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

CÂY TÌM KIẾM

ONE LOVE. ONE FUTURE.

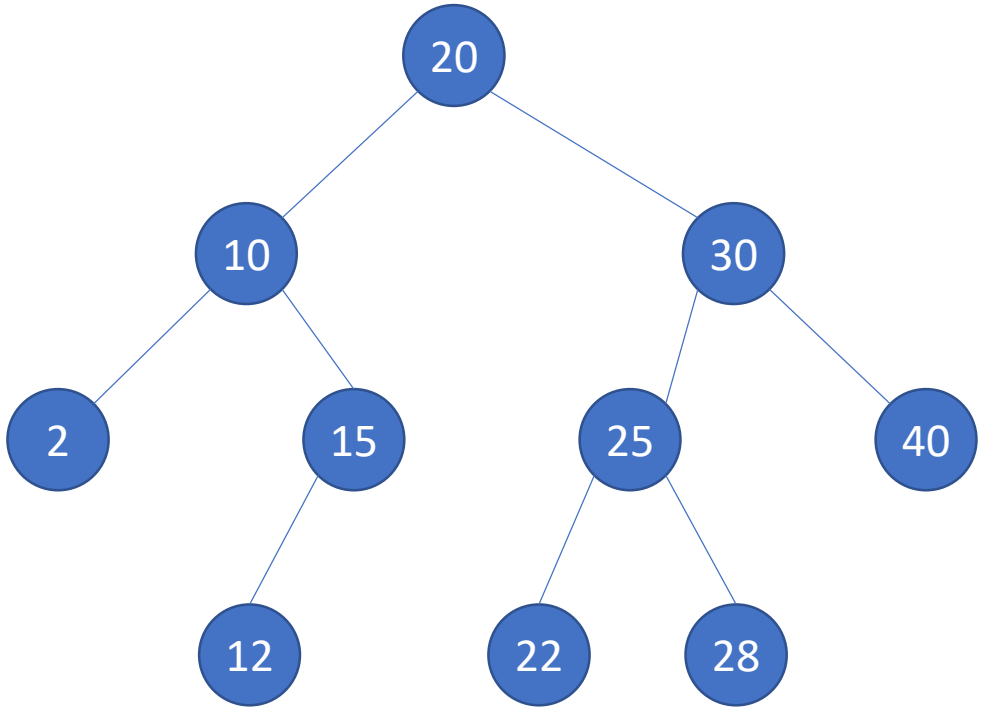
NỘI DUNG

- Cây nhị phân tìm kiếm
- Bài tập thao tác thêm khóa vào cây nhị phân tìm kiếm (P.06.14.01)
- Bài tập thao tác thêm, loại bỏ khóa khỏi cây nhị phân tìm kiếm (P.06.14.02)
- Bài tập dựng cây nhị phân tìm kiếm dựa vào dãy khóa duyệt theo thứ tự trước (P.06.14.03)

CÂY NHỊ PHÂN TÌM KIẾM

- Cây nhị phân tìm kiếm
  - Khóa của mỗi nút lớn hơn khóa tất cả các nút của cây con trái và nhỏ hơn khóa tất cả các nút của cây con phải
- Cấu trúc dữ liệu mỗi nút

```
struct Node {
    key // khóa của nút
    leftChild // con trỏ/tham chiếu đến nút con trái
    rightChild //con trỏ/tham chiếu đến nút con phải
}
```



CÂY NHỊ PHÂN TÌM KIẾM

- Tìm kiếm một khóa k trên cây nhị phân tìm kiếm
  - Nếu k bằng khóa của nút gốc thì trả về con trỏ đến nút gốc
  - Nếu k lớn hơn khóa của nút gốc thì tìm kiếm khóa k trên cây con phải (đệ quy)
  - Nếu k nhỏ hơn khóa của nút gốc thì tìm kiếm khóa k trên cây con trái (đệ quy)

```
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);
}
```

