# CSC 317: Data Structures and Algorithm Analysis

Dilip Sarkar

Department of Computer Science
University of Miami

## Outline I

## Activity-Selection Problem: Definition

- $S = \{a_1, a_2, \cdots, a_n\}$ be set of of $n$ proposed **activities**.

  Each $a_i$ has a start time $s_i$ and a finish time $f_i$.

  They use a common resource.
- Example: Let $S$ be a set of $n$ classes, and

  the common resource is a class room.

  We want to schedule maximum number of classes in the class room.
- The **problem is finding a maximum-size subset** $A$ such that these **activities** in $A$ **are compatible**
- What **compatible** mean?

  No two classes require the room during the same time.

  Before a class $a_i$ starts, the room is free and

  The next class $a_j$ does not start until the $a_i$ is not finished.

## A Example of Activity-Selection Problem

- $S = \{a_1 =< 8:00 \text{ am}, 9:15 \text{ am} >, a_2 =< 9:00 \text{ am}, 9:50 \text{ am} >,$

  $a_3 =< 9:30 \text{ am}, 10:25 \text{ am} >, a_4 =< 10:00 \text{ am}, 10:50 \text{ am} >\}$
- We have three solutions:
  - $A_k = \{a_1, a_3\} = \{< 8:00 \text{ am}, 9:15 \text{ am} >, < 9:30 \text{ am}, 10:25 \text{ am} >\}$
    $A'_k = \{a_1, a_4\} = \{< 8:00 \text{ am}, 9:15 \text{ am} >, < 10:00 \text{ am}, 10:50 \text{ am} >\}$
    $A^*_k = \{a_2, a_4\} = \{< 9:00 \text{ am}, 9:50 \text{ am} >, < 10:00 \text{ am}, 10:50 \text{ am} >\}$
- Since we do not know the value of $k$, in a brute-force method we must consider all possible values of $k$.

  **Notation:** Let $c[i, j]$ be the maximum solution for the subset $S_{i,j}$.

$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \emptyset; \\ \max_{a_k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & \text{if } S_{ij} \neq \emptyset. \end{cases}$$

- This is too expensive. Let us consider a greedy method.

## A greedy Algorithm for Activity-Selection Problem

- Sort the activities in monotonically increasing order of finish time
  $i$ is the index of the activity $a_i$
  $s_i$ is the start time of the activity $a_i$, and
  $f_i$ is the finish time of the activity $a_i$.

**Example**

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $s_i$ | 1 | 3 | 0 | 5 | 3 | 5 | 6 | 8 | 8 | 2 | 12 |
| $f_i$ | 4 | 5 | 6 | 7 | 9 | 9 | 10 | 11 | 12 | 14 | 16 |

- A solution: $A_k = \{a_1, a_4, a_8, a_{11}\}$
  How we get this solution:

Step 1: Choose the activity that finishes first
Since the list is sorted, we simply select the first activity on the sorted list
Step 2: Exclude all activities that start before the selected activity finishes
(that is, exclude $a_2$ and $a_3$ in our case)

- Repeat steps 1 and 2 with the remaining set.

## Prove Optimality of the Solutions I

- **Notation:**
- Let $S_k = \{a_i \in S : s_i \geq f_k\}$.
  $S_k$ is the set of activities that starts after $a_k$ finishes.
  **Examples:** $S_1 = \{a_4, a_5, \cdots, a_{11}\}$, $S_5 = S_6 = \{a_{11}\}$
- The greedy choice works if $a_1$ is selected, then
- an optimal solution includes $a_1$, and an optimal solution to $S_1$

**Theorem**

*Consider any nonempty subproblem $S_k$, and let $a_m \in S_K$ with the earliest finish time. Then $a_m$ is included in some maximum-size subset of mutually compatible activities of $S_k$.*

## Prove Optimality of the Solutions II

**Theorem**

*Consider any nonempty subproblem $S_k$, and let $a_m \in S_K$ with the earliest finish time. Then $a_m$ is included in some maximum-size subset of mutually compatible activities of $S_k$.*

**Proof.**

- Let $A_k$ be a maximum-size subset of mutually compatible activities in $S_k$ and let $a_j$ be the activity in $A_k$ with earliest finish time.
- If $a_j = a_m$, we are done.
- If $a_j \neq a_m$, then

