COMP20007 Workshop Week 12

LAB	**Preparation:** - *have draft papers and pen ready, or ready to work on whiteboard* 1. String Search & Horspool's Algorithm : Q 12.1, 12.2, 12.3 2. Huffman Coding: Questions 12.4, 12.5 Revision: 12.8: Quicksort & Mergesort 12.6: Complexity 12.7: Solving Recurrences OR Sample Exam Papers?

String Searching

Input:

A (long) text `T[0..n-1]`. Example: `T`= “SHE SELLS SEA SHELLS”, with `n`=20

A (short) pattern `P[0..m-1]`. Example: `P`=“HELL”, `m`=4.

Output: index `i` such that `T[i..i+m-1]=P[0..m-1],` or NOTFOUND

(Naïve) **Brute-Force Algorithm:**

complexity O(nm) (max= (n-m+1)*m character comparison)

- shift pattern from left to right on the text, **exactly 1 position each time**
- compare pattern with text character-by-character from left to right (stopping at first mis-match)

Horspool's Algorithm:

Complexity: O(nm) (worst-case, like brute-force), but often faster in practice.

Key Ideas:

- Compare the pattern with the text by scanning **from right to left.**
- Shift the pattern to the right as far as possible with each step.
- Preprocess the pattern to determine shift lengths.

Horspool’s Algorithm: ideas

a	b	a	b	b	a	y	a	d	a	a	b	a	b	c	a

a	b	c	a	b

mismatch found at the second comparison
no matter where mismatch happens, the shift is totally decided by the last character in T (b in this case)

Short Rule: Shift until having the first match of character on P with that last character

a	b	a	b	b	a	b	a	d	b	a	b	a	b	c	a

a	b	c	a	b

How to shift Pattern?

- Use the last character in text as the pivot
- Shift the pattern at least one position.
- Shift until the pivot character matches its first occurrence in the pattern, or until the pattern completely passes the pivot if no match is found.

Note: here, we use “last character in text” to refers to “text character aligned with the pattern's last character”

The Horspool's Algorithm

The task: Searching for a pattern P (such as "abcab" that has length m=5) in a text T (such as "ababyayb aabacca", having length n=16).

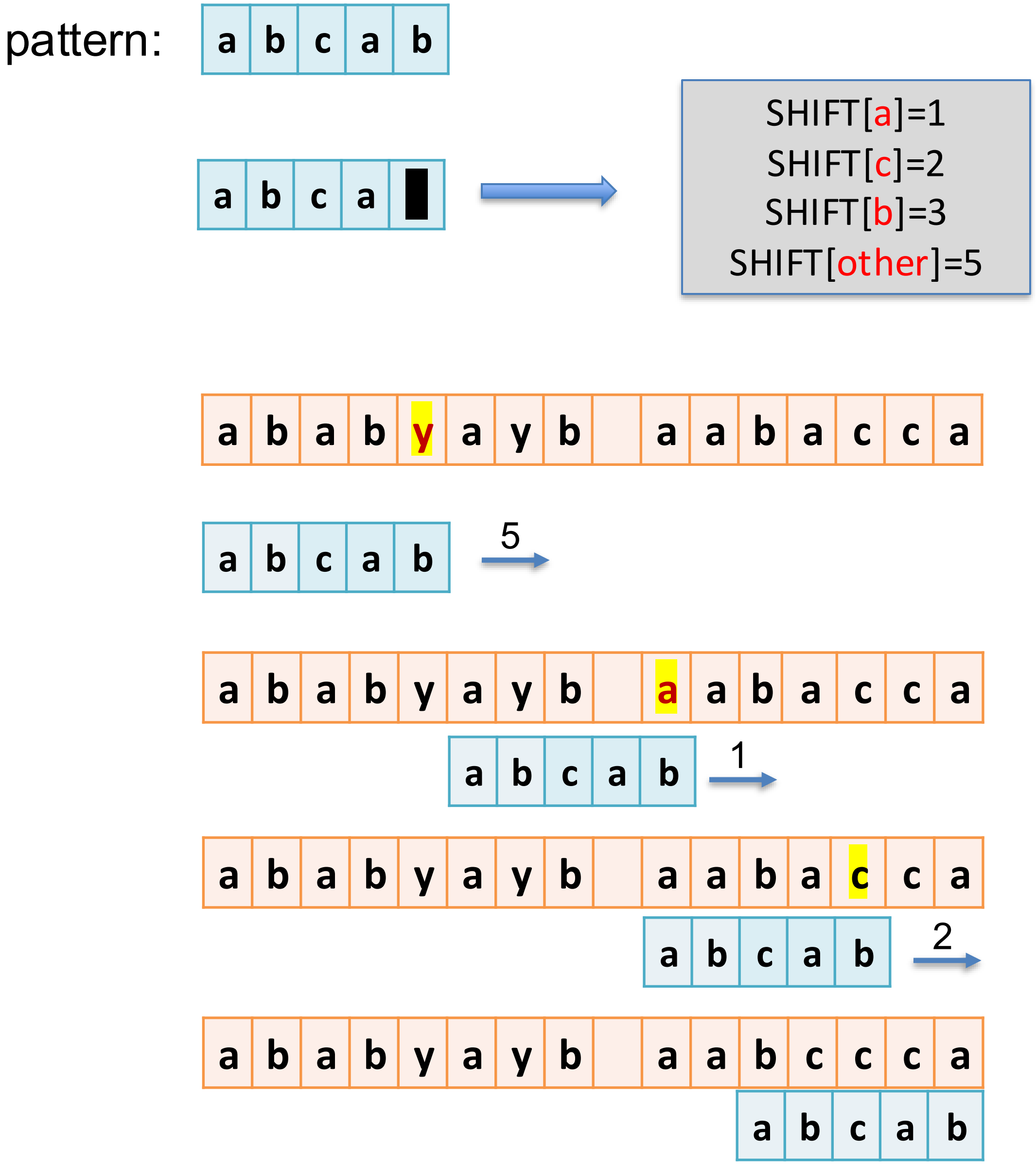
Searching for pattern P in text T

Stage 1: building SHIFT[x] for all x, by:

