CSE 100:
AVERAGE CASE ANALYSIS OF FIND
Average case analysis of a “successful” find

Given a BST having:

• N nodes $x_1, .. x_N$, such that $\text{key}(x_i) = k_i$

How many compares to locate a key in the BST?

1. Worst case:

2. Best case:

3. Average case:
Given a BST having:

- N nodes \(x_1, \ldots, x_N\) such that \(\text{key}(x_i) = k_i\)
- Probability of searching for key \(k_i\) is \(p_i\)

What is the expected number of comparisons to find a key?

A. \(\sum_{i=1}^{N} p_i \cdot (\text{No. of comparisons to find } k_i)\)

B. \(\sum_{i=1}^{N} p_i \cdot x_i\)

C. \(\frac{\sum_{i=1}^{N} \text{No. of comparisons to find } k_i}{N}\)
Number of compares to find key $k_i$ is related to the Depth of $x_i$ in the BST

- **Depth** of node $x_i$: No. of nodes on the path from the root to $x_i$ inclusive

- Notation for depth of $x_i$: 

![BST Diagram](image-url)
Given a BST having:
• N nodes $x_1, .. x_N$ such that $\text{key}(x_i) = k_i$
• Probability of searching for key $k_i$ is $p_i$

What is the expected number of comparisons to find a key?

A. $\sum_{i=1}^{N} p_i \cdot (\text{No. of comparisons to find } k_i)$

B. $\sum_{i=1}^{N} p_i \cdot x_i$

C. $(\sum_{i=1}^{N} \text{No. of comparisons to find } k_i) / N$
Probabilistic Assumption #1

• Probabilistic Assumption #1:
  All keys are equally likely to be searched (how realistic is this)?

• Thus $p_1 = \ldots = p_N = 1/N$ and the average number of comparisons in a successful find is:

$$D_{avg}(N) = \sum_{i=1}^{N} p_i d(x_i) = \sum_{i=1}^{N} \frac{1}{N} d(x_i) = \frac{1}{N} \left( \sum_{i=1}^{N} d(x_i) \right)$$

$$\sum_{i=1}^{N} d(x_i) = \text{total node depth}$$
Calculating total node depth

What is the total node depth of this tree?
A. 3
B. 5
C. 6
D. 9
E. None of these
Calculating total node depth

What is the total node depth of this tree?
A. 3  
B. 5  
C. 6  
D. 9  
E. None of these

So far:

Given a particular BST with $N$ nodes, we found a way to calculate average #comparisons to find a node in the BST

But that’s not enough... Why?
Calculating total node depth

- In a complete analysis of the average cases, we need to look at all possible BSTs that can be constructed with same set of N keys
- What does the structure of the tree relate to?
How many possible ways can we insert three elements into a BST?

- Suppose \( N=3 \) and the keys are \( (1, 2, 3) \)
How many possible ways can we insert three elements into a BST?

• Suppose N=3:
  
  (1,2,3); (1,3,2); (2,1,3); (2,3,1); (3,1,2); (3,2,1)

  6 possible trees

What is the total number of possibilities for an N-node BSTs?

A. \(N^N\)
B. \(N!\)
C. \(e^N\)
D. \(N*N\)
E. None of these
Given a set of N keys: The structure of a BST constructed from those keys is determined by the order the keys are inserted.

Example: N=3. There are N! = 3! = 6 different orders of insertion of the 3 keys. Here are resulting trees:
Probabilistic assumption #2

- We may assume that each key is equally likely to be the first key inserted; each remaining key is equally likely to be the next one inserted; etc.
- This leads to **Probabilistic Assumption #2**
  
  *Any insertion order (i.e. any permutation) of the keys is equally likely when building the BST*

- This means with 3 keys, each of the following trees can occur with probability 1/6
Average Case for successful Find: Brute Force Method

3, 1, 5
1, 3, 5
1, 5, 3
5, 1, 3
5, 3, 1
3, 5, 1
Average # of comparisons in a single tree

- Let $D(N)$ be the expected total depth of BSTs with $N$ nodes, over all the $N!$ possible BSTs, assuming that Probabilistic Assumption #2 holds.

$$D(N) = \sum_{\text{all BSTs } T_j \text{ with } N \text{ nodes}} \left( \text{probability of } T_j \right) \left( \text{Total Depth}(T_j) \right)$$

$$= \sum_{\text{all BSTs with } N \text{ nodes}} \left( \frac{1}{N!} \right) \left( \sum_{i=1}^{N} d(x_i) \right)$$

