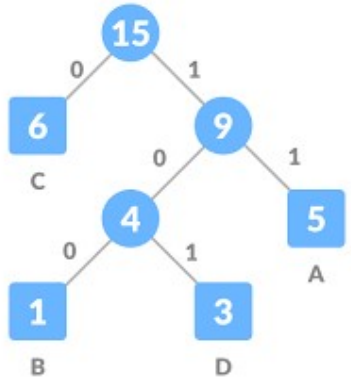
Algorithms Analysis and Design

Chapter 5

Greedy Technique
Part 3



Huffman Code

Compression techniques

- Text can be encoded by representing each character by a bit string (code)
- Each character has a frequency (how often it appears in the text)
- **The goal** is to represent the data using the minimum number of bits→ data compression
- Fixed-length encoding scheme:
  - Each character is represented using the same number of bits (fixed-size code)
  - e.g. ASCII code
- Variable-length encoding scheme:
  - Different characters are encoded using different number of bits (variable-size code)
  - e.g. Huffman coding

Example of fixed-length encoding

- Given a text contains "**AAUP**"
- ASCII : 8 bits for each character

  size = 4 * 8 = 32 bits
- Can we reduce this size?
- We have 3 different characters, two bits are enough to encode each character

  A (00) , U (01) , P (10) size = 4 * 2 = 8 bits

To reduce the size further, we will use a variable-length encoding scheme in which we assign a variable length (variable number of bits) to characters depending upon their frequency in the given text.

Huffman Coding

- **Huffman Coding** is a technique of compressing data to reduce its size without losing any of the details.

Lossless compression technique

- Huffman's algorithm achieves data compression **by finding the best variable length binary encoding scheme for the symbols** that occur in the file to be compressed.
- Huffman Coding is generally useful to compress the data in which **there are frequently occurring characters**.
- Huffman's greedy algorithm uses a table of the frequencies of occurrence of each character to build up an optimal way of representing each character as a binary string.

Huffman coding

- **Huffman codes** use variable–length binary codes to represent characters
  - Use **short bit-strings** to represent **the most frequently** used characters
  - Use longer bit-strings to represent less frequently used characters

→ It is possible to represent text in **less space** than if fixed-length code is used

The more frequent a symbol occurs, the shorter should be the Huffman binary word representing it.

Space is saved

- Huffman codes: compressing data (savings of 20% to 90%)
- Example: Consider a file of 100,000 characters contains only the characters { A – F } with the frequencies shown in the table.

only three bits are enough to encode 6 different characters

Letter to be encoded	Frequency (thousands)	Fixed-length code	Size (thousands)	Variable-length code	Size (thousands)
A	45	000	**135**	0	**45**
B	13	001	**39**	101	**39**
C	12	010	**36**	100	**36**
D	16	011	**48**	111	**48**
E	9	100	**27**	1101	**36**
F	5	101	**15**	1100	**20**
			300,000		**224,000**

- Using the fixed-length encoding (3-bits codeword) the file can be encoded in **300,000 bits**.
- Using the variable-length code shown, the file can be encoded in 224,000 bits

The size is reduced by 25.33%

300000 − 224000

300000

Prefix code

- **Prefix code (also known prefix-free code)** is a code in which **NO** character is represented by a bit-string that is the initial segment of a bit-string that represents another character.
- A code is called a prefix (free) code if no codeword is a prefix of another one.
  - Example: {a = 0, b = 110, c = 10, d = 111} is a prefix code.
  - Example**:**

  If **A** is represented by **100**, **B** by **101** and **T** by **100101** **is not** a prefix code encode **100101**??

  → **T** ? OR **AB**?!

The Huffman code is a prefix-free code

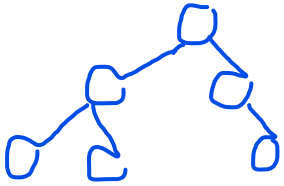
Huffman coding

- **The problem is**

  given a table of characters and their frequencies, construct a Huffman code that represents the text using as few bits as possible

- **Representation**

  A Huffman code may be defined by a **binary** tree → **Huffman coding Tree**



Binary Tree

- **definition**

  A binary tree is a tree data structure in which each node has at most two children, referred to as the left child and the right child

Root, children, leaf nodes, subtrees

Types of binary trees: Full binary tree, complete binary tree, balanced binary tree ….

