COEN279 - Design and Analysis of Algorithms

Greedy Algorithm

Dr. Tunghwa Wang

Fall 2016

Announcement

- Grader: Xiaoyu Li
  - Email: natalienew2016@gmail.com
  - Future submissions to her: CC to me
    - Filename should contain your name and type of homework: for example, TunghwaWang-Bonus2
    - You may resubmit, same file names: new file replaces the old one.
- Programming assignment #1:
  - Submission:
    - When you run "perl Submit P1", you should see the message of successful submission to natalienew2016@gmail.com and twang1@cse.scu.edu.
    - How did you send the submission to grader@engr.scu.edu?
  - Future assignments:
    - P2: will be sent to grader and me
    - P3: will not be sent to grader but only to me

Announcement

- Mid-term Exam:
  - Total duration is 100 minutes.
  - You may want to go to bathroom before starting the exam.
  - Read the honor code, then sign and date below it.
  - Read all questions first in order to plan the order of questions:
    - Question number has nothing to do with the difficulty of questions.
    - You should work first on problems which you have higher confidence.

Announcement

- Mid-term Exam:
  - You may bring in summary notes and text book.
  - You may not use cellular phone, laptop computer, or any electronic devices.
  - There are 8 questions to cover:
    - Mathematical background
      - Mathematical induction
      - Probability and Statistics
    - Complexity analysis
    - Heap
    - Tree algorithm
    - Divide and conquer algorithm
    - Dynamic programming algorithm

Be Greedy

- When solving a problem, the greedy paradigm is usually utilized to get an optimal solution:
  - Characterize the structure of an optimal solution.
  - Develop a recursive solution.
  - Show that if we make a greedy choice, then only one sub-problem remains.
  - Prove that it is always safe to make the greedy choice.
  - Develop a recursive algorithm that implement the greedy strategy.
  - Convert the recursive algorithm into iterative implementation.

Greedy Algorithm

- Thus:
  - Cast the problems one in which we make a choice and are left with only one sub-problem.
  - Prove that there is always an optimal solution to the original problem that makes the greedy choice, so that the greedy choice is always safe.
  - Demonstrate optimal substructure by showing that, having make the greedy choice, what remains is a sub-problem with the property that if we combine the optimal solution to the sub-problem with the greedy choice we have made, we arrive at an optimal solution to the original problem.

Greedy Algorithm

- A greedy algorithm always makes the choice that looks the best at the step of sequence of choices:
  - Select the local maximal choice in the hope that this choice will lead to a globally optimal solution.
  - It does not always yield optimal solution.
  - However, to many problems, it does.
- Note:
  - For such problems, dynamic programming algorithm normally also works.
    - When we do greedy algorithm, only one sub-problem remains: greedy choice.
  - The differences:
    - Greedy algorithm typically is top-down (make the choice then solve sub-problem) rather than bottom-up (solve related sub-problems then make a choice).
    - Greedy algorithm has much better performance in both time and space complexity.
    - For lots of problems, while dynamic programming algorithm works, the greedy algorithm does not work.

Activity Selection Problem

- Definition:
  - S is a set of n activities $\{ a_1, a_2, \ldots, a_n \}$ utilizing a resource
  - For each activity a_i for all $1 \le i \le n$:
    - s_i is the time to start using the resource.
    - f_i is the time to finish using the resource.
    - Without loosing the generality, we assume that $f_i \le f_j$ if $1 \le i < j \le n$.
  - Solution:
    - Select maximum-size subset of activities to use the resource.
  - Note:
    - Activity a_i, with start time s_i and finish time f_i, is compatible with activity a_j, with start time s_j and finish time f_j, if the time intervals $[s_i,f_i)$ and $[s_j,f_j)$ do not overlap.

