Tutorial

1. 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 $h(k) = k \mod L$.

Show the hash table after insertion of records with the keys

$$17 \quad 6 \quad 11 \quad 21 \quad 12 \quad 33 \quad 5 \quad 23 \quad 1 \quad 8 \quad 9$$

Can you think of a better data structure to use for storing the records in each slot?

2. 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 $h(k) = k \mod L$.

Show the hash table after insertion of records with the keys

$$17 \quad 7 \quad 11 \quad 33 \quad 12 \quad 18 \quad 9$$

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?

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

3. Huffman code generation

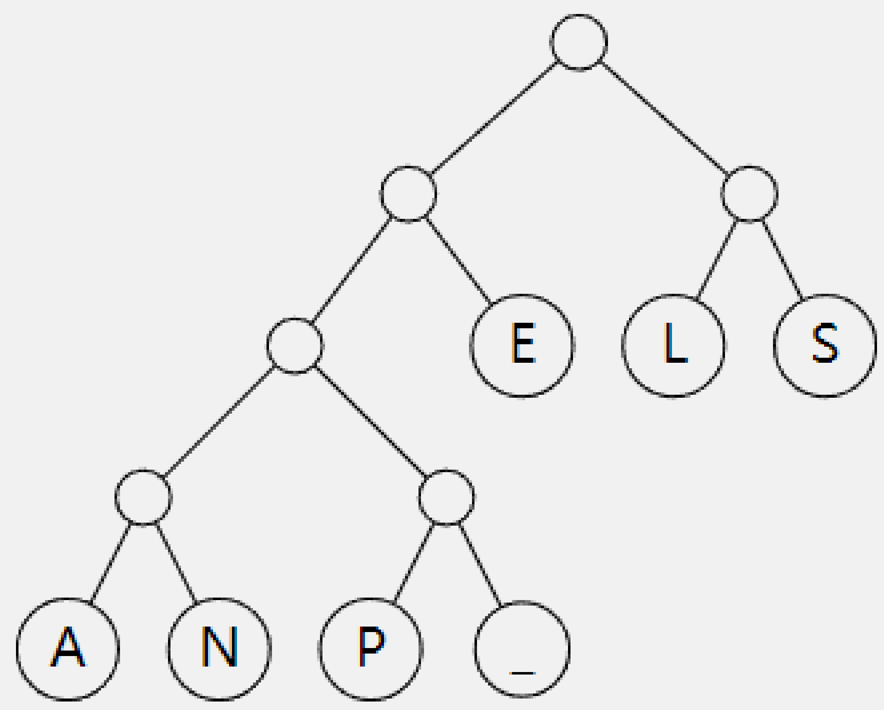
Huffman's Algorithm generates prefix-free code trees for a given set of symbol frequencies. Using these algorithms generate two code trees based on the frequencies in the following message:

`losslesscodes`

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

4. Canonical Huffman decoding

The following 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:

`00101001000011010011100111110011111001010010100111110001011111`

5. Asymptotic Complexity Classes (Revision)

For each pair of the following functions, indicate whether $f(n) \in \Omega(g(n))$, $f(n) \in O(g(n))$ or both (in which case $f(n) \in \Theta(g(n))$).

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

The following result may be useful,

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \begin{cases} 0 & \text{implies that } f(n) \text{ has a smaller order of growth than } g(n), \\ c & \text{implies that } f(n) \text{ has the same rder of growth than } g(n), \\ \infty & \text{implies that } f(n) \text{ has a larger order of growth than } g(n). \end{cases}$$

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

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

b. Perform Quicksort on the array from (a). 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 from (a).

7. (Optional Homework) Karp-Rabin Hashing

In this question we will use Karp-Rabin hashing to solve a string search problem.

For an alphabet with a characters, and some positive number m (we usually select m to be prime, why?) the Karp-Rabin hash function for a string of n characters S="$s_0 s_1 \ldots s_{n-1}$" is given by:

$$h(S) = \sum_{i=0}^{n-1} a^{n-1-i} \cdot \mathrm{chr}(s_i) \mod m.$$

Here $\mathrm{chr}(s)$ gives the index of the character s in the alphabet, *i.e.*, $\mathrm{chr}(s) \in \{0, 1, \ldots, a-1\}$.

Consider the following example. We have the alphabet { A, B, C, D }, and as such $a = 4$. Let $m = 11$.

We are going to search for the string P="CAB" in the string T="CADACAB".

Since $|P| = 3$ you'll have to compute the hash for each substring of 3 characters in T. To start you off, consider the first subtrings $T[0 \ldots 2]$="CAD",

$$\begin{aligned} h(\text{"CAD"}) &= a^2 \cdot \mathrm{chr}(\mathrm{C}) + a \cdot \mathrm{chr}(\mathrm{A}) + \mathrm{chr}(\mathrm{D}) \mod m \\ &= 4^2 \cdot 2 + 4 \cdot 0 + 3 \mod 11 \\ &= 35 \mod 11 \\ &= 2 \end{aligned}$$

Now we can, in constant time, compute the successive three characters, by using the following formula:

$$h(s_1 s_2 \ldots s_k) = a(h(s_0 s_2 \ldots s_{k-1}) - a^{k-1} \cdot \mathrm{chr}(s_0)) + \mathrm{chr}(s_k) \mod m$$

So, we can compute h("ADA") like so:

$$\begin{aligned} h(\text{"ADA"}) &= 4\left(h(\text{"CAD"}) - 4^2 \cdot 2\right) + 0 \mod m \\ &= 4(2 - 32) + 0 \mod 11 \\ &= 4(-30) + 0 \mod 11 \\ &= 4(-30 \mod 11) \mod 11 \\ &= 4(3) \mod 11 \\ &= 12 \mod 11 \\ &= 1 \end{aligned}$$

Complete the string search by computing all of the required hashes (including $h(P)$) and determining whether or not $h(P)$ matches any of the hash values for the substrings of T.

How does this enable us to search for P in T in $O(|T|+|P|)$ time? What might go wrong?