Binary tree vs Binary search tree

Applications: Min Heap, Max Heap, Binary Search Tree

Building a Huffman Tree

Each node contains:

- A character (**ch**)
- A frequency (**freq**)
- and left and right members (**left**, **right**)

After the Huffman coding tree is constructed, a left member of a node references its left child, and a right member of a node references its right child or,

if the node is a **terminal** vertex, its left and right members are null.

Building a Huffman Tree

الخطوات

- Put all leaf nodes in **priority queue** keyed on **frequency**
  - Select the two nodes with the lowest frequencies, create a parent node of them
  - Assign the sum of the children's frequencies to the parent node and insert it into the queue
  - Assign code 0, 1 to the two branches of the tree, and delete the children from the queue
  - Repeat until the queue has only one node left
- The code for a character is obtained by following the path from the root to the character and noting the bits on the edges of the path

Example 1

Suppose the following string is to be sent over a network:

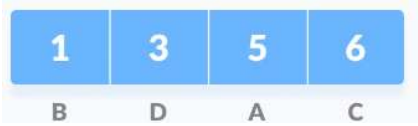
BCAADDDCCACACAC

Using Huffman coding algorithm, find the variable-length prefix-free encoding scheme to reduce the size of the string.

1. Calculate the frequency of each character in the string.

2. Sort the characters in increasing order of the frequency. These are stored in a priority queue Q

Char	Frequency
A	5
B	1
C	6
D	3



Example 1

Suppose the following string is to be sent over a network:

BCAADDDCCACACAC

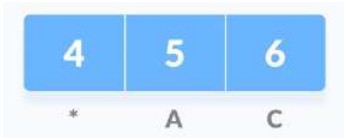
Using Huffman coding algorithm, find the variable-length prefix-free encoding scheme to reduce the size of the string.

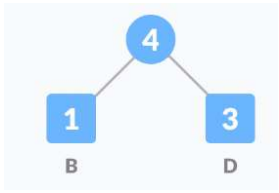
Char	Frequency
A	5
B	1
C	6
D	3

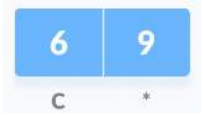
3. Select the two nodes with the lowest frequencies and create a parent of them (z). Assign the minimum frequency to the left child of z and assign the second minimum frequency to the right child of z. Set the value of the z as the sum of the above two minimum frequencies.
4. Remove these two minimum frequencies from Q and add the sum into the list of frequencies (* denote the internal nodes).

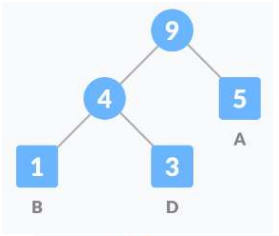
5. Insert node z into the tree.

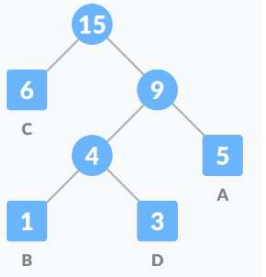
6. Repeat until the queue has only one node left







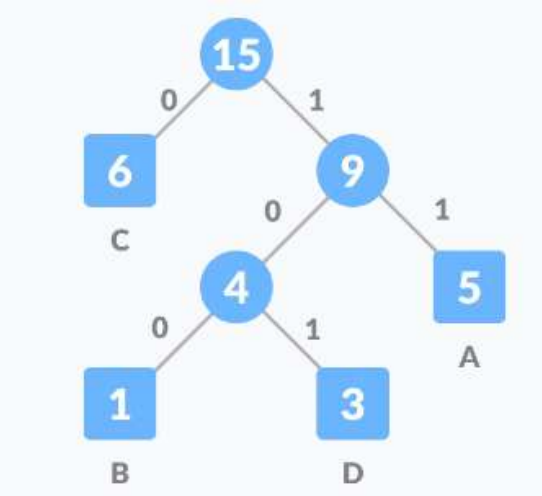




② huffman tree

7. For each non-leaf node, assign **0** to the left edge and **1** to the right edge

Char	Frequency	code	size
A	5	11	5*2 = 10
B	1	100	1 * 3 = 3
C	6	0	6 * 1 = 6
D	3	101	3 * 3 = 9
			28 bits