- If Assumption #1 also holds, the average # comparisons in a successful find is

$$D_{\text{avg}}(N) = \frac{D(N)}{N}$$

The computationally intensive part is constructing $N!$ trees to compute $N!$ total depth values: This is a brute force method!
How do we compute $D(N)$?

$$D(N) = \sum_{\text{all BSTs with } N \text{ nodes}} \left( \frac{1}{N!} \right) \left( \sum_{i=1}^{N} d(x_i) \right)$$

We need an equation for $D(N)$ that does not involve computing $N!$ total depth values (in a brute force fashion)

**Key Idea:** We will build a recurrence relation for $D(N)$ in terms of $D(N-1)$
And then solve that recurrence relation to give us a sum over $N$
(instead of $N!$)
What are we interested in finding?
A. The expected total depth over all possible BSTs with N keys
B. A recurrence relation for the expected total depth
C. Both A and B
Towards a recurrence relation for average BST total depth

What are we interested in finding?
A. The expected total depth over all possible BSTs with N keys
B. A recurrence relation for the expected total depth
C. Both A and B

Assume we have solved the smaller versions of the problem
So, we know $D(i) \ \forall i<N$
Towards a recurrence relation for average BST total depth

Define the following sub-problem:

• Find the expected depth of all trees where the root node is the $i^{th}$ largest key.

Note: the rest of the keys can be organized in any fashion in the left and right subtrees of the root.
Towards a recurrence relation for average BST total depth

Define the following sub-problem:
Find the expected depth of all trees where the root node is the \( i \)th largest key.

Which of the following best describes the consequence of fixing the root to be the \( i \)th largest key:
A. We have described our original problem in terms of a smaller version of itself
B. We are restricted to trees with a fixed number of nodes in the left and right subtrees of the root
C. We can describe our original problem as \((N-1)!\) such sub problems
Towards a recurrence relation for average BST total depth

- Define $D(N|i)$ as expected total depth of a BST with $N$ nodes, assuming that $T_L$ has $i$ nodes (and $T_R$ has $N-i-1$ nodes)
Average case analysis of find in BST

• Given N nodes, how many such subsets of trees are possible as i is varied?

A. \( N \)
B. \( N! \)
C. \( \log_2 N \)
D. \( (N-1)! \)
Probability of subtree sizes

- Let $P_N(i) = \text{the probability that } T_L \text{ has } i \text{ nodes}$
- It follows that $D(N)$ is given by the following equation

$$D(N) = \sum_{i=0}^{N-1} P_N(i)D(N \mid i)$$
Probability of subtree sizes

- Let $P_N(i) = \text{the probability that } T_L \text{ has } i \text{ nodes}$
- It follows that $D(N)$ is given by the following equation

$$D(N) = \sum_{i=0}^{N-1} P_N(i) D(N \mid i)$$

What is the value of $P_N(i)$?

Hint: use assumption #2, any of the $N$ keys are equally likely to be inserted first

A. $N$
B. It depends on $i$ and $N$
C. It depends only on $N$
Towards a recurrence relation for average BST total depth

- We defined $D(N|i)$ as the expected total depth of a BST with $N$ nodes, assuming that $T_L$ has $i$ nodes (and $T_R$ has $N-i-1$)

What is $D(N|i)$ in terms of $D(i)$ & $D(N-i-1)$?

Hint: all nodes in each subtree are 1 deeper in tree $T$

A. $D(i) + D(N-i-1)$
B. $D(i) + D(N-i-1) + 1$
C. $D(i) + D(N-i-1) + N$

$$D(N) = \sum_{i=0}^{N-1} P_N(i)D(N|i)$$
Towards a recurrence relation for average BST total depth

What is $D(N|i)$ in terms of $D(i)$ & $D(N-i-1)$?

