CMSC 441: Algorithms
Greedy Algorithms

Hillol Kargupta, Professor
http://www.cs.umbc.edu/~hillol/
hillol@gl.umbc.edu
Greedy Algorithms

- Greedy algorithms have the following property: Continuously finding the local optimum leads to the global optimum solution.
- In simple words, be greedy at every step!
- A greedy algorithm always makes the choice that looks best at the moment.
- Examples:
  - Gas station problem to minimize the number of gas stops
  - Activity selection problem
  - Huffman code for data compression
  - Fractional knapsack problem
  - Minimum spanning tree: Prim’s algorithm
Example: Minimize gas stops

- Problem statement:
  - Goal is to minimize the number of gas stops.
  - Input: start & end positions, exact locations of all gas stations along the way. Exactly m miles can be covered with one full tank of gas, irrespective of speed. Assume that initially gas tank is full.
  - Output: a list of gas stops.

- Greedy idea: Go as far as you can go before stopping for gas!

<table>
<thead>
<tr>
<th>Gas station locations</th>
</tr>
</thead>
<tbody>
<tr>
<td>s ____________________</td>
</tr>
</tbody>
</table>
Algorithm find_gas_stops():
current position = start position;
while (current_position < end_position)
    compute the position at which car will run out of gas.
    if (that position < end position) then
        find closest gas station before reaching that position.
        output fill up gas at that gas station.
        current position = that gas station location
    else
        current position = end position; /* reached */
Proof

• Prove that the algorithm finds an optimal solution, i.e. another solution with less number of gas stations does NOT exist.
• Induction or contradiction can be used. Contradiction will work well.
• Proof:
  • Assume that our algorithm does not find an optimal solution, i.e. there exists another better solution.
  • Our solution is \( s_1 = <g_{s1}, g_{s2}, g_{s3}, \ldots g_{sk}> \), that better solution is \( s_2 = <g_{s1}', g_{s2}', g_{s3}', \ldots g_{sk-1}'> \).
  • Continues in the next slide…
Proof …

• Compare the gas stops, starting with the first gas stop:
  - There are three possibilities:
    - $gs_i = gs_i'$ : Continue to compare next gas stops.
    - $gs_i > gs_i'$ : This is strange, $s_1$ is taking lead. Without affecting the outcome, we can make $gs_i' = gs_i$ and collapse this one to previous case.
    - $gs_i < gs_i'$ : As per the algorithm, every time we are selecting last possible gas station before running out of gas. $gs_i'$ is beyond $gs_i$ (note that previous gas stop is same for both solutions) $\Rightarrow$ car will run out of gas using $s_2$ $\Rightarrow$ $s_2$ is invalid.
  - If $gs_i = gs_i'$ for all $k-1$ gas stops, $s_1$ makes one more stop $gs_k$ to avoid running out of gas before reaching $t$. $s_2$ does not make that stop $\Rightarrow$ car will run out of gas $\Rightarrow$ $s_2$ is invalid.
• So, we have proven that, for all cases, $s_2$ is invalid $\Rightarrow$ such a better solution does not exist.
• So, $s_1$ is an optimal solution $\Rightarrow$ our algorithm is an optimal algorithm.
Details

- Input: start position \( s \), end position \( t \), gas station location array \( g[n] \), \( m \) miles/full tank.
- Let us say the current position is \( x \) with full tank of gas, then the car will run out of gas at position \( x+m \).
- To find the closest gas station: Do binary search in \( g[] \) for position \( x+m \), assuming that \( g[] \) is in sorted order.
- There are totally \( n \) gas stations. What is the running time of algorithm?
- Well, for finding each gas stop, time complexity is \( O(\lg n) \).
- Let us say that the total number of gas stops is \( k \). Then, time complexity is \( O(k \lg n) \).
- Worst case: \( O(n \lg n) \) [gas stations are sparse].
- Best case: \( O( ((t-s)/m) \ast \lg n) \) [lots of gas stations].
Activity selection problem

- Given a set of n proposed activities that wish to use the resource, goal is to select a maximum-size set of mutually compatible activities.
- Each activity has start time and finish time. Two activities are compatible if they do not overlap.
- Greedy idea: The sooner an activity is finished, we can schedule another activity.
- Algorithm:
  - Initialize the output set to nil.
  - Consider each activity in the order of increasing finish time:
    - If the activity starts after the finish time of last activity in output set, then
      - Include it in the output set.
- For more details, pages 330 to 332 including figure 17.1
Huffman codes

- Widely used and effective technique for compressing data
- Savings of 20% to 90% are typical depending on file characteristics.
- Binary character code to represent each character:
  - Fixed length code: each char is assigned same fixed length codeword; a=000, b=001, c=010, d=011, e=100, f=101.
  - Variable length code: much better than fixed length code, by giving frequent chars short codeword and infrequent chars long codeword; a=0, b=101, c=100, d=111, e=1101, f=1100
  - Prefix code are codes in which no codeword is also a prefix of some other codeword.
  - An optimal code for a file is always represented by a full binary tree, in which every nonleaf node has two children.
Huffman codes …

- Figure 17.4: Fixed-length code and optimal prefix code for
  chars: frequencies = a:45, b:13, c:12, d:16, e:9, f:5
- Total # of bits of encoded file = \( \text{freq}_1 \times \text{length}(\text{code}_1) + \text{freq}_2 \times \text{length}(\text{code}_2) + \ldots + \text{freq}_k \times \text{length}(\text{code}_k) \)
- Huffman invented a greedy algorithm that constructs an optimal prefix code called Huffman code.
- Idea is to start with \(|C|\) leaves and perform a sequence of \(|C|-1\) “merging” operations to create the final tree.
- Greedy property: Smaller the frequency, make the code longer to improve the compression.
- Priority queues can be used to find the two least-frequent objects to merge together.
Properties of Huffman’s Algorithm

- Complexity?
- Correct and
- Optimal
Results

- Let $J$ be the set of all internal nodes.

- Prove that the total cost of a tree for a code $\text{code} = \sum_{i \in J} f(i)$
Fractional Knapsack Problem

- A thief robbing a store finds \( n \) items; the \( i \)th item is worth \( v_i \) dollars and weights \( w_i \) pounds. He wants to take as valuable a load as possible, but he can carry at most \( W \) pounds in his knapsack.

- The thief can take fractions of items in this case. (There is another problem called 0-1 knapsack problem in which each item is either taken or left behind. No fractions allowed).

- Greedy property: Take the item with greatest value first, i.e. item with max. \( \frac{v_i}{w_i} \) value.

- Algorithm: Consider all items in the order of decreasing value. Keep including each item until the weight limit \( W \) is reached. Note that the last item may have to fractionally included.

- Proof by contradiction?
Prism’s algorithm

- Problem: Given a weighted undirected graph $G = (V,E)$, the goal is to find the minimum spanning tree, i.e. a tree that connects all nodes with minimum total cost.

- Algorithm:
  - Start with any node as the covered node.
  - At any time, there are two sets of nodes: covered nodes and uncovered nodes.
  - In each step, find the cheapest edge among all the edges that connect any covered node to any uncovered node.
  - Stop when all nodes are covered.

---

Covered nodes

```
       e1
      /   \
 e2     e3
      /     /
     .     .
     /     /
    ck     .
```

Uncovered nodes