

# CSC373 Worksheet 2 Solution

July 27, 2020

1)  $[a_{11} = [12, 16]]$

$i$	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 red arrow points from  $s_{11} = 12$  to  $f_8 = 11$ . A blue arrow labeled  $k$  points up to  $s_{11} = 12$ .

3)  $[a_{11} = [12, 16], a_2 = [8, 11], a_4 = [5, 7]]$

$i$	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 red arrow points from  $s_4 = 5$  to  $f_2 = 5$ . A blue arrow labeled  $k$  points up to  $s_4 = 5$ .

2)  $[a_{11} = [12, 16], a_2 = [8, 11]]$

$i$	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 red arrow points from  $s_8 = 8$  to  $f_7 = 10$ . A blue arrow labeled  $k$  points up to  $s_8 = 8$ .

3)  $[a_{11} = [12, 16], a_2 = [8, 11], a_4 = [5, 7], a_1 = [1, 4]]$

$i$	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 blue arrow labeled  $k$  points up to  $s_1 = 1$ .

1.

This approach is a greedy algorithm because algorithm

- 1) Has the greedy choice: selecting the last activity to start that is compatible with all previously selected activities
- 2) Has the greedy choice that is always part of optimal solution:

## Claim:

Consider any nonempty subproblem  $S_k$ . Let  $a_m$  be an activity in  $S_k$  with the last activity to start that is compatible with all previously selected activities. 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 the last activity to start that is compatible with all previously selected activities.

If  $a_j = a_m$ , we are done, since we have shown that  $a_m$  is the maximum-size subset of mutually compatible activities of  $S_k$ .

If  $a_j \neq a_m$ , let the set  $A'_k = A_k = \{a_j\} \cup \{a_m\}$  be  $A_k$  but substituting  $a_m$  for  $a_j$ . The activities in  $A'_k$  are disjoint, which follow because the activities in  $A_k$  are disjoint,  $a_j$  is the first activity in  $A_k$  to finish, and  $s_j \leq s_m$ .

Since  $|A'_k| = |A_k|$ , we conclude that  $A'_k$  is a maximum-size subset of mutually compatible activities of  $S_k$ , and it includes  $a_m$ .  $\square$

### Notes:

- Greedy Algorithm
  - Always makes the choice that looks best at the moment
    - \* Locally optimal solution leads to globally optimal solution
- Activity-selection Problem (Greedy algorithm using dynamic programming)
  - Goal: Selecting maximum size set of mutually compatible activities

### 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

- Suppose a set exists  $S = \{a_1 = [s_1, f_1), a_2 = [s_2, f_2), \dots, a_n = [s_n, f_n)\}$ 
  - \*  $a_i$  represents an  $i^{th}$  activity
  - \*  $s_i$  represents starting time
  - \*  $f_i$  represents finishing time
  - \*  $0 \leq s_i < f_i < \infty$
  - \*  $a_1, \dots, a_n$  sorted in monotonically increasing order of finish time

i.e.

$$f_1 \leq f_2 \leq f_3 \leq \dots \leq f_{n-1} \leq f_n$$

- \*  $a_i$  and  $a_j$  are **compatible**, if intervals  $[s_i, f_i)$  and  $[s_j, f_j)$  don't overlap

i.e

$$s_i \geq f_j \text{ and } s_j \geq f_i$$

- Steps
  1. Think about dynamic programming solution
    - \* Construct optimal solution using two subproblems

$S_{ij}$ : activities that start after activity  $a_i$  finishes and before activity  $a_j$  starts

i.e.

$$S_{19} = \{a_4 = [5, 7), a_6 = [5, 9), a_7 = [6, 10)\}$$

$A_{ij}$ : maximum set of mutually compatible activities in  $S_{ij}$  (including  $a_k$ )

- $A_{ik} = A_{ij} \cap S_{ik}$
- $A_{kj} = A_{ij} \cap S_{kj}$
- $A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$
- So,  $|A_{ij}| = |A_{ik}| + |A_{kj}| + 1$

- \* Verify that optimal solution  $A_{ij}$  must include optimal solution to the two subproblems for  $S_{kj}$

Let  $A'_{kj}$  be another mutually compatible activities in  $S_{kj}$  where  $|A'_{kj}| > |A_{kj}|$ .

