Fundamental Algorithms
Topic 10: Greedy Algorithms and Data Compression

By Adam Sheffer

Greedy Algorithms

- A greedy algorithm.
  - At each step, the algorithm performs the choice that looks best locally at that moment.
  - No global long-term strategy.
Lightest Paths

• Problem.
  ◦ Find lightest paths from a vertex $s$ in a graph with edge weights.

• Dijkstra’s algorithm is greedy.
  ◦ At each step it adds to the tree the lightest edge it found.

Lightest Paths

• Problem.
  ◦ Find lightest paths from a vertex $s$ in a graph with edge weights.

• The greedy strategy fails when there are negative weights.

• Bellman-Ford is not greedy. It has a global strategy.
Transporting Materials

- We need to choose materials to place on a ship.
  - We can transport at most $n$ tons.
  - We have a list of materials. For each material, we know how many tons of it are available and the profit when transporting such a ton.
  - Our goal is to maximize the profit.

<table>
<thead>
<tr>
<th>Material</th>
<th>Amount (tons)</th>
<th>Profit per ton</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oil</td>
<td>300</td>
<td>500</td>
</tr>
<tr>
<td>Gold</td>
<td>20</td>
<td>1000</td>
</tr>
<tr>
<td>Chicken soup</td>
<td>20,000</td>
<td>100</td>
</tr>
<tr>
<td>Lizards</td>
<td>20</td>
<td>300</td>
</tr>
</tbody>
</table>

Example

- Maximize profit when we can transport up to 300 tons.

<table>
<thead>
<tr>
<th>Material</th>
<th>Amount (tons)</th>
<th>Profit per ton</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oil</td>
<td>300</td>
<td>500</td>
</tr>
<tr>
<td>Gold</td>
<td>20</td>
<td>1000</td>
</tr>
<tr>
<td>Chicken soup</td>
<td>20,000</td>
<td>100</td>
</tr>
<tr>
<td>Lizards</td>
<td>20</td>
<td>600</td>
</tr>
</tbody>
</table>
Greedy Solution

- **Solution.**
  - Repeat as long as there is room left: Out of the remaining materials, take as much of the *most profitable material* as possible.
  - *m materials.*
  - Running time: $O(m \log m)$.

The Knapsack Problem

- Our *bag can hold up to n pounds.*
  - We have a list of items that we can put in the bag. *For each item, we have its weight and its price.*
  - We wish to put items in the bag, *maximizing the sum of their prices.*
Maximize the Price

- Greedy algorithm fails!
- Equivalent to solving \( P \) vs \( NP \).

Principles of the Greedy Approach

- When **trying to use a greedy approach**:  
  - Find a **greedy step** that can be performed repeatedly  
  - Harder part: **Prove that this greedy step leads to an optimal solution**.  
- Some greedy approaches do not always lead to an optimal solution.  
  - **Dijkstra's greedy** approach fails when there are negative weights.  
  - **Shipping works** but **knapsack fails**.
Discuss in Groups

• Problem.
  ◦ We wish to have $n$ cents using as few coins as possible.
  ◦ We are allowed to use quarters (25c), dimes (10c), nickels (5c), and pennies (1c).
  ◦ Describe an efficient greedy algorithm for the problem.
  ◦ Will your algorithm always find an optimal solution?

Optimal Solution

• Why does the greedy algorithm work when there are quarters (25c), dimes (10c), nickels (5c), and pennies (1c)?
  ◦ Optimality can be proved by induction on $n$.
  ◦ (Not in this course)
Discuss in Groups

- Problem.
  - We used quarters (25c), dimes (10c), nickels (5c), and pennies (1c).
  - Make up a different set of coins for which the greedy algorithm fails.
  - The set must include pennies, so that we cannot get stuck.
  - It suffices to have two other types of coins.

Lecture Scheduling

- Problem.
  - We manage a lecture hall at our college.
  - We are given a set of n lectures that wish to use our hall, each with a start time and end time.
  - We wish to assign as many lectures as possible to the hall, without choosing two lectures that overlap.
Example

- $s_i$ – start time.
- $f_i$ – finish time.

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>$f_i$</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>16</td>
</tr>
</tbody>
</table>

- We cannot take both lectures 3 and 6.
- We can choose lectures 3, 9, and 11.
- Can you do better?

Greedy Approaches

- $s_i$ – start time.
- $f_i$ – finish time.

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>$f_i$</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>16</td>
</tr>
</tbody>
</table>

- Can you suggest greedy approaches for this problem?
A Greedy Approach

- $f_i$ – finish time.
- Rename the lectures so that $f_1 \leq f_2 \leq \cdots \leq f_n$.
- **Greedy approach:** What would locally be the best choice to make?
  - Take the lecture that ends first.
  - This choice leaves the largest vacant time period for additional lectures.

