

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

CON TRỎ - CẤP PHÁT ĐỘNG - ỨNG DỤNG



Nội dung

1. Biến và vùng nhớ RAM
2. Toán tử & và *
3. Khái niệm con trỏ
4. Cú pháp khai báo
5. Con trỏ có giá trị NULL
6. Cấp phát bộ nhớ
7. Con trỏ và hàm
8. Con trỏ và chuỗi
9. Con trỏ trong mảng 1 chiều
10. Con trỏ và cấu trúc
11. Bài tập



Ý tưởng: Đặt nghi vấn

Xem chương trình sau

```
#include <iostream>
using namespace std;

int main()
{
    char c = 'F';
    int x = 10;
    float y = 1.8;

    return 0;
}
```

Các biến **c**, **x**, **y** được quản lý
như thế nào?



Vùng nhớ RAM

Vùng nhớ RAM (Random Access Memory)

- RAM là vùng nhớ được sử dụng để lưu trữ các dữ liệu và chương trình đang hoạt động trong thời gian thực.
- Trong RAM có nhiều cell, mỗi cell trong RAM có một **địa chỉ** duy nhất để truy cập vào cell đó.
- Vùng nhớ RAM dùng để lưu trữ dữ liệu và dữ liệu có thể được truy cập bất kỳ vị trí nào trong RAM một cách ngẫu nhiên và **nhanh chóng**. Tức là có thể truy cập vào bất kỳ cell nào trong RAM một cách nhanh chóng.
- Nhưng, RAM là vùng nhớ tạm thời, tức là dữ liệu chỉ **tồn tại** trong khi máy tính **đang hoạt động** và bị **xóa** khi máy tính **tắt**.

Địa chỉ	cell
0xf919a0	
0xf919a1	1 byte
0xf919a2	
0xf919a3	
0xf919a4	
0xf919a5	
0xf919a6	
0xf919a7	
0xf919a8	
0xf919a9	
0xf919aa	
0xf919ab	
0xf919ac	
.....



Vùng nhớ RAM

Biến và vùng nhớ RAM

```
#include <iostream>
using namespace std;

int main()
{
    char c = 'F';
    int x = 10;
    float y = 1.8;

    return 0;
}
```

Địa chỉ	cell
0xf919a0	
0xf919a1	'F'
0xf919a2	
0xf919a3	10
0xf919a4	
0xf919a5	
0xf919a6	
0xf919a7	
0xf919a8	
0xf919a9	1.8
0xf919aa	
0xf919ab	
0xf919ac	
0xf919ad	
0xf919ae	
0xf919af	
.....

c

x

y

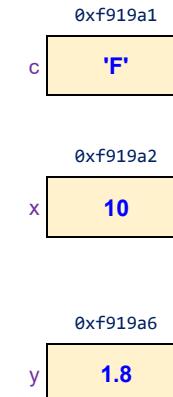
Địa chỉ	cell
0xf919a0	
0xf919a1	'F'
0xf919a2	
0xf919a3	10
0xf919a4	
0xf919a5	
0xf919a6	
0xf919a7	
0xf919a8	
0xf919a9	1.8
0xf919aa	
0xf919ab	
0xf919ac	
0xf919ad	
0xf919ae	
0xf919af	
.....

c

x

y

Cách vẽ gọn hơn



Toán tử & và *

Toán tử &

- Toán tử & dùng để lấy địa chỉ của biến.
- Phương pháp dùng:

`&tên_bien` x `value`

Toán tử *

- Toán tử * được sử dụng để truy cập vào giá trị mà một con trỏ đang trỏ đến.
- (Toán tử * dùng để truy cập giá trị mà một địa chỉ đang quản lý.)
- Phương pháp dùng:

`*địa_chỉ_bien`



Toán tử & và *

Minh họa

```
#include <iostream>
using namespace std;
int main()
{
    int x = 36;
    cout << "Gia tri bien x = " << x << endl;
    cout << "Dia chi bien x : " << &x << endl;
    printf("Dia chi bien x : %p", &x); printf("\n");
    printf("Dia chi bien x : %x", &x);

    return 0;
}
```

x `36`



Toán tử & và *

Minh họa

```
#include <iostream>
using namespace std;
int main()
{
    int x = 36;
    cout << "Gia tri bien x = " << x << endl;
    cout << "Dia chi bien x : " << &x << endl;
    cout << "Gia tri bien x = " << *(&x) << endl;
    return 0;
}
```

x 0xb369
36

Khái niệm con trỏ (pointer)

Con trỏ - pointer

- Con trỏ là một kiểu dữ liệu đặc biệt dùng để **lưu địa chỉ** trong bộ nhớ.
- Con trỏ cho phép truy cập và thay đổi dữ liệu của **biến** được quản lý bởi một **địa chỉ**.
- Sử dụng con trỏ thì cần cẩn thận và chắc chắn sử dụng chính xác để tránh các vấn đề liên quan đến quản lý bộ nhớ.



Cú pháp khai báo

Kiểu_Dữ_Liệu* ten_con_tro;

Trong đó:

- **Kiểu_Dữ_Liệu** là kiểu dữ liệu của biến mà con trỏ sẽ trỏ tới. Ví dụ: int, float, char, double... hoặc kiểu dữ liệu khác.
- *** gọi là toán tử con trỏ, đặt sau kiểu dữ liệu để khai báo là một con trỏ.**
- **ten_con_tro** tên của con trỏ.



Minh họa

Ví dụ khai báo con trỏ

```
int* iP;
float* fP;

int *p1, *p2, *p3;

SinhVien* pSV;
ToaDo* pTD;
```

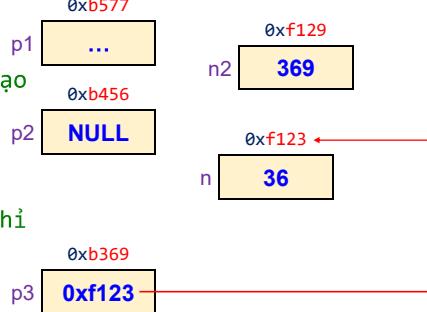


Con trỏ có giá trị NULL

- Khi khai báo kiểu con trỏ và không gán giá trị địa chỉ nào hết thì con trỏ đó chưa khởi tạo.
- Nếu gán giá trị **NULL** cho con trỏ thì gọi là **con trỏ NULL**, sẽ không trỏ vào địa chỉ của biến nào cả.

