Data Structure and Advanced Programming

Homework 12

B08705034 資管一 施芊羽

1.

In my opinion, I think that the using hash-table will be the most efficient and less-memory-used way to implement an English dictionary if a good, well-defined hashing function is chosen.

Sorted Array-based Implementation

For a sorted array-based implementation, the Big-O of the retrieval function will be $O(\log n)$ while using binary search.

Sorted Linked-based Implementation

For a sorted linked-based implementation, the Big-O of the retrieval function will be $O(n)$ since we have to traverse through all the nodes and their pointer `next` to retreive the desired English word.

Binary Search Tree

For a binary search tree, the Big-O of the retrieval function will be $O(\log n)$

Hashing

For hashing, the Big-O of the retrieval will be $O(1)$ if a good hash function is chosen. Also, the using of hashing will save memory space at the same time.

2.

a.

For this hashing function, the keys will not likely be distributed different index after the hashing since most English words have the sum of positions in alphabet of their letters less than 2048. In case of two keys has the same sum of positions in alphabet of their letter, such like “act” and “cat” will occur to point to the same index.

b.

The function is not able to distribute strings evenly to the table since position in alphabet of first letters of key will be in the range of $[1,26]$ since there are only 26 alphabets. Like for "BUT" and "BEAR" will get the same index after hashing. It's not appropriate for the strings are random since they will only get their index in the range between $[1,26]$ after the hashing, and the rest of the table will be a waste; if the strings are English words, the table will more likely to distribute the words evenly although they'll still get indices between 1 and 26.

c.

The appropriateness of this function is not quite good since the smaller key will have more probability to be used as the $(key \times random)$ becomes a number not larger than key. Hence, the number x that is larger y will be likely to be point to y, thus, 1 can be the index after hashing of $1 \le key \le 9999$ while 9999 can only be the index after hashing for $key = 9999$.

d.

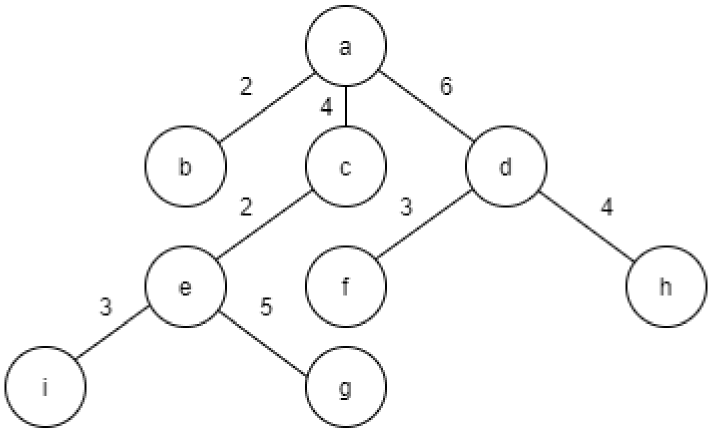
If the function is chosen, first it'll take a bit time since the `for-loop` will run for 100000 times. For the case of $key = 1000$ and $key = 100$ the hash function will both end up returning 0 which occur to hash to the same location. However, this function can make different `hashIndex` more likely.

3.

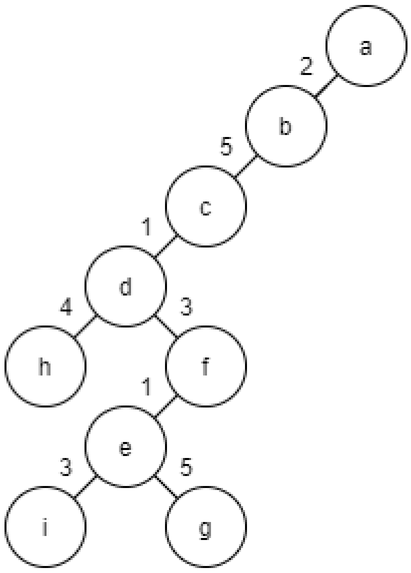
Adjacency Matrix

	a	b	c	d	e	f	g	h	i
a	0	2	4	6	0	0	0	0	0
b	2	0	5	0	0	0	0	0	0
c	4	5	0	1	2	0	0	0	0
d	6	0	1	0	0	3	0	4	0
e	0	0	2	0	0	1	5	0	3
f	0	0	0	3	1	0	4	0	0
g	0	0	0	0	5	4	0	0	0
h	0	0	0	4	0	0	0	0	0
i	0	0	0	0	3	0	0	0	0

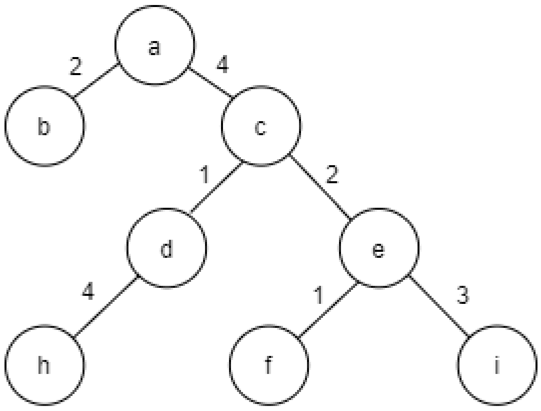
BFS



DFS



Minimum Spanning Tree



a
b
c
d
e
h
f
i
2
4
1
2
4
1
3