HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

C BASIC

BINARY SEARCH TREE

ONE LOVE. ONE FUTURE.

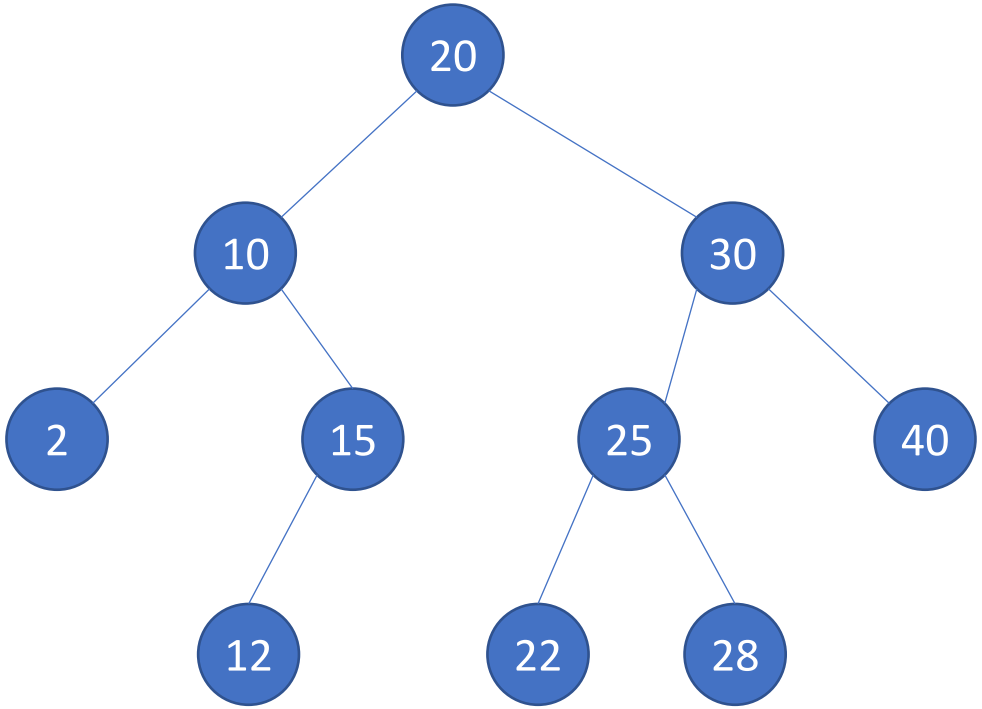
CONTENT

- Binary search tree
- Inserting a key into a binary search tree (P.06.14.01)
- Removing a key from a binary search tree (P.06.14.02)
- Constructing a binary search tree from a sequence of pre-order traversal (P.06.14.03)

BINARY SEARCH TREE

- Binary search tree
  - The key of a node is larger than all keys of the left subtree and smaller than all keys of the right subtree
- Data structure of each node

```
struct Node {
    key // key
    leftChild // pointer to the left child node
    rightChild // pointer to the right child node
}
```



BINARY SEARCH TREE

- Find a key in a binary search tree
  - If k is equal to the key of the root node then return the pointer of the root node
  - If k is larger than the key of the root node then find k on the right subtree (recursion)
  - If k is smaller than the key of the root node then find k in the left subtree (recursion)

```
Find(r, k) {
  if r = NULL then return NULL;
  if r.key = k then return r;
  if r.key < k then
      return Find(r.rightChild, k);
  else
      return Find(r.leftChild, k);
}
```

BINARY SEARCH TREE

- Insert a key into a binary search tree
  - If the tree is empty then create a new node with key k and return the pointer to the node
  - If k is equal to the key of the root node then return the pointer of the root node
  - If k is larger than the key of the root node then insert k into the right subtree (recursion)
  - If k is smaller than the key of the root node then insert k into the left subtree (recursion)

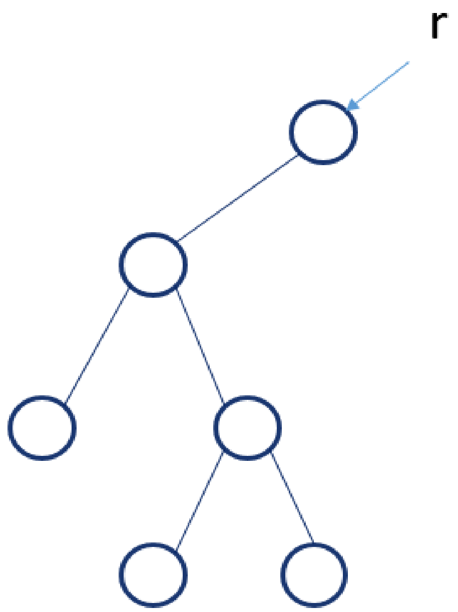
```
Insert(r, k) {
  if r = NULL then return Node(k);
  if r.key = k then return r;
  if r.key < k then
      r.rightChild = Insert(r.rightChild, k);
  else
      r.leftChild = Insert(r.leftChild, k);
  return r;
}
```