1. set SHIFT[x]= length(P) for all x, then
2. for each x in P, *except for the last one*: SHIFT(x)= distance from the last appearance of x to the end of P

Stage 2: searching

1. align P to the left of T
2. compare characters *backwardly* from the last character of P until the start or until finding a mismatch:
3. if no mismatch found: return solution
4. otherwise, let e be the last character in T, shift pattern SHIFT[e] positions, then back to step 2



Peer Activity: Horspool's Algorithm

Using Horspool's algorithm, how many character comparisons are needed to search for the pattern SHRIKE in the text BURMESE-SHRIKE ?

A) 3
B) 8
C) 11
D) 9

Other class exercises with Horspool's Algorithm

1. Search for HAT in BIRDWATCHER
2. Build the shift dictionary for patterns:
   a) ABCD
   b) ABBA

Horspool's Algorithm: [group/individual] exercies

Q12.1: Use Horspool's algorithm to search for the pattern `GORE` in the string `ALGORITHM`

`ALGORITHM`

`GORE`

Q12.2: How many character comparisons will be made by Hor-spool's algorithm in searching for each of the following patterns it the binary text of one million zeros?

`T= 000000000..0`

`a) P= 01001`

`b) P= 00010`

`b) P= 01111`

Q12.3: Using Horspool's method to search in a text of length n for a pattern of length m, what does a worst-case example look like?

[Check soln] Exercises with Horspool's Algorithm

Q12.1: Use Horspool's algorithm to search for the pattern `GORE` in the string `ALGORITHM`

```
ALGORITHM                  SHIFT value: R:1   O:2    G:3   others:4
GORE          1 comparison, shift 2
  GORE        1 comparison, shift 4
              done, NOTFOUND
```

Q12.2: How many character comparisons will be made by Horspool's algorithm in searching for each of the following patterns it the binary text of one million zeros?

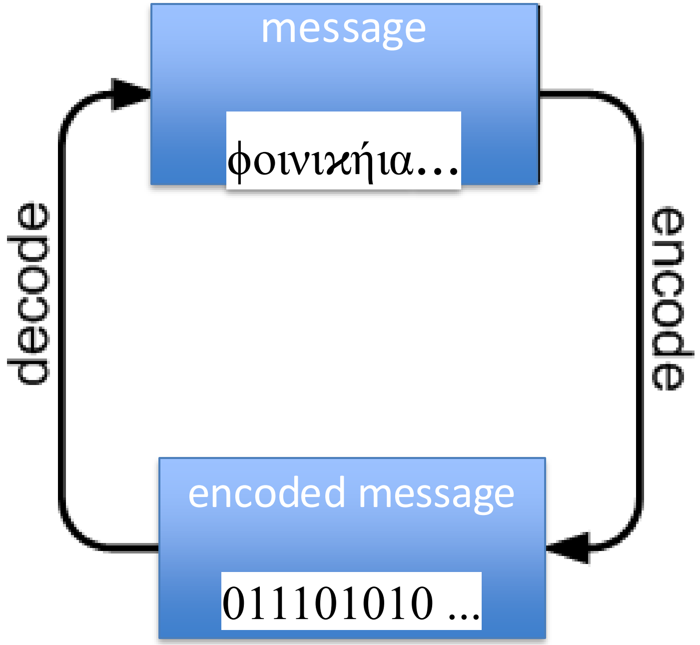
```
  (a) 01001      (b) 00010       (c) 01111
T = 000000000..0
a)  999,996     (1 comparison, shift 1)
b)  999,996     (2 comparison, shift 2)
c)  249,999     (1 comparison, shift 4)
```