CÂY NHỊ PHÂN TÌM KIẾM

- Thêm một khóa k vào cây nhị phân tìm kiếm
  - Nếu cây là rỗng thì ta tạo mới nút có khóa bằng k và trả về con trỏ/tham chiếu đến nút mới tạo này
  - Nếu k bằng khóa của nút gốc thì trả về con trỏ đến nút gốc (khóa tồn tại thì không them nữa)
  - Nếu k lớn hơn khóa của nút gốc thì thêm khóa k vào cây con phải (đệ quy)
  - Nếu k nhỏ hơn khóa của nút gốc thì thêm khóa k vào cây con trái (đệ quy)

```
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;
}
```

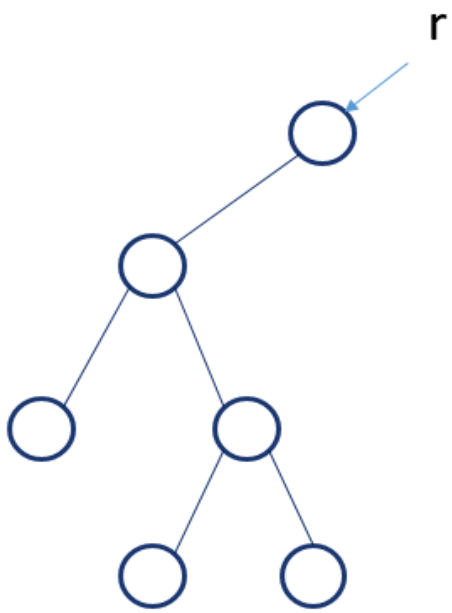
CÂY NHỊ PHÂN TÌM KIẾM

- Loại bỏ một nút có khóa bằng *k* khỏi cây nhị phân tìm kiếm
  - Nếu cây là rỗng thì ta trả về con trỏ NULL
  - Nếu *k* lớn hơn khóa của nút gốc thì thực hiện loại bỏ nút có khóa bằng k trên cây con phải (đệ quy)
  - Nếu *k* nhỏ hơn khóa của nút gốc thì thực hiện loại bỏ nút có khóa bằng k trên cây con trái (đệ quy)
  - Nếu *k* bằng khóa của nút gốc
    - Tìm nút có khóa lớn nhất trên cây con trái hoặc nút có khóa nhỏ nhất trên cây con phải để thay thế vào nút gốc

```
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;
}
```

CÂY NHỊ PHÂN TÌM KIẾM

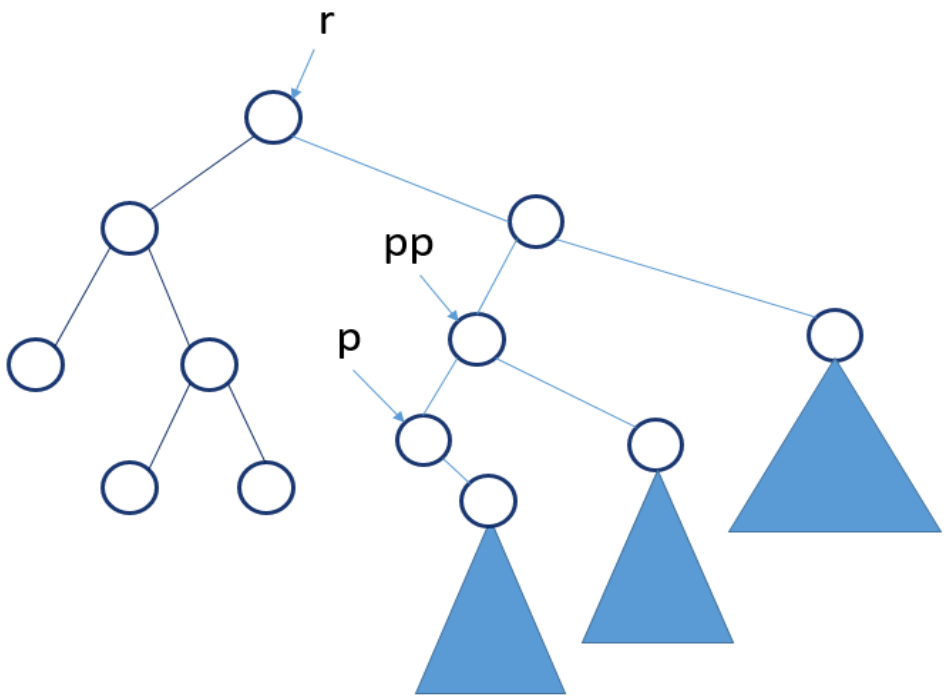
- Loại bỏ một nút gốc của một cây nhị phân tìm kiếm
  - Nếu nút gốc không có con phải thì trả về con trỏ/tham chiếu đến nút con trái




```
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;
}
```