Ví dụ:

```
int* p1; //Con trỏ chưa khởi tạo
int* p2 = NULL; //Con trỏ NULL
int n = 36;
int* p3 = &n; //Con trỏ có địa chỉ
```



Cấu trúc Dữ liệu và Giải thuật

13



Tính kích thước: sizeof()

size hoặc sizeof()

- Để xác định kích thước của **kiểu dữ liệu** hoặc **biến** thì dùng size hoặc sizeof(). Đơn vị: byte
- Cú pháp:

```
sizeof(kiểu_dữ_liệu)
sizeof ten_bien
```

Ví dụ:

```
double x = 9.6;
double* p;

cout << sizeof(int) << endl;
cout << sizeof(double) << endl;
cout << sizeof x << endl;
cout << sizeof p << endl;
```

4
8
8
4

Kích thước của một con trỏ trong C++ có thể khác nhau trên các nền tảng và trình biên dịch khác nhau.
32-bit (4 byte)
64-bit (8 byte)

Cấu trúc Dữ liệu và Giải thuật

14



Khái niệm cấp phát bộ nhớ

Cấp phát bộ nhớ tĩnh (Static Memory Allocation)

- Được thực hiện bằng cách khai báo mảng hoặc biến với kích thước cụ thể tại thời điểm biên dịch.
- Các biến và mảng được cấp phát bộ nhớ tĩnh tồn tại trong suốt thời gian chạy chương trình và không thay đổi kích thước khi chạy.

Cấp phát động (Dynamic Memory Allocation)

- Cho phép cấp phát bộ nhớ trong quá trình chạy chương trình và giải phóng bộ nhớ khi không cần sử dụng nữa.

Cấu trúc Dữ liệu và Giải thuật

15



Phương pháp cấp nhát bộ nhớ

Ngôn ngữ	Cấp phát	Thu hồi – Giải phóng
C++	new	delete
C	malloc() calloc() hoặc realloc()	free()

Cấu trúc Dữ liệu và Giải thuật

16



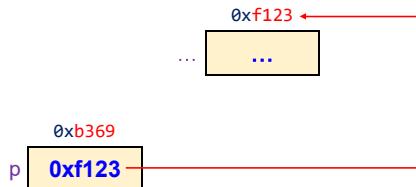
Cấp phát động

```
#include <iostream>
using namespace std;

int main()
{
    int* p = new int;
    if(p == NULL)
    {
        cout << "Cap phat khong thanh cong" << endl;
    }
    return 0;
}
```

Cấu trúc Dữ liệu và Giải thuật

17



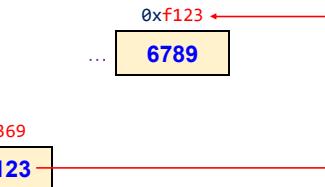
Cấp phát động

```
#include <iostream>
using namespace std;

int main()
{
    int* p = new int;
    if(p == NULL)
    {
        cout << "Cap phat khong thanh cong" << endl;
    }
    *p = 6789;
    return 0;
}
```

Cấu trúc Dữ liệu và Giải thuật

18



Vừa cấp phát động vừa khởi tạo giá trị

Cú pháp

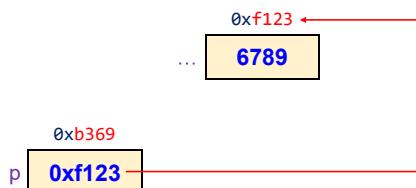
```
kiểu_dữ_liệu* ten_p = new kiểu_dữ_liệu(ghi_trị);
```

- Ví dụ:

```
int* p = new int(6789);
cout << *p; //6789
```

Cấu trúc Dữ liệu và Giải thuật

19



Thu hồi – Giải phóng

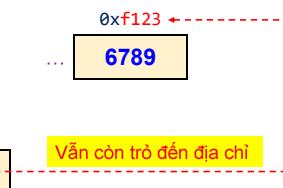
- Dùng toán tử `delete` để thu hồi vùng nhớ.
- Sau khi thu hồi vùng nhớ bằng `delete` thì con trỏ vẫn trỏ đến vùng nhớ.
- Có thể gọi tham chiếu nhưng kết quả không lường trước được.
- => Hiện tượng lac con trỏ.

```
int* p = new int(6789);
cout << "Gia tri p = " << p << endl;
cout << *p << endl << endl; //6789

delete p;
cout << "Gia tri p = " << p << endl;
cout << *p << endl; //Giá trị ...
```

Cấu trúc Dữ liệu và Giải thuật

20



Vẫn còn trỏ đến địa chỉ



Thu hồi – Giải phóng

Kết quả

Gia trị p = 0xe91970
6789

Gia trị p = 0xe91970
15302288

Gia trị p = 0x781970
6789

Gia trị p = 0x781970
7896720

Gia trị p = 0xf41970
6789

Gia trị p = 0xf41970
16023184



Thu hồi – Giải phóng

- Để tránh con trỏ bị thất lạc thì sau khi delete gán giá trị **NULL** cho con trỏ.
- Minh họa:

```
int* p = new int(6789);
cout << "Gia tri p = " << p << endl;
cout << *p << endl << endl; //6789
```

```
delete p;
p = NULL;
```

```
cout << "Gia tri p = " << p << endl;
//cout << *p << endl;
```

...
...

0xb369
p **NULL**



Con trỏ và hàm

- Con trỏ có thể dùng làm **tham số** của hàm hoặc **kiểu dữ liệu** của hàm.
- Ví dụ:

```
int* input()
{
    int* pi = new int;
    cout << "Nhập giá trị: ";
    cin >> *pi;

    return pi;
}

int main()
{
    int* p = input();
    cout << "Giá trị = " << *p << endl;
    return 0;
}
```



Con trỏ và hàm

- Ví dụ

```
void hoan_vị(int* a, int* b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}
int main()
{
    int x = 3, y = 6;
    cout << "Trước hoán vị. x = " << x << "    y = " << y << endl;

    hoan_vị(&x, &y);
    cout << "Sau hoán vị. x = " << x << "    y = " << y << endl;

    return 0;
}
```



Con trỏ và mảng 1 chiều

Các địa chỉ trong mảng 1 chiều

```
int a[] = {5, 3, 7, 5, 9, 1};
cout << "Địa chỉ mang: " << a << endl;
for(int i = 0; i < 6; i++)
{
    cout << i << " " << &a[i] << endl;
}

double b[] = {5.1, 3.6, 7.9, 5.3, 9.3, 1.9};
cout << "Địa chỉ mang: " << b << endl;
for(int i = 0; i < 6; i++)
{
    cout << i << " " << &b[i] << endl;
}
```

Địa chỉ mang: 0x61fea0

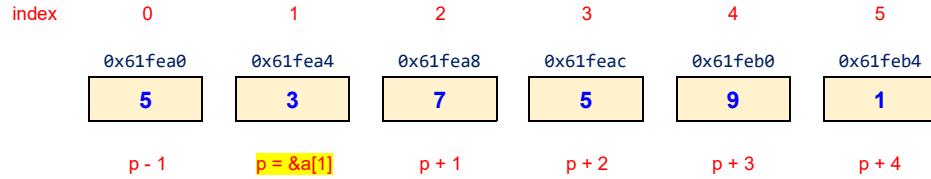
0	0x61fea0
1	0x61fea4
2	0x61fea8
3	0x61feac
4	0x61feb0
5	0x61feb4