BINARY SEARCH TREE

- Remove a key from a binary search tree
  - If the tree is empty then return NULL
  - If *k* is larger than the key of the root node then remove k from the right subtree (recursion)
  - If k is smaller than the key of the root node then remove k from the left subtree (recursion)
  - If *k* is equal to the key of the root node
    - Find the node having the largest key in the left subtree or the node having the smallest key in the right subtree to replace the root node

```
Remove(r, k) {
  if r = NULL then return NULL;
  if r.key = k then
     return RemoveRoot(r);
  if r.key < k then
      r.rightChild = Remove(r.rightChild, k);
  else
      r.leftChild = Remove(r.leftChild, k);
  return r;
}
```

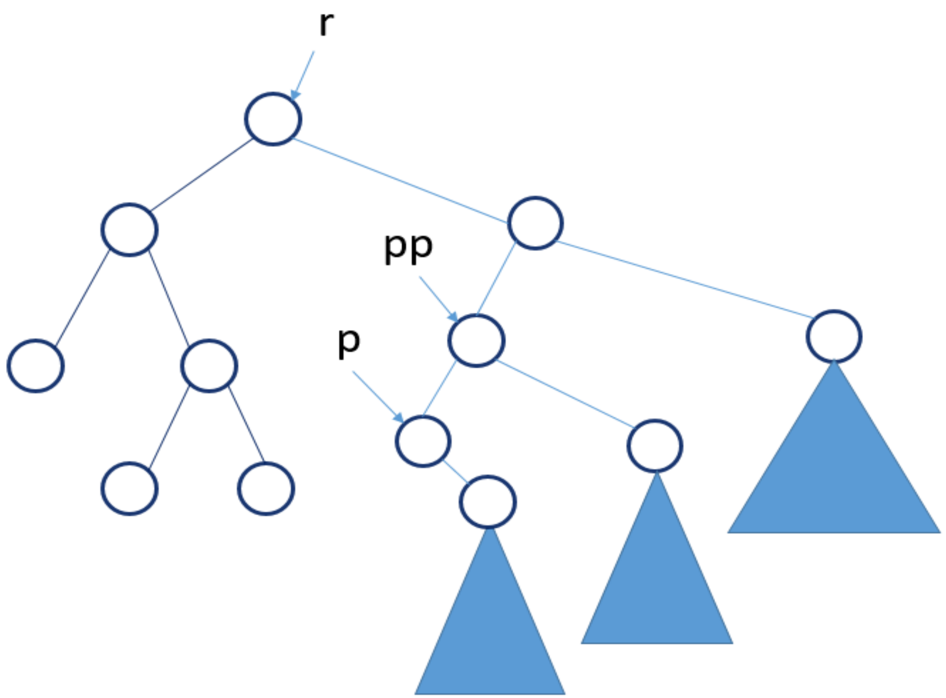
- Remove the root node of a binary search tree
  - If the root node does not have a right child node then return the pointer to the left child node




```
RemoveRoot(r) {
   if r = NULL then return NULL;
   tmp = r;
   if r.rightChild = NULL then {
     r = r.leftChild; free(tmp); return r;
   }
   p = r.rightChild; pp = r;
   if p.leftChild = NULL then {
     r.key = p.key; tmp = p;  r.rightChild = p.rightChild;
     free(tmp); return r;
   }
   while p.leftChild !=  NULL do { pp = p; p = p.leftChild; }
   pp.leftChild = p.rightChild;  r.key = p.key; free(p);
   return r;
}
```

BINARY SEARCH TREE

- Remove the root node from a binary search tree
  - If the root node has a right child node then find the node having the smallest key on the right subtree to replace




```
RemoveRoot(r) {
   if r = NULL then return NULL;
   tmp = r;
   if r.rightChild = NULL then {
     r = r.leftChild; free(tmp); return r;
   }
   p = r.rightChild; pp = r;
   if p.leftChild = NULL then {
     r.key = p.key; tmp = p;  r.rightChild = p.rightChild;
     free(tmp); return r;
   }
   while p.leftChild !=  NULL do { pp = p; p = p.leftChild; }
   pp.leftChild = p.rightChild;  r.key = p.key; free(p);
   return r;
}
```

INSERTING A KEY TO A BINARY SEARCH TREE (P.06.14.01)