Without encoding (i.e. using ASCII), the total size of the string was 15 * 8 = **120** bits
After encoding the size is reduced to **28** bits (**neglecting the size of the header**)

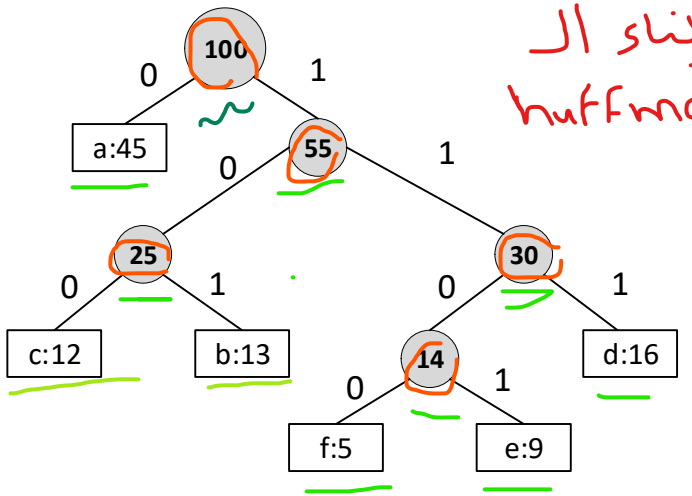
Note that the most frequently used character is encoded using the shortest code

Example 2

- Assume we are given a data file that contains only 6 symbols, namely **a, b, c, d, e, f** with the following frequency table:

char	Frequency (thousands)	Variable-length code	Size (thousands)
a	45	0	**45**
b	13	101	**39**
c	12	100	**36**
d	16	111	**48**
e	9	1101	**36**
f	5	1100	**20**
	100		**224,000**

- Find a variable length prefix-free encoding scheme that compresses this data file as much as possible?



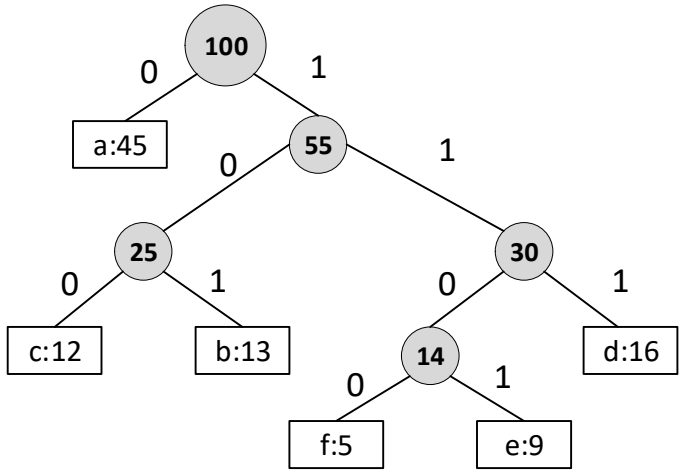
بناء ال huffman tree .

Encoding Data

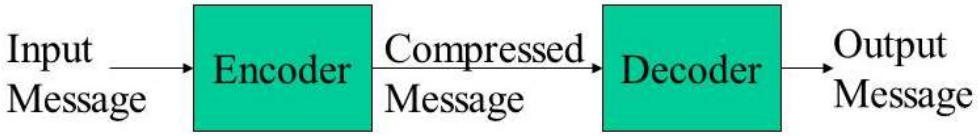
Once a Huffman code has been generated, data may be encoded simply by replacing each symbol with it's code

تشفير

- CAB → 1000101
- FACE → 110001001101
- FADE →
- BAD →



Decoding



- The encoding table tat maps each character with its corresponding code is sent along with the compressed file (as a header). This table will be used to decode the message.