Then we could use  $A'_{kj}$  in a solution to subproblem of  $S_{ij}$

Then we have  $|A_{ik}| + |A'_{kj}| + 1 > |A_{jk}| + |A_{kj}| + 1 = |A_{ij}|$  mutually compatible activities

This contradicts assumption that  $A_{ij}$  is an optimal solution

- \* Verify that optimal solution  $A_{ij}$  must include optimal solution to the two subproblems for  $S_{ik}$

The same applies for activities in  $S_{ik}$

2. Observe that only one choice - greedy choice, and that when we make the greedy choice, only one subproblem remains

- \* Steps

1. Make a greedy choice
  - Choose an activity that makes the most resource possible (intuition)
  - Choose an activity that finishes the earliest (intuition)
2. Solve a subproblem: Find activities that start after  $a_1$  finishes
3. Verify that making greedy choices always arrive at optimal solution

### **Theorem 16.1 (Page 418):**

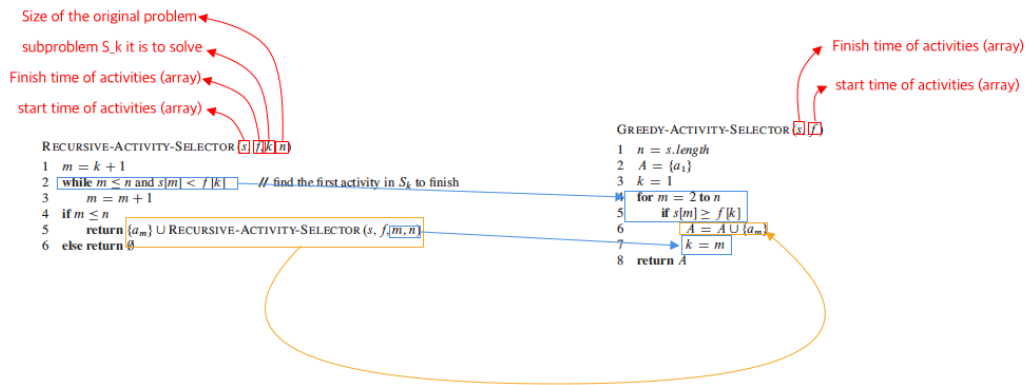
Consider any non-empty subproblem  $S_k$ , and let  $a_m$  be an activity in  $S_k$  with the earliest finish time. Then  $a_m$  is included in some maximum-size

subset of mutually compatible activities of  $S_k$

### 3. Develop recursive greedy solution



### 4. Convert the recursive algorithm into iterative one



### 2. • Greedy Choice

- Choose  $x_i$  that is greater than the current maximum as the upper bound of unit length closed interval
- Choose  $x_i$  that is smaller than the current minimum as the lower bound of unit length closed interval

**Example:**

$$\{0, 1, 2, 3, 4, 5\} \rightarrow [0, 5]$$

$$\{0, -1, 3, 5, 2\} \rightarrow [-1, 5]$$

- Optimal Substructure

Let  $I$  be the following instance of the problem: Let  $n$  be the number of items, and let  $x_i$  be the  $i^{th}$  point in the set.

Let  $A = [x_{\min}, x_{\max}]$  be the solution. The greedy algorithm works by assigning  $x_{\min} = \min(x_{\min}, x_n)$  and  $x_{\max} = \max(x_{\max}, x_n)$ , and then continuing by solving the subproblem

$$I' = (n - 1, \{x_1, \dots, x_{n-1}\}) \quad (1)$$

until  $n = 0$ .

We need to show that the strategy gives optimal solution.

