CSE 101- Winter ‘16
Discussion Section
February 1st 2016
Topics for today

Greedy algorithms

- Proofs using the exchange argument
- Minimum spanning trees
  - Prim’s Algorithm
  - Kruskal’s Algorithm
- Huffman Coding
  - Problem Specification
  - Example
  - Prefix-codes
Greedy Algorithms

- Make the decision that gives you the most immediate benefit
- Not always optimal, but can be
- Useful for providing an approximation when solving the exact optimal solution would be prohibitively expensive or very complicated
How to Approach Greedy Algorithms

1. Identify greedy parameter—the value you want to increase the most with each iteration

2. Figure out how you want to order the elements you will be selecting

3. Define your selection procedure
Greedy Algorithms

- Not always optimal
- Need proof of optimality
Sample problem

• Suppose that you are driving from San Diego to Harmony, CA, a drive of D miles. Suppose your can can drive m miles on a full tank of gas.

• You know that there are n gas stations $s_1, \ldots, s_n$ along the way. Each gas station $s_i$ is at a distance $d_i$ miles from the start of the drive, where $0 < d_{i-1} < d_i < D$ for all $i$.

• You want to determine the sequence of gas stations you should stop at to minimize the number of times you have to stop for gas along the drive. Find an algorithm that will produce one such sequence of gas stations.
The algorithm

- What is the greedy parameter?
The algorithm

● What is the greedy parameter?
  ▪ The amount of distance traveled between stopping for gas
● Keep driving as far as you can before stopping for gas. Choose the farthest away gas station reachable from the previously chosen gas station.
● Proof of optimality: ??
The Exchange Method

- Assume for contradiction that some optimal solution exists that is different from the solution produced by your greedy method.
- Eventually its solution must differ from your greedy solution.
- It may be possible to change parts of the assumed optimal solution without changing its optimality.
- Prove that the greedy method does just as well or better than the optimal method.
Proof that Greedy is Optimal

- Assume for contradiction that there exists an optimal solution produced by some algorithm that produces a smaller sequence of gas stations than our greedy algorithm.
- We don’t specify anything about the algorithm so this proof will generalize to all possible solutions.
- Call the optimal solution $s_1, \ldots, s_k$ and the greedy solution $g_1, \ldots, g_m$, where $m$ and $k$ are positive integers.
Applying the Exchange Method

- Call the optimal solution \( s_1, \ldots, s_k \) and the greedy solution \( g_1, \ldots, g_m \), where \( m \) and \( k \) are positive integers.
- The solutions may be the same for a while, but eventually there will be some index \( i \) for which \( s_i \neq g_i \).
- Furthermore, \( g_i \) must be farther along than \( s_i \) because our algorithm chooses the farthest away possible gas station.
- Therefore, the optimal solution has fallen behind our greedy solution.
Applying the Exchange Method

- In order for the optimal solution to be better than the greedy solution, it must catch up and beat the greedy solution.
- However, since the greedy solution is ahead, then every gas station reachable by the optimal solution is also reachable by the greedy solution.
- Since the greedy solution chooses the one that is farthest away, the optimal solution can never beat the greedy solution.
Applying the Exchange Method

- We assumed that this optimal solution was better than our greedy solution, but we just proved that the greedy solution must be equivalent or better.

- Therefore, it is impossible to have a solution that beats the greedy solution.
Minimum Spanning Tree

Problem:
• Given a connected graph G with positive edge weights, find a minimum weight set of edges that connects all of the vertices.

Eg:
A **minimum spanning tree** is a subgraph of an undirected weighted graph $G$, such that

- it is a tree (i.e., it is acyclic)
- it covers all the vertices $V$
  - contains $|V| - 1$ edges
- the total cost associated with tree edges is the minimum among all possible spanning trees
- not necessarily unique

Two well known algorithms: Prim’s and Kruskal’s
Prim’s Algorithm

- Remove all edges from the graph and choose a starting vertex.
- Re-add the smallest edge that would connect a formerly unconnected vertex to the connected component containing the starting vertex.
- Repeat until all vertices are connected.
Prim’s Algorithm

Select any vertex

A

Select the shortest edge connected to that vertex

AB  3
Prim’s Algorithm

Select the shortest edge connected to any vertex already connected.

AE 4
Select the shortest edge connected to any vertex already connected.

Prim's Algorithm
Prim’s Algorithm

Select the shortest edge connected to any vertex already connected.

DC 4
Select the shortest edge connected to any vertex already connected.

Prim’s Algorithm

EF 5
Prim’s Algorithm

All vertices have been connected.

The solution is

- AB 3
- AE 4
- ED 2
- DC 4
- EF 5

Total weight of tree: 18
Kruskal’s Algorithm

- Remove all edges from the graph and sort them by ascending weights
- Iterating through the sorted edges, add them back to the graph unless adding them would create a cycle
- Every edge you add should connect two connected components
- Stop when all vertices are connected
List the edges in order of size:

- ED 2
- AB 3
- AE 4
- CD 4
- BC 5
- EF 5
- CF 6
- AF 7
- BF 8
- CF 8
Kruskal’s Algorithm

Select the shortest edge in the network

ED 2
Select the next shortest edge which does not create a cycle

ED  2
AB  3
Select the next shortest edge which does not create a cycle

- ED 2
- AB 3
- CD 4 (or AE 4)

Kruskal’s Algorithm
Select the next shortest edge which does not create a cycle

ED  2
AB  3
CD  4
AE  4
Kruskal’s Algorithm

Select the next shortest edge which does not create a cycle

ED 2
AB 3
CD 4
AE 4
BC 5 – forms a cycle
EF 5
All vertices have been connected.