CÂY NHỊ PHÂN TÌM KIẾM

- Loại bỏ một nút gốc của một cây nhị phân tìm kiếm
  - Nếu nút gốc có nút con phải thì tìm nút có khóa nhỏ nhất của cây con phải để thay thế nút gốc




```
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;
}
```

BÀI TẬP THAO TÁC THÊM KHÓA VÀO CÂY NHỊ PHÂN TÌM KIẾM (BM14.01)

- Cho một cây nhị phân tìm kiếm T (bắt đầu từ cây rỗng). Thực hiện 1 dãy các thao tác thêm khóa vào T và duyệt theo thứ tự trước, thứ tự sau trên T
  - insert k: thêm nút có khóa bằng k vào T (nếu nút có khóa k chưa tồn tại)
  - preorder: in dãy khóa được thăm bằng duyệt theo thứ tự trước trên T (sau mỗi khóa ghi ra 1 ký tự SPACE)
  - postorder: in dãy khóa được thăm bằng duyệt theo thứ tự sau trên T (sau mỗi khóa ghi ra 1 ký tự SPACE)
- Dữ liệu
  - Mỗi dòng chứa thông tin về 1 thao tác thuộc 1 trong 3 dạng trên
  - Dữ liệu vào kết thúc bởi dòng chứa #
- Kết quả
  - Ghi ra trên mỗi dòng, kết quả của 1 thao tác preorder và postorder gặp ở đầu vào

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

BÀI TẬP THAO TÁC THÊM KHÓA VÀO CÂY NHỊ PHÂN TÌM KIẾM - MÃ GIẢ

- Thuật toán
  - Mỗi bước lặp, đọc thông tin về 1 thao tác ở đầu vào và thực hiện thao tác chèn hoặc duyệt cây tương ứng.

```
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);
  }
}
```

BÀI TẬP THAO TÁC THÊM KHÓA VÀO CÂY NHỊ PHÂN TÌM KIẾM - CODE

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct Node{
    int key;
    struct Node* leftChild;
    struct Node* rightChild;
}Node;
Node* root;
Node* makeNode(int k){
    Node* p = (Node*)malloc(sizeof(Node));
    p->key = k; p->leftChild = NULL;
    p->rightChild = NULL;
    return p;
}
```

```c
Node* insert(Node* r, int k){
    if(r == NULL) r = makeNode(k);
    else if(r->key > k)
        r->leftChild = insert(r->leftChild, k);
    else if(r->key < k)
        r->rightChild = insert(r->rightChild,k);
    return r;
}
```

BÀI TẬP THAO TÁC THÊM KHÓA VÀO CÂY NHỊ PHÂN TÌM KIẾM - code

```
void preOrder(Node* r){
    if(r == NULL) return;
    printf("%d ",r->key);
    preOrder(r->leftChild);
    preOrder(r->rightChild);
}
void postOrder(Node*r){
    if(r == NULL) return;
    postOrder(r->leftChild);
    postOrder(r->rightChild);
    printf("%d ",r->key);
}
```