**Correct Solution:**

- 1) Consider the left-most interval.
- 2) Set the left most point  $x$  in the set as its value (since we know it must contain the leftmost point)
- 3) For any point that is within the unit distance of the point  $x$  (i.e.  $[x, x + 1]$ ), remove the points since they are covered
- 4) Move to the next closest point not covered by the unit interval of  $x$ , and repeat until all points in the set are covered.
- 5) Since each step has a clearly optimal choice for where to put the leftmost interval, the final solution is optimal

**Notes:**

- I stopped because it's taking too much time.
- I struggled on this problem.

- I had trouble understanding the meaning of unit interval
  - I felt there is missing knowledge regarding optimal substructure
  - I felt tunnel visioned to provide one interval that covers all
- I had difficulty arguing why the algorithm is correct
  - i.e. How can i generate a claim?
- Unit length
  - $[1, 25, 2.25]$  includes all  $x_i$  such that  $1.25 \leq x_i \leq 2.25$ .
- Greedy-choice property and optimal substructure to problem are the two key ingredients
- Summary of Steps for Greedy Algorithm
  1. Determine the optimal structure of the problem
  2. Develop a recursive solution.
  3. Show that if we make the greedy choice, then only one subproblem remains
  4. Prove that it is always safe to make the greedy choice
  5. Develop a recursive algorithm that implements the greedy strategy
  6. Convert the recursive algorithm to an iterative algorithm
- Criteria for Greedy Algorithm
  1. Greedy-choice property
    - Exists if we can assemble a globally optimal solution by making a locally optimal (greedy) choices
  2. Optimal Substructure
    - Exists if an optimal solution to the problem contains within it optimal solutions to subproblems.
- Greedy vs Dynamic Programming
  - 0-1 Knapsack Problem

1)



Capacity: 50 lbs  
Current: 50lbs

**0-1 Knapsack Problem**

item 1  
weight: 10 lbs  
worth: \$60  
value: \$6/lbs

item 2  
weight: 20 lbs  
worth: \$100  
value: \$5/lbs

item 3  
weight: 30 lbs  
worth: \$120  
value: \$4/lbs

2)



Capacity: 50 lbs  
current: 40 lbs

item 1  
weight: 10 lbs  
worth: \$60  
value: \$6/lbs

item 2  
weight: 20 lbs  
worth: \$100  
value: \$5/lbs

item 3  
weight: 30 lbs  
worth: \$120  
value: \$4/lbs

3)



Capacity: 50 lbs  
current: 20 lbs

item 1  
weight: 10 lbs  
worth: \$60  
value: \$6/lbs

item 2  
weight: 20 lbs  
worth: \$100  
value: \$5/lbs

item 3  
weight: 30 lbs  
worth: \$120  
value: \$4/lbs

Uh oh. This is not greedy!

– Fractional Knapsack Problem

1)



Capacity: 50 lbs  
Current: 50lbs

Fractional Knapsack Problem

item 1  
weight: 10 lbs  
worth: \$60  
value: \$6/lbs

item 2  
weight: 20 lbs  
worth: \$100  
value: \$5/lbs

item 3  
weight: 30 lbs  
worth: \$120  
value: \$4/lbs

2)



Capacity: 50 lbs  
current: 40 lbs

item 1  
weight: 10 lbs  
worth: \$60  
value: \$6/lbs

item 2  
weight: 20 lbs  
worth: \$100  
value: \$5/lbs

item 3  
weight: 30 lbs  
worth: \$120  
value: \$4/lbs

3)



Capacity: 50 lbs  
current: 20 lbs

item 1  
weight: 10 lbs  
worth: \$60  
value: \$6/lbs

item 2  
weight: 20 lbs  
worth: \$100  
value: \$5/lbs

item 3  
weight: 30 lbs  
worth: \$120  
value: \$4/lbs

item 2  
weight: 20 lbs  
worth: \$100  
value: \$5/lbs



4)



