***BINARY TREES (ABSTRACT DATA TYPE)***

D

E

P

T

H

ROOT

HOG

DOG

HAT

ELF

CAT

HEN

FOX

LEAVES

35

45

HAT

At a given node,

* left subtree will have all the smaller numbers that you have in that list of things
* right subtree will have all the larger numbers that you have in that list of things

40

25

15

30

42

Subtrees

My list 🡪 {15, 25, 30, 35, 40, 42, 45}

I can apply bisection method to find a number where we look at the middle of the list (35) and left side should be smaller than middle value, right side should be larger than middle value.

Always divide 2 and look to middle while you didn’t find your number.

If I have 1 000 000 entries, I will find my entry in only log(n) time comparisons.

Search you are gonna do is depend of depth of the tree.

Depth: how many branching at most you can have.

Binary search tree will have several things:

* things to create a tree
* things to insert an entry
* things to remove an entry
* something to destroy the tree
* function to search an entry
* counting the entries in tree
* function that tells us depth of the tree
* converting tree to a list

SEARCHING

Linked list is like 1 branch of the entire tree. So we need 2 branches at most.

typedef struct node{

struct node \* left;

int data;

struct node \* right;

}tree;

/\* received t is pointing to root first \*/

int search(tree \*t, int key){

int r = 0;

if (t -> data == key) r = 1;

else if (t -> data < key) r = search(t->right, key);

else r = search(t->left, key);

return r;

}

tree \* search2(tree \* t, int key){

if (t!=NULL && t->data!=key{

if (t->data < key) t = search2(t->right, key);

else t = search2(t->left, key);

}

return t;

}

/\* not recursively \*/

int search(tree \*t, int key){

int r;

while (t != NULL && t->data != key){

if (t->data < key) t = t->right;

else t = t->left;

}

if (t == NULL) r = 0;

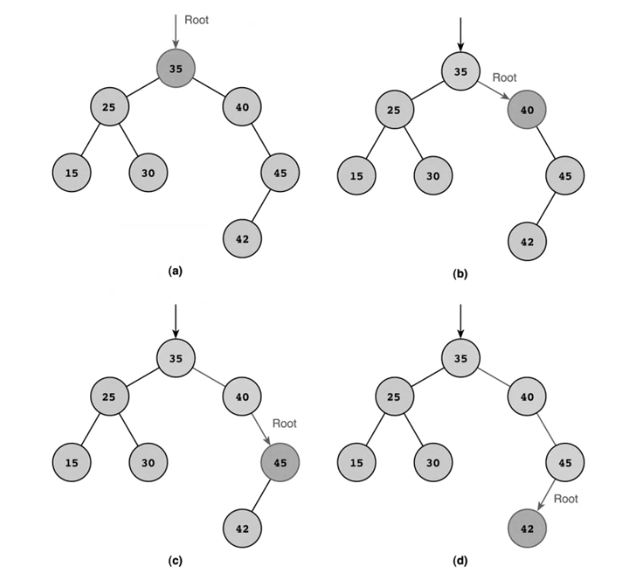
else r = 1;

You can also “return t;” instead of these lines. Of course function definition would be “tree \*”. Now function is returning the node.

return r;

}

Binary tree search for 42:

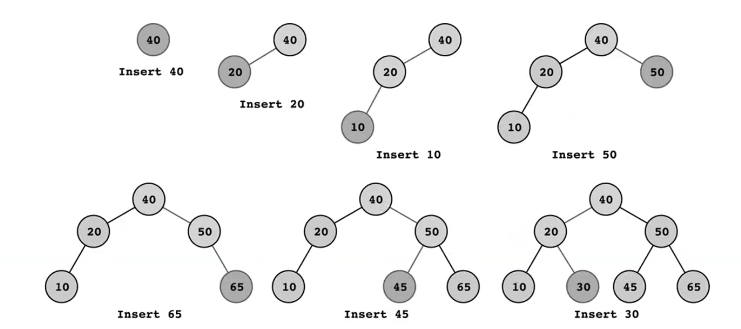


|  |  |  |
| --- | --- | --- |
| NULL | 40 |  |

|  |  |  |
| --- | --- | --- |
|  | 45 | NULL |

|  |  |  |
| --- | --- | --- |
| NULL | 40 | NULL |

INSERTING



tree \*t;

tree \*n = (tree\*)malloc(sizeof(tree)); /\* inserting 40 \*/

n->data = 40;

n->left = NULL;

n->right = NULL;

t = n;

n = (tree \*)malloc(sizeof(tree)); /\* inserting 20 \*/

n->data = 20;

n->left = n->right = NULL;

t->left = n;

/\* FUNCTION IMPLEMENTATION \*/

tree \* insert(tree \*t, int key){

tree \*n;

if (t == NULL){

n = (tree\*)malloc(sizeof(tree));

n->data = key;

n->left = n->right = NULL;

return n;

}

if (t->data == key) return;

if (t->data < key){

if (t->right == NULL) t->right = insert(t->right, key);

else insert(t->right, key);

}

else{

if (t->left == NULL) t->left = insert(t->left, key);

else insert(t->left, key);

}

return t;

}

DELETING

let’s delete 50

parent

40

50

20

children

45

65

10

I have to have the parent of 50 which is 40.

What about the children?

* We can take the children and insert them 1 at a time to the tree.
  + This is not good because I already have the tree but I am building the tree again.
* … You will learn another strategies in another courses.

PRINTING

Algorithm to print in ascending order:

* recursively print left child till you reach a leaf
* print current entry
* print right child

void print(tree \*t){

if (t == NULL) return;

if (t->left == NULL && t->right == NULL) printf(“%d\n”, t->data);

else{

print(t->left);

printf(“%d\n”, t->data);

print(t->right);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void print(tree \*t){

if (t != NULL){

print(t->left);

printf(“%d\n”, t->data);

print(t->right);

}

}