**PUNE INSTITUTE OF COMPUTER TECHNOLOGY, PUNE - 411043 Department of Computer Engineering** 

**S.No.-27, Pune Satara Road, Dhankawadi, Pune-411043**

**Data Structures and Algorithms Laboratory**

**Batch-IV (H4)**

**Name: Rushikesh Vidye**

**Roll No.: 21471**

**Class: SE4**

**Assignment No. 2**

**Title:** A Dictionary stores keywords and its meanings. Provide facility for adding new keywords, deleting keywords, updating values of any entry. Provide facility to display whole data sorted in ascending/ Descending order. Also find how many maximum comparisons may require for finding any keyword. Use Binary Search Tree for implementation.

**Software Requirement:**

a) OS : Microsoft Windows 10.

b) Browser: Google Chrome.

c) VS Code.

**Hardware Requirement:**

a) Processor: Intel Core i5-8265U.

b) Ram: 8 GB DDR4 2800Mhz.

**Theory:**

**Binary Search Tree:**

In computer science, a binary search tree (BST), also called an ordered or sorted binary tree, is a rooted binary tree whose internal nodes each store a key greater than all the keys in the node's left subtree and less than those in its right subtree. A binary tree is a type of data structure for storing data such as numbers in an organized way. Binary search trees allow binary search for fast lookup, addition and removal of data items, and can be used to implement dynamic sets and lookup tables. The order of nodes in a BST means that each comparison skips about half of the remaining tree, so the whole lookup takes time proportional to the binary logarithm of the number of items stored in the tree. This is much better than the linear time required to find items by key in an (unsorted) array, but slower than the corresponding operations on hash tables. Several variants of the binary search tree have been studied.

**Binary Search Tree** is a node-based binary tree data structure which has the following properties:

* The left subtree of a node contains only nodes with keys lesser than the node’s key.
* The right subtree of a node contains only nodes with keys greater than the node’s key.
* The left and right subtree each must also be a binary search tree.

200px-Binary_search_tree.svg

**Searching a key**   
For searching a value, if we had a sorted array we could have performed a binary search. Let’s say we want to search a number in the array what we do in binary search is we first define the complete list as our search space, the number can exist only within the search space. Now we compare the number to be searched or the element to be searched with the mid element of the search space or the median and if the record being searched is lesser we go searching in the left half else we go searching in the right half, in case of equality we have found the element. In binary search we start with **‘n’** elements in search space and then if the mid element is not the element that we are looking for, we reduce the search space to **‘n/2’** and we go on reducing the search space till we either find the record that we are looking for or we get to only one element in search space and be done with this whole reduction.

Search operation in binary search tree will be very similar. Let’s say we want to search for the number, what we’ll do is we’ll start at the root, and then we will compare the value to be searched with the value of the root if it’s equal we are done with the search if it’s lesser we know that we need to go to the left subtree because in a binary search tree all the elements in the left subtree are lesser and all the elements in the right subtree are greater. Searching an element in the binary search tree is basically this traversal in which at each step we will go either towards left or right and hence in at each step we discard one of the sub-trees. If the tree is balanced, we call a tree balanced if for all nodes the difference between the heights of left and right subtrees is not greater than one, we will start with a search space of **‘n’**nodes and when we will discard one of the sub-trees we will discard **‘n/2’** nodes so our search space will be reduced to **‘n/2’** and then in the next step we will reduce the search space to **‘n/4’** and we will go on reducing like this till we find the element or till our search space is reduced to only one node. The search here is also a binary search and that’s why the name binary search tree.

**Insertion of a key**   
A new key is always inserted at the leaf. We start searching a key from the root until we hit a leaf node. Once a leaf node is found, the new node is added as a child of the leaf node.

