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:

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

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 ℓ characters then the hash value $h_M(S)$ is given by,

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

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 $K = 3$.

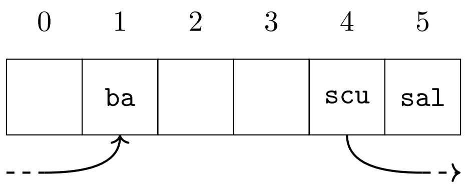
The first string `sal` has a hash value of $h_6(\texttt{sal}) = 5$ so it is inserted into index 5:

0	1	2	3	4	5
					`sal`

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

0	1	2	3	4	5
				`scu`	`sal`

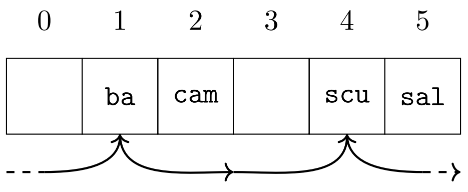
Now the hash value of `ba` is also $h_6(\texttt{ba}) = 4$, so there is a collision. Since the step size is 3 we will try to insert this string into index $(h_6(\texttt{ba}) + K) \mod M = (4 + 3) \mod 6 = 1$:



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

0	1	2	3	4	5
	`ba`	`cam`		`scu`	`sal`

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`).

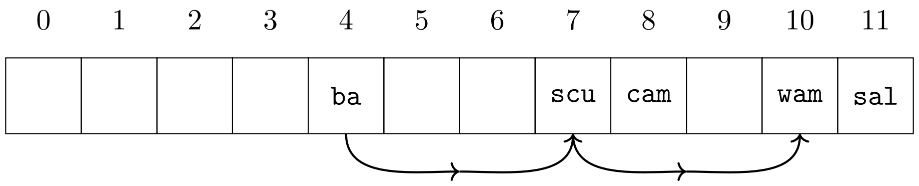
The new hash values are:

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

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

0	1	2	3	4	5	6	7	8	9	10	11
				`ba`			`scu`	`cam`			`sal`

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
```

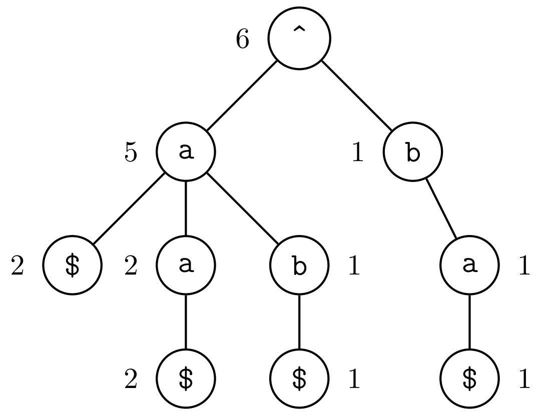
Problem 2 (a)

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"`.

Your task in this problem is to read in N strings containing between 1 and 99 lowercase letters and use them to build a character level trie. For example the input to create the trie above contains $N = 6$ on the first line followed by the 6 strings:

```
6
ab
a
aa
ba
aa
a
```

Your program must print the pre-order traversal of the trie after inserting all N strings. For example:

```
$ ./a2 p2a < tests/p2a-in-1.txt
^
a
$
a
$
b
$
b
a
$
```

Note that the children must be stored (and thus, traversed) in alphabetic order, where `$` comes before `a`, which comes before `b` *etc.*

Problem 2 (b)

2 Marks

In this problem you will construct a trie, as in *Problem 2 (a)*, and use this trie to output all of the prefixes of length exactly K, along with their corresponding frequencies.

The prefixes must be output in alphabetic order.

The input will have one line containing N and K followed by N strings (one per line). For example `tests/p2b-in-2.txt` contains:

```
15 3
algebra
algorithmics
already
algebra
algae
albert
algorithm
algorithms
again
algorithms
artistic
albania
artemis
alg
algorithms
```

In this example your program should output the following:

```
$ ./a2 p2b < tests/p2b-in-2.txt
aga 1
alb 2
alg 9
alr 1
art 2
```

Problem 2 (c)

2 Marks

In this problem you will again construct a character level trie, as in *Problem 2 (a)*, and use it the solve to following problem.

You are provided with a *word stub*, which is the start of a word which is yet to be completed. Your program should predict the probabilities of the possible word completions, based on the set of words provided to your program.

For example if the word stub is `"alg"` then you should calculate the probability of each possible word W with prefix `"alg"` according to the following formula:

$$\Pr(W|\text{stub} = \texttt{"alg"}) = \frac{\text{Freq}(W)}{\text{Freq}(\text{prefix} = \texttt{"alg"})}$$

The input to your program will contain N (the number of strings) on the first line, the word stub on the second line, and then N lines containing the N strings (one per line).

For example, consider the input `tests/p2c-in-2.txt`:

```
15
alg
algebra
algorithmics
already
algebra
algae
albert
algorithm
algorithms
again
algorithms
artistic
albania
artemis
alg
algorithms
```

The word stub given in `"alg"`. The possible word completions are `"algorithms"`, `"algebra"`, `"algorithm"`, `"algorithmics"`, `"algae"` and `"alg"` with frequencies of 3, 2, 1, 1, 1 and 1 respectively.

Your program should output the probability (formatted to 2 decimal places) followed by the word completion of up to 5 word completions (there may be fewer possibilities). Your program should output the most probable word completions, in descending order of probability, breaking ties alphabetically where `a` comes before `aa`.

The output in this example should be:

```
$ ./a2 p2c < tests/p2c-in-6.txt
0.33 algorithms
0.22 algebra
0.11 alg
0.11 algae
0.11 algorithm
```

Analysis Problems

Problem 3

3 Marks – Up to 1/2 page

In the lectures you saw a variety of sorting algorithms with different advantages and drawbacks. In this problem you are a software consultant who is specialised in sorting. You have a set of case studies, given by clients, each one with a range of requirements. The task is to select a *good* sorting algorithm for each case and *justify* your choice, *in a maximum of 2 sentences.* Here is an example:

Example case: an airplane factory wants to improve the embedded system in its airplanes. The system uses a range of sensors that constantly collect large amounts of data from wind currents outside the airplane. This data needs to be sorted as a preprocessing step to ease their visualisation by the pilot. The sorting algorithm should be completely in-place, as extra memory usage should be minimised in an embedded system. The algorithm should also have good performance even in the worst case, since it should be robust to hijacking. What algorithm would you use?

Solution: Heapsort, because it is in-place and has guaranteed $\Theta(n \log n)$ worst case performance, which makes it robust to adversarial attacks.

It's important to keep in mind that this is a *subjective* question and there might be more than one correct answer. Many real world scenarios do not have a single correct solution—this is why your argument about your chosen algorithm is important.

(a) The Melbourne City Library has introduced a program where people can return books in local "drop-off" boxes in their neighbourhoods. For each box, a staff member collects the returned books, sorts them manually and deposits them in a chute at the library, where they are scanned and their record is stored in an array. At the end of the day, the array will contain a sequence of book records starting with all books from the first collected box, all books from the second collected box and so on.

The array needs to be sorted as soon as the last box arrives, to inform the librarian before the end of the day. Therefore, the main requirement is that the algorithm should be as fast as possible in this scenario. However, the algorithm should also maintain the original order of any duplicate books in case multiple copies of the same book are returned in the same day.

(b) A mobile phone manufacturer is planning to revolutionise the market by launching a phone with a battery that lasts for a whole week. The developers of the phone firmware asked for you help to decide on a sorting algorithm to be added to their code.

The sorting function would be general as it needs to sort arbitrary records and it does not need to be fast as most use cases will only have a few hundreds of records maximum. However, the developers are concerned about making it energy efficient: while they managed to keep record comparisons at a very low energy cost (1 nanoWatt), record swapping is not as efficient (5 microWatts, 5000x times higher consumption).

(c) In astronomy, one method used to detect the presence of a new planet is to look for changes of luminosity in stars. An observatory has a collection of electronic telescopes that constantly collects luminosity values. Each telescope is focused on a small set of 50 stars. At every *microsecond*, it returns a luminosity value for each of its 50 stars: that value is an integer between

