview.
- For the whole scoop on B-trees (Actually B+ trees), take CSSE 333, 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
- For all keys in T_i , $K_i \leq T_i < K_{i+1}$
  K_i is the smallest key that appears in T_i

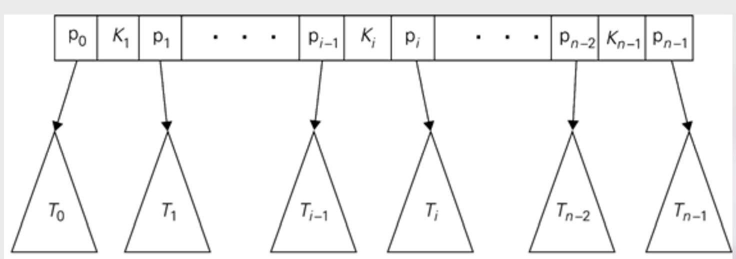


FIGURE 7.7 Parental node of a B-tree

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)

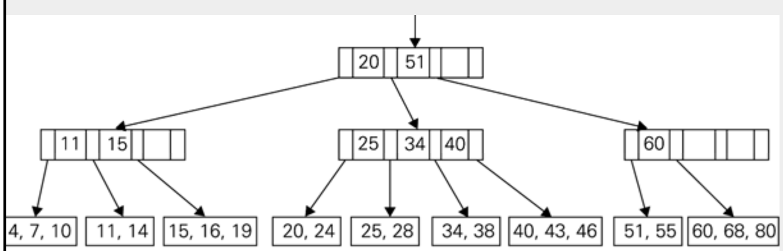


FIGURE 7.8 Example of a B-tree of order 4

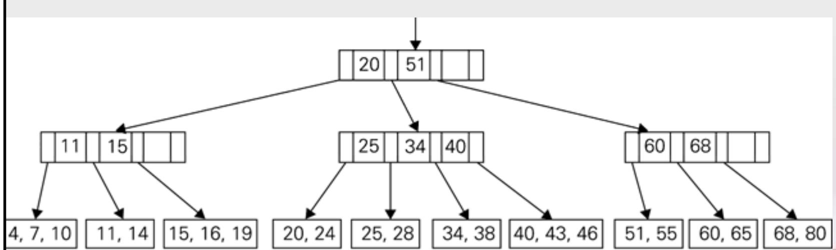


FIGURE 7.9 B-tree obtained after inserting 65 into the B-tree in Figure 7.8

Search for an item

- Within each parent or leaf node, the keys 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

Dynamic programming

- Used for problems with recursive solutions and overlapping subproblems
- Typically, we save (memoize) solutions to the subproblems, to avoid recomputing them.
- You should look at the simple examples from Section 8.1 in the textbook.

Dynamic Programming Example

- Binomial 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 recursive definition.
  - A good practice problem for you
- 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: $C(n,k) = C(n-1,k) + C(n-1,k-1)$ for $n > k > 0$
$C(n,0) = 1$, $C(n,n) = 1$ for $n \geq 0$