Activity Selection Problem

- Optimal construct:
  - Let S_{ij} be the optimal solution after a_i finishes and before a_j start using the resource and c[i,j] is the associated size of S_{ij}.
    - Then the optimal solution is S_{1n}.
  - For any a_k in S_{ij}, we have c[i,j] = c[i,k] + c[k,j] + 1.
  - Thus, c[i,j] =
    - If S_{ij} is empty: 0
    - Otherwise: maximum_k (c[i,k] + c[k,j] + 1)
  - Note:
    - If we do dynamic programming, we need to resolve all possible k's to get the optimal solution to the sub-problem S_{ij}.

Activity Selection Problem

- Greedy choice:
  - Is there a greedy choice?
    - First finish time: f_k
      - Works and seems natural and straight forward.
    - Last start time: s_k
      - Actually also works thought not straight forward.
    - First start time: s_k
      - What if the order is given in start time instead of finish time?
        - We assume that $s_i \le s_j$ if $1 \le i < j \le n$.
        - Intuitively, it is not optimal if we hit a very long running job early!
    - Minimum duration: $f_k - s_k$
      - What if the duration $\{ d_1, d_2, \ldots, d_n \}$ is provided instead of finish time $\{ f_1, f_2, \ldots, f_n \}$?
      - We need to consider both s_i and d_i because we cannot wait an activity started late.
    - Minimum overlaps: ?
      - Fifth: Difficult to describe input!
  - Thus let S_k be defined as the set all of a_i such that $s_i \ge f_k$.
    - If a_1 is in the optimal solution, then it can be reduced to sub-problem S_1.

Activity Selection Problem

- Greedy choice is safe:
  - If a_m has the earliest finish time in S_k, then it is in the maximum-size activity subset of mutually compatible activities.
    - If S_k has a maximum-size activity subset A_k with a_i having the earliest finish time:
      - If $a_m = a_i$, then it is done.
      - Otherwise, $A_k - a_i + a_m$ is also a maximum-size activity subset because $f_m \leq f_i$.
  - Thus we can use greedy algorithm instead of dynamic programming:
    - Greedy choice is the next activity with earliest finish time.
- Top-down recursion:
  - Call activity_select_recur(s, f, 0, N).
  - Time complexity: $\Theta(N^2)$
  - Space complexity: $\Theta(1)$

```
integer_set_t activity_select_recursive(
    time_t[N] s,
    time_t[N] f,
    integer k)
{ integer m = k + 1;

  while m ≤ n and s[m] < f[k]
    m = m + 1;
  if m ≤ n
    return activity_select_recursive(s, f, m, n) ∪ {m};
  else
    return empty_set;
}
```

Activity Selection Problem

- Top-down iteration:
  - Call activity_select(s, f).
  - Time complexity: Θ(N)
  - Space complexity: Θ(1)

```
integer_set_t activity_select_iteration(
    time_t[N] s,
    time_t[N] f)
{ integer_set_t A = { 1 };
  integer k = 1, m;

  for m = 2 to n
    if s[m] s[k] ≥ f[k]
    {
      A = A ∪ { m };
      k = m;
    }
  return A;
}
```

Huffman Codes

- Definition:
  - Given a file of characters from a character set S = { a_1, a_2, …, a_n } together with the frequencies { f_1, f_2, …, f_n }, compress the file to optimal size:
    - Fixed-length bit-code
    - Variable-length bit-code
- Issue:
  - Fixed-length bit-code is easier but gives low compression ratio.
  - Thus we work on variable-length bit-code hoping to achieve higher compression ratio.
    - Assign shortest bit-code to character with highest frequency.
    - Need to design scheme to avoid ambiguity.