3. *Proof.* Let  $T$  be a binary tree corresponding to an optimal prefix code and suppose that  $T$  is not full. Let node  $n$  have a single child  $x$ . Let  $T'$  be the tree obtained by removing  $n$  and replacing it by  $x$ . Let  $m$  be leaf node which is descendent of  $x$ . Then we have:

**My work:**

$$B(T') \leq \sum_{c \in C \setminus \{m\}} c.freq \cdot d_T(c) + m.freq \cdot d_{T'}(m) \quad (1)$$

$$= \sum_{c \in C \setminus \{m\}} c.freq \cdot d_T(c) + m.freq \cdot (d_T(m) - 1) \quad (2)$$

$$< \sum_{c \in C \setminus \{m\}} c.freq \cdot d_T(c) + m.freq \cdot d_T(m) \quad (3)$$

$$= \sum_{c \in C} c.freq \cdot d_T(c) \quad (4)$$

$$= B(T) \quad (5)$$

which contradicts the fact that  $T$  was optimal. Therefore every binary tree corresponding to an optimal prefix code is full

□

**Notes:**

- Optimal Substructure

- A problem is said to have optimal substructure if an optimal solution can be constructed from optimal solutions of its subproblems.

- Huffman Codes

- Is an algorithm that uses greedy algorithm for lossless (without loss of data) data compression
- Has two types of codewords

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

- \* Fixed Length Code
  - has codeword with the same length
- \* Variable Length
  - has codeword that may be of different lengths
- Constructs optimal prefix codes
  - \* Means no codeword is a prefix of some other codewords

e.g.

The following is not prefix codes

a - 110

b - 1101

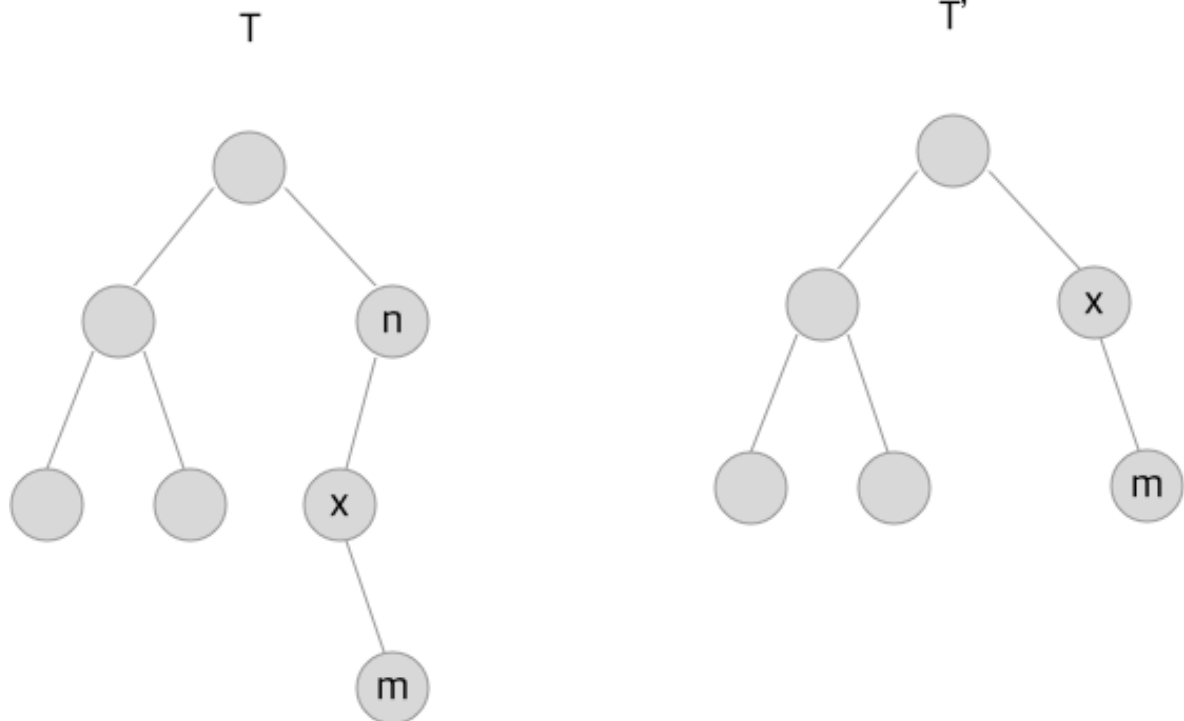
e.g.

