PICT, PUNE

Subject: Data Structures & Algorithms

Roll No. 21118

Assignment No: 02

Date: 20-Feb-2021

Problem Definition:

A dictionary stores keywords & it's meanings. Provide faility for adding new keywords, deleting keywords, updating values of an entry. Provide faility to display whole data stored in ascending / descending oder. Also find how many maximum comparisons may require for finding any word. Use binary search tree for implementation.

Learning Objectives:

1) To understand practical implementation & usage of non linear data structure binary search tree (BST). for solving problems of different domains.
2) To implement functionality of dictionary & do the cost analysis of various methods.

Learning Outcomes:

1) Understand BST as an ADT to design algorithm for specific problem.
2) Apply & analyze non-linear data structure binary tree to solve real world problems.

Concept Related Theory:

Tree & Binary Search Tree:

1) Tree can be considered as an undirected graph where there is exactly one path from any node to any other node.
2) Binary Tree is a tree where each node has at most two childs.
3) Binary search tree (BST) is a special case of

PICT, PUNE

binary tree where left child of node contains values lower than root value & right child of node contains values higher than root value. This property is followed in entire tree.

eg. of binary search tree:

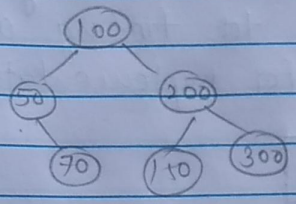

fig Binary Search Tree

note that value at each node is greater than its left child & smaller than its right child

Dictionary functionality:

It is an abstract data type composed of collection of key, value pairs, such that each possible key appears at most once in the collection.

following are the common operations on dictionary

i) Search value of key in the collection

ii) Remove the key & it's associated value from the collection.

iii) Add new key value pair in the collection.

iv) Update the value of key in the collection.

The major data structures used to implement dictionary functionalities are hash tables & search trees.

Class Structure & Pseudo Code:

- Two main classes are used:

Roll. No. 21118

PICT, PUNE

A) Node class: Used to ~~each~~ store each node in the binary tree.

```
class Node {
    string key, val;
    Node* lchild, *rchild;
public:
    Node(string k="", string v="") {
        key=k, val=v;
    }
    void printNode() {
        // code to print key & val pair.
    }
};
```

B) BST class: This represents binary tree/dictionary

```
class BST {
    Node* root;
public:
    BST() { root = NULL; }
    void setRoot(Node* rt) { root = rt; }
    Node* getRoot() { return root; }
    Node* Insert(Node* root, string k, string v);
    Void Update(Node* root, string k, string v);
    int Search(Node* root, string k);
    Node* deleteNode(Node* root, string k);
    void lexoPrint(Node* root);
    void decPrint(Node* root);
};
```

Pseudo Codes:

A) Search operation: Searches key in the dictionary.

if found returns no. of comparisons else returns -1.

Search (Node* root, string key) {

1. if (root == NULL)
   return -1;
2. else if (root->key matches with key)
   return 1;
3. search in left subtree, if key is less tha root->key
   l = Search (root->left, key)
4. else search in right subtree.
   r = Search (root->right, key);
5. if both l & r are -1
   return -1.
6. else if l is -1 return r+1;
7. else if r is -1 return l+1;

2> Insert Operation:

Insert (Node* root, key, val) {

1. if root is NULL:
   return new node with key & val fields.
2. if key < root->key:
   root->leftchild = Insert(root->left, key, val);
3. else if key > root->key;
   root->rightchild = Insert (root->right, key, val);
4. return root;

3> Update:

Update (root, key, val):

Roll No. 21118

PICT, PUNE

1. If (root is NULL):
   return;
2. else if (root->key is same as key):
   set root->val to val.
3. else if (key < root->key)
   Update (root->leftchild, key, val).
4. else if (key > root->key)
   Update (root->rightchild, key, val).

4) Delete.

Delete (root, key) : // return root of updated tree.
1. if root is NULL
   return root.
2. if key < root->key
   root->lchild = Delete (root->lchild, key).
3. else if (key > root->key
   root->rchild = Delete (root->rchild, key).
4. else : // when root->key is same as key
   4.1 if root->lchild is NULL:
      4.1.1 temp ← root
      4.1.2 root ← root->rchild
      4.1.3 deallocate memory of temp.
   4.2 follow steps of 4.1 for lchild if rchild is NULL.
   4.3 else :
      4.3.1 temp ← root->rchild
      4.3.2 while (temp->lchild)
         temp ← temp->rchild.
      4.3.3. swap the node values at temp & root.