**Algorithm:**

1: Insert Node :

1. Create a new BST node and assign values to it.

2. insert(node, key)

     i) If root == NULL,

         return the new node to the calling function.

     ii) if root=>data < key

         call the insert function with root=>right and assign the return value in root=>right.

        root->right = insert(root=>right,key)

     iii) if root=>data > key

         call the insert function with root->left and assign the return value in root=>left.

         root=>left = insert(root=>left,key)

3. Finally, return the original root pointer to the calling function.

2: Search Node:

1. Compare root with search element
   1. Return root if true
2. Check if element belongs to left subtree.
   1. Traverse to left subtree.
   2. Go to step 1
3. Traverse to right subtree.
   1. Traverse to right subtree
   2. Go to step 1

**Time Complexity:**

|  |  |  |
| --- | --- | --- |
| Sr.No | Methods | Complexity |
| 1 | Insert Node() | O(n) |
| 2 | Search Node | O(n) |

**Conclusion:**

Hence we learnt and implemented binary search tree concept into real world application like Dictionary.

**Code:**

|  |
| --- |
| #pragma once  // Headers and Standard Library Declaration  #include<iostream>  #include<string>  using namespace std;  // Class and Member function Declaration  //Class Node For Stoaring Word,Meaning left anf Right Pointers of BST  class Node{      private:      string word,meaning;      Node \*left,\*right;      public:      friend class Dictionary;      Node(string word,string meaning);      Node(Node\* other);      void printWordAndMeaning();};  //Class Dictionary for Stoaring Node class objects and all important function impletation to make a working dictionary  class Dictionary{      private:      Node\* root;      int counter;      public:      Dictionary();      Dictionary(Node\* root);      Node\* GetRoot();      int GetCounter();      void SetRoot(Node\* root);      void SetCounter(int count);      void AddNode(string word,string meaning);      void InOrder(Node\* root);      void Dictionary::PreOrder(Node\* root);      void ReverseInOrder(Node\* root);      Node\* SearchWord(Node\* root,string word);      Node\* MinimumValueNode(Node\* root);      Node\* DeleteNode(Node\* root,string word);      void UpdateNode(Node\* root,string searchWord,string newSword,string meaning);};  // Node Class Function Defination Start  Node::Node(string word,string meaning){      //cout<<"Inside Node Construct"<<endl;      this->word = word;      this->meaning = meaning;      left = right = NULL;}  Node::Node(Node\*other){      this->word = other->word;      this->meaning = other->meaning;      this->left = other->left;      this->right = other->right;}  void Node:: printWordAndMeaning(){      cout<<"\t"<<this->word<<" : "<<this->meaning<<endl;}  // END Of Node Class Function  // START OF Dictionary Class Function Defination  Dictionary::Dictionary()  { this->root = NULL;      this->counter = 0;}  Dictionary::Dictionary(Node\* root)  { this->root = new Node(root);      this->counter = 0;}  Node\* Dictionary::GetRoot()  {return this->root;}  int Dictionary::GetCounter()  {return this->counter;}  void Dictionary::SetRoot(Node\* root)  {this->root = root;}  void Dictionary::SetCounter(int count)  {this->counter = count;}  // Function to Add New Node to Binary Tree  void Dictionary::AddNode(string word,string meaning){      Node\* move,\*newWord;      newWord = new Node(word,meaning);      if(this->root == NULL){          this->root = newWord;      }else{          move = this->root;          while (1){             if(newWord->word < move->word){                 if(move->left == NULL) {                      move->left = newWord; }                 else{                   move = move->left; }}             else if(newWord->word > move->word) {                 if(move->right == NULL) {                     move->right = newWord; }                 else{                      move = move->right; }             }else{                 break;}}}}  // Basically Using this function to print Dictonary in Ascending Order  void Dictionary::InOrder(Node\* root){      if(root != NULL){          InOrder(root->left);          root->printWordAndMeaning();          InOrder(root->right);}}  void Dictionary::PreOrder(Node\* root){      if(root != NULL){          cout<<root->word<<endl;          PreOrder(root->left);          PreOrder(root->right);}}  void Dictionary::ReverseInOrder(Node\* root){       if(root != NULL){          ReverseInOrder(root->right);          root->printWordAndMeaning();          ReverseInOrder(root->left);}}  // Searching Word In Dictionary Simple Taversal Algorithm  Node\* Dictionary::SearchWord(Node\* root,string word){      this->counter++;      if(root == NULL || root->word == word)          return root;      if(root->word < word)          return SearchWord(root->right,word);      return SearchWord(root->left,word);}  // Used inside DeletNode function to find smalled valued node from right side of tree  