The following is prefix codes

a - 110

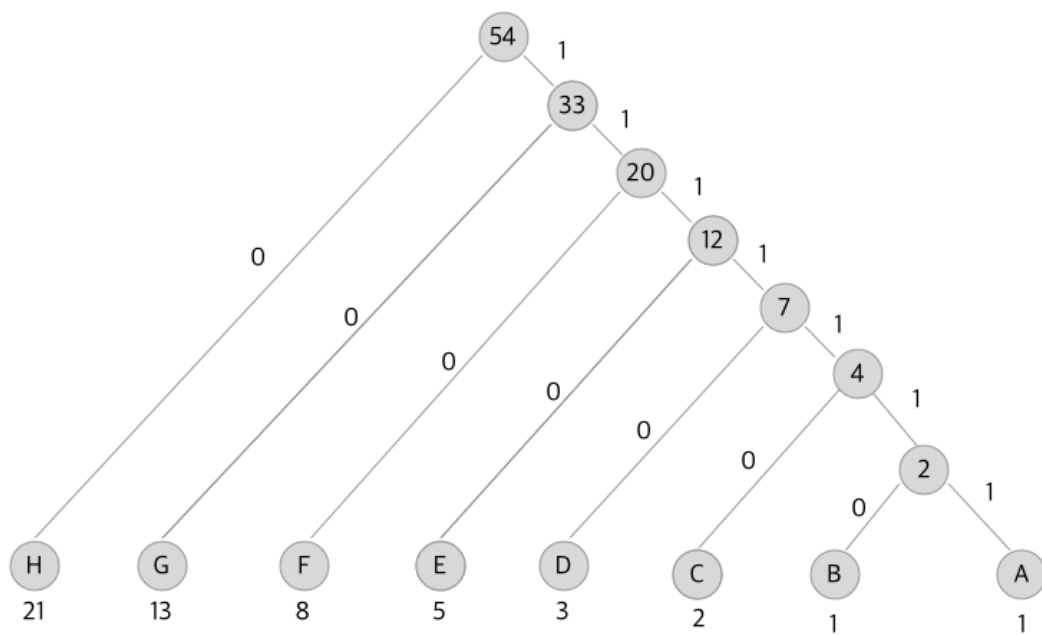
b - 111

- Realized that I should learn with the solution. Otherwise, it will take too much time.
- Learned that the author used another but very similar tree  $T'$  to show the cost of bits in  $T$  is not minimum, which is the condition of prefix codes.
- Learned that the solution feels very similar to the proof of optimal substructure on page 416.
- Learned that the tree  $T$  and  $T'$  looks as follows:

