

# MA/CSSE 473

## Days 29-30

Optimal BSTs



### MA/CSSE 473 Days 29-30

- **Student Questions?**
- Expected Lookup time in a Binary Search Tree
- Optimal static Binary Search Tree
- Using the recursive formula with dynamic programming
- Greedy Algorithms
- Background for file compression (Huffman's algorithm)



## Recap: Optimal linked list order

- Suppose we have  $n$  distinct data items  $x_1, x_2, \dots, x_n$  in a linked list.
- Also suppose that 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?
  - $\sum_{i=1}^n i p_i$
- How can we minimize this number?
  - Place the elements in the list in decreasing order of probability.
- What about an unsuccessful search?



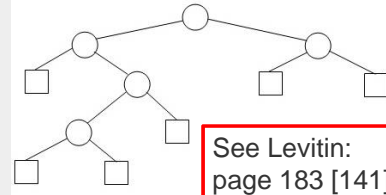
## Optimal Binary Search Trees

- Suppose we have  $n$  distinct data keys  $K_1, K_2, \dots, K_n$  (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
- This discussion follows Reingold and Hansen, *Data Structures*. An excerpt on optimal static BSTs is posted on Moodle. I use  $a_i$  and  $b_i$  where Reingold and Hansen use  $\alpha_i$  and  $\beta_i$



## Recap: 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 (We proved this by induction earlier in the term)



## What contributes to the expected number of probes?

- Frequencies, depth of node
- For successful search, number of probes is one more than the depth of the corresponding internal node
- For unsuccessful, number of probes is equal to the depth of the corresponding external node



## Optimal BST Notation

- Keys are  $K_1, K_2, \dots, K_n$
- Let  $v$  be the value we are searching for
- For  $i = 1, \dots, n$ , let  $a_i$  be the probability that  $v$  is key  $K_i$
- For  $i = 1, \dots, n-1$ , let  $b_i$  be the probability that  $K_i < v < K_{i+1}$ 
  - Similarly, let  $b_0$  be the probability that  $v < K_1$ , and  $b_n$  the probability that  $v > K_n$

- Note that 
$$\sum_{i=1}^n a_i + \sum_{i=0}^n b_i = 1$$

- We can also just use *frequencies* instead of *probabilities* when finding the optimal tree (and divide by their sum to get the probabilities if we ever need them). That is what we will do.
- Should we try exhaustive search of all possible BSTs? **Answer on next slide**



## Aside: How many possible BST's

- Given distinct keys  $K_1 < K_2 < \dots < K_n$ , how many different Binary Search Trees can be constructed from these values? **When  $n=20$ ,  $c(n)$  is almost  $10^{10}$**
- Figure it out for  $n=2, 3, 4, 5$
- Write the recurrence relation
- Solution is the **Catalan number**  $c(n)$

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

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

**Wikipedia Catalan article has five different proofs of**

$$c(n) = \binom{2n}{n} \frac{1}{n+1}$$



## Recap: Optimal Binary Search Trees

- Suppose we have  $n$  distinct data items  $K_1, K_2, \dots, K_n$  (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
- Let  $v$  be the value we are searching for
- For  $i = 1, \dots, n$ , let  $a_i$  be the probability that  $v$  is item  $K_i$
- For  $i = 1, \dots, n-1$ , let  $b_i$  be the probability that  $K_i < v < K_{i+1}$
- Similarly, let  $b_0$  be the probability that  $v < K_1$ , and  $b_n$  the probability that  $v > K_n$
- Note that 
$$\sum_{i=1}^n a_i + \sum_{i=0}^n b_i = 1$$
but we can also just use frequencies when finding the optimal tree (and divide by their sum to get the probabilities if needed)



## What not to measure

- Earlier, we introduced the notions of external path length and internal path length
- These are too simple, because they do not take into account the frequencies.
- We need *weighted* path lengths.