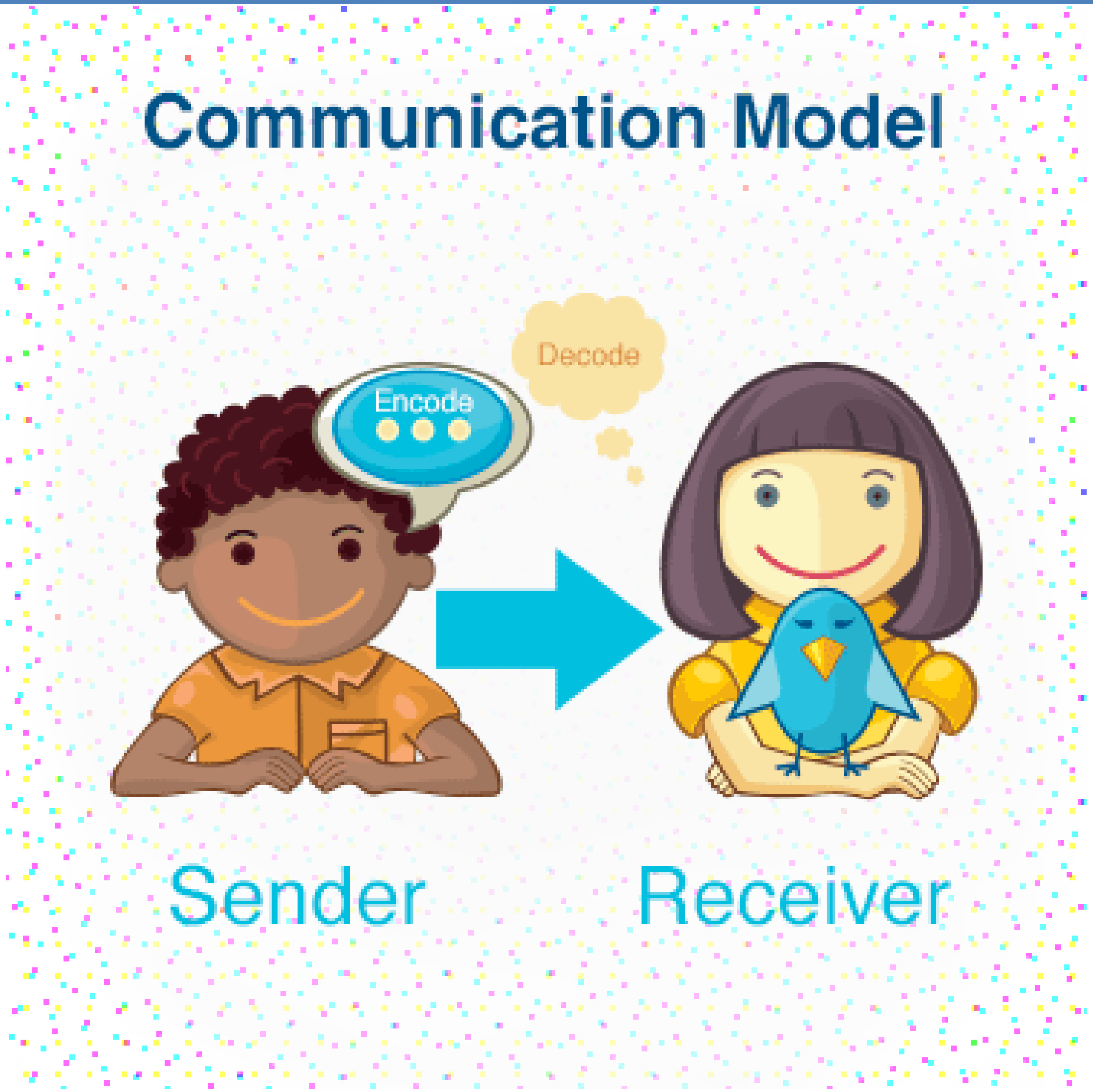
Q12.3: Using Horspool's method to search in a text of length n for a pattern of length m, what does a worst-case example look like?

text: 1111111111....1111111

pattern: 0111

Coding: used for storage, communication, compression, …

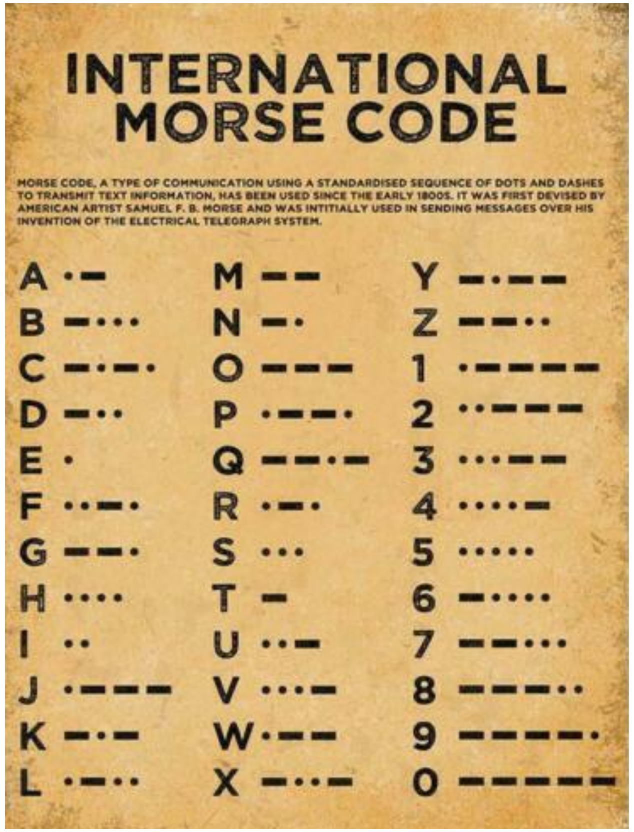




Compression: the encoded message normally has smaller size (in bits)

Encoding/Compressing: each symbol of the message is replaced with a bit pattern called its codeword
Decoding/De-compressing: encoded message is converted to the original message using codeword table

Symbol	ASCII codeword
A	01000001
B	01000010
C	01000011
D	01000100
E	01000101
F	01000110
G	01000111
H	01001000
…	



fixed-length encoding

AB → 0100000101000010

variable-length encoding

AB → 

codeword length	same bit-length for all codewords	shorter codewords for more-frequent symbols
space-efficient?	No (unless uniform distribution)	Yes, space-efficient
random access to k-th symbol	Yes	No, need to decode previous k-1 symbols first

The task:

Input: a message T such as `that cat, that bat, that hat` over some alphabet

Output: an encoded message T' with the guarantee that T can be reproduced from T'. For example

T'= 11100011110100010111…

ASCII code: each letter is replaced by a 8-bit codeword → need 224 bits for the above T

Principle of Data Compression: Use less number of bits (shorter codeword) for symbol that appears more frequently → variable-length encoding

Encoding (Compressing)

Input message: `that cat, that bat, that hat`

1. Gather Statistics: build table of frequencies, aka weight table:

a	b	c	h	t	,	_
6	1	1	4	9	2	5

2. Coding: build the *code table*

   - various methods, such as Huffman coding

3. Encoding: replace each symbol of the message by its codeword

Huffman Coding = a method for building *minimum-redundancy prefix-free* code

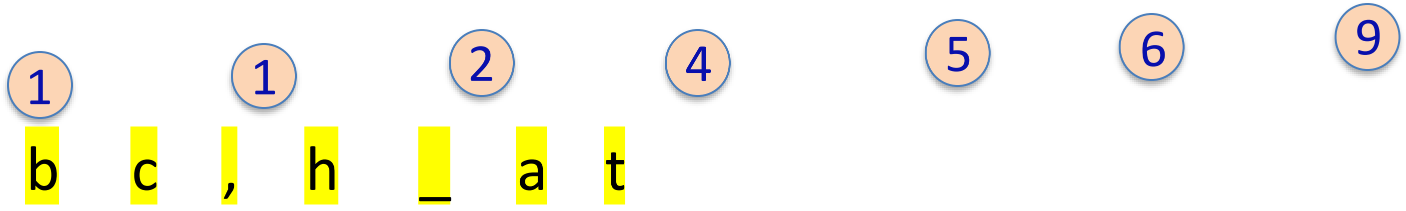
Building Huffman code = building a binary tree in *bottom-up* manner:

- **Create a node for each weight.**
- Repeatedly join the two smallest weights, forming a parent node with their summed weight, until only a single root remains.
- Assign a zero-bit to one edge and a one-bit to the other for each node.