Hint: all nodes in each subtree are 1 deeper in tree $T$

A. $D(i) + D(N-i-1)$
B. $D(i) + D(N-i-1) + 1$
C. $D(i) + D(N-i-1) + N$
Average total depth of a BST with $N$ nodes

$$D(N) = \sum_{i=0}^{N-1} P_N(i) D(N \mid i)$$

$$D(N) = \sum_{i=0}^{N-1} \frac{1}{N} [D(i) + D(N - i - 1) + N]$$

$$= \frac{1}{N} \sum_{i=0}^{N-1} D(i) + \frac{1}{N} \sum_{i=0}^{N-1} D(N - i - 1) + N$$

True or false: The term in the blue box is equal to the term in the red box
A. True
B. False
• Note that those two summations just add the same terms in different order; so

\[ D(N) = \frac{2}{N} \sum_{i=0}^{N-1} D(i) + N \]

• ... and multiplying by \( N \),

\[ ND(N) = 2 \sum_{i=0}^{N-1} D(i) + N^2 \]

• Now substituting \( N-1 \) for \( N \),

\[ (N-1)D(N-1) = 2 \sum_{i=0}^{N-2} D(i) + (N-1)^2 \]

• Subtracting that equation from the one before it gives

\[ ND(N) - (N-1)D(N-1) = 2D(N-1) + N^2 - (N-1)^2 \]

• ... and collecting terms finally gives this recurrence relation on \( D(N) \):

\[ ND(N) = (N + 1)D(N-1) + 2N - 1 \]
N*D(N) = (N+1) *D(N-1) + 2N – 1

How does this help us, again?
A. We can solve it to yield a formula for D(N) that does not involve N!
B. We can use it to compute D(N) directly
C. I have no idea, I’m totally lost
Through unwinding and some not-so-complicated algebra (which you can find in your reading, a.k.a. Paul’s slides) we arrive at:

\[ND(N) = (N+1)D(N-1) + 2N - 1\]

No N! to be seen! Yay!

And with a little more algebra, we can even show an approximation:

\[D(N) = 2(N+1) \sum_{i=1}^{N} \frac{1}{i} - 3N\]

Conclusion: The average time to find an element in a BST with no restrictions on shape is \(\Theta(\log N)\).
The importance of being balanced

• A binary search tree has average-case time cost for Find = \( \Theta (\log N) \):
What does this analysis tell us:
• On an average things are not so bad provided assumptions 1 and 2 hold
• But the probabilistic assumptions we made often don’t hold in practice
  • Assumption #1 may not hold: we may search some keys many more times than others
  • Assumption #2 may not hold: approximately sorted input is actually quite likely, leading to unbalanced trees with worst-case cost closer to \( O(N) \) when \( N \) is large
• We would like our search trees to be balanced
The importance of being balanced

- We would like our search trees to be balanced
- Two kinds of approaches
  - Deterministic methods guarantee balance, but operations are somewhat complicated to implement (AVL trees, red black trees)
  - Randomized methods (treaps, Randomized Search Trees from our result) – deliberate randomness in constructing the tree helps!!
    - Operations are simpler to implement
    - Balance not absolutely guaranteed, but achieved with high probability
- We will return to this topic later in the course
Changing gears: Data Compression Problem

- What do we do with data?
- What is the encoding scheme that would result in the shortest binary representation?
- Why is this question important?
How do we encode data?

• Step 1: Figure out the ‘alphabets’ that constitute the data

• The data is a sequence of the these alphabets

• Think on these:
  • What are the alphabets in a black and white image?
  • What are the alphabets in a colored image?
  • What are the alphabets in a text file?
How do we encode data?

- Step 2: Determine the binary code word for each alphabet
- Step 3: Replace each alphabet by its code word
- For example if the alphabet was ‘s’, ‘p’, ’a’, ‘m’ we might define the following encoding:

<table>
<thead>
<tr>
<th>Alphabet</th>
<th>Code word</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td></td>
</tr>
<tr>
<td>p</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td></td>
</tr>
<tr>
<td>m</td>
<td></td>
</tr>
</tbody>
</table>
Fixed length encoding

- In fixed length, each alphabet is represented using a fixed number of bits.
- For example if the alphabet was ‘s’, ‘p’, ’a’, ‘m’ we might define the following encoding:

<table>
<thead>
<tr>
<th>Alphabet</th>
<th>Code word</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>00</td>
</tr>
<tr>
<td>p</td>
<td>01</td>
</tr>
<tr>
<td>a</td>
<td>10</td>
</tr>
<tr>
<td>m</td>
<td>01</td>
</tr>
</tbody>
</table>

For a dictionary consisting of M symbols, what is the minimum number of bits needed to encode each symbol (assume fixed length binary codes)?