Decoding Data

- start from the root
- read the encoded data one bit at a time
- branch according to the bits encountered in the input
- Once the bits read match a code for a character, write out the character and start collecting bits again

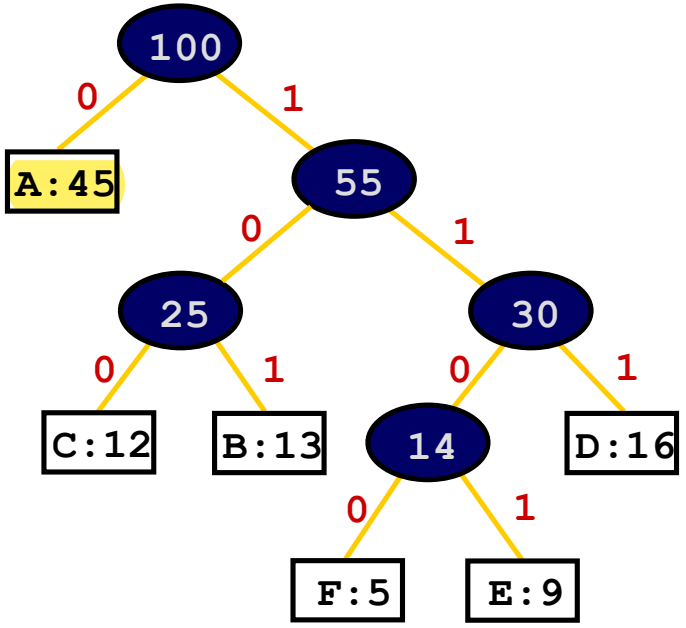
- Use binary tree to represent prefix codes for easy decoding

Decoding Example

- Decode the following code: 000101100

0 -> A
0 -> A
0 -> A
101 -> B
100 -> C

000101100 -> AAABC



Recall: Huffman coding prevents an ambiguity in the decoding process by using the concept of **prefix-free code**

Time complexity of Huffman coding algorithm

```
HUFFMAN(C)   // C is a set of n characters
1  n ← |C|
2  Q ← C     // Q is implemented as a binary min-heap        O(n)
3  for i ← 1 to n − 1
4      do allocate a new node z
5      left[z] ← x ← EXTRACT-MIN(Q)                          O(lg n)
6      right[z] ← y ← EXTRACT-MIN(Q)                         O(lg n)
7      f[z] ← f[x] + f[y]
8      INSERT(Q, z)                                          O(lg n)
9  return EXTRACT-MIN(Q)                  ▷ Return the root of the tree.
```

Recall: the cost of insert and remove operations for the priority queue when it is implemented as a heap data structure is **O(logn)**

Total computation time = $O(n \lg n)$

Time complexity of Huffman coding algorithm

- A **min-priority queue Q**, is used to identify the two least-frequent objects to merge together.
- The running time of Huffman's algorithm assumes that Q is implemented as a binary min-heap.
- Binary Heap is generally preferred for priority queue implementation because heaps provide better performance compared to arrays or LinkedList
- The **for** loop in lines 3-8 is executed exactly n - 1 times, and since each heap operation requires time O(lg n), the loop contributes O(n lg n) to the running time. Thus, the total running time of HUFFMAN on a set of n characters is **O(n lg n).**

Key Points

- Reduce size of data by 20%-90% in general
- If no characters occur more frequently than others, then no advantage over ASCII
- Encoding:
  - Given the characters and their frequencies, perform the algorithm and generate a code. Write the characters using the code
- Decoding:
  - Given the Huffman tree, figure out what each character is (possible because of prefix property)

Application on Huffman code

- Both the .mp3 and .jpg file formats use Huffman coding at one stage of the compression

Exercise

- Assume we are given a data file that contains only 9 symbols, namely **a, b, c, d, e, f, g, h, i** with the following frequency table:

- Find a variable length prefix-free encoding scheme that compresses this data file as much as possible?

char	Frequency
A	15
B	6
C	7
D	12
E	25
F	4
G	6
H	1
I	15