Input message: `that_cat,_that_bat,_that_hat`

Table of frequencies:

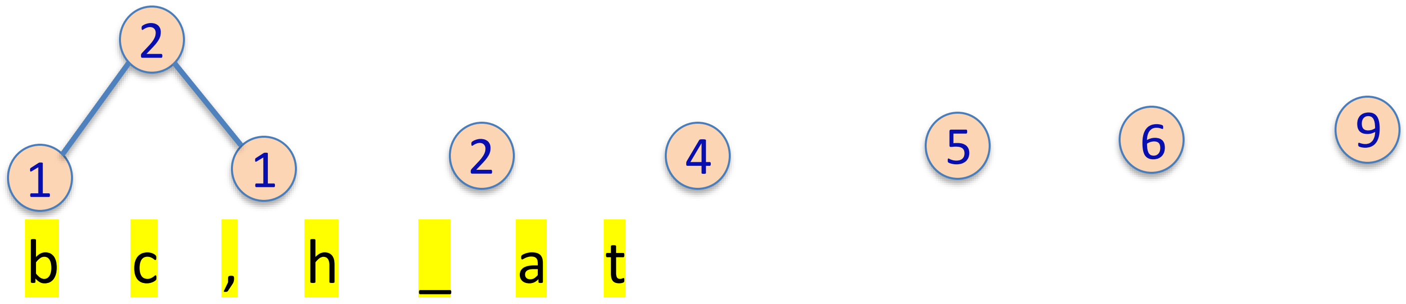
a	b	c	h	t	,	_
6	1	1	4	9	2	5



Huffman Coding = a method for building *minimum-redundancy prefix-free* code

Building Huffman code = building a binary tree in *bottom-up* manner:

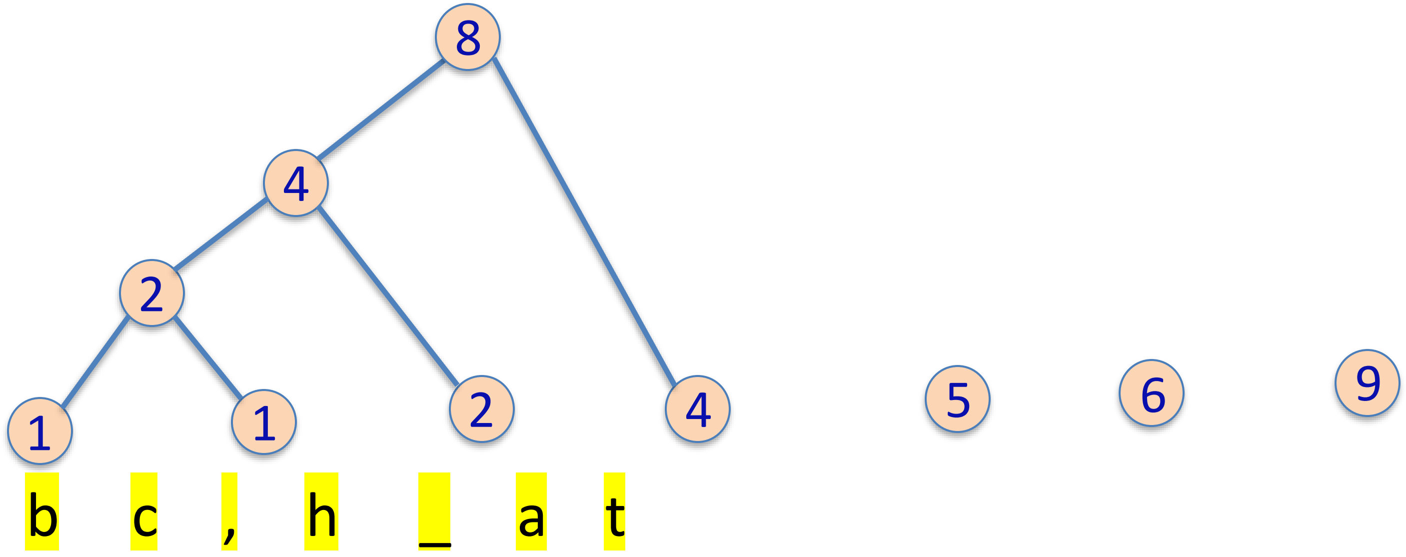
- Create a node for each weight.
- **Repeatedly join the two smallest weights, forming a parent node with their summed weight, until only a single root remains.**
- Assign a zero-bit to one edge and a one-bit to the other for each node.