More Formally

- $f_1 \leq f_2 \leq \cdots \leq f_n$.
- $L_i$ – lecture ending at time $f_i$.
- **Claim.** There is an optimal choice of lectures that includes $L_1$.
  - Consider an optimal set $S$ of lectures.
  - $L'$ – first lecture in $S$, ending at time $f'$.
  - (If $L' = L_1$ then we are done.)
  - We have $f_1 \leq f'$, so $L_1$ does not overlap with any lecture of $S$, except for $L'$.
  - Replacing $L_1$ with $L'$ in $S$ leads to a set of lectures of the same size.
More Formally

- \( f_1 \leq f_2 \leq \cdots \leq f_n \).
- \( L_i \) – lecture ending at time \( f_i \).
- Claim. There is an optimal choice of lectures that includes \( L_1 \).
- After choosing \( L_1 \), we discard lectures that overlaps with \( L_1 \). We remain with a smaller problem to solve.
- Repeat until no lectures are left.
  - If the first remaining lecture is \( L_9 \), then exists an optimal solution containing also \( L_9 \).
  - (Same proof as in the previous slide.)

Example

- \( s_i \) – start time.
- \( f_i \) – finish time.

<table>
<thead>
<tr>
<th>( i )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_i )</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>( f_i )</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>16</td>
</tr>
</tbody>
</table>

- Use our greedy algorithm to solve this.
The Algorithm

- Algorithm.
  - Repeat until no lectures are left.
    - Take the lecture $L^*$ with the **earliest ending time**.
    - **Discard every lecture that overlaps** with $L^*$.
  - Leads to optimal solution.
- What is the best **running time** that you can get for this algorithm? **How?**

Running Time

- $n$ lectures.
- Sort lecture by finish time: $O(n \log n)$.
- $F$ – latest finish time among lecture we took. At first $F = 0$.
- **For $i$ from 1 to $n$:**
  - If $s_i < F$, discard $L_i$.
  - If $s_i \geq F$, add $L_i$ to our set. Set $F = f_i$.
- This process takes $O(n)$ time.
- Total running time: $O(n \log n)$. 
Data Compression

**Problem.**
- We are given a big text file.
- We wish to compress the file, so that the amount of storage used is smaller, but no information is lost.
Warm-up

- We know that **no digits appear in the text**.
- We want to compress a text of the form:
  
  \[ \text{aaaaaaaaaabcbcbcbcbcaaaaadefdefdef} \]
  
  ◦ We can write \[ \text{9a5bc4a3def} \].
  ◦ Instead of **21** characters, we only store **11**.
  ◦ Useful only for very special types of texts.

Warm-up 2

- We know that the **symbol \$ is not used in the text**.
- Set \$ = “**round**” and replace each instance of “round” with \$.
- Better idea: set \$ = “**round and round**”.

---

**The Wheels On The Bus**

The wheels on the bus go
round and round,
round and round,
round and round.
The wheels on the bus go
round and round,
All the way to town.
Characters and Bits

- **Recall.** Computers store data in binary format: zeros and ones.
- **Example.**
  - A standard way of representing the capital letters.
  - Every letter takes 7 bits.

<table>
<thead>
<tr>
<th>ASCII Alphabet</th>
</tr>
</thead>
<tbody>
<tr>
<td>A  1000001</td>
</tr>
<tr>
<td>B  1000010</td>
</tr>
<tr>
<td>C  1000011</td>
</tr>
<tr>
<td>D  1000100</td>
</tr>
<tr>
<td>E  1000101</td>
</tr>
<tr>
<td>F  1000110</td>
</tr>
<tr>
<td>G  1000111</td>
</tr>
<tr>
<td>H  1001000</td>
</tr>
<tr>
<td>I  1001001</td>
</tr>
<tr>
<td>J  1001010</td>
</tr>
<tr>
<td>K  1001011</td>
</tr>
<tr>
<td>L  1001100</td>
</tr>
<tr>
<td>M  1001101</td>
</tr>
</tbody>
</table>

Huffman Codes

- Huffman coding is **probably the most common compression technique today.**
  - It is used in **zip files.**
  - It is used in **mp3 files.**
  - It is used in common **formats of image files** (specifically in cases where we don’t want to lose any data).
Fixed-length Codewords

- Text file with **100,000 characters**.
  - For simplicity, assume that the **only characters in the file are a,b,c,d,e,f**.
  - We can represent each letter using **three binary digits (bits)**.

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency (in thousands)</td>
<td>45</td>
<td>13</td>
<td>12</td>
<td>16</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>Fixed-length codeword</td>
<td>000</td>
<td>001</td>
<td>010</td>
<td>011</td>
<td>100</td>
<td>101</td>
</tr>
</tbody>
</table>

- The size of the file is **100,000 · 3 = 300,000** bits.

Variable-length Codewords

- Text file with **100,000 characters**.
  - For simplicity, assume that the **only characters in the file are a,b,c,d,e,f**.
  - We use fewer bits for common characters.