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

	0	1	2	. . .	k-1	k
0	1					
1	1	1				
.						
.						
.						
n-1					$C(n-1,k-1)$	$C(n-1,k)$
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]
```

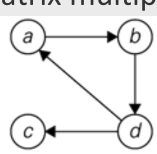
Time efficiency: Θ(*nk*)

Space efficiency: Θ(*nk*)

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

- We ask this question for a given directed graph G: for each of vertices, (A,B), 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**



(a)

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

(b)

$$T = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 1 & 1 & 1 & 1 \\ b & 1 & 1 & 1 & 1 \\ c & 0 & 0 & 0 & 0 \\ d & 1 & 1 & 1 & 1 \end{array}$$

(c)

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

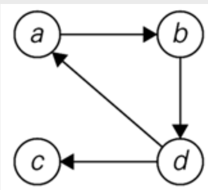
- Similar to binomial coefficients algorithm
- Assumes that the vertices have been numbered

  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 $v_i=w_0 \rightarrow w_1 \rightarrow \ldots \rightarrow w_s=v_j$, where
    - s >=1, and
    - for all t = 1, …, s-1, w_t is v_m for some $m \leq k$

      i.e, none of the intermediate vertices are numbered higher than k
- Note that the transitive closure T is $R^{(n)}$

$R^{(k)}$ example

- $R^{(k)}[i][j]$ is 1 iff there is a path in the directed graph
  $v_i = w_0 \rightarrow w_1 \rightarrow \ldots \rightarrow w_s = v_j$, where
  - $s > 1$, and
  - for all $t = 2, \ldots, s-1$, w_t is v_m for some $m \leq k$
- **Example:** assuming that the node numbering is in alphabetical order, calculate $R^{(0)}$, $R^{(1)}$, and $R^{(2)}$



$A =$

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	0	1	0

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 $R^{(k)}[i][j]$ is
  $R^{(k-1)}[i][j]$ **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?

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 $R^{(n)}$

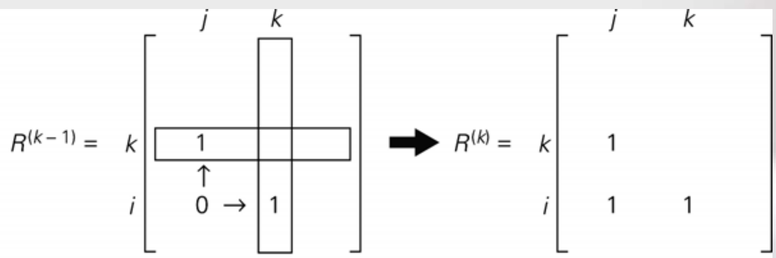
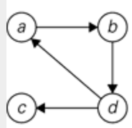


FIGURE 8.3 Rule for changing zeros in Warshall's algorithm



$R^{(0)}$ =

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	0	1	0

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 $R^{(1)}$.

$R^{(1)}$ =

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	**1**	1	0

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 $R^{(2)}$.

$R^{(2)}$ =

	a	b	c	d
a	0	1	0	**1**
b	0	0	0	1
c	0	0	0	0
d	1	1	1	**1**

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 $R^{(3)}$.

$R^{(3)}$ =

	a	b	c	d
a	0	1	0	1
b	0	0	0	1
c	0	0	0	0
d	1	1	1	1

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 $R^{(4)}$.

$R^{(4)}$ =

	a	b	c	d
a	**1**	1	**1**	1
b	**1**	**1**	**1**	1
c	0	0	0	0
d	1	1	1	1

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.

Floyd's algorithm

- All-pairs shortest path
- A **network** is a graph whose edges are labeled by (usually) non-negative numbers. We store those edge numbers as the values in the adjacency matrix for the graph
- A **shortest path** from vertex u to vertex v is a path whose edge sum is smallest.
- Floyd's algorithm calculates the shortest path from u to v for each pair (u, v) od vertices.
- It is so much like Warshall's algorithm, that I am confident you can quickly get the details from the textbook after you understand Warshall's algorithm.

Dynamic Programming Example

OPTIMAL BINARY SEARCH TREES

Warmup: Optimal linked list order

- Suppose we have n distinct data items $x_1, x_2, \ldots, x_n$ in a linked list.
- Also suppose that we know the probabilities $p_1, p_2, \ldots, p_n$ that each of the items is the one we'll be searching for.
- Questions we'll attempt to answer:
  - 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?
- $p_1 = ½, p_2 = ¼, \ldots, p_{n-1} = 1/2^{n-1}, p_n = 1/2^{n-1}$
  - 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?

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

- The next slide shows the evaluation of the last two summations in Maple.
  - Good practice for you? prove them by induction

Calculations for previous slide

```
> sum(i/2^i, i=1..n-1)+ n/2^(n-1);
```

$$-2\left(\frac{1}{2}\right)^{n} n-2\left(\frac{1}{2}\right)^{n}+2+\frac{n}{2^{(n-1)}}$$

```
> simplify(%);
```

$$-2^{(1-n)}+2$$

```
> sum(i/2^(n+1-i), i=2..n) + 1/2^(n-1);
```

$$n-1+\frac{1}{2^{(n-1)}}$$

What if we don't know the probabilities?

1. Sort the list so we can at least improve the average time for unsuccessful search
2. Self-organizing list:
   - Elements accessed more frequently move toward the front of the list; elements accessed less frequently toward the rear.
   - Strategies:
     - Move ahead one position (exchange with previous element)
     - Exchange with first element
     - Move to Front (only efficient if the list is a linked list)

Q3

Optimal Binary Search Trees

- Suppose we have n distinct data keys K_1, K_2, …, K_n (in increasing order) that we wish to arrange into a Binary Search Tree
- This time the expected number of probes for a successful or unsuccessful search depends on the shape of the tree and where the search ends up

Q4