Địa chỉ mang: 0x61fe70

0	0x61fe70
1	0x61fe78
2	0x61fe80
3	0x61fe88
4	0x61fe90
5	0x61fe98

Con trỏ và mảng 1 chiều

```
int a[] = {5, 3, 7, 5, 9, 1};
```



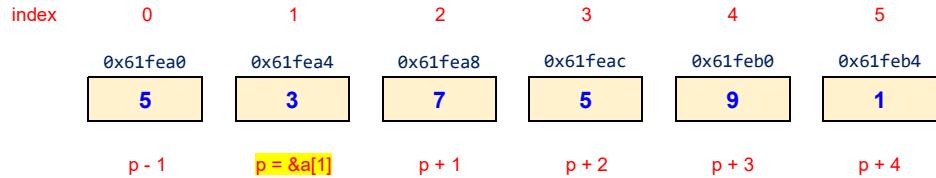
Phép cộng và trừ hai địa chỉ con trỏ

- Khi con trỏ **cộng/trừ** với một **số nguyên**, **địa chỉ** của con trỏ **sẽ tăng/giảm** lên một số lượng bộ nhớ tương ứng với **số nguyên nhân** vào kích thước của **kiểu dữ liệu** mà con trỏ đang trỏ tới.



Con trỏ và mảng 1 chiều

```
int a[] = {5, 3, 7, 5, 9, 1};
```

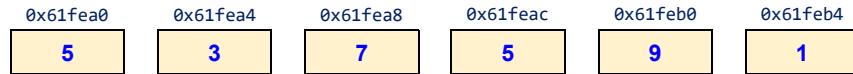


Ví dụ:

```
int a[] = {5, 3, 7, 5, 9, 1};
int* p = &a[1];
cout << *(p - 1) << endl;
cout << *(p + 1) << endl;
```

Con trỏ và mảng 1 chiều

```
int a[] = {5, 3, 7, 5, 9, 1};
```



```
for(int i = 0; i < 6; i++)
{
    cout << a + i << endl;
}
```

```
for(int i = 0; i < 6; i++)
{
    cout << *(a + i) << endl;
}
```



Xuất mảng

- Hàm xuất mảng áp dụng kỹ thuật con trỏ

```
void xuat(int* a, int n)
{
    for(int i = 0; i < n; i++)
    {
        cout << *(a + i) << endl;
    }
}
```

```
int main()
{
    int a[] = {5, 3, 7, 5, 9, 1};
    int n = 6;

    xuat(a, n);

    return 0;
}
```

Khởi tạo giá trị mảng

- Hàm khởi tạo giá trị mảng áp dụng kỹ thuật con trỏ

```
void khoiTao(int* a, int& n)
{
    n = 6;
    *(a + 0) = 3;
    *(a + 1) = 1;
    *(a + 2) = 6;
    *(a + 3) = 9;
    *(a + 4) = 7;
    *(a + 5) = 8;
}
```

```
int main()
{
    int a[100];
    int n = 0;

    khoiTao(a, n);
    xuat(a, n);

    return 0;
}
```



Khởi tạo giá trị mảng

- Hàm khởi tạo giá trị mảng áp dụng kỹ thuật con trỏ

```
void nhap(int* a, int& n)
{
    cout << "So luong can nhap: ";
    cin >> n;
    for(int i = 0; i < n; i++)
    {
        cout << "\tNhập a[" << i << "] = ";
        cin >> *(a + i);
    }
}
```

```
int main()
{
    int a[100];
    int n = 0;

    nhap(a, n);
    xuat(a, n);

    return 0;
}
```



Con trỏ và mảng 1 chiều

Cú pháp cấp phát bộ nhớ cho mảng 1 chiều

Kiểu_Dữ_Liệu* ten_mang = new Kiểu_Dữ_Liệu[kích_thước];

- Ví dụ:

```
int main()
{
    int* a = new int[100];
    int n = 0;

    khoiTao(a, n);
    xuat(a, n);
    return 0;
}
```



Tính tổng: áp dụng con trỏ mảng

Yêu cầu: Viết hàm tính tổng các giá trị trong mảng?

Phân tích:

- Input:
 - Mảng a => `int* a`
 - Số lượng n => `int n`
- Output:
 - Tổng các giá trị => return giá trị

```
int tinhTong(int* a, int n)
{
    int t = 0;
    for(int i = 0; i < n; i++)
    {
        t = t + *(a + i);
    }
    return t;
}
```



Bài tập nhỏ

```
#include <iostream>
using namespace std;
int main()
{
    int a[] = {1, 3, 5, 7, 9, 2, 4, 6, 8};
    cout << *(a + 3) + *(a + 7);
    return 0;
}
```

Nguồn: Đề thi



Con trỏ và cấu trúc

Khai báo

```
Kiểu_Cấu_Trúc* ten_bien;
```

Cấp phát

```
ten_bien = new Kiểu_Cấu_Trúc();
```

Vừa khai báo vừa cấp phát

```
Kiểu_Cấu_Trúc* ten_bien = new Kiểu_Cấu_Trúc();
```



Truy xuất thuộc tính

Phương pháp

```
ten_bien->thuoc_tinh
```

• Ví dụ:

```
struct Diem
{
    int x;
    int y;
};
int main()
{
    Diem a = {3, 5};
    Diem* p = &a;

    cout << "Hoanh do x = " << p->x << endl;
    cout << "Tung do y = " << p->y << endl;
    return 0;
}
```



Truy xuất thuộc tính

Minh họa

```
void xuat(Diem d)
{
    cout << "(" << d.x << " ; " << d.y << ")\n";
}

Diem* b = new Diem;
b->x = 6;
b->y = 8;

xuat(*b);

Diem* c = new Diem;
(*c).x = 7;
(*c).y = 9;

xuat(*c);
```

Cấu trúc Dữ liệu và Giải thuật

37

Vừa cấp phát vừa khởi tạo

Minh họa

```
Diem* d = new Diem({1, 3});

xuat(*d);
```

Cấu trúc Dữ liệu và Giải thuật

38



Cấp phát mảng cấu trúc

Minh họa

```
void khoiTao(Diem* d, int& n)
{
    n = 3;
    *(d + 0) = {1, 2};
    *(d + 1) = {2, 6};
    *(d + 2) = {3, 1};
}
```

```
void xuatMang(Diem* d, int n)
{
    for(int i = 0; i < n; i++)
    {
        xuat(*(d + i));
    }
}
```