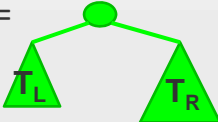


## Weighted Path Length

$$C(T) = \sum_{i=1}^n a_i [1 + \text{depth}(x_i)] + \sum_{i=0}^n b_i [\text{depth}(y_i)]$$

- If we divide this by  $\sum a_i + \sum b_i$  we get the average search time.

- We can also define it recursively:

- $C(\square) = 0$ . If  $T =$   , then

**Note:**  $y_0, \dots, y_n$   
are the external  
nodes of the tree

$C(T) = C(T_L) + C(T_R) + \sum a_i + \sum b_i$ , where the summations are over all  $a_i$  and  $b_i$  for nodes in  $T$

- It can be shown by induction that these two definitions are equivalent (good practice problem).



## Example

- Frequencies of vowel occurrence in English
- : A, E, I, O, U
- a's: 32, 42, 26, 32, 12
- b's: 0, 34, 38, 58, 95, 21
- Draw a couple of trees (with E and I as roots), and see which is best. (sum of a's and b's is 390).



## Strategy

- We want to minimize the weighted path length
- Once we have chosen the root, the left and right subtrees must themselves be optimal EBSTs
- We can build the tree from the bottom up, keeping track of previously-computed values



## Intermediate Quantities

- Cost: Let  $C_{ij}$  (for  $0 \leq i \leq j \leq n$ ) be the cost of an optimal tree (not necessarily unique) over the frequencies  $b_i, a_{i+1}, b_{i+1}, \dots, a_j, b_j$ . Then
- $C_{ii} = 0$ , and 
$$C_{ij} = \min_{i < k \leq j} (C_{i,k-1} + C_{kj}) + \sum_{t=i}^j b_t + \sum_{t=i+1}^j a_t$$
- This is true since the subtrees of an optimal tree must be optimal
- To simplify the computation, we define
- $W_{ii} = b_i$ , and  $W_{ij} = W_{i,j-1} + a_j + b_j$  for  $i < j$ .
- Note that  $W_{ij} = b_i + a_{i+1} + \dots + a_j + b_j$ , and so
- $C_{ii} = 0$ , and 
$$C_{ij} = W_{ij} + \min_{i < k \leq j} (C_{i,k-1} + C_{kj})$$
- Let  $R_{ij}$  (root of best tree from  $i$  to  $j$ ) be a value of  $k$  that minimizes  $C_{i,k-1} + C_{kj}$  in the above formula



## Code

```
# initialize the main diagonal
for i in range(n + 1):
    R[i][i] = i
    W[i][i] = b[i]
    # Draw this cell of the table in the given window.
    drawSquare(i, i, W[i][i], C[i][i], R[i][i], win, indent, squareSize)
# Now populate each of the n upper diagonals:
for d in range(1, n+1): # fill in this diagonal
    # The previous diagonals are already filled in.
    for i in range(n - d + 1):
        j = i + d; # on the dth diagonal, j - i = d
        opt = i + 1 # until we find a better one
        for k in range(i+2, j+1):
            if C[i][k-1]+C[k][j] < C[i][opt-1]+C[opt][j]:
                opt = k
        R[i][j] = opt
        W[i][j] = W[i][j-1] + a[j] + b[j]
        C[i][j] = C[i][opt-1] + C[opt][j] + W[i][j]
        # Draw this cell of the table in the given window.
        drawSquare(i, j, W[i][j], C[i][j], R[i][j], win, indent, squareSize)
```



## Results

R00:	0	R01:	1	R02:	2	R03:	2	R04:	3	R05:	4
W00:	0	W01:	66	W02:	146	W03:	230	W04:	357	W05:	390
C00:	0	C01:	66	C02:	212	C03:	418	C04:	754	C05:	936
		R11:	1	R12:	2	R13:	3	R14:	3	R15:	4
		W11:	34	W12:	114	W13:	198	W14:	325	W15:	358
		C11:	0	C12:	114	C13:	312	C14:	624	C15:	798
				R22:	2	R23:	3	R24:	4	R25:	4
				W22:	38	W23:	122	W24:	249	W25:	282
				C22:	0	C23:	122	C24:	371	C25:	532
						R33:	3	R34:	4	R35:	4
						W33:	58	W34:	185	W35:	218
						C33:	0	C34:	185	C35:	346
								R44:	4	R45:	5
								W44:	95	W45:	128
								C44:	0	C45:	128
										R55:	5
										W55:	21
										C55:	0

**How to  
construct the  
optimal tree?**

**Analysis of the  
algorithm?**

- Constructed by diagonals, from main diagonal upward
- What is the optimal tree?





