COMP20007 Workshop Week 9

	Preparation: - *have draft papers and pen ready* - open `ws10.pptx`/pdf from `github.com/anhvir/c207` - open `wokshop10.pdf` (from `LMS`), and - download lab files from `LMS`
1	Hashing: Problems T1, T2
2	Huffman Coding: Problems T3, T4
3	Revision on demands: Complexity (problem T5) Solving Recurrences and others
LAB	Lab: playing with hashing code

Hashing

- What? Why? How?
- Collision, separate chaining, open addressing.

T1: Separate chaining

Consider a hash table in which the elements inserted into each slot are stored in a linked list. The table has a fixed number of slots `L=2`. The hash function to be used is

`h(k)=k mod L`.

a) Show the hash table after insertion of records with the keys
`17 6 11 21 12 33 5 23 1 8 9`

b) Can you think of a better data structure to use for storing the records that overflow each slot?

T2: Open addressing

Consider a hash table in which each slot can hold one record and additional records are stored elsewhere in the table using linear probing with steps of size `i=1`. The table has a fixed number of slots `L=8`. The hash function to be used is `h(k)=k mod L`.

a) Show the hash table after insertion of records with the keys
`17 7 11 33 12 18 9`

b) Repeat using linear probing with steps of size `i = 2`. What problem arises, and what constraints can we place on `i` and `L` to prevent it?

c) Can you think of a better way to find somewhere else in the table to store overflows?

T3,T4: Huffman Coding

Huffman Coding, encoding and decoding

Build Huffman code for [a:3, b:4, c:2, d:4, e:7]

T3,T4: Huffman Coding

Build Huffman code for [a:3, b:4, c:2, d:4, e:7]

Note: there are different versions of Huffman code, all we need to do is to choose a way and keep consistency. For instance:

- maintaining current weight/frequency table in increasing order, taking into account the alphabetic order
- always choosing the first 2 smallest weights to join
- when assigning code, always set 0 to the left, 1 to the right edge

T3: Huffman Code Generation

Huffman's Algorithm generates prefix-free code trees for a given set of symbol frequencies. Using these algorithms generate two code trees based on the frequencies in the following message:

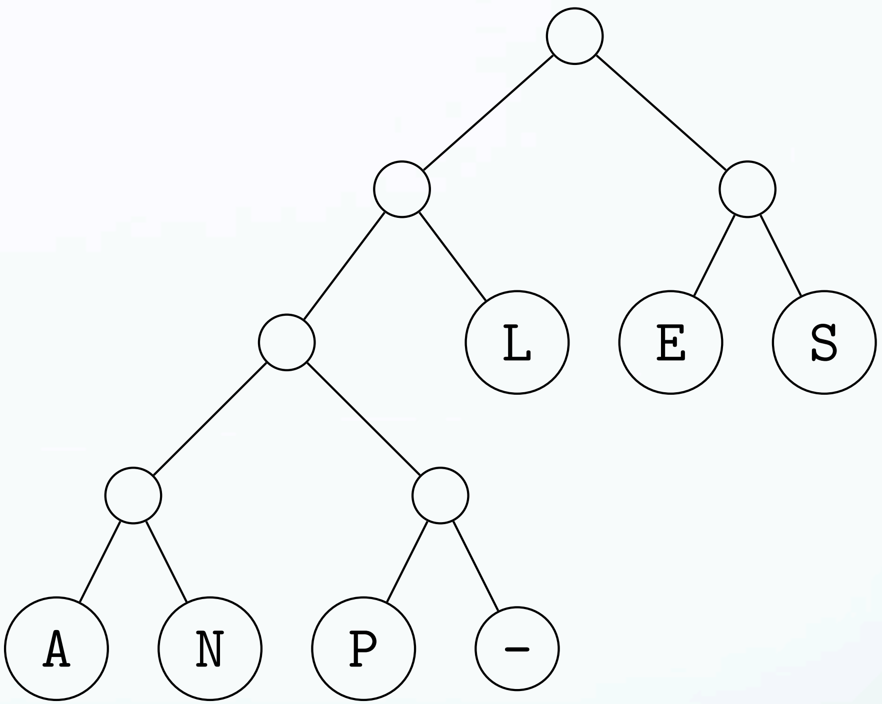
`losslesscodes`

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

T4: 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:



001001100000011100011011011110011110110100010011011110001101111

Revision 1: Complexity Analysis – `03.pdf` & `04.pdf`

$1 \prec \log n \prec n^{\varepsilon} \prec n^{c} \prec n^{\log n} \prec c^{n} \prec n^{n}$ where $0 < \varepsilon < 1 < c$

$(\log n)^{\alpha} \prec (\log n)^{\beta}$ and $n^{\alpha} \prec n^{\beta}$ where $0 < \alpha < \beta$

$O(\, f(n) + g(n)\,) = O(\, \max\{f(n), g(n)\}\,)$
$O(c\, f(n)) = O(\, f(n)\,)$
$O(\, f(n) \times g(n)\,) = O(f(n)) \times O(g(n))$

note: these 3 also applied to big-Θ

$1+2+ \ldots + n = n(n+1)/2 = \Theta(n^2)$
$1^2 + 2^2 + \ldots + n^2 = n(n+1)(2n+1)/6 = \Theta(n^3)$
$1 + x + x^2 + \ldots + x^n = (x^{n+1}-1)/(x-1) \quad (x \neq 1)$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \begin{cases} 0 & f(n) = O(g(n)) \\ c & f(n) = \Theta(\,g(n)\,) \\ \infty & f(n) = \Omega(g(n)) \end{cases}$$

$$\lim_{n\to\infty} \frac{t(n)}{g(n)} = \lim_{n\to\infty} \frac{t'(n)}{g'(n)}$$

R1 exercises: Problem T5

For each of the following cases, indicate whether $f(n)$ is $O(g(n))$, or $\Omega(g(n))$, or both (that is, $\Theta(g(n))$)

(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 $g(n) = n^3 + 20n^2$.

R2: Recurrences T(n)= ? T(<n) + $f(n)$ and T(1)=c

- Apply the Master Theorem (`10.pdf`) if possible (ie. if having a, b, and $\boldsymbol{\Theta}(n^d)$ or $O(n^d)$)
- Otherwise, using substitution to expand until T(1)

Note that we can easily make mistakes with substitution. *Do it step by step using draft paper, don't rush.*

Exercises: Supposing T(1)=1, solve the following using substitution *and* using master theorem when possible:

a) $T(n)= 3T(n\text{-}1) + 1$

b) $T(n)= T(n/3) +1$

R3: `05.pdf`: exhaustive string search, knapsack

- exhaustive string search = naïve search
- exhaustive knapsack = find all subsets of a set

Exercises:

R4: graphs

- `06.pdf`: graph concepts
- `07.pdf`: DFS and BFS, topological sort
- `08.pdf` : Prim & Dijkstra

Exercises:

R5: Sorting algorithms

- `11.pdf`: Sorting algorithm properties, *insertion sort* & *selection sort*
- `12.pdf`: *top-down mergesort*, *quicksort with Lomuto partitioning*, *quicksort with Hoare partitioning*

Exercises: For each of the above 5 algorithms:

- is it input-sensitive, in-place, stable? What's the complexity? Best case and worst case?
- Show how it works on:

`EXAMPLE`

R6: Binary Heap (`13.pdf`)

- Binary Heap: complexity of insertion, removeMin, heapify
- Heap Sort and its complexity

Exercises:

- Show how to insert into an originally-empty min-heap:

EXAMPLE

R7: Search Trees (`14.pdf`)

- BST and AVL
- 2-3 Tree

Exercises: For each of the above 2 types of search trees:

- What is the complexity of insertion, of search?
- Perform the insertion into originally-empty tree:

  `TREBALNCD`

R8: Hashing

- `15.pdf`: Hashing.

Exercises:

R9: Huffman Coding

- `16.pdf`: Coding and Huffman Coding

Exercises:

Anything missing in our list of revision?

Of course, there is no guarantee that the list is complete. You can fill in things like:

- stacks and queues, priority queues