Cấu trúc Dữ liệu và Giải thuật

39



Đệ quy tự trỏ

Minh họa

```
int main()
{
    Diem* arr = new Diem[100];
    int n = 0;

    khoiTao(arr, n);
    xuatMang(arr, n);

    return 0;
}
```

```
struct Node
{
    int data;
    Node* pNext;
};
```

```
struct NODE
{
    int data;
    NODE* pLeft;
    NODE* pRight;
};
```

Cấu trúc Dữ liệu và Giải thuật

40



Áp dụng con trỏ cho mảng 1 chiều

BÀI 1

Xây dựng mảng 1 chiều lưu trữ các số thực. Sau đó, viết hàm thực hiện các chức năng sau:

1. Tạo giá trị ngẫu nhiên (số thực, 2 chữ số sau dấu chấm) thuộc khoảng (-268; 339). Số lượng thuộc đoạn [10; 20]
2. Xuất (in) mảng vừa tạo.
3. Tính tổng các giá trị có trong mảng.
4. Tính tổng, tích các giá trị có trong mảng (viết 1 hàm).
5. Đếm số lần xuất hiện 1 phần tử x bất kỳ.



Áp dụng con trỏ cho mảng 1 chiều

BÀI 1 (Tiếp theo)

Xây dựng mảng 1 chiều lưu trữ các số thực. Sau đó, viết hàm thực hiện các chức năng sau:

6. Trả về các phần tử chẵn (chẵn-lẻ dựa vào giá trị nguyên của số thực) có trong mảng.
7. Tìm vị trí phần tử nhỏ nhất trong mảng.
8. Tạo một mảng đảo ngược.
9. Trả về các số âm trong mảng.
10. Trả về số lượng các giá trị trong mảng một chiều thuộc đoạn [x, y].
11. Trả về các giá trị trong mảng một chiều thuộc đoạn [x, y].



Bài tập

Bài 2

Áp dụng kỹ thuật cấp phát động cho mảng một chiều. Viết HÀM thực hiện các yêu cầu sau:

1. Tạo mảng số thực có các giá trị sau: | 0.8 | 5.6 | 9.1 | 7.3 | 10 | 5.9 | 7.2 | 9.3 | 8.0 | 8.7 |
2. Xuất mảng (in) ra màn hình.
3. Kiểm tra mảng có toàn dương.
6. Kiểm tra mảng có đối xứng.
7. Kiểm tra trong mảng có chứa số nguyên tố, số chính phương.
8. Tìm một giá trị có trong mảng.



Bài tập

Bài 2 (tiếp theo...)

9. Tìm các số nguyên tố có trong mảng.
10. Tìm các số chính phương có trong mảng.
11. Trả về các số âm trong mảng.
12. Tìm các số dương trong mảng và tính tổng của chúng.
13. Đảo ngược các phần tử trong mảng.



Bài tập

Bài 3

Viết chương trình quản lý và tính tiền lương cho các nhân viên trong một công ty. Nhân viên có các thông tin sau: mã nhân viên, họ tên, lương cơ bản, số ngày làm và lương hằng tháng.

Công thức tính tiền lương hằng tháng cho nhân viên như sau:

Lương hằng tháng = lương căn bản + số ngày làm việc * 180.000 đ

Nếu lương hằng tháng hơn 8.000.000 thì thưởng thêm 5%.

Nếu lương cơ bản dưới 5.000.000 thì phụ cấp thêm 10%.

Áp dụng cấp phát động. Xây dựng chương trình đáp ứng các yêu cầu sau:

1. Tạo dữ liệu các nhân viên dùng để kiểm thử (10 nhân viên)
2. Xuất danh sách nhân viên trong công ty.

Bài tập

Bài 3 (tiếp theo...)

Chương trình đáp ứng các yêu cầu sau:

3. Tính và cập nhật tiền lương cho các loại nhân viên.
4. Tính tổng lương hằng tháng mà công ty phải trả cho toàn bộ nhân viên.
5. Tính tổng lương hằng tháng các nhân viên có lương cơ bản < 5 triệu.
6. Tìm nhân viên theo theo mã nhân viên.
7. Tìm các nhân viên có lương hằng tháng thấp nhất.
8. Tìm các nhân viên có lương cơ bản cao nhất cao nhất.



Hỏi - Đáp



CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

DANH SÁCH LIÊN KẾT ĐƠN



Nội dung

1. Ý tưởng – Giải pháp
 2. Kiến trúc danh sách liên kết đơn
 3. Các bước xây dựng
 4. Các thao tác
 5. Bài tập

Cấu trúc Dữ liệu và Giải thuật

2

Ý tưởng

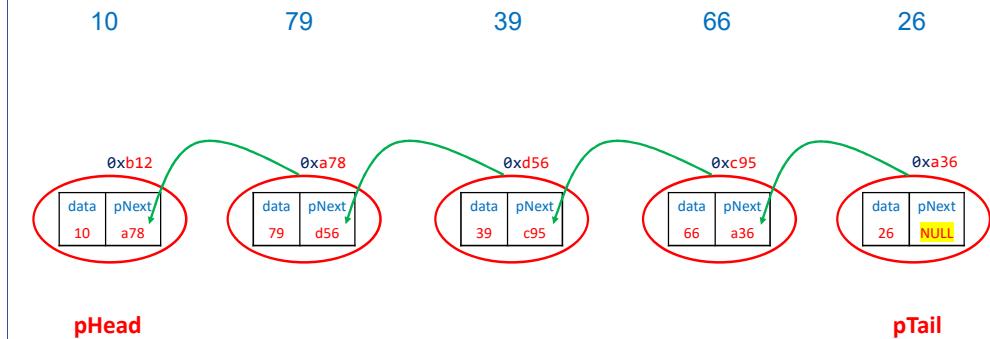
Vấn đề 1: Muốn lưu trữ các số:	10	79	39	66	26	50
• Chúng ta làm sao?	x	y	z	t	g	h

Vấn đề 2: Muốn lưu trữ nhiều số hơn thì làm sao?

index	0	1	2	3	4	5	6	...	499
value	10	79	39	66	26	50		...	

- Có giải pháp nào tốt hơn nữa không?

