Computer Science 161
Spring 2021 Lecture 17:
Dynamic Programming:
Optimal [Offline] Binary Search Trees

2

Offline Optimal Binary Search Trees

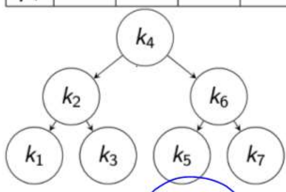
- In ICS 46, you saw "online" search trees
  - Additions happened one at a time
  - Resolve addition before next request
  - Had to maintain "balance"
  - Did not know probability distribution of requests.
- Today we will look at "offline" search trees
  - Know full set of keys at beginning
  - Know probability distribution of requests
  - Want to minimize expected lookup time
  - Even if that means bad lookup for some

3

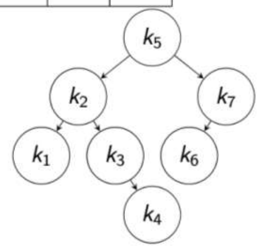
Examples of Binary Search Trees

i	1	2	3	4	5	6	7
p_i	.13	.21	.11	.01	.22	.08	.24

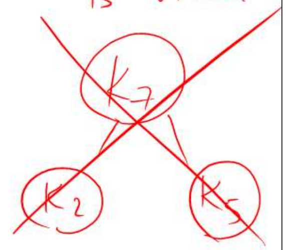
BST property is vital.



E[lookup] = 2.69



E[lookup] = 2.12



4

Problem Statement

- Input: n probabilities, $p_1 \ldots p_n$
- p_i is probability of looking up ith key.
- Goal: build binary search tree.
  - Minimize expected lookup cost.

Check for understanding

- Suppose we have d_i (depth of each node)
- Root has $d_i = 1$, its children have $d_i = 2$, etc.
- What is the expected lookup cost of this tree?

$$\sum p_i d_i$$

5

Creating the Dyn Prog Algorithm

$- \sum_{k \in LHS} P_k \cdot (d_k) + \sum P_k (d_k + 1)$

d_k = depth in OPT

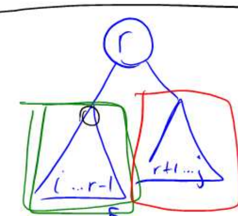
left hand side

smallest ↓ ↓ largest   using only these keys

Define $\mathtt{OPT}(i, j)$: *cost* of opt tree keys i through j

- Base cases: $j < i$ return 0
  $i == j$ return P_i
- Which key(s) can be the root of a binary search tree consisting of keys i through j?

⟶ The root is in range $i..j$ inclusive



- Cost of BST, rooted at r, has keys i through j?

OPT tree on smaller keys

Cost of LHS + cost RHS

$$\boxed{OPT(i, r-1) + \sum_{k=i}^{r-1} P_k} + OPT(r+1, j) + \sum_{k=r+1}^{j} P_k + P_r$$

$$OPT(i, r-1) + OPT(r+1, j) + \sum_{k=i}^{j} P_k \quad \uparrow \min_{i \le r \le j}$$

6

First make recursive solution

$\mathtt{OPT}(i, j)$:
  if $j < i$ **then**
    return 0
  else if $j = i$ **then**
    return p_i
  else
    $r = i$ roots[i,j] = r
    min = $\mathtt{OPT}(i, r-1)$ + $\mathtt{OPT}(r+1, j)$ + $\sum p_k$
    for r = i+1 ... j
      val = $OPT(i, r-1) + OPT(r+1, j) + \sum P_k$
      if val < min
        min = val
        roots[i,j] = r
  return min

$O(n)$

7 # Iterative Version: Topological Order

- Caution: some recursive calls to *higher* values.
- We can't iterate increasing i and j together.
- OPT$[i, j]$ will make calls to:
  - OPT$[i,\ r-1]$ for $i \le r \le j$
  - OPT$[r+1,\ j]$ for $i \le r \le j$

~~No!: for $i=1..n$ for $j=i+1...n$ //fill OPT$[i,j]$~~

$\delta = 2+3$

- For example, OPT$[2, 5]$ will call:
  - OPT$[2, 1]$ and OPT$[3, 5]$ $(r = 2)$
  - OPT$[2, 2]$ and OPT$[4, 5]$ $(r = 3)$
  - OPT$[2, 3]$ and OPT$[5, 5]$ $(r = 4)$
  - OPT$[2, 4]$ and OPT$[6, 5]$ $(r = 5)$

8 # Iterative Version: Memoize the Data

for $i \leftarrow 1 \ldots n$ **do**
  OPT$[i, i-1] \leftarrow 0$
  OPT$[i, i] \leftarrow p_i$

for $\delta = 1 \ldots n-1$
  for $i = 1 \ldots n-\delta$
    $j = i+\delta$
    //fill in OPT$[i,j]$ here

9

Table looks like

$\delta = 3$

$\delta = 1$ values

	k_1	k_2	k_3	k_4	k_5	k_6	k_7
k_1	.13						
k_2	0	.21					
k_3		0	.11				
k_4			0	.01			
k_5				0	.22		
k_6					0	.08	
k_7						0	.24

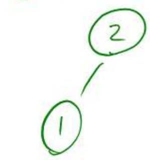
$\delta = 2$ values

10

How to get the tree itself?

	k_1	k_2	k_3	k_4	k_5	k_6	k_7
k_1	0.13	0.47	0.69	0.72	1.28	1.52	2.12
k_2		0.21	0.43	0.46	1	1.17	1.73
k_3			0.11	0.13	0.47	0.63	1.19
k_4				0.01	0.24	0.4	0.95
k_5					0.22	0.38	0.92
k_6						0.08	0.4
k_7							0.24

OPT tree on 1...2



what value of r minimized OPT[1,7]?

$$\mathrm{OPT}[1,2] \overset{?}{=} \begin{cases} r=1 & \cancel{\mathrm{OPT}[1,0]}^{\,0} + \cancel{\mathrm{OPT}[2,2]}^{\,.21} + .34 \\ r=2 & \cancel{\mathrm{OPT}[1,1]}^{\,.13} + \cancel{\mathrm{OPT}[3,2]}^{\,0} + .34 \end{cases}$$

.47

11

How to get the tree itself?

for $i \leftarrow 1 \ldots n$ **do**
  $\mathrm{OPT}[i, i-1] \leftarrow 0$
  $\mathrm{OPT}[i, i] \leftarrow p_i$ roots[i,i] = i
for $\delta = 1$ to $n-1$ **do**
  for $i = 1$ to $n - \delta$ **do**
    $j = i + \delta$
    // OPT[i,j] gets filled in here.
    // some value r minimized OPT[i,j] = ...
    roots [i,j] = that value of r

12

Function printTree

```
Node * printTree(roots, i, j)
    if j < i return nullptr
    elif i == j return Node (i^th key)
    else
        r = roots [i,j]
        Node *n = new Node (r^th key)
        n -> left = printTree(roots, i, r-1)
        n -> right = printTree(roots, r+1, j)
        return n
```