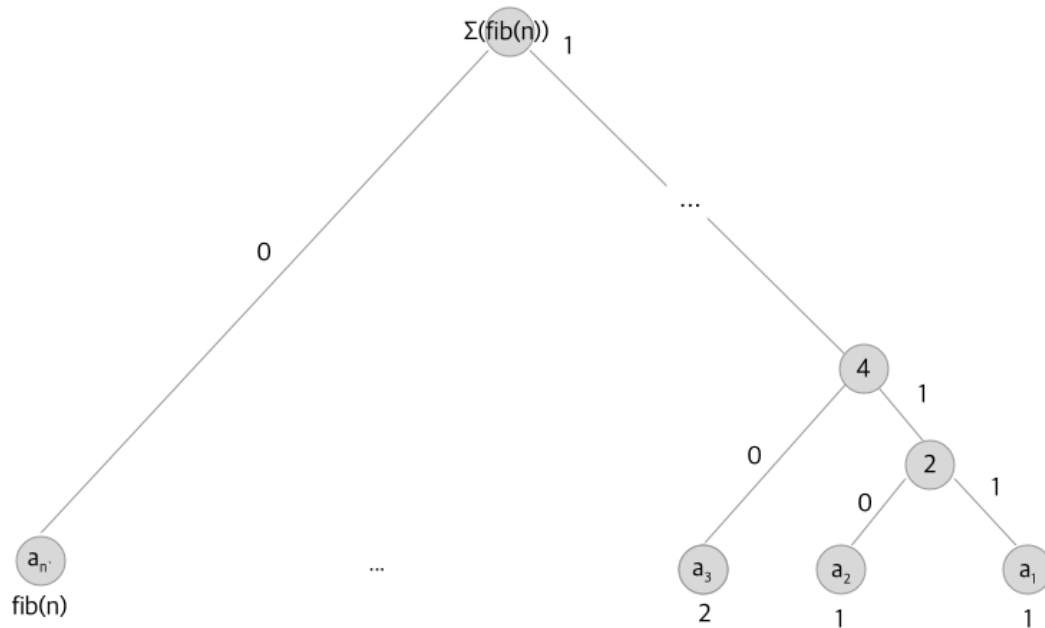


4. **Solution:**

- Finding optimal Huffman code



- Generalizing answer to find the optimal code when the frequencies are first  $n$  fibonacci numbers



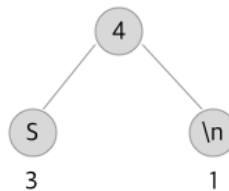
### Notes

- Constructing Huffman Code

#### Example:

char	A	E	I	S	T	P	\n
Freq	10	15	12	3	4	13	1

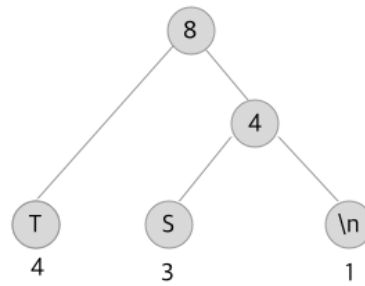
- Take the 2 chars with the lowest frequency



- Make a 2 leaf node tree from them

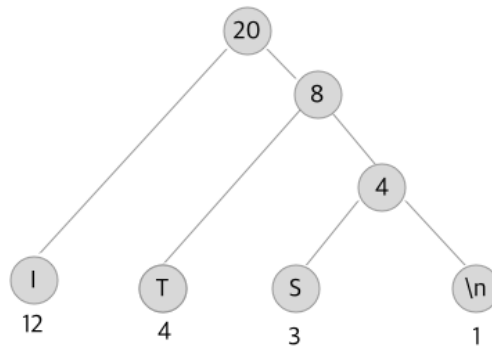
2)

char	A	E	I	S	T	P	\n
Freq	10	15	12	<del>3</del>	<del>4</del>	13	<del>1</del>



3)

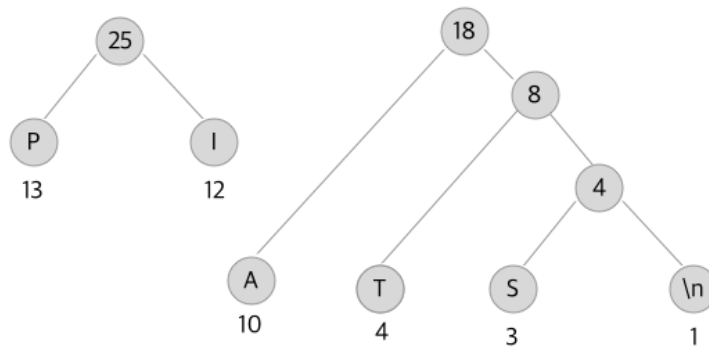
char	A	E	I	S	T	P	\n
Freq	10	15	<del>12</del>	<del>3</del>	<del>4</del>	13	<del>1</del>