Kiến trúc



Cấu trúc Dữ liệu và Giải thuật

3

Cấu trúc Dữ liệu và Giải thuật

4

Các bước xây dựng

Bước #1: Tạo cấu trúc Node



```
struct Node{
    int data;
    Node* pNext;
};
```

Bước #2: Khởi tạo Node từ value
(Viết hàm chuyển value thành Node)

0xb12



```
Node* initNode(int value)
{
    Node* p = new Node;
    p->data = value;
    p->pNext = NULL;
    return p;
}
```

Bước #3: Tạo cấu trúc LIST



Bước #4: Khởi tạo LIST

```
void initList(List& l)
{
    l.pHead = NULL;
    l.pTail = NULL;
}
```

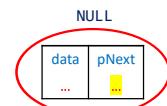
Cấu trúc Dữ liệu và Giải thuật

5

Khai báo – Khởi tạo Node và List

- Khai báo danh sách và khởi tạo danh sách

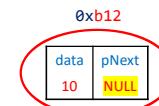
```
List l;
initList(l);
```



pHead => NULL
pTail => NULL

- Khai báo Node và khởi tạo Node

```
Node* p1 = initNode(10);
Node* p2 = initNode(79);
Node* p3 = initNode(39);
```



Cấu trúc Dữ liệu và Giải thuật

6

Thêm Node vào List

10 79 39

Node* p = initNode(.....);

```
addTail(l, p);
addHead(l, p);
```

10

Cấu trúc Dữ liệu và Giải thuật

7

Thêm Node vào List

10 79 39

Node* p = initNode(.....);

addHead(l, p);

79 —> 10

Cấu trúc Dữ liệu và Giải thuật

8

Thêm Node vào List

10 79 39
Node* p = initNode(.....);

addTail(l, p);



Cấu trúc Dữ liệu và Giải thuật

9

Thêm Node vào List

10 79 39 50
Node* p = initNode(.....);

addHead(l, p);



Cấu trúc Dữ liệu và Giải thuật

10

Thêm Node vào List

10 79 39 50 26
Node* p = initNode(.....);

addTail(l, p);



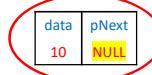
Cấu trúc Dữ liệu và Giải thuật

11

Thêm Node tại vị trí đầu List

10 79 39
Node* p = initNode(.....);

P => 0xd56



NULL



pHead => NULL
pTail => NULL

```
void addHead(List& l, Node* p)
{
    if(l.pTail == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        p->pNext = l.pHead;
        l.pHead = p;
    }
}
```

Cấu trúc Dữ liệu và Giải thuật

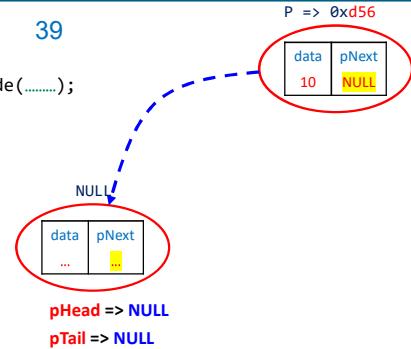
12



Thêm Node tại vị trí đầu List

10 79 39

Node* p = initNode(.....);



```
void addHead(List& l, Node* p)
{
    if(l.pTail == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        p->pNext = l.pHead;
        l.pHead = p;
    }
}
```

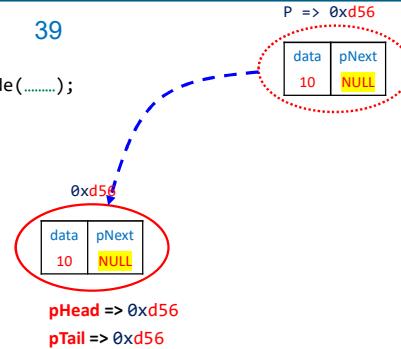
Cấu trúc Dữ liệu và Giải thuật

13

Thêm Node tại vị trí đầu List

10 79 39

Node* p = initNode(.....);



```
void addHead(List& l, Node* p)
{
    if(l.pTail == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        p->pNext = l.pHead;
        l.pHead = p;
    }
}
```

Cấu trúc Dữ liệu và Giải thuật

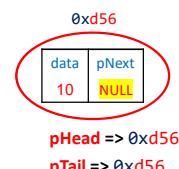
14



Thêm Node tại vị trí đầu List

10 79 39

Node* p = initNode(.....);



```
void addHead(List& l, Node* p)
{
    if(l.pTail == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        p->pNext = l.pHead;
        l.pHead = p;
    }
}
```

Cấu trúc Dữ liệu và Giải thuật

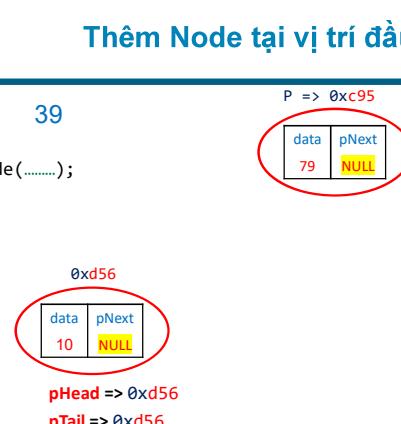
15



Thêm Node tại vị trí đầu List

10 79 39

Node* p = initNode(.....);



```
void addHead(List& l, Node* p)
{
    if(l.pTail == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        p->pNext = l.pHead;
        l.pHead = p;
    }
}
```

Cấu trúc Dữ liệu và Giải thuật

16



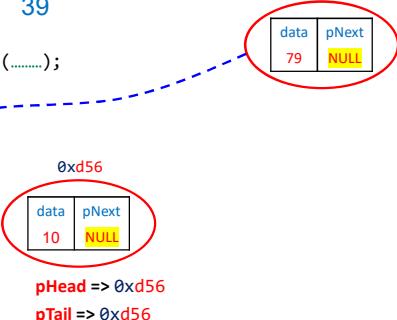
Thêm Node tại vị trí đầu List

10 79 39

Node* p = initNode(.....);

addHead(l, p);
pHead => 0xd56
pTail => 0xd56

P => 0xc95



```
void addHead(List& l, Node* p)
{
    if(l.pTail == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        p->pNext = l.pHead;
        l.pHead = p;
    }
}
```

Thêm Node tại vị trí đầu List

10 79 39

Node* p = initNode(.....);
addHead(l, p);

0xc95
data | pNext
79 | NULL
pHead => 0xd56
pTail => 0xd56

0xd56
data | pNext
10 | NULL

```
void addHead(List& l, Node* p)
{
    if(l.pTail == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        p->pNext = l.pHead;
        l.pHead = p;
    }
}
```



Thêm Node tại vị trí đầu List

10 79 39 Node* p = initNode(.....);

addHead(l, p);

0xc95
data | pNext
79 | NULL
pHead => ...?... ←----- pHead => 0xd56
pTail => 0xd56

```
void addHead(List& l, Node* p)
{
    if(l.pTail == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        p->pNext = l.pHead;
        l.pHead = p;
    }
}
```

Thêm Node tại vị trí đầu List

10 79 39 Node* p = initNode(.....);
addHead(l, p);

