21163

Page No.
Date: / /

Assignment 8

Title: AVL tree

Problem Statement:
A dictionary has keywords & meanings Provide facility for adding new keywords delete, update, display & finding complexity to find a word.

Objective:
To study dynamic programming & implement AVL tree.

Outcome:
I will be able to implemen AVL tree. & get the benefits of it while searching.

Requirements:
1) 64 bit OS
2) Editor, g++, linker, loadr.
3) CPU, RAM.

Theory:
AVL tree is such that it is always balanced. No two leaf nodes are farther than 1 in height from the root node.
In dynamic programming we find the best solution to a problem at run time.

Page No. _____
Date: / /

- Pseudo Codes:

```
add ( Node *t , string key , string meaning ) {
      if (t==NULL)
          t = new Node()
      else {
          if (key == keyword)
              t -> right = add(t->right, k, m)
              if (balance == -2)
                    t = RR(t)
              else
                    t = RL(t)
          }
          else if (key < keyword) {
              t -> left = add (t->left , k, m)
              if (balance == 2) {
                    if key < left->key
                          t = LL(t)
                    else
                          t = LR(t)
              }
          }
          else {
              print "present"
          }
      }
      t -> balance = height (t)
      return t ;
}
```

```
findbalance (Node *t){

    int lh, rh
    if (r = NULL)
        return 0;
    if (t->left == NULL)
        lh = 0
    else
        lh = (t->left->balance)+1
    if (t->right == NULL)
        rh = 0
    else
        rh = t->right->balance +1
    return (lh - rh)
}
```

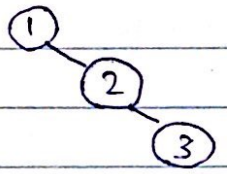
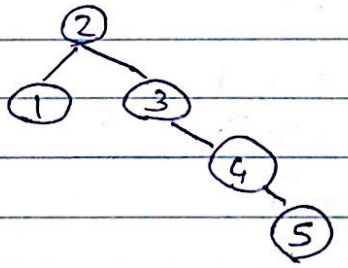
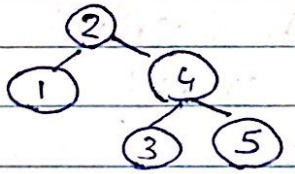
```
Node * RR (Node *t){
        t = rotateleft (t)
        return t
}
```

```
Node * LR (Node *t){
        t->left = rotateleft (t->left)
        t = rotateRight (t)
        return(t);
}
```

```
Node *RL(Node *t){
    t->right = rotateRight(t->right)
    t = rotateleft(t)
    return t
}

Node *LL(Node *t)
    t = rotateRight(t)
    return t
}

Node *rotateleft(Node *t){
    Node *r = t->right
    t->right = r->left
    r->left = t;
    t->balance = height(t)
    r->balance = height(r)
    return r;
}

Node *rotateRight(Node *t){
    Node *r = t->left
    t->left = t->right
    t->right = r
    t->balance = height(t)
    r->balance = height(r)
}
```

```
Node *deleteNode(Node *root, string key)
    if (root = NULL)
        return root;
    if (root->keyword.compare(key) < 0)
        root->left = deleteNode(root->left, key)
        ret root;
    else() {
        root->right = deleteNode(root->right, key)
        ret root
    }
    if (root->left == NULL)
        ret root->right
    else if (root->right == NULL)
        ret root->left
    else {
        succ parent = root->right
        succ = root->right
        while (succ->left != NULL) {
            succ parent = succ
            succ = succ->left
        }
        Succparent->left = succ->right
        root->keyword = succ->keyword
        delete succ
        ret root
    }
}
```

- Test Cases:

Input	Operation	Result
1, 2, 3, 4, 5	delete (5)	deleted Success
	delete (5)	Not found
	delete (3)	deleted success
	update (4, no)	updated meaning success.

Conclusion:
Hence I learnt to successfully construct an AVL tree from scratch using C++.