3. If the node has summed value that is higher than any other values in the table, then repeat 1 and 2 in another tree

4)

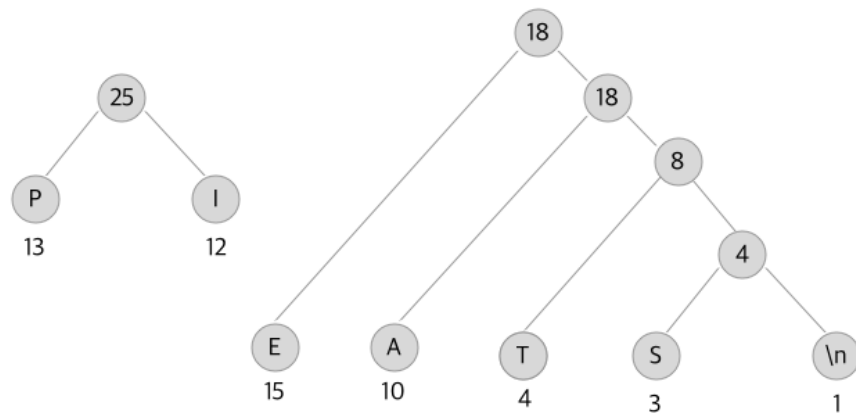
char	A	E	I	S	T	P	\n
Freq	<del>18</del>	15	<del>12</del>	<del>3</del>	<del>4</del>	<del>13</del>	<del>1</del>



4. Attach an additional node to the subtree with the smallest value

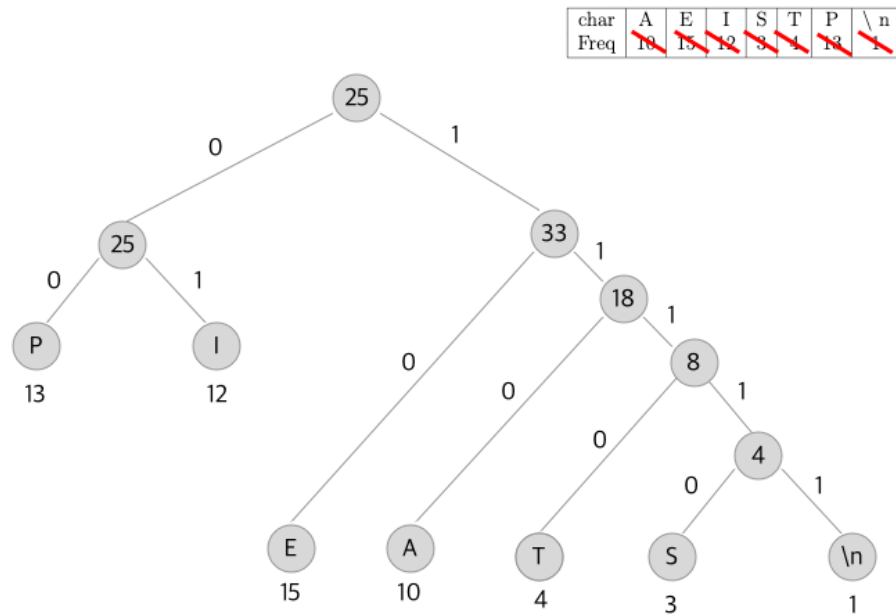
5)

char	A	E	I	S	T	P	\n
Freq	<del>18</del>	15	<del>12</del>	<del>3</del>	<del>4</del>	<del>13</del>	<del>1</del>



5. Repeat step 4 above until done

5)



5. Instead of grouping together the two with lowest frequency into pairs that have the smallest total frequency, we will group together the three with lowest frequency in order to have a final result that is a ternary tree. The analysis of optimality is almost identical to the binary case. We are placing the symbols of lowest frequency lower down in the final tree and so they will have longer codewords than the more frequently occurring symbols.

