Dynamic Programming III

CSci 4041: Algorithms and Data Structures

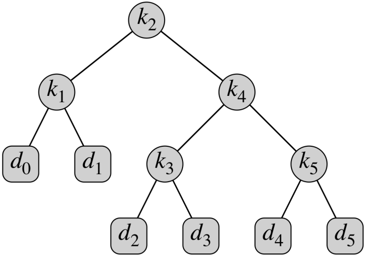
Instructor: Arindam Banerjee

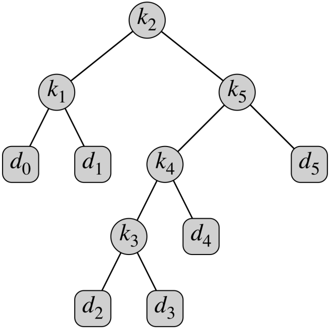
March 31, 2015

Optimal Binary Search Tree (BST)

- Expected time in binary search trees
  - Depends on frequency of queries
  - Keys queried frequently should be towards the top
- Sequence of sorted keys $K = \langle k_1, k_2, \ldots, k_n \rangle$
  - Probability of querying k_i is p_i
- Also $(n+1)$ dummy keys $d_0, d_1, \ldots, d_n$
  - d_0 represents values less than k_1
  - d_n represents values more than k_n
  - d_i represents all values between k_i and k_{i+1}
  - Probability of querying d_i (between k_i and k_{i+1}) is q_i

Example: Optimal BST





i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

- Expected cost of (a) is 2.80, of (b) is 2.75

Expected Cost of Search

- Total probably: successful or unsuccessful search

$$\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1$$

- Cost for a node = depth of the node + 1
- Expected cost over all nodes

$$\begin{aligned} E[\text{cost}(T)] &= \sum_{i=1}^{n} (\text{depth}_T(k_i) + 1) \cdot p_i + \sum_{i=0}^{n} (\text{depth}_T(d_i) + 1) \cdot q_i \\ &= 1 + \sum_{i=1}^{n} \text{depth}_T(k_i) \cdot p_i + \sum_{i=0}^{n} \text{depth}_T(k_i) \cdot q_i \end{aligned}$$

- Goal: Find BST with minimum expected cost
  - Cannot evaluate each BST, total number is $\Omega(4^n / n^{3/2})$

Example: Expected Cost of Search

i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

node	depth	probability	contribution
k_1	1	0.15	0.30
k_2	0	0.10	0.10
k_3	2	0.05	0.15
k_4	1	0.10	0.20
k_5	2	0.20	0.60
d_0	2	0.05	0.15
d_1	2	0.10	0.30
d_2	3	0.05	0.20
d_3	3	0.05	0.20
d_4	3	0.05	0.20
d_5	3	0.10	0.40
Total			2.80

Step 1: Structure of an Optimal BST

- Consider any subtree of a BST
  - Has keys in contiguous range $k_i, \ldots, k_j$, $1 \leq i \leq j \leq n$
  - Has dummy keys $d_{i-1}, d_i, \ldots, d_j$ as leaves
- Optimal substructure: Assume T is optimal
  - Let T' be a subtree with keys $k_i, \ldots, k_j$
  - Then T' is optimal for keys $k_i, \ldots, k_j$, dummy keys $d_{i-1}, \ldots, d_j$
- The 'cut-and-paste' argument
  - If a different T'' had lower cost, we can use it to get a better overall solution compared to T
  - Contradicts optimality of overall solution T

Step 1: Structure of an Optimal BST

- For keys $k_i, \ldots, k_j$, one key k_r will be root
  - Left subtree $k_i, \ldots, k_{r-1}$, dummy keys $d_{i-1}, \ldots, d_{r-1}$
  - Right subtree $k_{r+1}, \ldots, k_j$, dummy keys $d_r, \ldots, d_j$
- Explore all k_r as possible root
  - Optimal subtrees on $k_i, \ldots, k_{r-1}$, and $k_{r+1}, \ldots, k_j$
- Need to consider empty subtrees
  - $k_r = k_i$, left subtree has no keys, only dummy key d_{i-1}
  - $k_r = k_j$, right subtree has no keys, only dummy key d_j

Step 2: A Recursive Solution

- Consider keys $k_i, \ldots, k_j$
- Denote $e[i, j]$ as the expected cost of optimal BST
  - $i \geq 1, j \leq n$, and $j \geq i - 1$
  - When $j = i - 1$, no key, only d_{i-1}, so $e[i, i-1] = q_{i-1}$
- For $j \geq i$, need to select a key k_r from $k_i, \ldots, k_j$
  - Subtree $k_i, \ldots, k_{r-1}$ and $k_{r+1}, \ldots, k_j$
  - The depth of keys in the subtrees increase by 1

Step 2: A Recursive Solution

- Denote

$$w(i,j) = \sum_{\ell=i}^{j} p_\ell + \sum_{\ell=j-1}^{j} q_\ell$$

- If k_r is the root of the optimal subtree containing $k_i, \ldots, k_j$

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

- Since $w(i,j) = w(i,r-1) + p_r + w(r+1,j)$, we have

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

- Since we need to minimize over k_r, we get the recursion

$$e[i,j] = \begin{cases} q_{i-1} & \text{if } j = i-1 \ , \\ \min_{i \le r \le j} \{e[i,r-1] + e[r+1,j] + w(i,j)\} & \text{if } i \le j \ . \end{cases}$$

Step 3: Computing the Expected Cost

- A direct recursive execution would be inefficient
- Working with contiguous indices, store $e[i,j]$ in a table
- Table $e[1 \dots n, 0 \dots n]$
  - First index runs till $(n+1)$, since $e[n+1,n]$ considers d_n
  - Second index starts from 0, since $e[1,0]$ considers d_0
- Table $root[i,j]$ stores root of subtree on $k_i, \dots, k_j$
- Also maintain table for $w(i,j)$, rather than recomputing
  - Avoid $\Theta(j-i)$ additions each time
  - Table $w(1 \dots n+1, 0 \dots n)$
  - Base case $w(i,i-1) = q_i$, and for $j \geq i$

$$w(i,j) = w(i,j-1) + p_j + q_j$$

Algorithm: Optimal BST

```
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
```

Example: Tables for Optimal BST

e

j \ i	1	2	3	4	5	6
5	2.75	2.00	1.30	0.90	0.50	0.10
4	1.75	1.20	0.60	0.30	0.05	
3	1.25	0.70	0.25	0.05		
2	0.90	0.40	0.05			
1	0.45	0.10				
0	0.05					

w

j \ i	1	2	3	4	5	6
5	1.00	0.80	0.60	0.50	0.35	0.10
4	0.70	0.50	0.30	0.20	0.05	
3	0.55	0.35	0.15	0.05		
2	0.45	0.25	0.05			
1	0.30	0.10				
0	0.05					

root

j \ i	1	2	3	4	5
5	2	4	5	5	5
4	2	2	4	4	
3	2	2	3		
2	1	2			
1	1				