Node\* Dictionary::MinimumValueNode(Node \*root){      Node\* move = root;       while(move && move->left != NULL)          move = move->left;      return move;}  // Finds Smalled note on right side of tree that can be used to replace node that is suppsed to be delated  Node\* Dictionary::DeleteNode(Node\* root,string word){      if(root == NULL)          return root;      if(word < root->word)          root->left = DeleteNode(root->left,word);      else if(word > root->word)          root->right = DeleteNode(root->right,word);      else{          if(root->left == NULL) {              Node\* temp = root->right;              delete root;              return temp; }          else if(root->right == NULL){              Node\* temp = root->left;              delete root;              return temp;}          Node\* temp = MinimumValueNode(root->right);          root->word = temp->word;          root->meaning = temp->meaning;          root->right = DeleteNode(root->right,temp->word);}      return root;}  // Search Node from bst and update it's data.  void Dictionary::UpdateNode(Node\* root,string searchWord,string newWord,string meaning){      Node\* UpdateNode = SearchWord(root,searchWord);      UpdateNode->word = newWord;      UpdateNode->meaning = meaning;}  // END Of Dictionary Class Member Functions  #include"header.h"  int main(){      Dictionary days;      Node\* newWord;      string word,meaning;      word = "MONDAY";      meaning = "1st Day Of Week";      days.AddNode(word,meaning);      word = "TUESDAY";      meaning = "2nd Day Of Week";      days.AddNode(word,meaning);      word = "WEDNESDAY";      meaning = "3rd Day Of Week";      days.AddNode(word,meaning);      word = "THURSDAY";      meaning = "4th Day Of Week";      days.AddNode(word,meaning);      word = "FRIDAY";      meaning = "5th Day Of Week";      days.AddNode(word,meaning);      word = "SATURDAY";      meaning = "WeenEnd Start";      days.AddNode(word,meaning);      word = "SUNDAY";      meaning = "WEEKEND OVER";      days.AddNode(word,meaning);      string strNewWord;      int ch;      do{          word = "";          meaning = "";          cout<<endl<<"\tDictionary of Days"<<endl<<"\t 1 : Display All words"<<endl<<"\t 2 : Add New Word do Dictionary"<<endl<<"\t 3 : Remove A Word From Dictionary"<<endl<<"\t 4 : Update a Existing Word and It's Meaning"<<endl<<"\t 5 : Display Words in Ascending Order"<<endl<<"\t 6 : Display Words in Descending Order"<<endl<<"\t 7 : Search Word In Dictionary"<<endl<<"\t 0 : Exit"<<endl;          cin>>ch;          switch(ch) {              case 1:                  cout<<endl<<" WORDS IN DICTIONARY "<<endl;                  days.PreOrder(days.GetRoot());              break;              case 2:                  cout<<endl<<"Enter New Word and its Meaning to add in Dictionary"<<endl;                  cin>>word;                  cout<<"Enter Meaning of Word"<<endl;                  cin.ignore();                  getline(cin,meaning);                  days.AddNode(word,meaning);                  cout<<"Word Added to Days Dictionary"<<endl;              break;              case 3:                  cout<<endl<<"Enter Word To Delete From Dictionary"<<endl;                  cin>>word;                  days.DeleteNode(days.GetRoot(),word);                  cout<<word<<" Deleted From Days Dictionary"<<endl;              break;              case 4:                  cout<<endl<<"Enter Enter Word You want To Update"<<endl;                  cin>>word;                  cout<<endl<<"Enter Enter New WORD AND It's Meaning to Update"<<endl;                  cin>>strNewWord;                  cout<<"Enter Meaning of Word"<<endl;                  cin.ignore();                  getline(cin,meaning);                  cin >> meaning;                  days.UpdateNode(days.GetRoot(),word,strNewWord,meaning);                  cout<<"Word Updated to Days Dictionary"<<endl;              break;              case 5:                  cout<<endl<<"Dictionary Words In Ascending order With Their Meaning"<<endl;                  cout<<endl<<endl<<"\t Word : Meaning"<<endl;                  days.InOrder(days.GetRoot());              break;              case 6:                  cout<<endl<<"Dictionary Words In Descending order With Their Meaning"<<endl;                  cout<<endl<<endl<<"\t Word : Meaning"<<endl;                  days.ReverseInOrder(days.GetRoot());              break;              case 7:                  cout<<endl<<"Enter a word to search from dictionary."<<endl;                  cin>>word;                  newWord = days.SearchWord(days.GetRoot(),word);                  cout<<endl<<endl<<"\t Word : Meaning"<<endl;                  newWord->printWordAndMeaning();                  cout<<"Counter : "<<days.GetCounter()<<endl;                  days.SetCounter(0);              break;              case 0:                  exit(0);              break;              default:                  cout<<endl<<"Wrong Input";              break; }}while(ch != 0);      return 0;} |