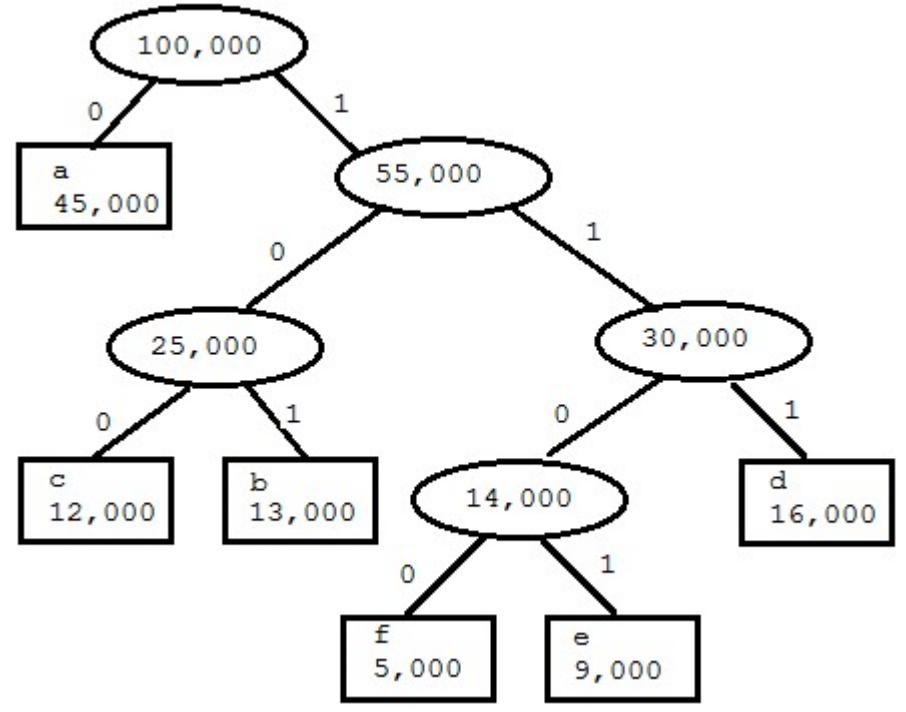
Huffman Codes

- Prefix codes:
  - An optimal code using full-tree:
    - Assign bit 0 to left branch
    - Assign bit 1 to right branch
  - Consideration:
    - No ambiguity
    - Simplify encoding/decoding

Huffman Codes

- Example:
  - Original file:
    - 100,000 characters
    - 800,000 bits
  - Fixed-length:
    - 3-bit code word
    - 300,000 bits
  - Variable-length:
    - 224,000 bits

	a	b	c	d	e	f
frequency	45000	13000	12000	16000	9000	5000
fixed-length codeword	000	001	010	011	100	101
variable-length	0	101	100	111	1101	1100



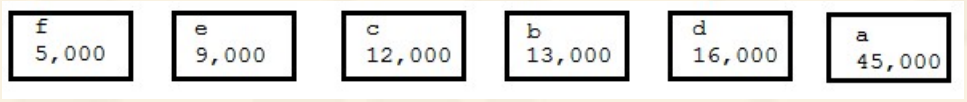
Huffman Codes

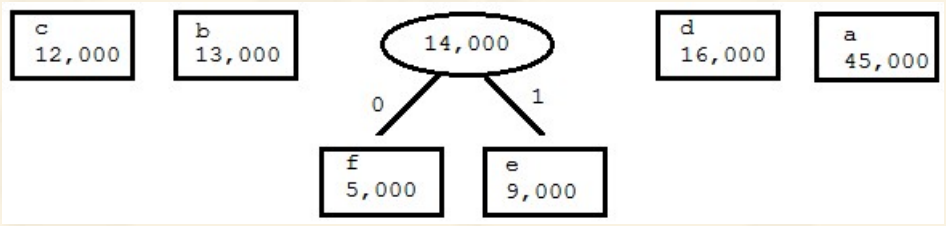
- Pseudo Code:
  - Q is initialized to heap of tree nodes:
    - Each tree has only one node of c[k]
  - Time complexity: Θ(NlgN)
    - Loop of N-1 iterations
    - Heap operation: Θ(lgN)
  - Space complexity: Θ(1)

