CSCE 411: Design and Analysis of Algorithms

Week 3: Greedy Algorithms

Date: January 27, 2026 *Nate Veldt, updated by Samson Zhou*

Course Logistics

- Greedy algorithms: Chapter 16
- Homework 2 is due this Friday

1 Making Change

Say a cashier has to give someone 68 cents in change to a customer for a purchase. What is the smallest number of coins the cashier can use to give this change, assuming they can give quarters (25 cents), dimes (10 cents), nickels (5 cents) and pennies (1 cent)?

Strategy for giving change:

- Choose coins one at a time
- At each step, choose the coin ____________________________

1.1 The Change-Making Problem

Assume we have n coins, and let

$$v_i = \underline{\hspace{8cm}}$$

$$m_i = \underline{\hspace{8cm}}$$

The Change-Making Problem Given an integer value $C > 0$ and coin values

$$v_1 > v_2 > v_3 > \cdots > v_n = 1,$$

find the number of coins m_i for $i = 1, 2, \ldots n$ so that

-

-

1.2 The greedy algorithm for change-making

```
GreedyCoinChange($C$, $v = [v_1, v_2, \ldots, v_n = 1]$)
  $n$ = length($v$)
  Let $m = [0..0]$ be an empty array of $n$ zeros
  for $i = 1$ to $n$ do
      while $C \geq v_i$ do

          $m[i] =$ ____________

          $C =$ __________
      end while
  end for
  Return $m$
```

Correctness: The greedy algorithm correctly gives change

Proof. At each step, we keep updated the amount left that we have to pay. We only add another coin of type i if v_i is less than this amount, so we never go over. We will always reach the exact amount, because C is an integer value and we can always add coins of values $v_n = 1$. $\square$

Is the greedy algorithm optimal?

Question 1. Which of the following do you think is true?

A The greedy algorithm works for some values of C when we use coin values (1,5,10,25) (like $C = 68$), but is sometimes not optimal when we use these coin values (1,5,10,25)

B The greedy algorithm is always optimal for giving change for coin values (1,5,10,25), but for other coin types $v = (v_1, v_2, \ldots, v_n)$ it may not be optimal

C The greedy algorithm is always optimal for the coin change problem, regardless of coin values, as long as $v_n = 1$.

D Any computational problem can be solved optimally by a greedy algorithm

2 Greedy Algorithm Basics

A greedy algorithm is an algorithm that

-

-

Elements of greedy algorithms. There are two key elements we look for when applying a greedy algorithm:

1. **Optimal substructure**:
2. **Greedy-choice property**: a globally optimal solution can be achieved by

   ____

Overall strategy for using greedy algorithms

1. Determine a notion of the ________________________________ that improves the global optimization problem.
2. Show how making one local choice leads to __________________________________
3. Prove that the greedy choice is __________: it leaves us with a subproblem which, when solves optimally and combined with the first greedy choice,

   ____

3 The Activity-Selection Problem

The activity-selection problem Let $(a_1, a_2, a_3, \ldots a_n)$ represent a set of activities, where:

- $s_i =$

- $f_i =$

where $s_i < f_i$ for $i = 1, 2, \cdots n$. We assume a half-open interval $[s_i, f_i)$ for the activity a_i. Assume these activities are ordered by finish times to that so that

---

The activity-selection problem

Example:

i	1	2	3	4	5
s_i	1	3	2	4	7
f_i	4	5	6	8	9

3.1 Optimal Substructure

Consider an activity-selection problem defined by $(a_1, a_2, \ldots, a_n)$ and the time interval $[s_1, f_n)$. If we know that an optimal solution contains activity a_k. Then:

A dynamic programming approach: Try each a_k, solving two smaller subproblems in each case, and taking the best answer.

3.2 A greedy choice lemma

Lemma 3.1. *There exists an optimal solution to the activity-selection problem defined by $(a_1, a_2, \ldots, a_n)$ that contains a_1 (the activity with the earliest finish time).*

Proof.

- Let S be an optimal solution, and let a_i be its activity with the earliest start and end time.
- a_1 has a finish time $f_1 \leq f_i$.
- This a_i overlaps with a_1, otherwise we could add a_1 to make S better
- If we replace a_i with a_1 to get S', then $|S| = |S'|$, and this is a valid non-overlapping solution, because a_i was the earliest activity in S.

3.3 An Optimal Greedy Algorithm for the Activity-Subset Problem

```
GreedyActivitySelection($s = (s_1, s_2, \ldots, s_n)$, $f = (f_1, f_2, \ldots, f_n)$)
  $n = \text{length}(s)$
  $S = \{a_1\}$
  $k = 1$
  for $i = 2$ to $n$ do
      if $s_i \geq f_k$ then
          $S = S \cup \{a_i\}$
          $k = i$
      end if
  end for
  Return $S$
```

4 Binary Codes

- An ________________ is a set of characters, e.g., $C = \{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}, \texttt{e}, \texttt{f}\}$

- A *binary code* for C is

Consider three possible codes for $C = \{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}, \texttt{e}, \texttt{f}\}$.

