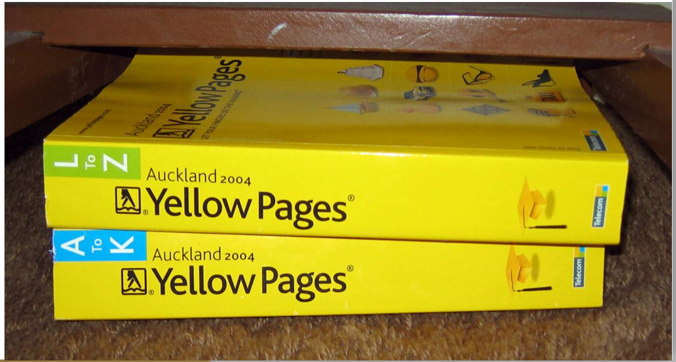
DSA: Project

Phonebook Directory

by:
Tanapat Samakit st122146
Pranpreya Samasutthi st122602

Contents



L To Z
Auckland 2004
Yellow Pages
A To K
Auckland 2004
Yellow Pages

What is our project?

Our project is about phonebook directory that will compare between 2 data structure in 2 application. The first structure that we have done is dictionary. The second structure is binary search tree & List. Each application consists of the same function including:

- Add (Insert) a new contact
- Search for a contact
- Remove an existing contact
- Delete all contacts
- Display all contacts

Work implemented - Dictionary

1. **Add (Insert) a new contact**

- Using Nested Dictionary to insert phonebook into a dictionary using “Name” as a key, and others as a value
- All details of contract which are name, number, email, DOB, and category will keep in the Phonebook List

2. **Search for a contact**

- Using Key of the dictionary to find the contract by “Name” key, number, email, DOB, and category values
- Show all information that related to input keyword

3. **Remove an existing contact**

- Remove Contact in Phonebook by using “Name”

4. **Delete all contacts**

5. **Display all contacts**

- Using Nested Dictionary to display all information of phonebook

Work implemented - BST & List

1. **Add (Insert) a new contact**

- Using BST structure to insert phonebook in to tree by using “Name” of the contract to be a node
- All details of contract which are name, number, email, DOB, and category will keep in the Phonebook List

2. **Search for a contact**

- Using BST structure to find the contract by “Name” node
- Show other information by compare between node.v and name in the phonebook list

3. **Remove an existing contact**

- Remove details in Phonebook List by using “Name”

4. **Delete all contacts**
5. **Display all contacts**

- Using BST structure to display all node.v
- Show other information by compare between node.v and name in the phonebook list

Experiment

Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

Experiment

	Dictionary	BST
Insert	Enter name: DSA Enter number: 123456 Enter e-mail address: dsa.mail Enter date of birth(dd/mm/yy): 1/1/2021 Enter category(Family/Friends/Work/Others): Work ['DSA', '123456', 'dsa.mail', '1/1/2021', 'Work'] Time Usage: 3600	Enter name: DSA Enter number: 123456 Enter e-mail address: dsa.mail Enter date of birth(dd/mm/yy): 1/1/2021 Enter category(Family/Friends/Work/Others): Work Time Usage: 25200
Search	Please enter the name of the contact yo Name: Marysa -> {'Phone': '181-817-77 Time Usage: 330200	Please enter your choice: 2 Enter name that you want to search: zena zena -> Number: 609-290-0944, Email: zbinner2v@jigsy Time Usage: 48400

- With BST you always have O(log(n)) operation, but resizing a hash table is a costly operation
- If you need to get keys in a sorted order you can get them traversing inorder tree. Sorting is not natural to a dictionary
- Doing statistics, like finding the closest lower and greater element, or range query.
- For this experiment, we search for the last contact in each phonebook to measure time spent.

Possible future work/limitations

Limitations:

- For BST, this data structure keeps value as a node, so it is not possible to search and display the contacts using other values than the “name”.
- For Dictionary, Hash tables might have a performance issue when they get filled up and need to reallocate memory (in the context of a hard real-time system). It needs more memory than they actually use

Future work:

- Dictionary is suitable for phonebook application more than BST. Improving Dictionary performance is future work because it’s slower than BST now. Therefore, improve Dictionary performance to be best case of Big-Oh is the best choice.

References

1. https://www.geeksforgeeks.org/implementing-a-contacts-directory-in-python/
2. python - Writing a dictionary to a csv file with one line for every 'key: value' - Stack Overflow
3. Assignment 8 - BST of DSA class:
   https://github.com/pranpreya/Data-Structures-Algorithms/blob/main/8-Pranpreya.py
4. https://stackoverflow.com/questions/5671445/dictionary-implementation-balance-binary-search-tree-v-s-hash-table