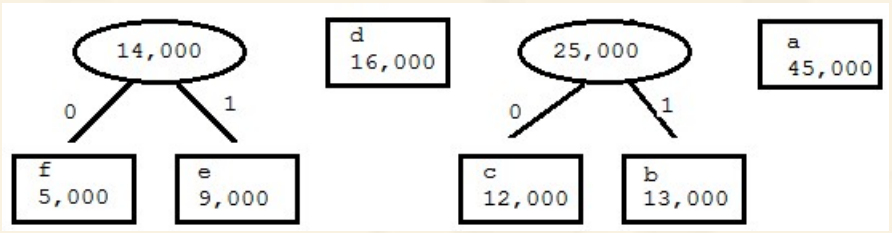
```
tree_node_t huffman(
    char_freq_t[N] c) // in ascending frequency
{ tree_node_t[N] q = c; // thus a min heap of trees
  integer k;

  for k = 1 to N-1
  { heap_node_t z;

    z.left = x = heap_extract_min(q);
    z.right = y = heap_extract_min(q);
    z.freq = x.freq + y.freq;
    heap_insert(q, z);
  }
  return heap_extract_min(q);
}
```

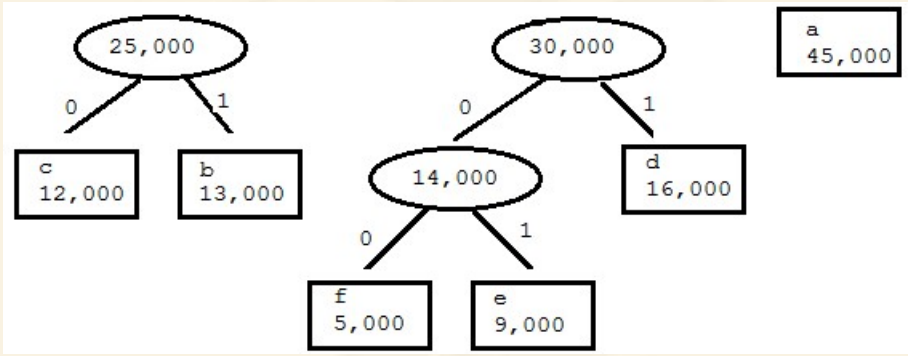
Huffman Codes

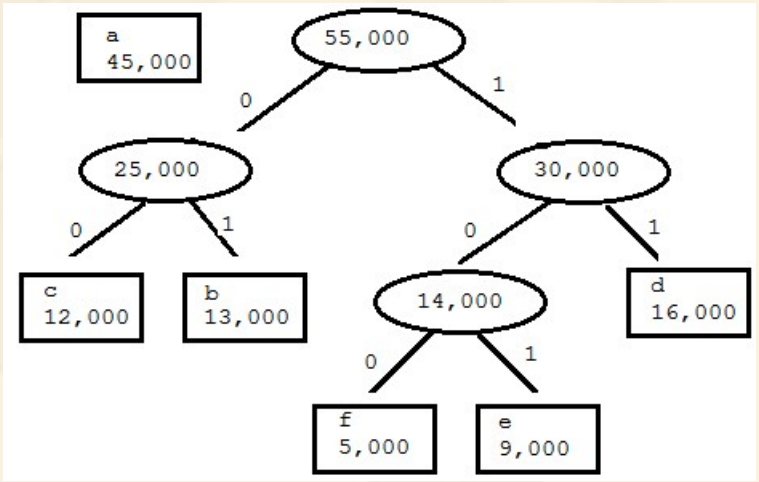
- Example:
  - Build the solution:

