



1.4.2 Hash Tables

Q1

Andrew is writing code to implement an allow list for a firewall to keep a track of websites that users on a network are permitted to visit. He has decided to store the address of each website, along with any extra details, in a hash table.

The hash table's hash function is carried out on the website's address which acts as the key.

The hash function works in the following way:

1. Discard the character up to and including the first dot.
2. Discard the characters including and to the right of the remaining leftmost dot.
3. Convert the characters to uppercase.
4. Add the ASCII values of the characters together and MOD by 100.

A	B	C	D	E	F	G	H	I	J	K	L	M
65	66	67	68	69	70	71	72	73	74	75	76	77
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
78	79	80	81	82	83	84	85	86	87	88	89	90

For example, `www.python.org.uk` gets hashed in the following manner:

Step 1:

`python.org.uk`

Step 2:

`python`

Step 3:

`PYTHON`

Step 4:

$(80 + 89 + 84 + 72 + 79 + 78) \text{ MOD } 100 = 82$

a) State what hashed value would be given by the website `www.pugs.co.uk`

[1 mark]

- 19





b) Write a function hash which takes a string and returns the hashed value.

You can assume you have access to the following three functions:

- `asc()` – this takes a character as a parameter and returns its ASCII value.
- `locate()` – this takes a string and a character as parameters and returns the index of the first instance of the character in the string.
- `upper()` – this takes a string as a parameter and returns the strings with all characters in uppercase.

[5 marks]

```
function hash(siteName)
    //remove up to and including first dot.
    firstDot = locate(siteName, ".")
    siteName = siteName.substring(firstDot+1, siteName.length - firstDot - 1)

    //remove second dot and after
    secondDot = locate(siteName, ".")
    siteName = siteName.substring(0, secondDot)
    siteName = upper(siteName)
    value = 0
    for i=0 to siteName.length-1
        value = value + asc(siteName.substring(i, 1))
    next i
    return value MOD 100
endfunction
```

c) The hash function that Andrew is using is producing many instances where two different keys generate the same hash value after the hash function is applied to them. This causes collisions.

Explain how collision handling methods can be applied to handle collisions.

[4 marks]

- Linear probing could be used
- Move through the structure one space at a time
- to find the next free space
- Chaining could be used
- Each location points (to the start of) a linked list.
- The new item is added to the end of the linked list.
- Use overflow area
- The new item is stored with the other values in the same area





The hash function is changed so there are no longer high numbers of collisions.

During busy periods the firewall is expected to check several addresses a second. It is anticipated that roughly 10 new addresses will be added to the allows list each day.

Andrew is considering whether a hash table (with the new hash function) is the best approach, or if the whitelist would be better stored in a linked list.

- d) Discuss whether a hash table or linked list is better to store acceptable websites. You should compare how each structure can be searched and has data added and come to a recommendation as to which is better for the allow list.

[12 marks]

- Searching of a Linked list involves starting at the first node and following the pointers until either the desired value is found, or the end of the list is reached, meaning the item isn't in the list.
- The bigger the linked list grows, the longer it takes to search.
- If a linked list doubles in size it will, on average, take twice as long to search.
- A list of size n takes on average $n/2$ checks.
- In Big O this is $O(n)$, or linear complexity.
- Searching of a hash table requires the key to be hashed and the correct location
- accessed.
- The time this takes is largely dependent on the time to create the hash.
- If we ignore collisions, the time to find an item will stay the same regardless of the size of the allow list.
- In other words it has $O(1)$ or constant complexity.
- As the allow list grows collisions become more likely.
- Linear probing and chaining means that once a location has been found the time taken grows linearly with the number of collisions that have occurred for that location,
- Nonetheless this is still going to perform significantly better than a linked list.
- If items are added to the end of the linked list then if the location of the last node is
- stored, that location can be ready made to point at the new item.
- The time to add items is constant.
- If they are added in some sort of order then the time to add items grows linearly due to the time spent searching for the right position. (Storing in order has the advantage that it is if an item isn't in the list this can be deduced once its location is passed, rather than waiting until the end.)
- Adding items to a hash table involves hashing the key and placing it in the correct
- location.
- This takes a constant amount of time..





- ..unless there are collisions then there is an overhead which grows with the number of collisions for that location.
 - Overall a hash table is likely to be the best option (assuming it has enough space and a good hashing algorithm which produces a hash quickly and with few collisions). It will give very consistent performance even as the whitelist grows.
-