**Output:**

|  |
| --- |
| Dictionary of Days  1 : Display All words  2 : Add New Word do Dictionary  3 : Remove A Word From Dictionary  4 : Update a Existing Word and It's Meaning  5 : Display Words in Ascending Order  6 : Display Words in Descending Order  7 : Search Word In Dictionary  0 : Exit  1  WORDS IN DICTIONARY  MONDAY  FRIDAY  TUESDAY  THURSDAY  SATURDAY  SUNDAY  WEDNESDAY  2  Enter New Word and its Meaning to add in Dictionary  Hello  Enter Meaning of Word  Greeting  Word Added to Days Dictionary  1  WORDS IN DICTIONARY  MONDAY  FRIDAY  Hello  TUESDAY  THURSDAY  SATURDAY  SUNDAY  WEDNESDAY  5  Dictionary Words In Ascending order With Their Meaning  Word : Meaning  FRIDAY : 5th Day Of Week  Hello : Greeting  MONDAY : 1st Day Of Week  SATURDAY : WeenEnd Start  SUNDAY : WEEKEND OVER  THURSDAY : 4th Day Of Week  TUESDAY : 2nd Day Of Week  WEDNESDAY : 3rd Day Of Week  7  Enter a word to search from dictionary.  MONDAY  Word : Meaning  MONDAY : 1st Day Of Week  Counter : 1  6  Dictionary Words In Descending order With Their Meaning  Word : Meaning  WEDNESDAY : 3rd Day Of Week  TUESDAY : 2nd Day Of Week  THURSDAY : 4th Day Of Week  SUNDAY : WEEKEND OVER  SATURDAY : WeenEnd Start  MONDAY : 1st Day Of Week  Hello : Greeting  FRIDAY : 5th Day Of Week  4  Enter Enter Word You want To Update  Hello  Enter Enter New WORD AND It's Meaning to Update  There General Kanobi  Enter Meaning of Word  Hello  Word Updated to Days Dictionary  1  WORDS IN DICTIONARY  MONDAY  FRIDAY  There  TUESDAY  THURSDAY  SATURDAY  SUNDAY  WEDNESDAY  Enter Word To Delete From Dictionary  There  There Deleted From Days Dictionary  1  WORDS IN DICTIONARY  MONDAY  FRIDAY  There  TUESDAY  THURSDAY  SATURDAY  SUNDAY  WEDNESDAY  0 |