Roll. No. 21118

PICT, PUNE

4.3.4. Delete (root→child, key):

4. return root

~~Write a program~~

4) Printing keys in ascending order. Inorder traversal of binary search tree gives the keys in ascending order.

```
AscendingPrint (root):
1 if root is NULL
      return.
2. AscendingPrint(root→lchild)
3. Print data of root.
4. AscendingPrint(root→rchild)
```

5) Descending Print:

```
DescendingPrint (root):
1. IF (root is NULL)
      return.
2. DescendingPrint (root→rchild)
3. Print data of root.
4. DescendingPrint (root→lchild)
```

Time & Space complexity Analysis

Method / Algorithm	Time Complexity	Auxiliary Space.
1) Insert.	The algorithm moves left/right side of node according to	The algorithm doesn't uses any additional data

Roll No 21118

PICT, PUNE

	data of node. If node at appropriate position it insert it. At most the algorithm require O(h) time where h is the height of the tree.	structure. It operates on original tree. Hence it require O(1) & const. auxiliary space.
2) Search.	To search a key in binary search tree we require O(h) time. where h is height of tree.	No additional data structure is used, hence auxiliary space required is O(1)
3) Update	for updating key we need to search for it & then it's const. time operation. for searching it takes O(h) time. ∴ Overall algorithm takes O(h) time.	No additional data structure hence O(1) time.
4) Delete.	The algorithm takes O(h) time in worst case. as it moves left/right according to tree data & deletes node if found.	No additional data structure hence O(1) time.
5) Increasing order Print & Decreasing Print	The algorithm traversal each node, hence takes O(n) time. where n ← no. of keys in dictionary	No additional space is require which depend on input size hence O(1) time algorithm.

Roll. No. 21118

PICT, PUNE

Testcases:

* The following tree/dictionary is constructed using insert operation multiple times.

Input given:	key	Mango	Apple	Orange	Grapes
	value	Green	Red	Orange	Black

Expected Output	Actual Ouput
lexographic Order of keys.	Order of keys obtained is as follow: Apple Grapes, Mango Orange

Operation	Expected Output	Actual Output
1) Searching a key "Orange"	The key & it's value should be printed & no. of comparisons also.	The key & value is printed as follows: Orange -> Orange. The comparisons needed:
2) Inserting new key "Banana"	The new key should be inserted & should be shown at appropriate position in lexographic order.	After inserting key, increasing order of keys is: Apple Banana Grapes Mango Orange
3) Updating value of key "Grapes" to "Purple"	The value of key should be updated if it is presents in the key	After updation key value pair is as follows: Apple -> ~~Green~~ Red, Banana -> Yellow, Grapes -> Purple, Mango -> Green, Orange -> Orange

Roll No. 21118

iv) deleting key "Banana" from dictionary	The key should be delete from dictionary, expected ascending order of keys: Apple Grapes Mango Orange	After deletion of key lexographic order is same as expected.
v) Search key "fruit" in dictionary	Since key is not present in tree, the message should be displayed.	The program outputs the message "key is not present in BST".
vi) Updating value of key "fruit" in dictionary.	Since key is not present in the dictionary, there should be no updation/change in dictionary	The lexographic order of keys & their values is same as before updating the dictionary
vii) Deleting a key "fruit" from dictionary	Since key is not present in the dictionary there should be no change ~~change~~ in the dictionary.	The lexographic order of keys & their values is same before & after deletion of key "fruit".

Real time applications of Dictionaries:

1) Dictionaries has wide range of application. Many standard libraries of languages contain predefined implementation of dictionaries eg. map in CH STL
2) Dictionaries are used to store key, value pares in JSON (Java Script Object Notation) which is a format for storing & transporting data.

Roll No. 21118

PICT, PUNE

Conclusion

Through this assignment, I learned implementation of dictionary using BST data structure. Implemented general methods of dictionary like search, insert, delete & update & studied about their real world applications.