0 and 10. This set of values needs to be sorted and sent to a central server in the observatory.

As this whole procedure happens *every microsecond*, the sorting algorithm needs to be extremely fast. Each telescope has a small amount of additional memory that can be used for that. The luminosity values are stored in an array in memory. You are allowed to use an **additional** array as auxiliary memory but the additional array should be no longer than the original array used to store the luminosity values.

Problem 4

3 Marks – Up to 1/2 page

You are a hacker that recently got hired by a cybersecurity company. The team needs to be preemptive against any adversarial attacks. Your role is to develop such attacks so the defense team can prepare mechanisms to avoid them.

In this problem you were able to inject code into the algorithm for inserting an element into a Binary Search Tree. Your goal is to use to **force the BST to always degrade to a "stick" (to the right), or more specifically, a linked list**.

The algorithm to insert an element into the BST is as follows.

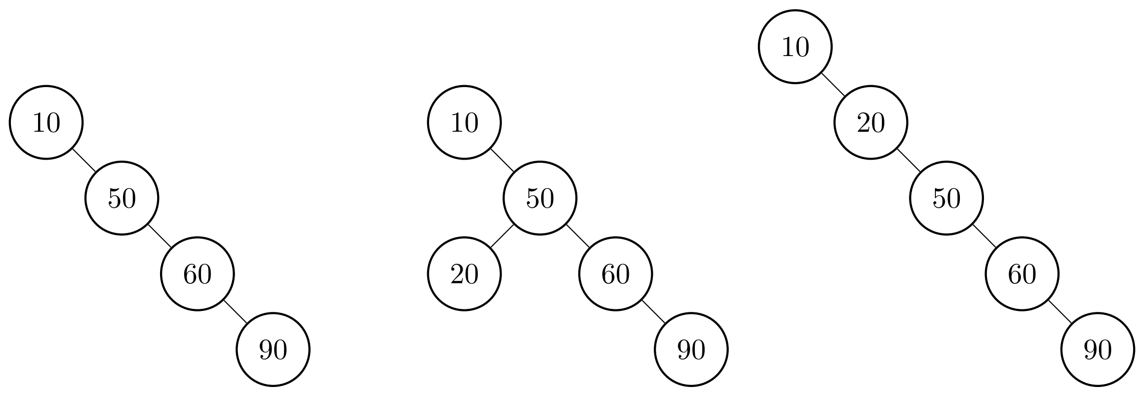
```
function BSTInsert(root, new):
    if new.value < root.value then
        if root.left = NULL then
            root.left ← new
            Stickify(new, root)
        else
            BSTInsert(root.left, new)
    else
        if new.value > root.value then
            if root.right = NULL then
                root.right ← new
                Stickify(new, root)
            else
                BSTInsert(root.right, new)
        else
            return // The element new.value is already in the tree
```

Here *root* and *new* are node data structures with fields *left* and *right* (which are pointers to other node data structures) and a field *value*.

Your task is to write the pseudocode for the algorithm Stickify function which takes the newly inserted node along with its parent and performs any necessary rotations to ensure the tree remains a "right stick".

If the BST is already a "right stick", then running the BSTInsert algorithm (with your Stickify function) should always leave the tree as a "right stick".

For example if we start with the tree on the left and insert 20 we will get the tree in the middle. After Stickify is run (with pointers to nodes 20 and 50 respectively).



Your algorithm should rearrange the tree by performing rotations (you may use RotateLeft(node) and RotateRight(node) without implementing these functions).

(i) Develop a set of rotations to ensure the BST is always a “stick”. You can safely assume that 1) the BST always starts as an empty structure, with elements inserted one-by-one and 2) the linked list should use the *right* children of each node in the tree.

(ii) Write pseudocode to perform the rotations at every insertion. The code is assumed to be injected in the blue line above. As before, it's safe to assume the BST is empty in the beginning.

You may use examples and diagrams to explain your algorithm. Your whole solution must be at most half a page in length.

Completing the Programming Problems

We have provided you with the skeleton code for this assignment. The provided files has the following directory structure:

```
provided_files/Makefile
  Makefile
  hash.c
  hash.h
  main.c
  text_analysis.c
  text_analysis.h
  tests/
    p1a-in-1.txt
    p1a-out-1.txt
    ...
    p2c-in-4.txt
    p2c-out-4.txt
```

You must not change the `main.c` file, however you are allowed to change any of the other files (without changing the existing type declarations and function signatures).

You should implement your solution to Problem 1 in the `hash.c` and `hash.h` files, and you should implement your solution to Problem 2 in the `text_analysis.c` and `text_analysis.h` files.

If you create new C modules then you must change the `Makefile` so that your program can be compiled on `dimefox` using the `make` command.

To run the program you must first compile with `make`, and then run the `a2` command with one of these command line arguments: `p1a`, `p1b`, `p2a`, `p2b`, `p2c`.

For example:

```
$ make
$ ./a2 p1a < tests/p1a-in-1.txt
```

The `pXX-in-X.txt` files contain the input your program will be provided for each test case, and the `pXX-out-X.txt` files contain the expected output. Your program must match the expected output exactly (be careful with whitespace).

Note: in this assignment, unlike Assignment 1, there will be 4 provided test cases, and 2 hidden test cases per subproblem.

Programming Problem Submission

You will submit your program via `dimefox submit`. Instructions for how to connect to the `dimefox` server can be found on the LMS.

You should copy all files required to compile and run your code to `dimefox`, this includes the `Makefile` and all `.c` and `.h` files.

It's recommended that you test your program on `dimefox` before submitting to make sure that your program compiles without warnings and runs as expected.

From the directory containing these files you should run `submit` and **list all files required to compile and run your program**. For example, to submit only the provided files we would run:

```
$ submit comp20007 a2 Makefile main.c hash.c hash.h
   ... text_analysis.c text_analysis.h
```

Note that you can also list all `.c` and `.h` files in the current directory with `*.c` and `*.h`, so the following command will be equivalent:

```
$ submit comp20007 a2 Makefile *.c *.h
```

For this to work correctly you should have your Assignment 2 code in its own subdirectory (*i.e.*, the only files in the directory should be the files you wish to submit).

You must then `verify` your submission, which will provide you with the outcome of each of the tests, and the number of marks your submission will receive.

```
$ verify comp20007 a2 > a2-receipt.txt
```

To view the result of the `verify` command you must run:

```
$ less a2-receipt.txt
```

`less` is a tool which allows you to read files using the command line. You can press `Q` on your keyboard to exit the `less` view.

You can submit as many times as you would like.

Any attempt to manipulate the submission system and/or hard-code solutions to pass the specific test cases we have provided will **result in a mark of 0 for the whole assignment.**

Completing the Written Problems

You will submit your solutions to the written problems to the LMS.

For Problem 4, which asks for pseudocode, we expect you to provide the same level of detail as the lectures and workshops do. Pseudocode which lacks sufficient detail or is too detailed (*e.g.*, looks like C code) will be subject to a mark deduction.

Your submission should be **typed and not handwritten** and submitted as a **single .pdf file**. Pseudocode should be formatted similarly to lectures/workshops, or presented in a monospace font.

The page limit for each problem is given (half to one full page per written problem). Submissions which go over the page limit, have too small a font size or are otherwise unreadable will be subject to a mark deduction.

Make sure you confirm that your written submission has been submitted correctly.

Mark Allocation

The total number of marks in this assignment is 20. The maximum marks for each problem are:

Problem 1 6 marks (3 per part)

Problem 2 7 marks (3 for (a), 2 for (b) and (c))

Problem 3 3 marks (1 per part)

Problem 4 3 marks

There is one additional mark awarded for "structure and style" for the C programming component. Of particular importance will be the structure of your C program in terms of how you separate your types and functions, as well as your commenting and choice of variable names.

There will be 6 test cases for each programming sub-problem, of which 4 are provided to you and 2 are hidden test cases.

The marks awarded for each programming sub-problem will be calculated by:

$$\text{Marks} = \text{Total Marks} \times \text{Test Cases Passed} \div 6$$

Late Policy

A late penalty of 20% per day (24 hour period) will be applied to submissions made after the deadline. The 20% applies to the *number of total marks*. This applies *per component*, *i.e.*,

$$\begin{aligned}\text{Grade} = &\max\left\{\text{Programming Grade} - 0.2 \times \text{Days Late} \times 14, 0\right\}\\ &+ \max\left\{\text{Written Submission Grade} - 0.2 \times \text{Days Late} \times 6, 0\right\}.\end{aligned}$$

For example, if you are 2 days late with the programming component but only 1 day late with the analysis component your grade for the programming component will be reduced by $0.4 \times 14 = 5.6$ and the grade for the analysis component will be reduced by $0.2 \times 6 = 1.2$.

Academic Honesty

You may make use of code provided as part of this subject's workshops or their solutions (with proper attribution), however you **may not** use code sourced from the Internet or elsewhere. Using code from the Internet is grounds for academic misconduct.

All work is to be done on an individual basis. All submissions will be subject to automated similarity detection. Where academic misconduct is detected, all parties involved will be referred to the School of Engineering for handling under the University Discipline procedures. Please see the Subject Guide and the “Academic Integrity” section of the LMS for more information.