Huffman Coding = a method for building *minimum-redundancy prefix-free* code

Building Huffman code =
building a binary tree in
bottom-up manner:

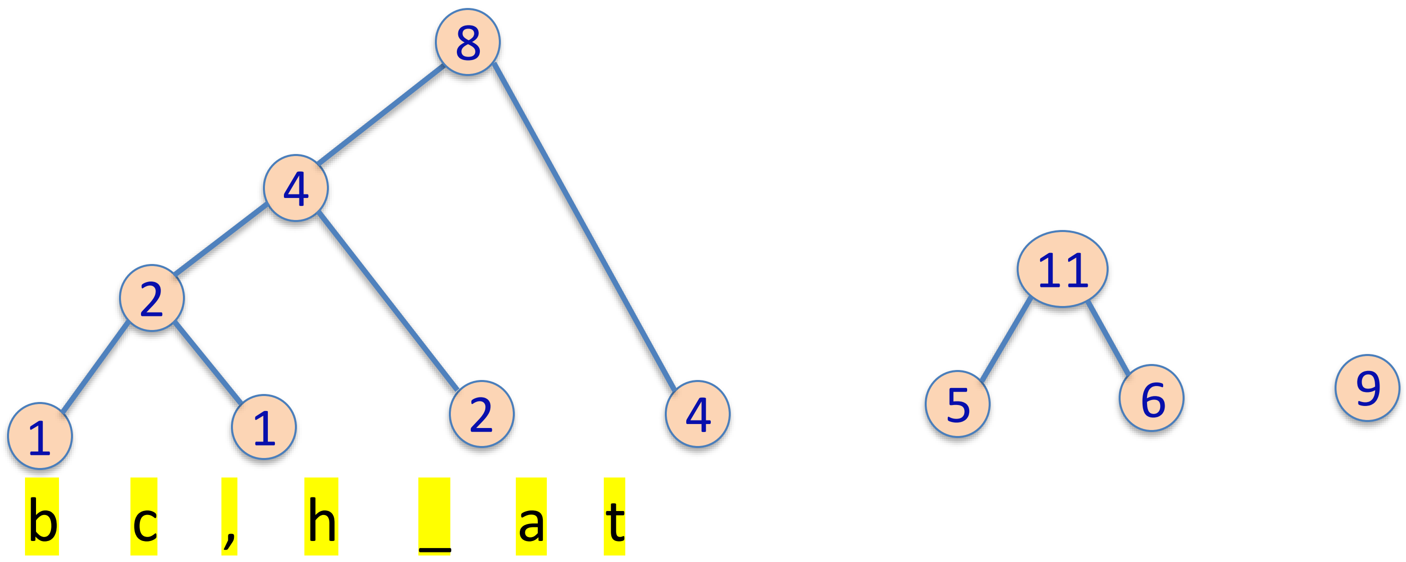
- Create a node for each weight.
- **Repeatedly join the two smallest weights, forming a parent node with their summed weight, until only a single root remains.**
- Assign a zero-bit to one edge and a one-bit to the other for each node.



Huffman Coding = a method for building *minimum-redundancy prefix-free* code

Building Huffman code = building a binary tree in *bottom-up* manner:

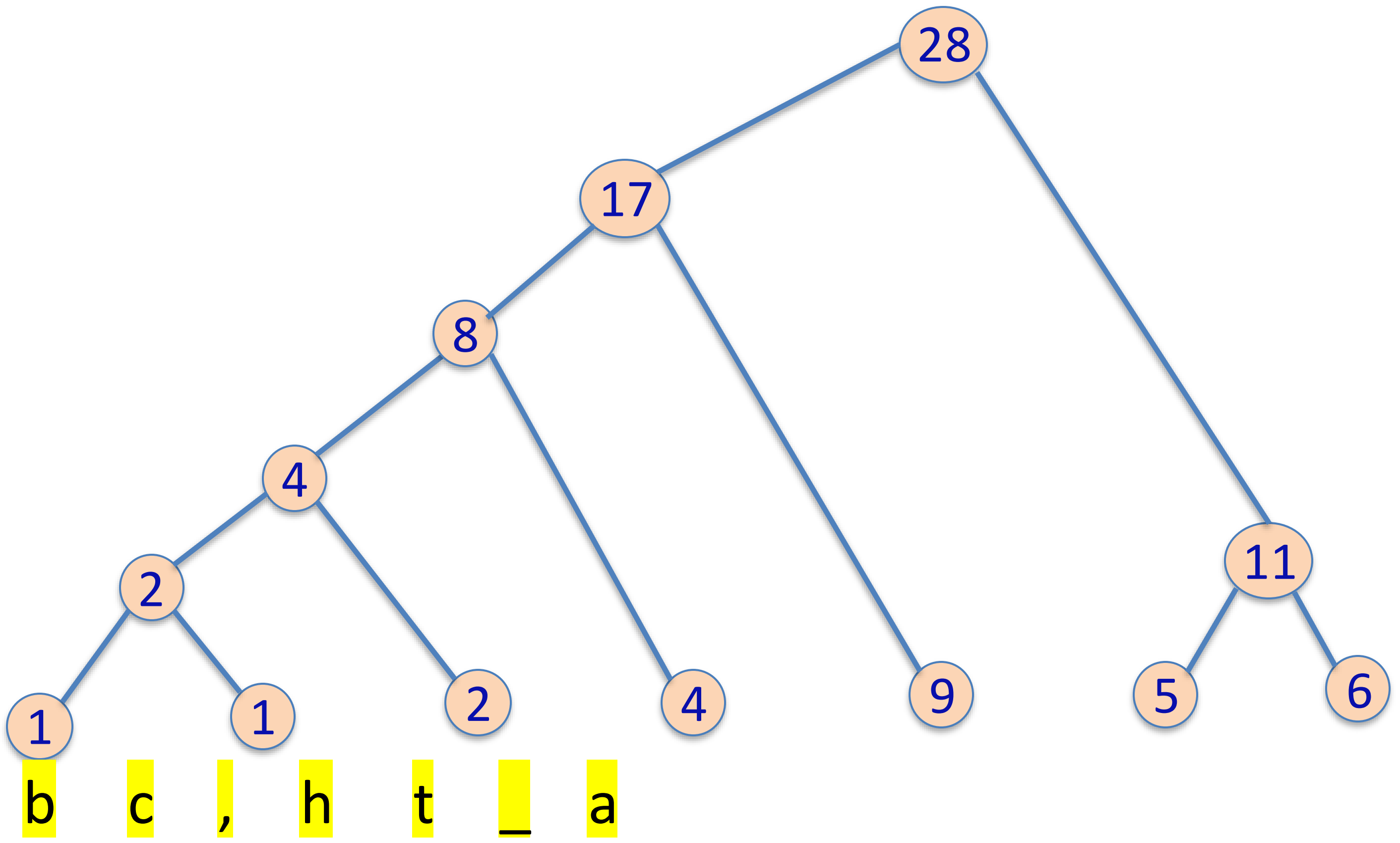
- Create a node for each weight.
- **Repeatedly join the two smallest weights, forming a parent node with their summed weight, until only a single root remains.**
- Assign a zero-bit to one edge and a one-bit to the other for each node.