0xc95
data | pNext
79 | d56
pHead => 0xc95

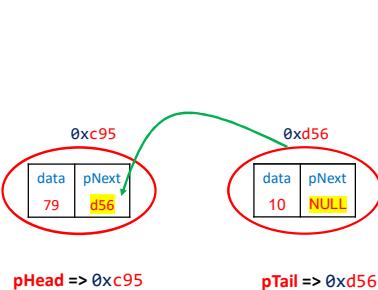
0xd56
data | pNext
10 | NULL
pTail => 0xd56

```
void addHead(List& l, Node* p)
{
    if(l.pTail == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        p->pNext = l.pHead;
        l.pHead = p;
    }
}
```



Thêm Node tại vị trí cuối List

10 79 39 Node* p = initNode(.....); `0xa36`



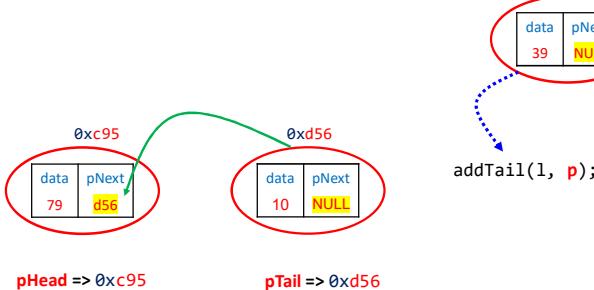
```
void addTail(List& l, Node* p)
{
    if(l.pHead == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        l.pTail->pNext = p;
        l.pTail = p;
    }
}
```

Cấu trúc Dữ liệu và Giải thuật

21

Thêm Node tại vị trí cuối List

10 79 39 Node* p = initNode(.....); `0xa36`



```
void addTail(List& l, Node* p)
{
    if(l.pHead == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        l.pTail->pNext = p;
        l.pTail = p;
    }
}
```

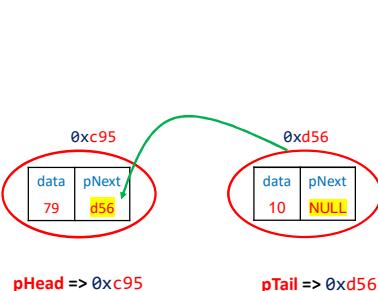
Cấu trúc Dữ liệu và Giải thuật

22



Thêm Node tại vị trí cuối List

10 79 39 Node* p = initNode(.....); `0xa36`



```
void addTail(List& l, Node* p)
{
    if(l.pHead == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        l.pTail->pNext = p;
        l.pTail = p;
    }
}
```

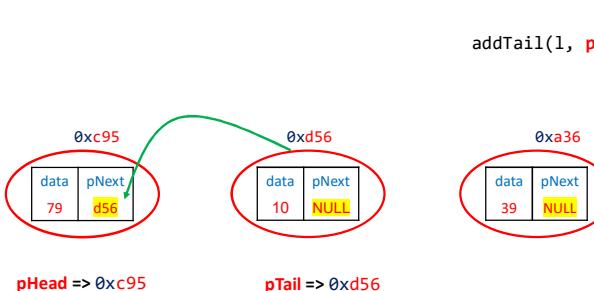
Cấu trúc Dữ liệu và Giải thuật

23

Thêm Node tại vị trí cuối List

10 79 39 Node* p = initNode(.....);

addTail(l, p);



```
void addTail(List& l, Node* p)
{
    if(l.pHead == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        l.pTail->pNext = p;
        l.pTail = p;
    }
}
```

Cấu trúc Dữ liệu và Giải thuật

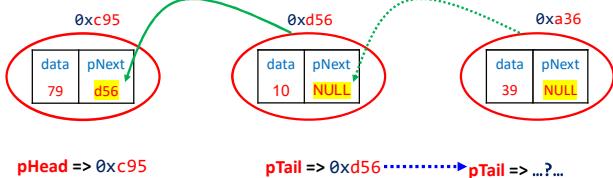
24



Thêm Node tại vị trí cuối List

10 79 39 Node* p = initNode(.....);

addTail(l, p);



```
void addTail(List& l, Node* p)
{
    if(l.pHead == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        l.pTail->pNext = p;
        l.pTail = p;
    }
}
```

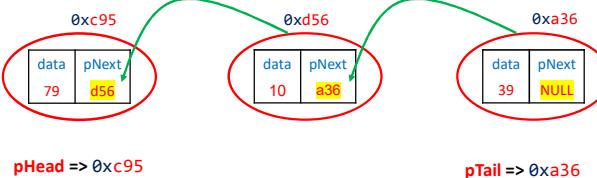
Cấu trúc Dữ liệu và Giải thuật

25

Thêm Node tại vị trí cuối List

10 79 39 Node* p = initNode(.....);

addTail(l, p);



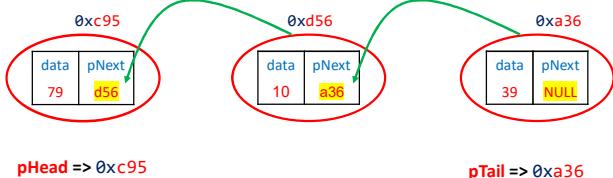
```
void addTail(List& l, Node* p)
{
    if(l.pHead == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        l.pTail->pNext = p;
        l.pTail = p;
    }
}
```

Cấu trúc Dữ liệu và Giải thuật

26



Thêm Node tại vị trí cuối List

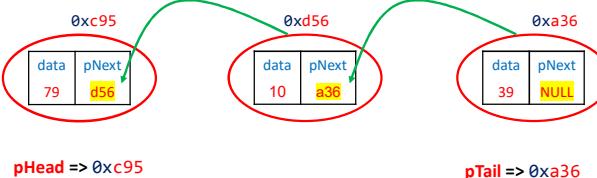


```
void addTail(List& l, Node* p)
{
    if(l.pHead == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        l.pTail->pNext = p;
        l.pTail = p;
    }
}
```

Cấu trúc Dữ liệu và Giải thuật

27

In danh sách



```
void printWh(List l)
{
    Node* p = l.pHead;
    while(p!=NULL)
    {
        cout << p->data << "\t";
        p = p->pNext;
    }
}
```

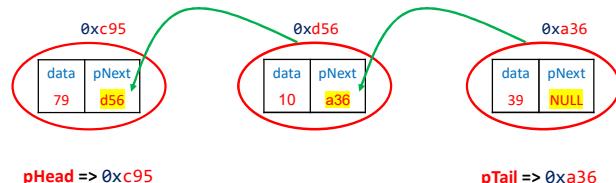
Cấu trúc Dữ liệu và Giải thuật

28



In danh sách

```
void printList(List l)
{
    for(Node* p = l.pHead; p != NULL; p = p->pNext)
    {
        cout << p->data << "\t";
    }
}
```



Cấu trúc Dữ liệu và Giải thuật

29



Viết hàm main

```
int main()
{
    List l;
    initList(l);

    Node* p = initNode(10);
    addHead(l, p);
    addHead(l, initNode(79));
    addTail(l, initNode(39));

    cout << "In danh sach:\n";
    printList(l); //79 10 39

    cout << endl;
    printWh(l);
    return 0;
}
```

In danh sach:
79 10 39
79 10 39

Cấu trúc Dữ liệu và Giải thuật

30



Tìm kiếm

Yêu cầu: Viết hàm tìm kiếm một giá trị có trong List hay không?