## Running time

- Most frequent statement is the comparison  
if  $C[i][k-1]+C[k][j] < C[i][opt-1]+C[opt][j]$ :
- How many times  
does it execute:

$$\sum_{d=1}^n \sum_{i=0}^{n-d} \sum_{k=i+2}^{i+d} 1$$

```
simplify(sum(sum(sum(1,k=i+2..i+d),i=0..n-d),d=1..n));
```

$$-\frac{1}{6}n + \frac{1}{6}n^3$$



Do what seems best at the moment ...

## GREEDY ALGORITHMS



## Greedy algorithms

- Whenever a choice is to be made, pick the one that seems optimal for the moment, without taking future choices into consideration
  - Once each choice is made, it is irrevocable
- For example, a greedy Scrabble player will simply maximize her score for each turn, never saving any “good” letters for possible better plays later
  - Doesn’t necessarily optimize score for entire game
- Greedy works well for the "optimal linked list with known search probabilities" problem, and reasonably well for the "optimal BST" problem
  - But does not necessarily produce an optimal tree



## Greedy Chess

- Take a piece or pawn whenever you will not lose a piece or pawn (or will lose one of lesser value) on the next turn
- Not a good strategy for this game either



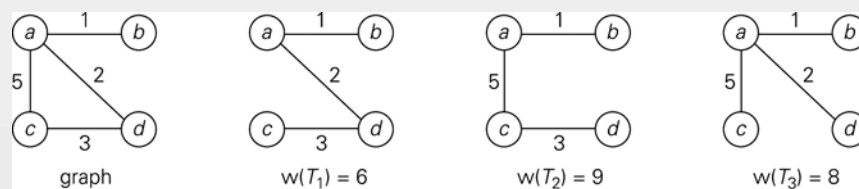
## Greedy Map Coloring

- On a planar (i.e., 2D Euclidean) connected map, choose a region and pick a color for that region
- Repeat until all regions are colored:
  - Choose an uncolored region  $R$  that is adjacent<sup>1</sup> to at least one colored region
    - If there are no such regions, let  $R$  be any uncolored region
  - Choose a color that is different than the colors of the regions that are adjacent to  $R$
  - Use a color that has already been used if possible
- The result is a valid map coloring, not necessarily with the minimum possible number of colors

<sup>1</sup> Two regions are *adjacent* if they have a common edge



## Spanning Trees for a Graph

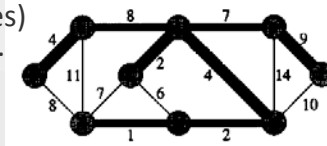


**FIGURE 9.1** Graph and its spanning trees;  $T_1$  is the minimum spanning tree



## Minimal Spanning Tree (MST)

- Suppose that we have a connected network  $G$  (a graph whose edges are labeled by numbers, which we call **weights**)
- We want to find a tree  $T$  that
  - spans the graph (i.e. contains all nodes of  $G$ ).
  - minimizes (among all spanning trees) the sum of the weights of its edges.
- Is this MST unique?
- One approach: Generate all spanning trees and determine which is minimum
- Problems:
  - The number of trees grows exponentially with  $N$
  - Not easy to generate
  - Finding a MST directly is simpler and faster



[More details soon](#)



## Huffman's algorithm

- Goal: We have a message that contains  $n$  different alphabet symbols. Devise an encoding for the symbols that minimizes the total length of the message.
- Principles: More frequent characters have shorter codes. No code can be a prefix of another.
- Algorithm: Build a tree from which the codes are derived. Repeatedly join the two lowest-frequency trees into a new tree.



Q8-10