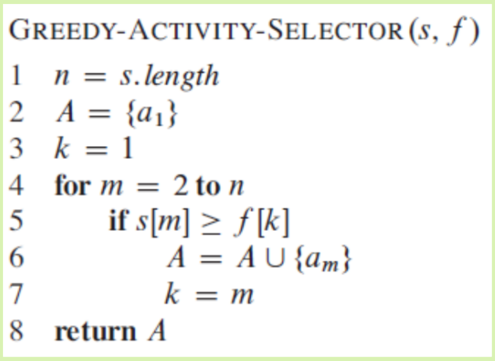
  we have to show that we can replace $a_j$ with $a_m$ and have mutually compatible maximum-size solution.
- It is easy to show this, because finish time of $a_m$ is earlier than that of the $a_j$.
- Thus, replacing $a_j$ with $a_m$ keep all activities mutually compatible.

$\square$

## Intuition Behind the Proof And the Pseudocode

**Example**

- Let $S_k = \{a_1 =< 5, 11 >, a_2 =< 6, 13 >, a_3 =< 13, 19 >, a_4 =< 18, 23 >\}$
- and $A_k = \{a_2, a_3\}$.
- In the solution set $A_k$, we don't have $a_1$.
- But we can replace $a_2$ in the solution set with $a_1$ and get another solution set $A'_k = \{a_1, a_3\}$.
- This new solution is a maximum-size solution and includes the earliest finish time activity $a_1$.



```
GREEDY-ACTIVITY-SELECTOR(s, f)
n = s.length
A = {a_1}
k = 1
for m = 2 to n
    if s[m] ≥ f[k]
        A = A ∪ {a_m}
        k = m
return A
```

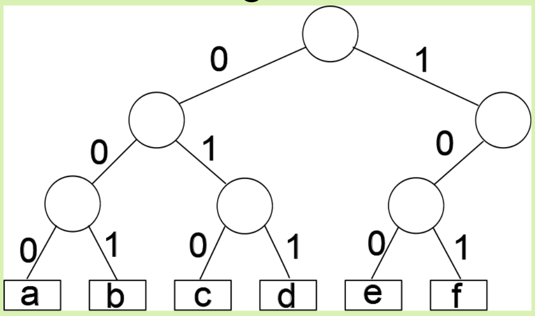
## Elements of Greedy Strategy

A greedy algorithm obtains an optimal solution to a problem by making a sequence **greedy**choices.

1. Cast the optimization problem as one in which we make a choice and are left with one subproblem to solve.
2. 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.
3. Demonstrate optimal substructure by showing that, having made the greedy choice, what remains is a subproblem with the property that if we combine an optimal solution to the subproblem with the greedy choice we have made, we arrive at an optimal solution to the original problem.

## Equal-Length Binary Codes for a Set of Symbols

Let us consider a set of 6 symbols: $S = \{a, b, c, d, e, f\}$
How many bits we need to encode a symbol? It depends on coding method.
Let us find # of bits ,$l$, for fixed length coding.
$\lfloor \log_2 6 \rfloor \leq l \leq \lceil \log_2 6 \rceil \Leftrightarrow 2 \leq l \leq 3$
$\Rightarrow$ We need three bits for a fixed length code.



- Finding a code: start from the root and walk to the symbol
  a = 000, b = 001, c = 010, d = 011, e = 100, and f = 101
- Notice the labels on the links:
  left link is marked with ‘0’ and right link is marked with ‘1’
  If we encode a string of any length, the number of bits we will need is **3** per symbol.
- **Can we design a better code?**

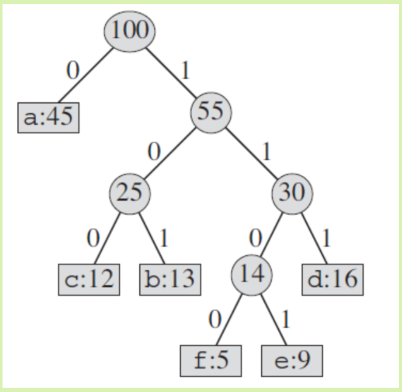
## Variable-Length Binary Codes for a Set of Symbols

- **Possibly.** If the frequency of occurrence are different.