Phân tích:

- Input:
 - List => List l
 - Giá trị cần tìm => int value
- Output:
 - True/false => return bool
- Mở rộng: Trả về địa chỉ

```
bool timGiaTri(List l, int value)
{
    for(Node* p = l.pHead; p != NULL; p = p->pNext)
    {
        if(value == p->data)
        {
            return true;
        }
    }

    return false;
}
```

Cấu trúc Dữ liệu và Giải thuật

31

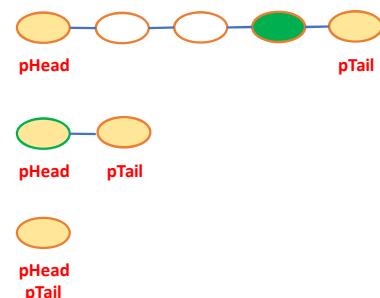


Tìm kiếm

Yêu cầu: Viết hàm tìm giá trị node kế cuối?

Phân tích:

- Input:
 - List => List l
- Output:
 - Có hay không => return bool
 - Giá trị tìm thấy => giá trị



- Mở rộng: Trả về địa chỉ

Cấu trúc Dữ liệu và Giải thuật

32



Kỹ thuật đếm

Yêu cầu: Viết hàm đếm các giá trị chẵn có trong List?

Phân tích:

- Input:
 - List \Rightarrow List l
- Output:
 - Số lượng phần tử chẵn \Rightarrow return int



Tìm giá trị lớn nhất – nhỏ nhất

Yêu cầu: Viết hàm tìm giá trị lớn nhất trong List?

Phân tích:

- Input:
 - List \Rightarrow List l
- Output:
 - Giá trị lớn nhất \Rightarrow return giá trị

```
int timGiaTriMax(List l)
{
    int m = ...giá trị đầu tiên...
    for(Node* p = l.pHead; p != NULL; p = p->pNext)
    {
        if(.....)
        {
            .....
        }
    }
    return m;
}
```

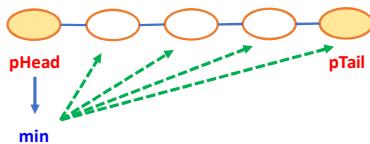


Tìm giá trị lớn nhất – nhỏ nhất

Yêu cầu: Viết hàm tìm giá trị nhỏ nhất trong List?

Phân tích:

- Input:
 - List \Rightarrow List l
- Output:
 - Giá trị lớn nhất \Rightarrow return giá trị



\Rightarrow Phương pháp tương tự tìm max.



Tính tổng

Yêu cầu: Viết hàm tính tổng các giá trị trong List?

Phân tích:

- Input:
 - List \Rightarrow List l
- Output:
 - Tổng các giá trị \Rightarrow return giá trị

```
int tinh(List l)
{
    int t = 0;
    for(Node* p = l.pHead; p != NULL; p = p->pNext)
    {
        t = //công thức tính tổng
    }
    return t;
}
```



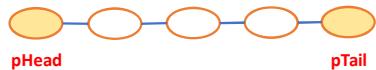
Xóa Node tại vị trí đầu List

Yêu cầu: Viết hàm xóa Node đầu tiên trong List?

Phân tích:

- Input:

- List => List



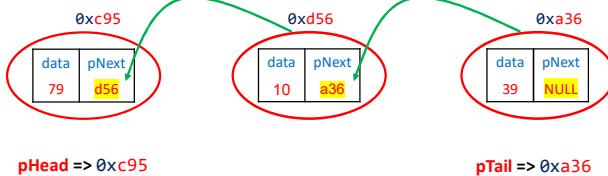
- Output:

- List đã xóa => List&
- Kết quả xóa => bool
- Giá trị đã xóa =>

```
int removeHead(List& l)
{
    return value;
}
```

Xóa Node tại vị trí đầu List

Minh họa



pHead => 0xc95

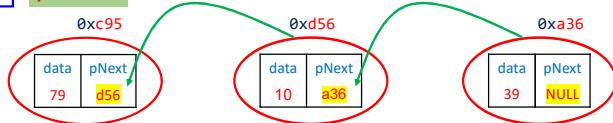
pTail => 0xa36



Xóa Node tại vị trí đầu List

Minh họa

#1 p => 0xc95



pHead => 0xc95

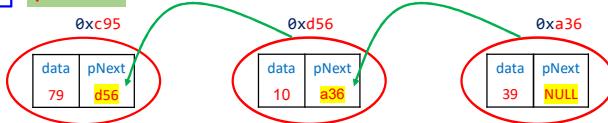
pTail => 0xa36



Xóa Node tại vị trí đầu List

Minh họa

#1 p => 0xc95



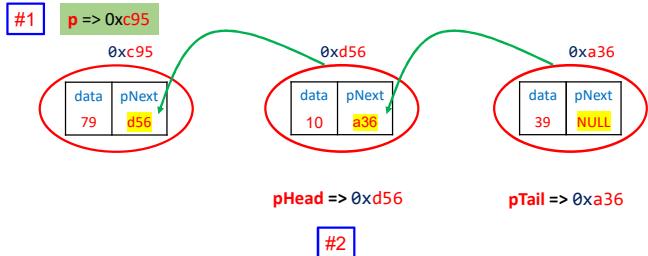
pHead => 0xc95 + pHead => #2

pTail => 0xa36



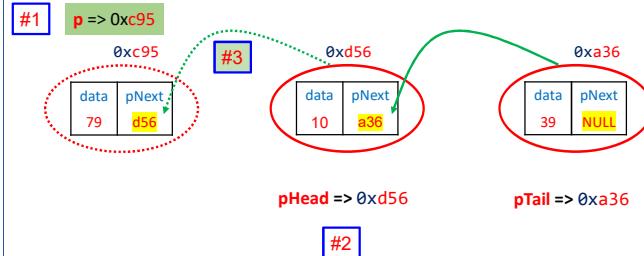
Xóa Node tại vị trí đầu List

Minh họa



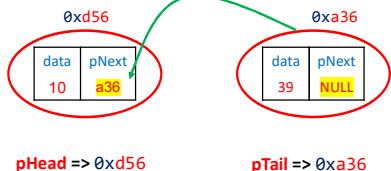
Xóa Node tại vị trí đầu List

Minh họa



Xóa Node tại vị trí đầu List

Minh họa



Xóa Node tại vị trí cuối List

Yêu cầu: Viết hàm xóa Node cuối trong List?