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency (in thousands)</td>
<td>45</td>
<td>13</td>
<td>12</td>
<td>16</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>Fixed-length codeword</td>
<td>000</td>
<td>001</td>
<td>010</td>
<td>011</td>
<td>100</td>
<td>101</td>
</tr>
<tr>
<td>Variable-length codeword</td>
<td>0</td>
<td>101</td>
<td>100</td>
<td>111</td>
<td>1101</td>
<td>1100</td>
</tr>
</tbody>
</table>

- The size of the file is

\[
45,000 \cdot 1 + 13,000 \cdot 3 + 12,000 \cdot 3 + 16,000 \cdot 3 + 9,000 \cdot 4 + 5,000 \cdot 4 = 224,000.
\]

- Better than the original 300,000.
Can We Read the Text?

- We need less space when using variable-length codewords.
  - Can we still read the text?
  - What is the text stored as 0101100?
  - The only way to read this is as 0-101-100, which is “abc”.

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency (in thousands)</td>
<td>45</td>
<td>13</td>
<td>12</td>
<td>16</td>
<td>9</td>
</tr>
<tr>
<td>Fixed-length codeword</td>
<td>000</td>
<td>001</td>
<td>010</td>
<td>011</td>
<td>100</td>
</tr>
<tr>
<td>Variable-length codeword</td>
<td>0</td>
<td>101</td>
<td>100</td>
<td>111</td>
<td>1101</td>
</tr>
</tbody>
</table>

Prefix Codes

- In a **prefix code**, no codeword is a prefix of another codeword.
  - Makes the decoding easier.
  - Can be thought of as a **tree with the letters as leaves**.

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency (in thousands)</td>
<td>45</td>
<td>13</td>
<td>12</td>
<td>16</td>
<td>9</td>
</tr>
<tr>
<td>Fixed-length codeword</td>
<td>000</td>
<td>001</td>
<td>010</td>
<td>011</td>
<td>100</td>
</tr>
<tr>
<td>Variable-length codeword</td>
<td>0</td>
<td>101</td>
<td>100</td>
<td>111</td>
<td>1101</td>
</tr>
</tbody>
</table>
Huffman’s Algorithm

- **Huffman’s algorithm** builds the tree in steps, using a greedy approach.
  - $n$ different **letters**.
  - We start with $n$ separate **trees**, each is a single **leaf** that corresponds to a **letter**.
  - The **weight** of a tree is the **number of letters in the file** that correspond to it.

```
f:5  e:9  c:12  b:13  d:16  a:45
```

Huffman’s Algorithm (cont.)

- **Huffman’s algorithm**.
  - Start with separate trees: each tree is one leaf that corresponds to a letter.
  - The **weight** of a tree is the **number of letters in the file** that correspond to it.
  - **Repeat** until there is only one tree.
    - Merge the two lightest **trees** by adding a common root for them.
    - **New weight** = sum of both original weights.

```
f:5  e:9  c:12  b:13  d:16  a:45
```
Create a Huffman Code

<table>
<thead>
<tr>
<th>Letter</th>
<th>frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>28</td>
</tr>
<tr>
<td>B</td>
<td>100</td>
</tr>
<tr>
<td>C</td>
<td>20</td>
</tr>
<tr>
<td>D</td>
<td>20</td>
</tr>
<tr>
<td>E</td>
<td>60</td>
</tr>
<tr>
<td>F</td>
<td>50</td>
</tr>
<tr>
<td>G</td>
<td>10</td>
</tr>
</tbody>
</table>
Why is this Greedy?

- Why do we say that Huffman's alg is greedy?
  - The less frequent a letters is, the lower it should be in the tree.
  - At each step, we take the least frequent subtrees we have, and place them lower in the tree we are building.

- Repeat until there is one tree.
  - Merge the two lightest trees by adding a common root for them.
  - New weight = sum of both original weights.

Huffman Codes are Optimal

- Huffman’s algorithm produces an optimal tree (minimizes the storage).
  - We skip the proof.
  - Intuitively, this can be proved similarly to proof of the lecture assignment problem.
    - Start with an optimal tree $T$.
    - Change $T$ so that the two least frequent letters have the same parent. The tree cannot become worse.
Discuss in Groups

- **Problem.**
  - Analyze the running time of Huffman’s algorithm.
  - You will need to decide on some details, such as what data structure to use.

- Start with separate trees, each is a single leaf that corresponds to a letter.
- **Repeat** until there is one tree.
  - **Merge the two lightest trees** by adding a common root for them.
  - New weight = sum of both original weights.

Discuss in Groups

- **Problem.**
  - A text file consists of 16 different letters.
  - There exists a number $X$ such that the frequency of each letter is larger than $X/2$ and at most $X$.
  - What is the structure of the tree of the Huffman code in this case?
Not Another Bad Joke!#$%

Three logicians walk into a bar. The bartender asks “Do you all want a beer?” The first says “I don’t know”. The second says “I don’t know”. The third says “Yes!”