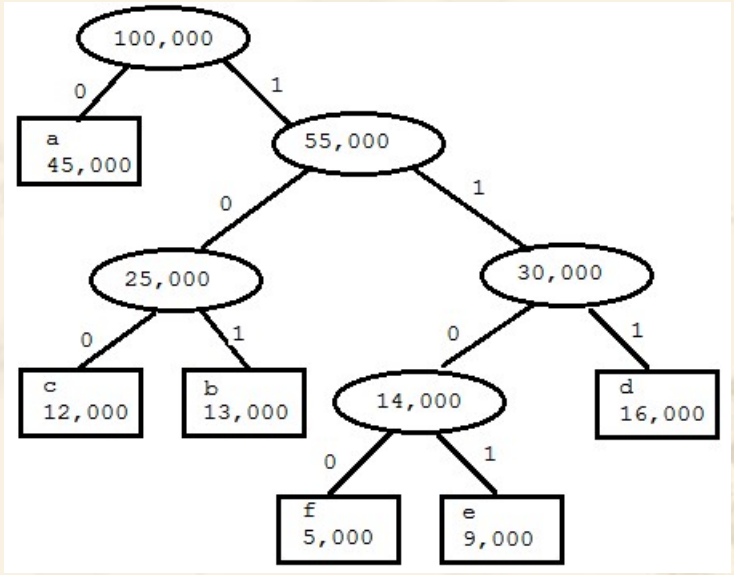












Huffman Codes

- Greedy algorithm?
  - Optimal construct:
    - Cost of tree T: B(T) = sum_c { c.freq*bit_length(codeword(d)) } for all c in C
    - Optimal solution should have minimum B(T).
  - Greedy choice:
    - Merging of 2 trees with minimum frequencies is the greedy choice.
  - Greedy choice is safe:
    - Let x and y be two characters with minimum frequencies. Then there exists an optimal prefix code for C in which x and y have the same length and differ only in the last bit.
    - Let z be the merged tree from x and y so that z.freq = x.freq + y.freq and C' = C - { x, y } + { z }. Let T' be the tree representing the optimal prefix code of C', then T, be the tree from T' by replacing z from leaf to non-leaf node with x and y as children, represents the optimal prefix code of C.

Matroids

- A theory describes cases which can be solved using greedy algorithm:
  - It is not true the other way:
    - Not all cases for greedy algorithm are matroids.
    - Neither one of activity selection and Huffman coding presented is.
  - Definition: A matroid is an ordered pair $M = \{ S, \mathscr{I} \}$ satisfying the conditions:
    - S is a finite set.
    - $\mathscr{I}$ is a non-empty set of subsets of S such that if $B \in \mathscr{I}$ and $A \subseteq B$, then $A \in \mathscr{I}$:
      - We call $\mathscr{I}$ an independent subsets of S with some property of being "independent".
      - $\varnothing \in \mathscr{I}$
      - We say that $\mathscr{I}$ is hereditary.
    - If $A \in \mathscr{I}$, $B \in \mathscr{I}$, and $|A| < |B|$, then there exists some element $x \in B\text{-}A$ such that $A \cup \{x\} \in \mathscr{I}$.
      - We say that M satisfies exchange property.

Matroids

- More:
  - If M = { S, $\mathscr{I}$ } is a matroid and x $\notin$ A, then we call x an extension of A if A U {x} $\in \mathscr{I}$.
  - If A is an independent subset in matroid M = { S, $\mathscr{I}$ }, then A is maximal if it has no extensions.
    - All maximal independent subsets in a matroid has the same size.
  - A matroid M = { S, $\mathscr{I}$ } is weighted if it is associated with a weight function w:
    - For every x $\in$ S, w(x) is defined and w(x) > 0.
    - For any A $\subseteq$ S, w(A) = $\text{sum}_{x \in S} w(x)$.

Matroids

- Example:
  - Matric matroid:
    - For a matrix M, let S = { r: r is row vector of M }, then an independent set $A \subseteq S$ is defined as being linearly independent.
      - Obviously, any $B \subseteq A$ still contains only linearly independent vectors of M.
      - If $A \in \mathscr{I}$, $B \in \mathscr{I}$, and $|A| < |B|$, then there exists some element $x \in B-A$ such that $A \cup \{x\} \in \mathscr{I}$ because at least one row vector can be linearly independent to all row vectors in A.
  - Graph matroid:
    - For a graph G = (V, E), let S = E, then an independent set $A \subseteq S$ is defined as being acyclic.
      - That is, G' = (V, A) forms a forest.
      - Obviously, any $B \subseteq A$ is still cyclic.
      - If $A \in \mathscr{I}$, $B \in \mathscr{I}$, and $|A| < |B|$, then there exists some element $x \in B-A$ such that $A \cup \{x\} \in \mathscr{I}$ because at least one edge does not form cycle to all edges in A.

Backup Slides