Huffman Coding = a method for building *minimum-redundancy prefix-free* code

Building Huffman code = building a binary tree in *bottom-up* manner:

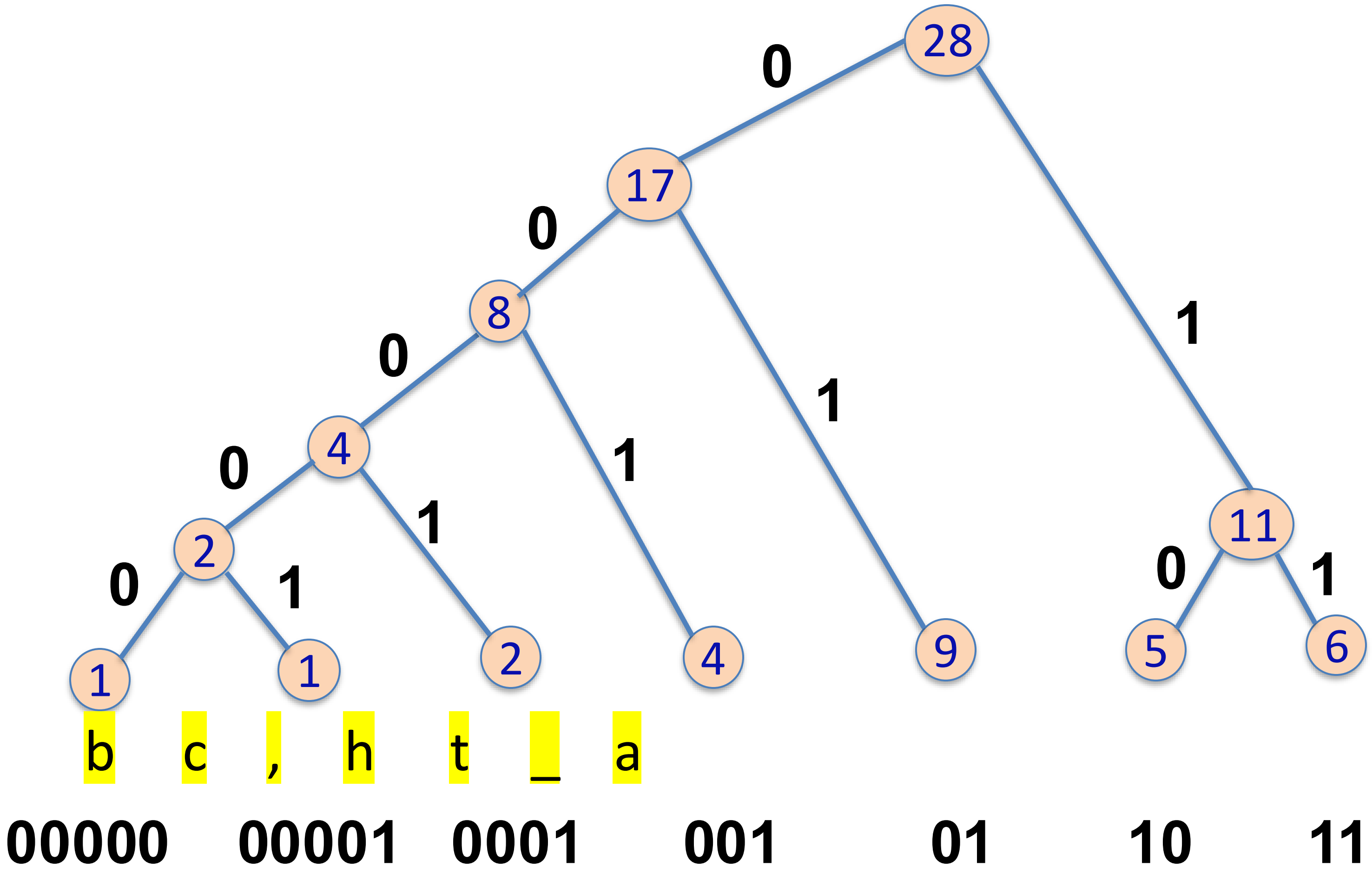
- Create a node for each weight.
- **Repeatedly join the two smallest weights, forming a parent node with their summed weight, until only a single root remains.**
- Assign a zero-bit to one edge and a one-bit to the other for each node.