### My Work:

- Proof of greedy-choice property for Huffman's Algorithm (Ternary)

#### Lemma (Modification of lemma on page 433):

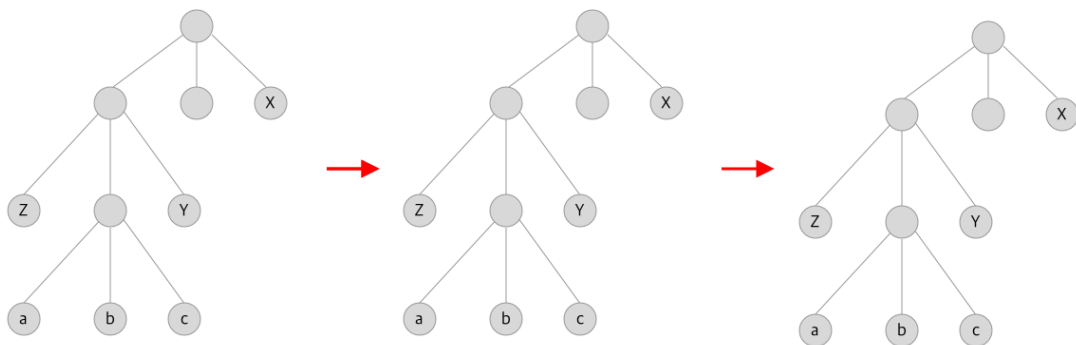
Let  $C$  be an alphabet in which each character  $c \in C$  has frequency  $c.freq$ . Let  $x$ ,  $y$  and  $z$  be two characters in  $C$  having the lowest frequencies. Then there exists an optimal prefix code for  $C$  in which the codewords for  $x$ ,  $y$  and  $z$  have the same length and differ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

*Proof.* Let  $a$ ,  $b$  and  $c$  be three characters that are sibling leaves of maximum depth in  $T$ .

Without loss of generality, we assume that  $a.freq \leq b.freq \leq c.freq$ ,  $x.freq \leq y.freq \leq z.freq$ . Since  $x.freq$ ,  $y.freq$  and  $z.freq$  are the three lowest leaf frequencies, in order, and  $a.freq$  and  $b.freq$  are three arbitrary frequencies, in order, we have  $x.freq \leq a.freq$ ,  $y.freq \leq b.freq$ , and  $z.freq \leq c.freq$ .

In the remainder of the proof, it is possible that we could have  $x.freq = a.freq$ ,  $y.freq = b.freq$ , and  $z.freq = c.freq$ . However, if we had  $x.freq = b.freq$  and  $x.freq = c.freq$ , then we would also have  $a.freq = b.freq = c.freq$ ,  $x.freq = y.freq = z.freq$ , and the lemma is trivially true. Thus, we will assume  $x.freq \neq b.freq$  and  $x.freq \neq c.freq$ , which means that  $x \neq b$  and  $x \neq c$ .

As the following image shows, we exchange the positions in  $T$  of  $a$  and  $x$  to produce a tree  $T'$ , and then we exchange the positions in  $T'$  of  $b$  and  $y$  to produce a tree  $T''$ , and then we exchange the position in  $T''$  of  $c$  and  $z$  to produce a tree  $T'''$  in which  $x$ ,  $y$  and  $z$  are sibling leaves of maximum depth. By equation (16.4), the difference in cost between  $T$  and  $T'$  is



$$B(T) - B(T') = \sum_{c \in C} c.freq \cdot d_T(c) - \sum_{c \in C} c.freq \cdot d_{T'}(c) \quad (6)$$

(7)

□

- Proof of optimal structure property for Huffman's Algorithm (Ternary)

### Notes:

- Lemma means “subsidiary or intermediate theorem in an argument of proof”
- Proof of greedy-choice property for Huffman's Algorithm (Binary)

### Lemma 16.2:



Let  $C$  be an alphabet in which each character  $c \in C$  has frequency  $c.freq$ . Let  $x$  and  $y$  be two characters 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.

*Proof.*

□

- Proof of optimal structure property for Huffman's Algorithm (Binary)