- Given a binary search tree T (starting from the empty tree). Perform a series of operations to insert keys into T and traverse with preorder and postorder in T.
  - insert k: insert a node with key k into T (if there is no node with key k)
  - preorder: print a sequence of keys visiting with preorder traversal in T (SPACE separator)
  - postorder: print a sequence of keys visiting with postorder traversal in T (SPACE separator)
- Data
  - Each line with information about an operation of one of the above three types
  - Input data ends with a line "#"
- Result
  - Each line is the result of a preorder or postorder from input data

stdin	stdout
insert 5 insert 9 insert 2 insert 1 preorder insert 8 insert 5 insert 3 postorder #	5 2 1 9 1 3 2 8 9 5

INSERTING A KEY TO A BINARY SEARCH TREE - PSEUDOCODE

- Algorithm
  - At each step, search information from input and perform the corresponding operation.

```
Run() {
  root = NULL;
  while true do {
     cmd = read a string from stdin;
     if cmd = “#” then break;
     if cmd = “insert” then {
        k = read an integer from stdin;
        root = Insert(root, k);
     }else if cmd = “preorder” then
        PreOrder(root);
     else if cmd = “postorder” then
        PostOrder(root);
  }
}
```

REMOVING A KEY FROM A BINARY SEARCH TREE (P.06.14.02)

- Given a binary search tree T (starting from the empty tree). Perform a series of operations to insert keys, remove keys, and traverse with preorder and postorder in T
  - insert k: insert a node with key k into T (if there is no node with key k)
  - remove k: remove key k from T
  - preorder: print a sequence of keys visiting with preorder traversal in T (SPACE separator)
  - postorder: print a sequence of keys visiting with postorder traversal in T (SPACE separator)
- Data
  - Each line with information about an operation of one of the above four types
  - Input data ends with a line "#"
- Result
  - Each line is the result of a preorder or postorder from input data

stdin	stdout
insert 5 insert 9 insert 2 insert 1 preorder insert 8 insert 5 insert 3 postorder #	5 2 1 9 1 3 2 8 9 5

REMOVING A KEY FROM A BINARY SEARCH TREE - PSEUDOCODE

- Algorithm
  - At each step, search information from input and perform the corresponding operation.

```
Run() {
  root = NULL;
  while true do {
     cmd = read a string from stdin;
     if cmd = "#" then break;
     if cmd = "insert" then {
        k = read an integer from stdin;
        root = Insert(root, k);
     } else if cmd = "remove" then {
        k = read an integer from stdin;
        root = Remove(root,k);
     }else if cmd = "preorder" then
        PreOrder(root);
     else if cmd = "postorder" then
        PostOrder(root);
  }
}
```

- Given a binary search tree T, each node has a key as an positive integer. Given a sequence of the keys with preorder traversal in T: k_1, k_2, . . ., k_n . Find the sequence of keys with postorder traversal in T.
- Data
  - Line 1: An positive integer n (1 <= n <= 50000)
  - Line 2: A sequence of keys k_1, k_2, . . ., k_n (1 <= k_i <= 1000000)
- Result
  - Write out a sequence of keys with postorder traversal in T or NULL if *T* is empty.

stdin	stdout
11 10 5 2 3 8 7 9 20 15 18 40	3 2 7 9 8 5 18 15 40 20 10

stdin	stdout
11 10 5 2 3 8 7 9 20 15 18 4	NULL

- Constructing a binary search T from a sequence of keys with preorder traversal.
  - The root node has key k_1
  - Find index i such that k_1 is larger $k_2, k_3, \ldots, k_i$ and k_1 is smaller than $k_{i+1}, k_{i+2}, \ldots, k_n$.
    - If there is no i then T does not exist
    - Otherwise, T T is constructed by the rules:
      - Create a tree with the root node k_1
      - Create a left subtree with a sequence of keys $k_2, k_3, \ldots, k_i$ and a right subtree with a sequence of keys $k_{i+1}, k_{i+2}, \ldots, k_n$.

```
BuildBST(k[1..n], L, R){
  if L > R then return NULL;
  r = Node(k[L]); // tạo nút gốc
  i = L + 1;
  while i <= R and a[i] < a[L] do { i = i + 1; }
  i = i - 1;
  for j = i+1 to R do
    if a[j] < a[L] then {
       ok = False; // biến tổng thể
       return NULL; // T không tồn tại
    }
  r.leftChild = BuildBST(k[1..n], 2, i);
  r.rightChild = BuildBST(k[1..n], i+1, R);
  return r;
}
```

HUST

hust.edu.vn fb.com/dhbkhn

THANK YOU !