2020 Semester 1
COMP20007 Design of Algorithms
School of Computing and Information Systems
The University of Melbourne

# Assignment 2

**Due: 7:00 PM AEST Thursday 4 June**

## Introduction

This assignment consists of 2 problems: 2 programming problems and 2 written problems. You will submit your solutions for the C programming component via `dimefox submit` and the written component via the LMS.

This assignment has a total of 20 marks and will contribute 20% to your final grade for this subject.

## Programming Problems

### Problem 1 (a)

**3 Marks**

You must create a C program which implements a string hash function, reads in a file containing $N$ strings and outputs the hash value for each string.

The strings will be between 1 and 256 characters long and will contain lowercase letters, uppercase letters and digits.

Each character will be mapped to a 6 digit binary string according to the following function:

$$\begin{aligned} \text{chr}(\texttt{a}) = 0 \quad \text{chr}(\texttt{b}) = 1 \quad \cdots \quad \text{chr}(\texttt{z}) = 25 \\ \text{chr}(\texttt{A}) = 26 \quad \text{chr}(\texttt{B}) = 27 \quad \cdots \quad \text{chr}(\texttt{Z}) = 51 \\ \text{chr}(\texttt{0}) = 52 \quad \text{chr}(\texttt{1}) = 53 \quad \cdots \quad \text{chr}(\texttt{9}) = 61 \end{aligned}$$

For example `A` maps to the binary string 011010.

The hash value for a string is computed by concatenating the binary strings for each of the characters which make up the string, and taking this *mod* $M$, for a given $M$.

For example if $M = 17$ the string `"pub"` would hash to 8. Since,

$$\text{chr}(\texttt{p}) = 15, \quad \text{chr}(\texttt{u}) = 20, \quad \text{chr}(\texttt{b}) = 1,$$

we have the concatenated binary string,

001111 010100 000001

This binary number represents the decimal number 62721. Taking 62721 mod 17 yields 8. We say that $h_{17}(\texttt{"pub"}) = 8$.

In general, if $S = c_0 c_1 \cdots c_{\ell-1}$ is a string of $\ell$ characters then the hash value $h_M(S)$ is given by,

$$h_M(S) = \sum_{i=0}^{\ell-1} \text{chr}(c_i) \times 64^{\ell-1-i} \mod M$$

We use 64 here because each binary string we're concatenating has length 6 and $2^6 = 64$.

Your program will be given input via `stdin`, where the first line of input contains the two integers $N$ and $M$ and the next $N$ lines contain each of the $N$ strings to be hashed (be careful that your program doesn't include the newline character in the string).

For example the input file `tests/p1a-in-1.txt` contains:

```
4 17
pub
Dijkstra
AB
comp20007
```

Your program should output the hash values of each of these strings, one per line. *E.g.,*

```
$ ./a2 p1a < tests/p1a-in-1.txt
8
16
8
13
```

**Hint:** As you can see, these numbers are going to get very big very quickly since we can have strings of up to 256 characters long. These numbers probably won't fit into a C `int`. You should use *Horner's Rule* to deal with this problem.

## Problem 1 (b)

**3 Marks**

In this problem you must implement a hash table with initial size $M$ capable of storing strings using the hash function $h_M$ described in *Problem 1 (a)*.

Your program will again be given $N$ strings which should be inserted (in the given order) into the hash table.

Collisions should be handled using linear probing with a step size of $K$.

If an element cannot be inserted into the table then the size of the hash table must be doubled (*i.e.,* $M_{\text{new}} = 2M_{\text{old}}$) and the strings already in the hash table must be rehashed and inserted (in the order in which they appear in the hash table). Finally the string which initially could not be inserted should be hashed and inserted into the larger hash table.

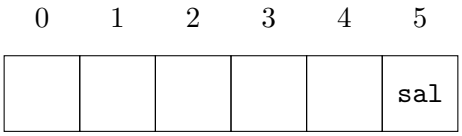
Your program will receive input from `stdin`, where the first line will contain the three integers $N, M$ and $K$ and the next $N$ lines will contain the $N$ strings to be inserted into the new larger hash table. Any memory associated with the hash table before the size was increased should be correctly freed.

Consider as an example the following input:

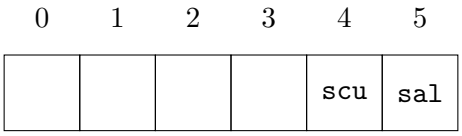
```
5 6 3
sal
scu
ba
cam
wam
```

So we will have $N = 5$ strings to insert into a hash table which initially has $M = 6$ slots, and we will use linear probing with step size $K = 3$.