```
int main(){
    root = NULL;
    while(1){
        char cmd[256];
        scanf("%s",cmd);
        if(strcmp(cmd,"#") == 0) break;
        else if(strcmp(cmd,"insert") == 0){
            int k;  scanf("%d",&k);  root = insert(root,k);
        }else if(strcmp(cmd,"preorder")==0){
            preOrder(root);   printf("\n");
        }else if(strcmp(cmd,"postorder")==0){
            postOrder(root);   printf("\n");
        }
    }
    return 0;
}
```

BÀI TẬP THAO TÁC THÊM, LOẠI BỎ KHÓA VÀO CÂY NHỊ PHÂN TÌM KIẾM (P06.1402)

- Cho một cây nhị phân tìm kiếm *T* (bắt đầu từ cây rỗng). Thực hiện 1 dãy các thao tác thêm, loại bỏ khóa khỏi *T* và duyệt theo thứ tự trước, thứ tự sau trên *T*
  - insert *k*: thêm nút có khóa bằng *k* vào *T* (nếu nút có khóa *k* chưa tồn tại)
  - remove *k*: loại bỏ nút có khóa bằng *k* khỏi *T*
  - preorder: in dãy khóa được thăm bằng duyệt theo thứ tự trước trên *T* (sau mỗi khóa ghi ra 1 ký tự SPACE)
  - postorder: in dãy khóa được thăm bằng duyệt theo thứ tự sau trên *T* (sau mỗi khóa ghi ra 1 ký tự SPACE)
- Dữ liệu
  - Mỗi dòng chứa thông tin về 1 thao tác thuộc 1 trong 3 dạng trên
  - Dữ liệu vào kết thúc bởi dòng chứa #
- Kết quả
  - Ghi ra trên mỗi dòng, kết quả của 1 thao tác preorder và postorder gặp ở đầu vào

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

BÀI TẬP THAO TÁC THÊM, LOẠI BỎ KHÓA VÀO CÂY NHỊ PHÂN TÌM KIẾM – MÃ GIẢ

- Thuật toán
  - Mỗi bước lặp, đọc thông tin về 1 thao tác ở đầu vào và thực hiện thao tác chèn, loại bỏ hoặc duyệt cây tương ứng.

```
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);
  }
}
```

BÀI TẬP THAO TÁC THÊM, LOẠI BỎ KHÓA VÀO CÂY NHỊ PHÂN TÌM KIẾM – CODE

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct Node{
    int key;
    struct Node* leftChild;
    struct Node* rightChild;
}Node;
Node* root;
Node* makeNode(int k){
    Node* p = (Node*)malloc(sizeof(Node));
    p->key = k; p->leftChild = NULL;
    p->rightChild = NULL;
    return p;
}
```

```
Node* insert(Node* r, int k){
    if(r == NULL) r = makeNode(k);
    else if(r->key > k)
        r->leftChild = insert(r->leftChild, k);
    else if(r->key < k)
        r->rightChild = insert(r->rightChild,k);
    return r;
}
```

BÀI TẬP THAO TÁC THÊM, LOẠI BỎ KHÓA VÀO CÂY NHỊ PHÂN TÌM KIẾM – CODE

```
Node* removeRoot(Node* r){
  if(r == NULL) return NULL;
  if(r->rightChild == NULL){
    Node* tmp = r;
    r = r->leftChild; free(tmp); return r;
  }
  Node* p = r->rightChild; Node* pp = r;
  if(p->leftChild == NULL){
    r->key = p->key; Node* tmp = p;
    r->rightChild = p->rightChild;
    free(tmp); return r;
  }
  while(p->leftChild != NULL){ pp = p; p = p->leftChild; }
  pp->leftChild = p->rightChild; r->key = p->key; free(p);
  return r;
}
```

```
Node* removeNode(Node* r, int k){
  if(r == NULL) return NULL;
  if(r->key == k)
    return removeRoot(r);
  else if(r->key < k)
    r->rightChild = removeNode(r->rightChild,k);
  else
    r->leftChild = removeNode(r->leftChild,k);
  return r;
}
```

BÀI TẬP THAO TÁC THÊM, LOẠI BỎ KHÓA VÀO CÂY NHỊ PHÂN TÌM KIẾM – CODE

```
void preOrder(Node* r){
    if(r == NULL) return;
    printf("%d ",r->key);
    preOrder(r->leftChild);
    preOrder(r->rightChild);
}
void postOrder(Node*r){
    if(r == NULL) return;
    postOrder(r->leftChild);
    postOrder(r->rightChild);
    printf("%d ",r->key);
}
```

```
int main(){
  root = NULL;
  while(1){
    char cmd[256];
    scanf("%s",cmd);
    if(strcmp(cmd,"#") == 0) break;
    else if(strcmp(cmd,"insert") == 0){
      int k;  scanf("%d",&k);        root = insert(root,k);
    }else if(strcmp(cmd,"remove")==0){
      int k;  scanf("%d",&k); root = removeNode(root,k);
    }else if(strcmp(cmd,"preorder")==0){ preOrder(root);   printf("\n");}
    else if(strcmp(cmd,"postorder")==0){ postOrder(root);   printf("\n");}
  }
  return 0;
}
```