Character:	a	b	c	d	e	f
example code 1:	000	001	010	011	100	101
example code 2:	0	101	100	111	1101	1100
example code 3:	0	1	10	01	11	00

(1)

Types of codes:

- **Fixed length code**:

- **Variable-length code**:

- **Prefix code**: the codeword for every $c \in C$ is

Question 2. Which of the codes in Table 1 is a prefix code?

A Example code 1

B Example code 2

C Example code 3

D Example codes 1 and 2

E Example codes 1, 2, and 3

4.1 Encoding and Decoding

Consider a string of characters from the alphabet $C = \{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}, \texttt{e}, \texttt{f}\}$ where the frequency of each character is given in the following table and we have two codes.

Character:	a	b	c	d	e	f
Frequency	0.45	0.13	0.12	0.16	0.09	0.05
FLC:	000	001	010	011	100	101
VLC:	0	101	100	111	1101	1100

(2)

4.2 The Optimal Prefix Code Problem

The optimal prefix code problem Given an alphabet C and a frequency $c.\textit{freq}$ for each $c \in C$ in a given string s,

Representing a prefix code as a tree Every prefix code for a string s can be represented as a binary tree T where

- Left branches are associated with
- Right branches are associated with
- Each leaf corresponds to a character $c \in C$, whose binary code is given by
- Each node is associated with the frequency of characters in its subtree

Example

Character:	`a`	`b`	`c`	`d`	`e`	`f`
Frequency	0.45	0.13	0.12	0.16	0.09	0.05
VLC:	0	101	100	111	1101	1100

The cost of the tree is give by

$$B(T) = \sum_{c \in C}$$

where $d_T(c)$ is the depth of c in the tree T.

Question 3. True or false: every binary tree (with ℓ leaves) gives a valid prefix code (for an alphabet with ℓ characters).

A True

B False

Theorem 4.1.

Proof.

Optimal prefix code problem:

Nice properties about the optimal tree In the optimal tree representation,

-

-

4.3 Huffman Codes

Huffman Code: a prefix code obtained by greedily building a tree representation for C

Huffman Code Tree Process

- Associate each character in C with a node, labeled by its frequency
- Identify two nodes x and y in C with the ______________________________
- Create new node z with frequency ____
- Make $z.left = x$, and $z.right = y$.
- Repeat the above procedure on the alphabet obtained by _________________________

Observe: x and y will be siblings in T at _____________________________

Activity Create a Huffman code for:

Character:	`a`	`b`	`c`	`d`	`e`	`f`
Frequency	0.45	0.13	0.12	0.16	0.09	0.05

```
HuffmanCode($C$)
  $n = |C|$
  $T \leftarrow$ empty graph
  $Q \leftarrow \emptyset$
  for $c$ in $C$ do
      Insert $c$ with value $c.freq$ into $T$ and $Q$
  end for
  for $i = 1$ to $n - 1$ do
      $x =$
      $y =$
      create node $z$







  end for
  Return $T$
```

4.4 Code and Illustration

4.5 Runtime Analysis

Given a set of objects C, with values $c.val$ for $c \in C$ and $n = |C|$, a binary min-heap for C is a binary tree such that

- All levels
- The value of a node is

It has the following operations:

- BuildBinMinHeap(C):
- ExtractMin(Q):
- Insert($Q, z, z.freq$):

Question 4. Assume we use a binary min heap to store and update Q in the pseudocode above. Then what is the overall runtime of creating a Huffman code, in terms of $n = |C|$?

A $O(\log n)$

B $O(n)$

C $O(n \log n)$

D $O(n^2)$

4.6 Optimality

It turns out that a Huffman code optimally solves the prefix code problem. The argument hinges on the following lemma.

Lemma 4.2. *Let C be an alphabet where $c.freq$ is the frequency of $c \in C$. Let x and y be the two characters in C that have the lowest frequencies.*

Then, there exists an optimal prefix code for C in which x and y have the same length code words and only differ in the last bit.

Equivalently:

Why does this imply optimality? Consider a slightly different (but equivalent) approach to defining a Huffman code:

1. Associate each character with a node, labeled by its frequency
2. Identify the two nodes x and y with the smallest frequencies
3. Create new node z with frequency $z.freq = x.freq + y.freq$.
4.
5. Create tree T from T' by making $z.left = x$, and $z.right = y$.

Claim This produces an optimal tree T.

Proof. Note that the cost of the tree T is

$$B(T) = B(T') + x.freq + y.freq \tag{3}$$

We prove the result by contradiction: suppose that T is not optimal. Then

We can assume that x and y are at maximum depth in R.

Let R' be the tree obtained by taking R and

Similar to (3) we have that:

We then get an impossible sequence of inequalities:

$$\begin{aligned} B(R') &= B(R) - x.freq - y.freq \\ &< B(T) - x.freq - y.freq \\ &= B(T') \end{aligned}$$