The solution is

- ED 2
- AB 3
- CD 4
- AE 4
- EF 5

Total weight of tree: 18
Huffman Coding

Problem Specification

- Given: A set of symbols and their weights (usually proportional to probabilities).
- Find: A set of code words with minimum average code word length.
Huffman Coding

- Proposed by Dr. David A. Huffman in 1952 “A Method for the Construction of Minimum Redundancy Codes”

- Applicable to many forms of data transmission
  - Our example: text files
Huffman Coding

- Fixed-length coding: all characters are allocated the same amount of space for their code words.

- But not all characters occur with the same frequency.

- Can save space using variable length-coding.
Huffman Coding

- Uses a specific method for choosing the representation for each symbol
- Produces *variable length* code words
- Shorter code words for symbols with higher frequency.
- Eg: ‘a’-010, ‘e’- 000, ‘l’- 11001
Algorithm

1. Scan text to be compressed and calculate the frequency/probability of each character.
2. Sort the characters based on number of occurrences in text.
3. Build Huffman code tree based on the sorted list.
4. Perform a traversal of tree to determine all code words.
5. Scan text again and create new file using the Huffman codes.
Consider the following short text:

\[ Eerie \text{ eyes seen near lake.} \]

Count up the occurrences of all characters in the text
Building a Tree
Scan the original text
Eerie eyes seen near lake.

• What is the frequency of each character in the text?

<table>
<thead>
<tr>
<th>Character</th>
<th>Frequency</th>
<th>Character</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>1</td>
<td>s</td>
<td>2</td>
</tr>
<tr>
<td>e</td>
<td>8</td>
<td>n</td>
<td>2</td>
</tr>
<tr>
<td>r</td>
<td>2</td>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>l</td>
<td>1</td>
<td>l</td>
<td>1</td>
</tr>
<tr>
<td>space</td>
<td>4</td>
<td>k</td>
<td>1</td>
</tr>
<tr>
<td>y</td>
<td>1</td>
<td>.</td>
<td>1</td>
</tr>
</tbody>
</table>
Building a Tree
Prioritize characters

• Create binary tree nodes with character and frequency of each character

• Place nodes in a priority queue (implemented using min-heap)
  – The lower the occurrence, the higher the priority in the queue
Building a Tree

• The queue after inserting all nodes

```
E
i
y
l
k
.
r
s
n
a
sp
e
1
1
1
1
1
1
2
2
2
4
8
```
Building a Tree

- While priority queue contains two or more nodes
  - Create new node
  - Dequeue node and make it left subtree
  - Dequeue next node and make it right subtree
  - Frequency of new node equals sum of frequency of left and right children
  - Enqueue new node back into queue
Building a Tree
Building a Tree

```
<table>
<thead>
<tr>
<th>y</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

| 1 |

<table>
<thead>
<tr>
<th>k</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>.</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>r</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sp</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

```

```
2

<table>
<thead>
<tr>
<th>E</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>i</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
```
Building a Tree
Building a Tree
Building a Tree
Building a Tree
Building a Tree
Building a Tree
Building a Tree
Building a Tree
Building a Tree
Building a Tree
Building a Tree
Building a Tree
What is happening to the characters with a low number of occurrences?
Building a Tree
Building a Tree
Building a Tree
Building a Tree
Encoding the File

Traverse Tree for Codes

• Perform a traversal of the tree to obtain new code words
• Going left is a 0 going right is a 1
• Code-word is only completed when a leaf node is reached
Encoding the File
Traverse Tree for Codes

<table>
<thead>
<tr>
<th>Char</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>0000</td>
</tr>
<tr>
<td>i</td>
<td>0001</td>
</tr>
<tr>
<td>y</td>
<td>0010</td>
</tr>
<tr>
<td>l</td>
<td>0011</td>
</tr>
<tr>
<td>k</td>
<td>0100</td>
</tr>
<tr>
<td>.</td>
<td>0101</td>
</tr>
<tr>
<td>space</td>
<td>0111</td>
</tr>
<tr>
<td>e</td>
<td>10</td>
</tr>
<tr>
<td>r</td>
<td>1100</td>
</tr>
<tr>
<td>s</td>
<td>1101</td>
</tr>
<tr>
<td>n</td>
<td>1110</td>
</tr>
<tr>
<td>a</td>
<td>1111</td>
</tr>
</tbody>
</table>
Encoding the File

• Rescan text and encode file using new code words

Eerie eyes seen near lake.

<table>
<thead>
<tr>
<th>Char</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>0000</td>
</tr>
<tr>
<td>i</td>
<td>0001</td>
</tr>
<tr>
<td>y</td>
<td>0010</td>
</tr>
<tr>
<td>l</td>
<td>0011</td>
</tr>
<tr>
<td>.</td>
<td>0100</td>
</tr>
<tr>
<td>space</td>
<td>0101</td>
</tr>
<tr>
<td>e</td>
<td>0110</td>
</tr>
<tr>
<td>r</td>
<td>0111</td>
</tr>
<tr>
<td>a</td>
<td>1000</td>
</tr>
<tr>
<td>n</td>
<td>1001</td>
</tr>
<tr>
<td>a</td>
<td>1110</td>
</tr>
</tbody>
</table>

• Why is there no need for a separator character?
Prefix codes

- No code word is a prefix of any other code.
- Huffman coding produces prefix codes.
- Eg:
  - \{1,23,44\} has the prefix property
  - \{1,23,14\} does not satisfy the prefix property
Encoding the File  
Results

- Have we made things any better?
- 73 bits to encode the text
- ASCII would take $8 \times 26 = 208$ bits
- If modified code used 4 bits per character are needed. Total bits $4 \times 26 = 104$. Savings not as great
Decoding the File

- Once receiver has tree it scans incoming bit stream
- 0 ⇒ go left
- 1 ⇒ go right

Eerie eyes seen near lake.
Thank You..!!