Example

| | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ |
|---|---|---|---|---|---|---|
| Frequency (in thousands) | 45 | 13 | 12 | 16 | 9 | 5 |

Before we do actual design, lets see a code tree.



| | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ |
|---|---|---|---|---|---|---|
| Frequency (in thousands) | 45 | 13 | 12 | 16 | 9 | 5 |
| Variable-length codeword | 0 | 101 | 100 | 111 | 1101 | 1100 |

## Benefits of Variable-Length Codes

Table from previous slide

| | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ |
|---|---|---|---|---|---|---|
| Frequency (in thousands) | 45 | 13 | 12 | 16 | 9 | 5 |
| Variable-length codeword | 0 | 101 | 100 | 111 | 1101 | 1100 |

- Suppose, we encode a string of 100 characters

  $a, b, c, d, e,$ and $f$ occurs exactly 45, 13, 12, 16, 9, and 5 times respectively.
  We use the code above. What is the length of the encoded string in bits?
  $45 \times 1 + 13 \times 3 + +12 \times 3 + 16 \times 3 + 9 \times 4 + 5 \times 4$
  $= 45 + 39 + 36 + 48 + 36 + 20$
  $= 224 \Rightarrow 2.24$ bits per characters
  $\Rightarrow$ a savings of 76 bits or 25.37%

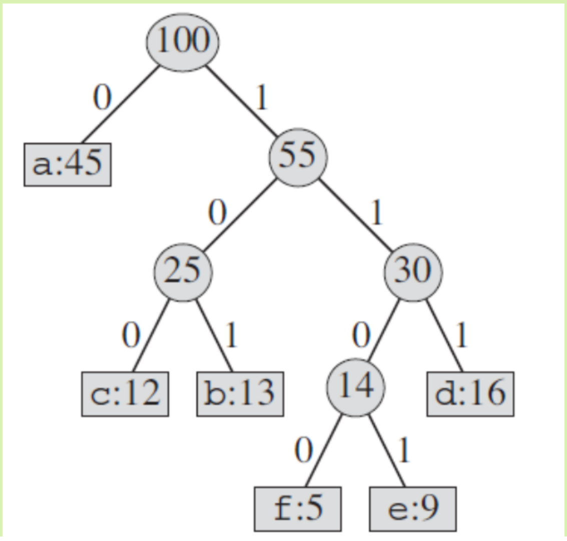
  - **Prefix codes:** code of a symbol is not a prefix of another code.
    Code of $a = 0$ is not beginning of codes for $b, c, d, e$ or $f$
    Code of $b = 101$ is not beginning of codes for $e$ and $f$

## Properties of Prefix Codes

**Prefix codes:** code of a symbol is not a prefix of another code.
Code of $a = 0$ is not beginning of codes for $b, c, d, e$ or $f$
Code of $b = 101$ is not beginning of codes for $e$ and $f$

Three lines above are from the previous slide
The *prefix* property is necessary for decoding

- Given string 01010, how do we decode it?
  Use the code tree



- Start at the root.
- If symbol is a '0' move left
- If symbol is a '1' move to right
- Repeat until a leaf node is reached
- You have a symbol now.
  For the next symbol, start at the root again.
- Following steps above we find $a\,b\,a$