Huffman Coding = a method for building *minimum-redundancy prefix-free* code

Building Huffman code = building a binary tree in *bottom-up* manner:

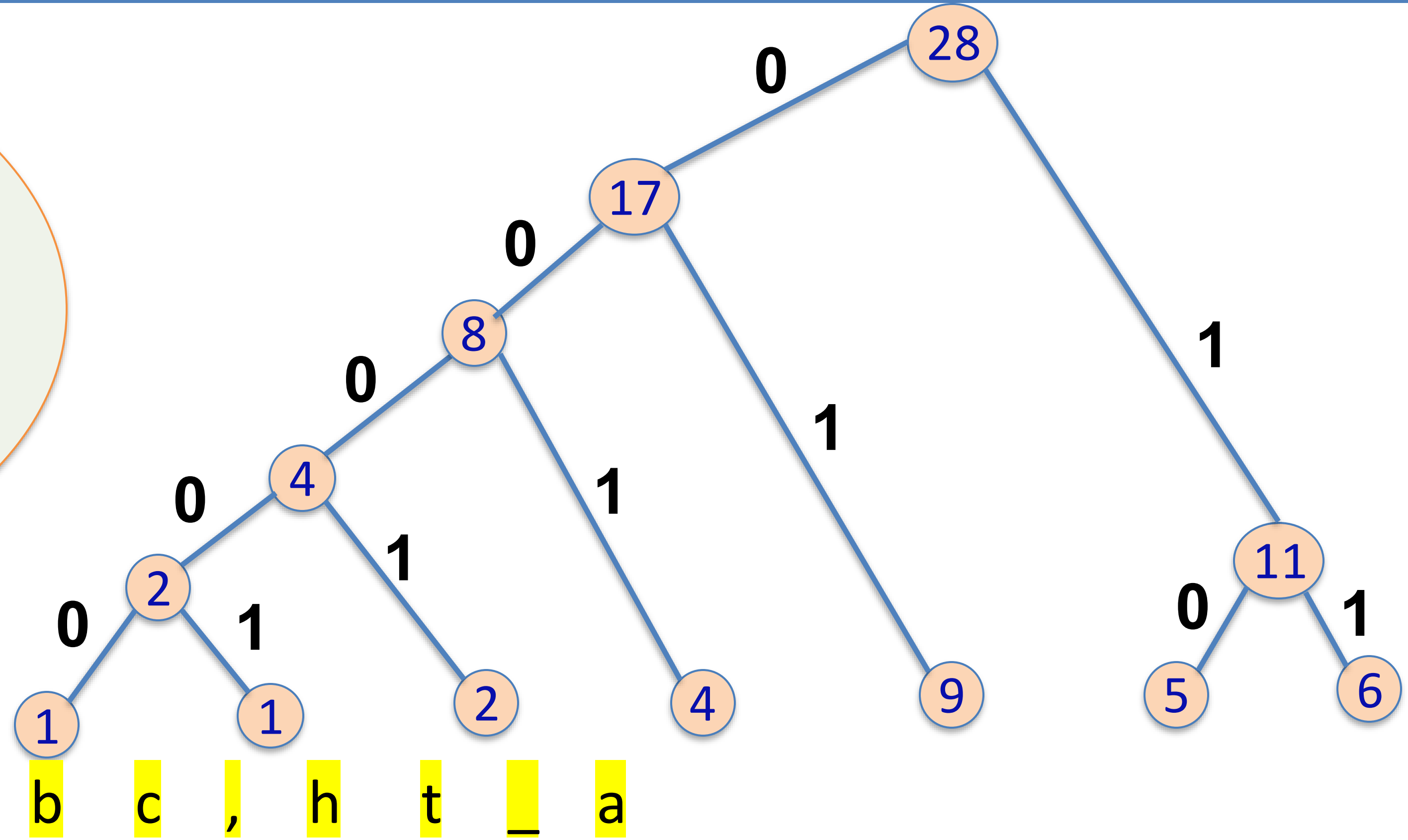
- Create a node for each weight.
- Repeatedly join the two smallest weights, forming a parent node with their summed weight, until only a single root remains.
- **Assign a zero-bit to one edge and a one-bit to the other for each node.**



Huffman Coding = a method for building *minimum-redundancy prefix-free* code

this code is prefix-free:
no codeword is a prefix
of another codeword

[so, decoding is
possible]



codewords: **00000** **00001** **0001** **001** **01** **10** **11**

`that cat, that bat, that hat` is encoded as:
01 001 11 01 10 00001 11 01 0001 ....
Total encoded length= 1*5 + 1*5 + 2*4 + 4*3 + 9*2 + 5*2 + 6*2 = 70 bits

Canonical Huffman Coding

→ There are different versions of Huffman code for a same weight table, all we need to do is to choose a way and keep consistency.

example rules for canonical Huffman's coding:

- when joining 2 weights into one, always make the smaller weight be the left child (hence, need to always keeps current roots in weight ordering)
- is same weight: simpler trees first + use alphabetical order
- when assigning code, always set 0 to the left edge, 1 to the right edge

[group/individual]Q 12.4 & Q12.5: Huffman Coding

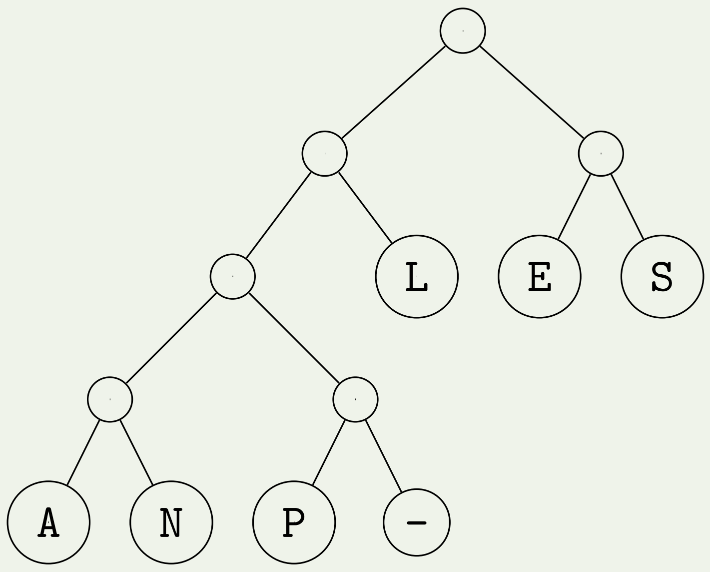
Q12.4: Using Huffman coding to generate two code trees based on for the message:

losslesscodes

What is the total length of the compressed message using the Huffman code?