BÀI TẬP DỰNG CÂY NHỊ PHÂN TÌM KIẾM TỪ DÃY DUYỆT THỨ TỰ TRƯỚC (Đề 1 Lab 03)

- Cho một cây nhị phân tìm kiếm *T*, mỗi nút có khóa là 1 số nguyên dương. Biết dãy khóa của các nút được thăm bằng phép duyệt *T* theo thứ tự trước là $k_1, k_2, \ldots, k_n$. Hãy tìm dãy khóa của các nút được thăm bằng duyệt *T* theo thứ tự sau.
- Dữ liệu
  - Dòng 1 chứa số nguyên dương n (1 <= n <= 50000)
  - Dòng 2 chứa dãy $k_1, k_2, \ldots, k_n$ (1 <= k_i <= 1000000)
- Kết quả
  - Ghi ra dãy khóa các nút được thăm bằng phép duyệt *T* theo thứ tự sau, hoặc ghi ra NULL nếu *T* không tồn tại

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

BÀI TẬP DỰNG CÂY NHỊ PHÂN TÌM KIẾM TỪ DÃY DUYỆT THỨ TỰ TRƯỚC – MÃ GIẢ

- Thuật toán dựng cây T dựa vào dãy khóa duyệt theo thứ tự trước
  - Nút gốc có khóa là k_1
  - Tìm chỉ số i sao cho k_1 lớn hơn $k_2, k_3, \ldots, k_i$ và k_1 nhỏ hơn $k_{i+1}, k_{i+2}, \ldots, k_n$.
    - Nếu không tồn tại chỉ số i như vậy thì T không tồn tại
    - Ngược lại, ta dựng cây T theo nguyên tắc:
      - Tạo nút gốc với khóa k_1
      - Tạo cây con trái với dãy khóa $k_2, k_3, \ldots, k_i$ và cây con phải với dãy khóa $k_{i+1}, k_{i+2}, \ldots, k_n$.

```
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;
}
```

BÀI TẬP DỰNG CÂY NHỊ PHÂN TÌM KIẾM TỪ DÃY DUYỆT THỨ TỰ TRƯỚC – CODE

```
#include <stdio.h>
#include <stdlib.h>
#define N 100001
int n;
int a[N];
int ok = 1;
typedef struct Node{
        int id;
        struct Node* leftChild;
        struct Node* rightChild;
}Node;
Node* makeNode(int id){
  Node* p = (Node*)malloc(sizeof(Node));
  p->id = id; p->leftChild = NULL;
  p->rightChild = NULL;  return p;
}
```

```
Node* build(int start, int end){
  if(start > end) return NULL;
  if(start == end){ return makeNode(a[start]); }
  Node* r = makeNode(a[start]);
  int i = start + 1;
  while(i <= end && a[i] < a[start]) i++;
  for(int j = i; j <= end; j++)
    if(a[j] < a[start]){ ok = 0; break; }

  r->leftChild =  build(start+1,i-1);
  r->rightChild = build(i,end);
  return r;
}
```

BÀI TẬP DỰNG CÂY NHỊ PHÂN TÌM KIẾM TỪ DÃY DUYỆT THỨ TỰ TRƯỚC – CODE

```
void postOrder(Node* r){
  if(r == NULL) return;
  postOrder(r->leftChild);
  postOrder(r->rightChild);
  printf("%d ",r->id);
}
```

```
int main(){
  scanf("%d",&n);
  for(int i = 1; i <= n; i++) scanf("%d",&a[i]);
  ok = 1;
  Node* root = build(1,n);
  if(ok == 0){
    printf("NULL"); return;
  }
  postOrder(root); printf("\n");
  return 0;
}
```

HUST

hust.edu.vn fb.com/dhbkhn

THANK YOU !