A. $2^M$    B. M    C. M/2    D. $\text{ceil}(\log_2 M)$    E. None of these
Code length and file size

- The longer the code length, the bigger the size of the binary representation of the data.

<table>
<thead>
<tr>
<th>Alphabet</th>
<th>Code word</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>00</td>
</tr>
<tr>
<td>p</td>
<td>01</td>
</tr>
<tr>
<td>a</td>
<td>10</td>
</tr>
<tr>
<td>m</td>
<td>01</td>
</tr>
</tbody>
</table>

Q: For the above text file that consists of 24 alphabets, what is the size of the binary encoded file using a fixed length encoding?

A. $2^{24}$
B. 24
C. 12
D. $\text{ceil}(\log_2 24)$
E. $2^*24$

In general the size of the binary file =
The longer the code length, the bigger the size of the binary representation of the data.

Q: For the above text file that consists of 24 alphabets, what is the size of the binary encoded file using a fixed length encoding?

A. $2^{24}$
B. 24
C. 12
D. $\text{ceil}(\log_2 24)$
E. $2*24$

In general the size of the binary file
= Average code length * number of alphabets
Variable length codes

Is code B better than code A?
A. Yes
B. No
C. Depends
Variable length codes

Text file

Symbol | Frequency
--- | ---
\(s\) | 0.6
\(p\) | 0.2
\(a\) | 0.1
\(m\) | 0.1

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s)</td>
<td>00</td>
</tr>
<tr>
<td>(p)</td>
<td>01</td>
</tr>
<tr>
<td>(a)</td>
<td>10</td>
</tr>
<tr>
<td>(m)</td>
<td>11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s)</td>
<td>0</td>
</tr>
<tr>
<td>(p)</td>
<td>1</td>
</tr>
<tr>
<td>(a)</td>
<td>10</td>
</tr>
<tr>
<td>(m)</td>
<td>11</td>
</tr>
</tbody>
</table>

Average length (code A) = 2 bits/symbol
Average length (code B) = \(0.6 \times 1 + 0.2 \times 1 + 0.1 \times 2 + 0.1 \times 2\) = 1.2 bits/symbol
Decoding variable length codes

Text file

Code A

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>00</td>
</tr>
<tr>
<td>p</td>
<td>01</td>
</tr>
<tr>
<td>a</td>
<td>10</td>
</tr>
<tr>
<td>m</td>
<td>11</td>
</tr>
</tbody>
</table>

Code B

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0</td>
</tr>
<tr>
<td>p</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>10</td>
</tr>
<tr>
<td>m</td>
<td>11</td>
</tr>
</tbody>
</table>

Decode the binary pattern 0110 using Code B?
A. spa
B. sms
C. Not enough information to decode
Decoding variable length codes

Variable length codes have to necessarily be prefix codes for correct decoding.

A *prefix code* is one where no symbol’s codeword is a prefix of another.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>00</td>
</tr>
<tr>
<td>p</td>
<td>01</td>
</tr>
<tr>
<td>a</td>
<td>10</td>
</tr>
<tr>
<td>m</td>
<td>11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0</td>
</tr>
<tr>
<td>p</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>10</td>
</tr>
<tr>
<td>m</td>
<td>11</td>
</tr>
</tbody>
</table>

Code B is not a prefix code. Why is this a problem?
Decoding variable length codes

Is code C better than code A and code B? (Why or why not?)
A. Yes
B. No

Code A

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>00</td>
</tr>
<tr>
<td>p</td>
<td>01</td>
</tr>
<tr>
<td>a</td>
<td>10</td>
</tr>
<tr>
<td>m</td>
<td>11</td>
</tr>
</tbody>
</table>

Code B

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0</td>
</tr>
<tr>
<td>p</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>10</td>
</tr>
<tr>
<td>m</td>
<td>11</td>
</tr>
</tbody>
</table>

Code C

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0</td>
</tr>
<tr>
<td>p</td>
<td>10</td>
</tr>
<tr>
<td>a</td>
<td>110</td>
</tr>
<tr>
<td>m</td>
<td>111</td>
</tr>
</tbody>
</table>

Symbol Frequency

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0.6</td>
</tr>
<tr>
<td>p</td>
<td>0.2</td>
</tr>
<tr>
<td>a</td>
<td>0.1</td>
</tr>
<tr>
<td>m</td>
<td>0.1</td>
</tr>
</tbody>
</table>
What is the best possible average length of a coded symbol with these frequencies?