## A Greedy Algorithm for Constructing Huffman Code

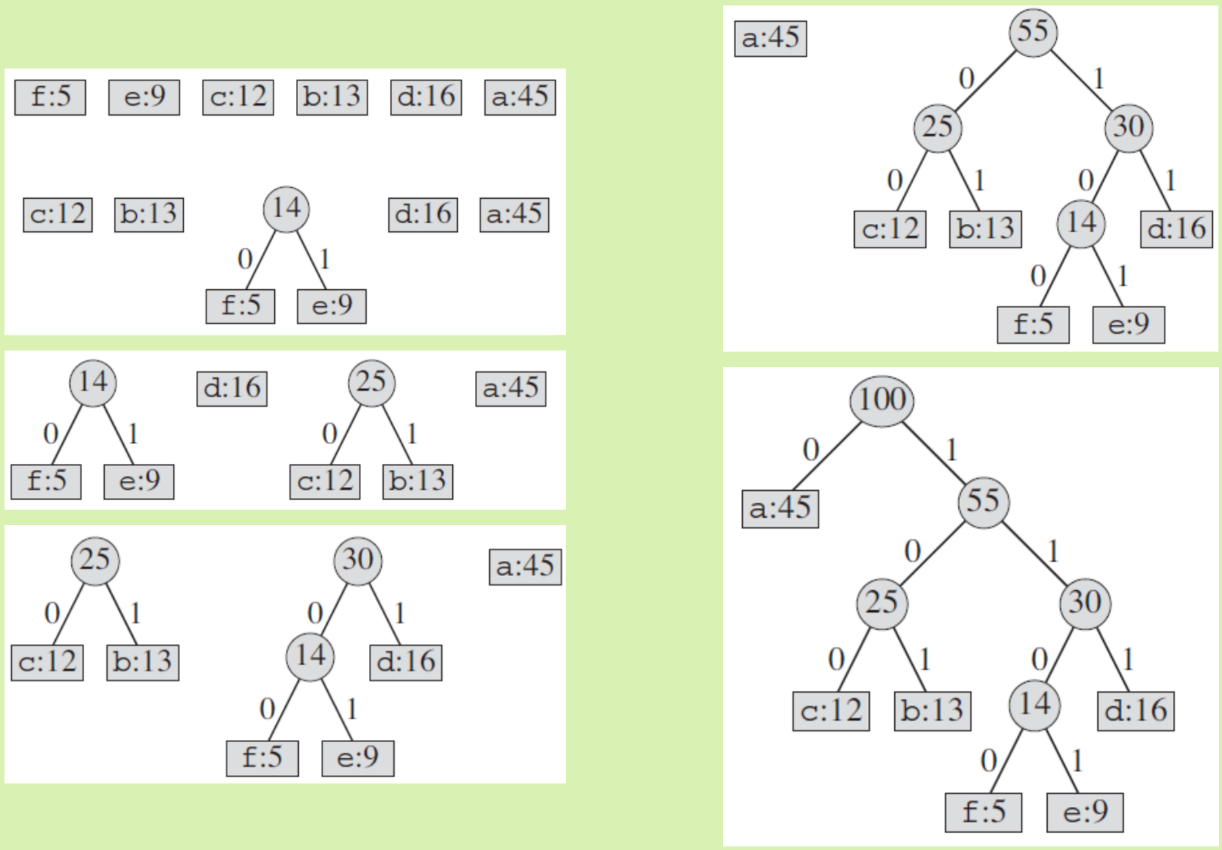
**Notations:**

- $C$ is a set of charters and $c \in C$
- $c.freq$ is the frequency of the character $c$
- $Q$ is a min-priority queue
- The algorithm builds the tree $T$ corresponding to the optimal code
- One node is added at each step

```
HUFFMAN($C$)
1  $n = |C|$
2  $Q = C$
3  for $i = 1$ to $n - 1$
4      allocate a new node $z$
5      $z.left = x = $ EXTRACT-MIN($Q$)
6      $z.right = y = $ EXTRACT-MIN($Q$)
7      $z.freq = x.freq + y.freq$
8      INSERT($Q, z$)
9  return EXTRACT-MIN($Q$)    // return the root of the tree
```

- **Note:** The left child has smaller value than the right child

## Illustration of Execution of Algorithm for Huffman Code


f:5 e:9 c:12 b:13 d:16 a:45
c:12 b:13 14 f:5 e:9 d:16 a:45
14 f:5 e:9 d:16 25 c:12 b:13 a:45
25 c:12 b:13 30 14 f:5 e:9 d:16 a:45
a:45 55 25 c:12 b:13 30 14 f:5 e:9 d:16
100 a:45 55 25 c:12 b:13 30 14 f:5 e:9 d:16
0
1

## Optimality of the Codes I

- **Definition:** A binary tree is **full**, if every internal node has two children.
- Hoffman code tree is a full binary tree.

  Let $B(T)$ denote the number for bits required to encode a file using codes represented by tree $T$.

$$B(T) = \sum_{c \in C} c.freq \times d_T(c)$$

where $c.freq$ is the frequency of character $c$, and $d_T(c)$ is the number of bits for encoding charter $c$.

**Lemma**

*Let $C$ be an alphabet in which each charter $c \in C$ has a frequency $c.freq$. Let $x$ and $y$ be two charters in $C$ having the lowest frequencies. Then there exists an optimal prefix code for $C$ in which the codewords for $x$ and $y$ have the same length and differ only in the last bit.*

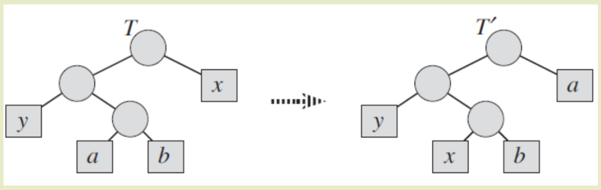
## Optimality of the Codes II

Proof.