Q12.5: The code tree was generated using Huffman's algorithm, and converted into a Canonical Huffman code tree.
Note: _ denotes space.

Assign codewords to the symbols in the tree, such that left branches are denoted 0 and right branches are denoted 1.



Use the resulting code to decompress the following message:
00100110000011100011011011110011110110100010011011110001101111

Check Q 12.4: Huffman Code Generation

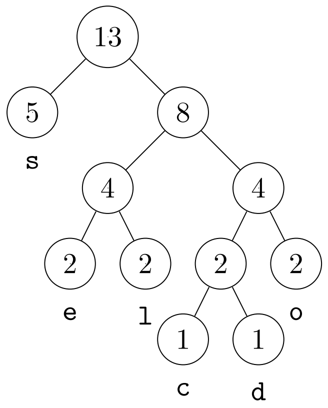
`losslesscodes`

Your solution: one tree could be

Frequency counts:

s	l	o	e	c	d
5	2	2	2	1	1

$\Longrightarrow$



$\Longrightarrow$

s	0
l	101
o	111
e	100
c	1100
d	1101

Hence, the encoded version of `losslesscodes` is:

101 111 0 0 101 100 0 0 1100 111 1101 100 0

Note:

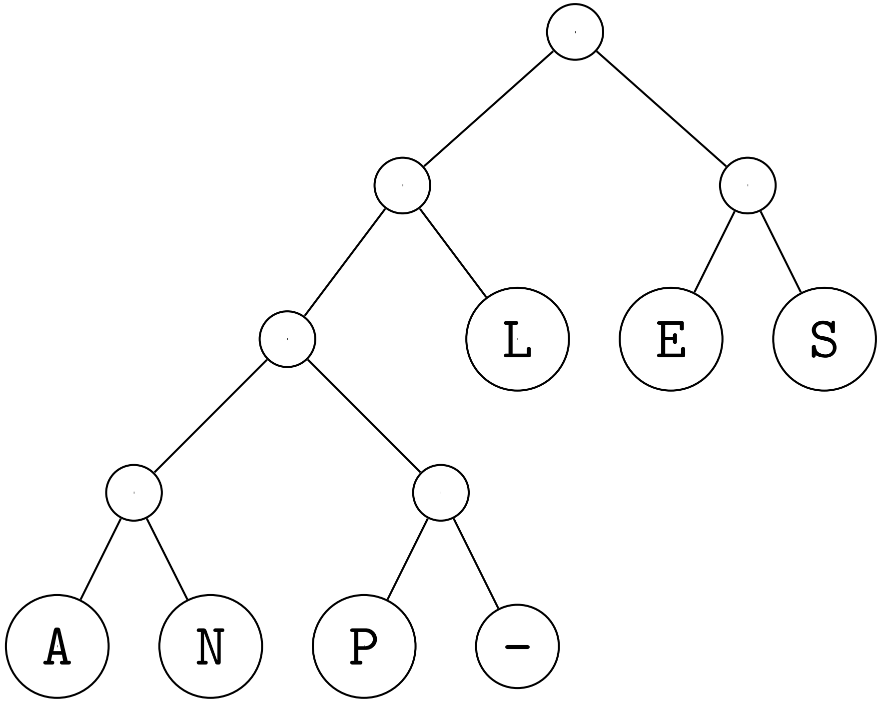
- give another tree
- other solutions are possible, but message length is always 31
- ➔ a need for a standard: canonical Huffman code

Check Q 12.5: Canonical Huffman decoding

The code tree was generated using Huffman's algorithm, and converted into a Canonical Huffman code tree. Note: _ denotes space.

Assign codewords to the symbols in the tree, such that left branches are denoted 0 and right branches are denoted 1.
Use the resulting code to decompress the following message:

symbol	codeword	length
A	0000	4
N	0001	4
P	0010	4
_	0011	4
L	01	2
E	10	2
S	11	2



001001100000111000110110111100111110110100010011011110001101111

Your soln: PLEASE LESS SLEEPLESSNESS

Subject Feedback

- There are two FEIT Subject Surveys available via your email:
  - One addresses workshops.
  - One addresses subject content.
- Please take some time now to provide honest, constructive feedback so that we can improve.

LAB: Revision exercises OR programming exercises

Q 12.8: Quicksort & Mergesort

Given the array:

[3, 8, 5, 2, 1, 3, 5, 4, 8]

a. Perform a single *Hoare Partition* on the array, taking the first element as the pivot

b. Perform Quicksort on the array. You may use whatever partitioning strategy you like (*i.e.*, you don't need to follow a particular algorithm).

c. Perform Mergesort on the array

Q 12.6: For each of the following cases, indicate whether $f(n)$ is $O(g(n))$, or $\Omega(g(n))$, or both (that is, $\boldsymbol{P}(g(n))$)

(a) $f(n) = (n^3+1)^6$ and $g(n) = (n^6+1)^3$,

(b) $f(n) = 3^{3n}$ and $g(n) = 3^{2n}$,

(c) $f(n) = \sqrt{n}$ and $g(n) = 10n^{0.4}$,

(d) $f(n) = 2\log_2\{(n+50)^5\}$ and $g(n) = (\log_e(n))^3$,

(e) $f(n) = (n^2+3)!$ and $g(n) = (2n+3)!$,

(f) $f(n) = \sqrt{n^5}$ and $g(n) = n^3 + 20n^2$.

Q 12.7: Solve the following recurrence relations. Give both a closed form expression in terms of n and a Big-Theta bound.

a) $T(n)= T(n/2)+1,\ T(1)= 1$

b) $T(n)= T(n-1) + n/5,\ T(0)= 0$

Algorithms are fun!

Thank You!

&

Good Luck!