A. 0
B. 0.67
C. 1.0
D. 1.57
E. 2.15
**Problem Definition**

Given a frequency distribution over $M$ symbols, find the optimal prefix binary code i.e. one that minimizes the average number of bits per symbol.

<table>
<thead>
<tr>
<th>Letter</th>
<th>freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0.6</td>
</tr>
<tr>
<td>p</td>
<td>0.2</td>
</tr>
<tr>
<td>a</td>
<td>0.1</td>
</tr>
<tr>
<td>m</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Solution involves thinking of binary codes as **Binary Trees**.
Binary codes as Binary Trees

```
          0  1
         / \ /
        0   1 0  1
       / \ / \ / \ /
      s p a m
```
Given the above binary tree, what is the binary encoding of the string “papa”
A. 11101110
B. 01100110
C. 01101000
D. None of the above
Decode the bitstream 110101001100 using the given binary tree
A. scam
B. mork
C. rock
D. korp
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>00</td>
</tr>
<tr>
<td>P</td>
<td>01</td>
</tr>
<tr>
<td>A</td>
<td>10</td>
</tr>
<tr>
<td>M</td>
<td>11</td>
</tr>
</tbody>
</table>

**Code B**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>0</td>
</tr>
<tr>
<td>P</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>10</td>
</tr>
<tr>
<td>M</td>
<td>11</td>
</tr>
</tbody>
</table>

**Code C**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>0</td>
</tr>
<tr>
<td>P</td>
<td>10</td>
</tr>
<tr>
<td>A</td>
<td>110</td>
</tr>
<tr>
<td>M</td>
<td>111</td>
</tr>
</tbody>
</table>
Problem Definition (revisited)

Input: The frequency of occurrence of each symbol

Output: Binary tree $T$ that minimizes the following objective function:

$$L(T) = \sum_{i \in \Phi} p_i \cdot Depth(i \text{ in } T)$$

Solution: Huffman Codes
Huffman coding is one of the fundamental ideas that people in computer science and data communications are using all the time - Donald Knuth
Huffman’s algorithm: Bottom up construction

- Build the tree from the bottom up!
- Start with a forest of trees, all with just one node
Huffman’s algorithm: Bottom up construction

• Build the tree from the bottom up!
• Start with a forest of trees, all with just one node
• Merge trees in the forest two at a time to get a single tree
Huffman’s algorithm: Bottom up construction

What should be the merge criterion?

A 6
B 4
C 4
G 1
H 2
Huffman’s algorithm: Bottom up construction

• Merging the nodes G and H increases the code length of which of the following symbols:

A. Symbols A, B and C
B. Symbol G
C. Symbol H
D. Both G and H
Huffman’s algorithm: Bottom up construction

• Choose the two smallest trees in the forest and merge them
Huffman’s algorithm: Bottom up construction

- Choose the two smallest trees in the forest and merge them

T1 now represents the meta symbol ‘GH’
What is the count associated with T1?

A. Max(count (G), count (H))
B. (count (G) + count (H))/2
C. (count (G) + count (H))
Huffman’s algorithm: Bottom up construction

- Choose the two smallest trees in the forest and merge them
- Repeat until all nodes are in the tree
Huffman’s algorithm: Bottom up construction

- A
- B
- C
- G
- H
- T

1. A, B (6, 4)
2. T (1)
3. G, H (7)
4. T (7)
5. T, C (10)
6. T, T (2)
Huffman’s algorithm: Bottom up construction

- Build the tree from the bottom up!
- Start with a forest of trees, all with just one node
- Choose the two smallest trees in the forest and merge them
- Repeat until all nodes are in the tree
You Try It!

<table>
<thead>
<tr>
<th>Letter</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>40</td>
</tr>
<tr>
<td>c</td>
<td>20</td>
</tr>
<tr>
<td>s</td>
<td>15</td>
</tr>
<tr>
<td>d</td>
<td>15</td>
</tr>
<tr>
<td>y</td>
<td>6</td>
</tr>
<tr>
<td>a</td>
<td>4</td>
</tr>
</tbody>
</table>

Build the tree and write down the codes for each of the symbols

Then encode the string “cya” using this code
Congratulations!

….and we look forward to seeing you in week 5.