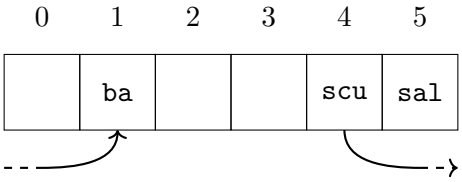
The first string `sal` has a hash value of $h_6(\texttt{sal}) = 5$ so it is inserted into index 5:



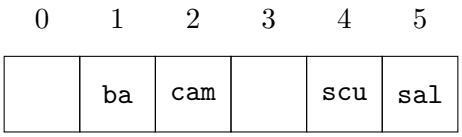

We then insert `scu` with $h_6(\texttt{scu}) = 4$ into index 4:



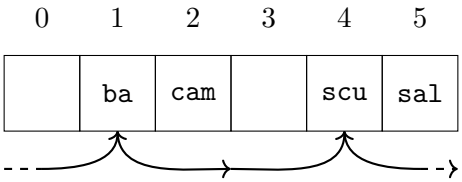

Now the hash value of `ba` is also $h_6(\texttt{ba}) = 4$, so there is a collision. Since the step size is 3 we will try to insert this string into index $(h_6(\texttt{ba}) + K) \mod M = (4 + 3) \mod 6 = 1$:




The string `cam` with $h_6(\texttt{cam}) = 2$ is then inserted:




We then try to insert `wam` with hash value $h_6(\texttt{wam}) = 4$, however we get a collision at both viable indices 4 and 1:

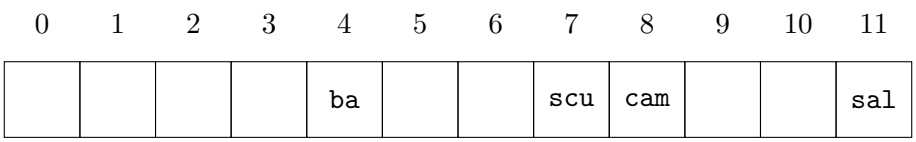



At this step we fail to insert `wam` into the hash table. So we increase the size of the hash table from 6 to 12 and rehash and re-insert all the existing elements (we do this in order of position in the existing hash table, *i.e.*, `ba`, `cam`, `scu`, and then `sal`).

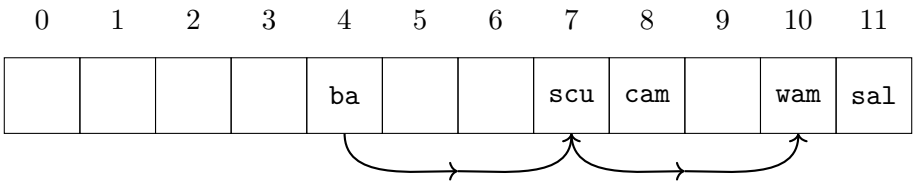
The new hash values are:

$$h_{12}(\texttt{ba}) = 4 \quad h_{12}(\texttt{cam}) = 8 \quad h_{12}(\texttt{scu}) = 4 \quad h_{12}(\texttt{sal}) = 11 \quad \text{and} \quad h_{12}(\texttt{wam}) = 4$$

So after reinserting all the existing elements the hash table looks like:



Then after inserting `wam` again we get:




The program must output the final state of the hash table with one line per slot. In this example the output should be:

```
$ ./a2 p1b < tests/p1b-in-2.txt
0:
1:
2:
3:
4: ba
5:
6:
7: scu
8: cam
9:
10: wam
11: sal
```

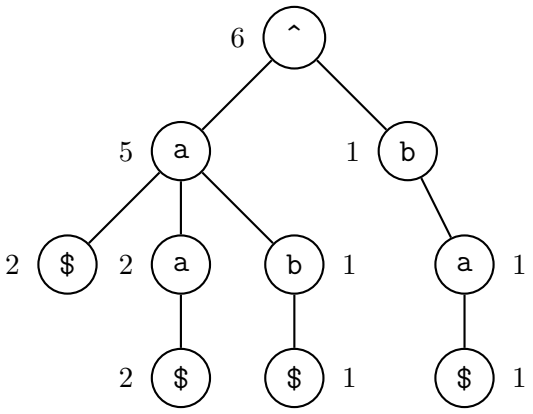
## Problem 2 (a)

**3 Marks**

For this problem you will create a *character level trie*. A *trie* is a tree where each node represents a character in a word (or a `"^"` or `"$"` which represent the start and the end of a word respectively).

Each path from the root to a leaf in a trie corresponds to a word in the data set which the trie represents. Each node has a corresponding frequency.

Consider the following trie as an example:




This trie represents the set of strings `"a"`, `"aa"`, `"ab"` and `"ba"` with frequencies 2, 2, 1 and 1 respectively. The frequencies associated with leaf nodes (note that leaf nodes are always `$` nodes) reflect the frequencies of the words ending at those leaf nodes.

On the other hand, the frequencies of the intermediate nodes reflect the frequencies of the corresponding prefix. For example the `a` node on the first layer indicates that there are 5 strings which start with `"a"`. What is important here is the path taken from the root to the node. If we are looking at the first `a` node on the second layer the path from the root is `^, a, a` and so the frequency of 2 indicates that there are two strings which begin with `"aa"`.