- Proof is by contradiction.
- Assume that there is an optimal tree $T$ where two lowest-frequency symbols $x$ and $y$ are not at the lowest level.

  Suppose symbols $a$ and $b$ are at the lowest level of $T$.

  Code-length of $a$ and $b$ are same (and differ only at the last bit).



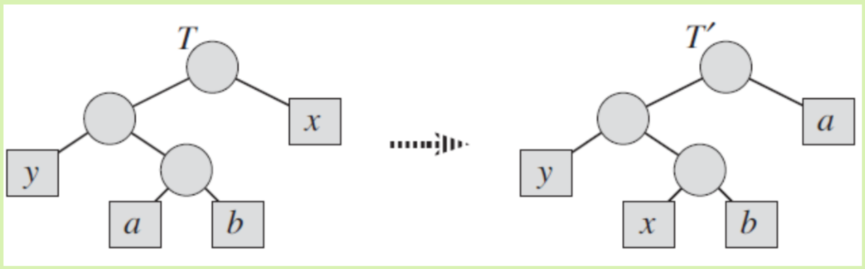
- Then construct a tree $T'$ by exchanging symbols $a$ and $x$.

  Show that $B(T) - B(T') \geq 0$
- Then construct a tree $T''$ by exchanging symbols $b$ and $y$.
- Show that $B(T') - B(T'') \geq 0$
- $\Rightarrow$ Assumption about optimal tree $T$ is not true.

  Let us do the algebra next.

$\square$

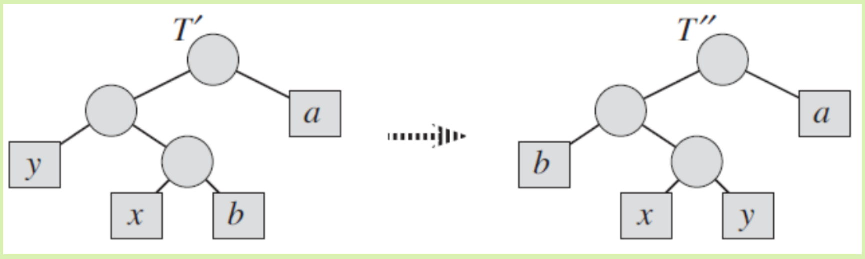
## Optimality of the Codes III



- Recall that, $B(T) = \sum_{c \in C} c.freq.d_T(c)$
- We want to show that $B(T) - B(T') \geq 0$

$$\begin{aligned}
B(T) - B(T') &= \textstyle\sum_{c \in C} c.freq.d_T(c) - \sum_{c \in C} c.freq.d_{T'}(c) \\
&= x.freq.d_T(x) + a.freq.d_T(a) - x.freq.d_{T'}(x) - a.freq.d_{T'}(a) \\
&= x.freq.d_T(x) + a.freq.d_T(a) - x.freq.d_T(a) - a.freq.d_T(x) \\
&= (a.freq - x.freq)(d_T(a) - d_T(x)) \\
&\geq 0
\end{aligned}$$

## Optimality of the Codes IV



- Following similar steps we can show that $B(T') - B(T'') \geq 0$
- Thus, $B(T) - B(T'') \geq 0$
- $\Rightarrow$ This contradicts the fact that $T$ is optimal.

  $T''$ is optimal and has lowest frequency character at the lowest level of the tree.

## Why Greedy method works?

- We get a best set of codes with a **greedy** algorithm!
- **Why the greedy choice works?**
- The answer is the next lemma.

**Lemma**

*Let $C$ be an alphabet in which each charter $c \in C$ has a frequency $c.freq$. Let $x$ and $y$ be two charters in $C$ having the lowest frequencies. Let $C'$ be the alphabet obtained from $C$ with characters $x$ and $y$ removed and a new character $z$ added, where $z.freq = x.freq + y.freq$. Let $T'$ be any tree representing an optimal prefix code for $C'$. Then the tree $T$, obtained from $T'$ by replacing the leaf node for $z$ with an internal node having $x$ and $y$ as children, represents an optimal prefix code for the alphabet $C$.*

**Proof.**

- The idea behind the proof is simple.

  If a greed choice is made, the number of characters is reduced by one

  This is because we replaced two selected characters with a ‘metacharacter’.
- If the tree $T'$ obtained from the reduced set of characters is optimal, then tree obtained after expanding the ‘meta-character’ is also optimal.
- The proof is by contradiction.

□