CS583 Lecture 05

Greedy Algorithms

Jyh-Ming Lien

some materials here are based on Prof. Shehu, and Prof. Wang's past lecture notes

Intro

- Greedy algorithm is algorithm that makes the locally optimal choice at each stage with the hope of finding the global optimum
- Greedy algorithm never changes the choices that have been made

Intro

- **Advantages**
  - Simple and Intuitive
  - Work for problems such as minimum spanning tree, shortest path problem, and data compression.
- **Disadvantages**
  - Be very careful when use it. May not work for many problems
  - But still provide good approximate solution

Outline

- Problems we are going to look at today
  - The Activity Selection Problem
  - Huffman coding
  - Knapsack problem(s)

Activity Selection

- Optimization problem
  - select a max-size subset of compatible activities

Activity	i	1	2	3	4	5	6	7	8	9	10	11
Start time	s_i	1	3	0	5	3	5	6	8	8	2	12
Finish time	f_i	4	5	6	7	8	9	10	11	12	13	14

  - possible subsets?
  - brute force approach?

Activity Selection

- This problem can be solved using dynamic programming!
  - sub-problem:
  - a recursive definition:

Activity Selection

- Converting it to a greedy algorithm

Activity Selection

Algorithm $\textsc{GreedyActivity}(s, f)$

$n \leftarrow |S|$

$A \leftarrow \{a_1\}$

$i \leftarrow 1$

for $m \leftarrow 2$ **to** n **do**

    if $s_m \geq f_i$ **then**

        $A \leftarrow A \cup \{a_m\}$

        $i \leftarrow m$

    endif

endfor

return A

Huffman Coding

- binary cipher

Letter	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed length encoding	000	001	010	011	100	101

- A message consisting of 100K a-f characters would require:

Huffman Coding

- fixed length vs. variable length coding

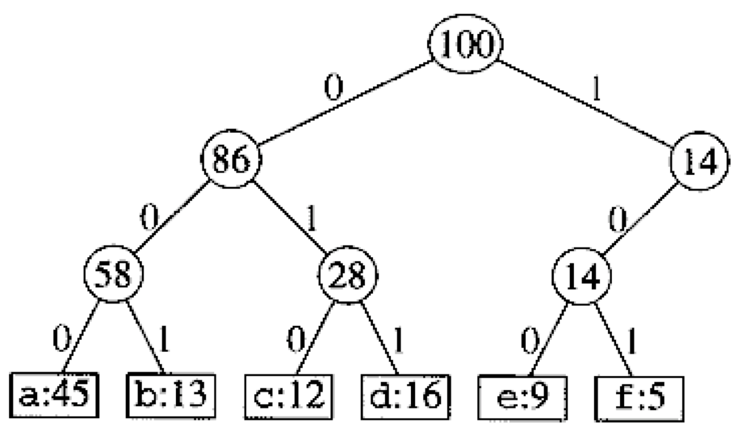
Letter	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length encoding	000	001	010	011	100	101
Variable-length encoding	0	101	100	111	1101	1100

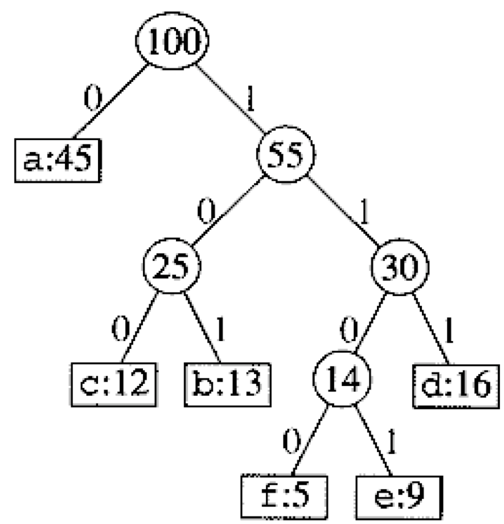
- 001011101 uniquely converts to:
  - this requires how many bits with fixed length coding?

Huffman Coding

- fixed vs. variable length coding

Letter	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length encoding	000	001	010	011	100	101
Variable-length encoding	0	101	100	111	1101	1100





Huffman Coding

- problem: minimize this:

$$B(C) = \sum_{i=1}^{n} f(a_i) \cdot L(c(a_i))$$

- Huffman developed a greedy algorithm for producing a minimum-cost prefix code. The code that is produced is called a Huffman Code

Huffman Coding

- Basic idea
  - **greedy**: low frequency letters should be at the bottom of the tree
  - build the encoding tree from bottom up

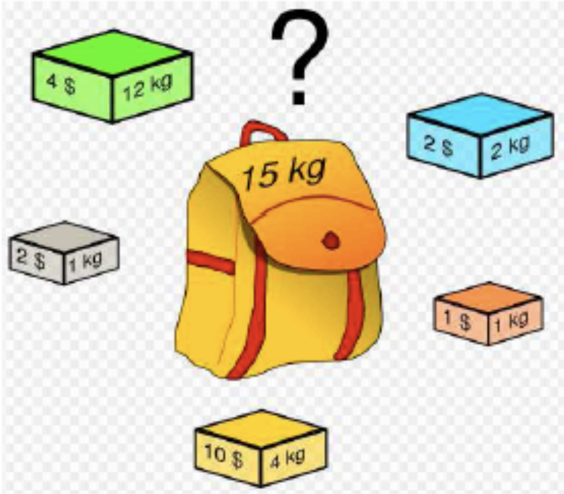
Huffman Coding

- greedy algorithm

Algorithm HUFFMAN(C)
$n \leftarrow |C|$
$Q \leftarrow C$
for $i \leftarrow 1$ **to** $n-1$ **do**
  {allocate a new node z}
  $left[z] \leftarrow x \leftarrow$ EXTRACT-MIN(Q)
  $right[z] \leftarrow y \leftarrow$ EXTRACT-MIN(Q)
  $f[z] \leftarrow f[x] + f[y]$
  INSERT(Q, z)
endfor
return EXTRACT-MIN(Q)

Knapsack

- Knapsack Problem: Given n objects, each object has weight w and value v, and a knapsack of capacity W, find most valuable items that fit into the knapsack



- Brute force approach
  - generate a list of all potential solutions
  - evaluate potential solutions one by one
  - when search ends, announce the solution(s) found
- What is the time complexity of the brute force algorithm?

Knapsack

- Dynamic programming approach
  - Assume that we want to compute the optimal solution $S(w, i)$ for capacity $w < W$ with i items
  - Assume that we know the optimal solutions $S(w', i')$ for all $w' \leq w$ and $i' \leq i$
  - Option 1: **Don't add** the k-th item to the bag, then $S(w, i) = S(w, i-1)$
  - Option 2: **Add** the k-the item to the bag, then $S(w, i) = S(w - w_i, i-1) + v_i$

w	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
12kg, \$4															
1kg, \$2															
2kg, \$2															
1kg, \$1															
4kg, \$10															

- Time complexity?

Knapsack

- greedy algorithm #1
  - Put object with smallest weight in knapsack first
  - Add objects (according to sorted order of weights) into knapsack as long as there is capacity
- result:
- time complexity:

Knapack

- greedy algorithm #2
  - focusing on maximizing profit while minimizing weight
  - add items with max $\frac{v_i}{w_i}$ first, where

    v_i and w_i are the value and weight of item i
- result:
- time complexity:

Fractional Knapack

- You can take fractions of an object
- **Problem**: Fit objects (taking even fractions of them) that give the maximum total profit
- The optimal solution of this problem can be obtained using a greedy algorithm
  - why?