Phân tích:

- Input:
 - List => List
- Output:
 - List đã xóa => List&
 - Giá trị đã xóa => return

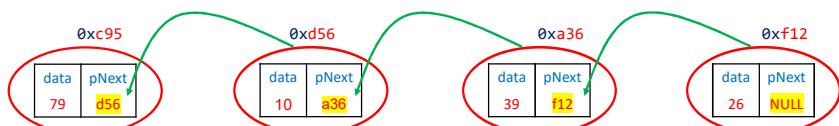
```
int removeTail(List& l)
{
    // Implementation

    return value;
}
```



Xóa Node tại vị trí cuối List

Minh họa



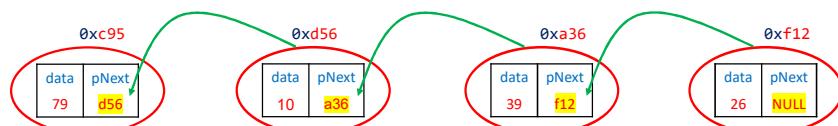
pHead => 0xc95

pTail => 0xf12



Xóa Node tại vị trí cuối List

Minh họa



pHead => 0xc95

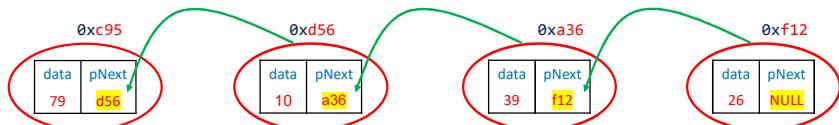
pTail => 0xf12



Xóa Node tại vị trí cuối List

Minh họa

#1
q => 0x...?....
#2
p => 0xf12



pHead => 0xc95

pTail => 0xf12



Xóa Node tại vị trí cuối List

Minh họa



pHead => 0xc95

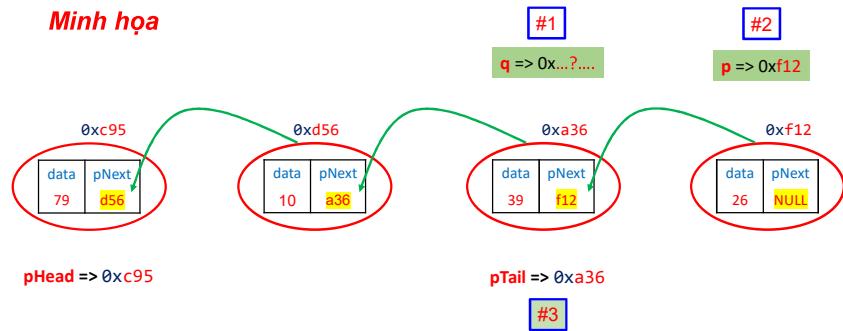
pTail => 0x...?... ← pTail => 0xf12

#3



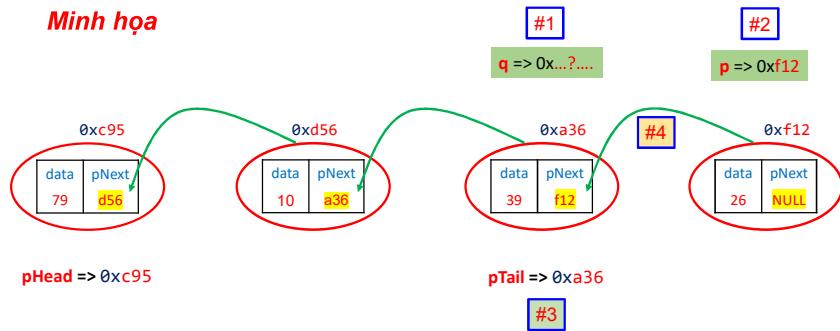
Xóa Node tại vị trí cuối List

Minh họa



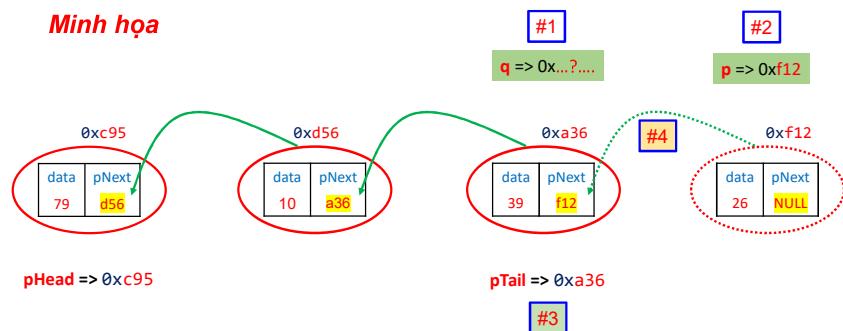
Xóa Node tại vị trí cuối List

Minh họa



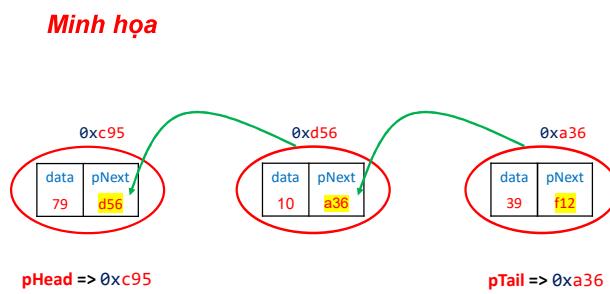
Xóa Node tại vị trí cuối List

Minh họa



Xóa Node tại vị trí cuối List

Minh họa





Bài tập

Viết chương trình cài đặt danh sách liên kết đơn quản lý các số nguyên.

Thực hiện theo yêu cầu sau:

1. Xây dựng cấu trúc node và cấu trúc danh sách liên kết đơn.
2. Viết hàm khởi tạo danh sách liên kết đơn và hàm khởi tạo địa chỉ node từ một số nguyên.
3. Viết hàm chèn node vào đầu danh sách.
4. Viết hàm chèn node vào cuối danh sách.
5. Viết hàm khởi tạo giá trị cho danh sách từ mảng một chiều.
6. Viết hàm in giá trị danh sách đã nhập.
7. Viết hàm in giá trị kèm địa chỉ của từng node trong danh sách.
8. Viết hàm kiểm tra danh sách có rỗng hay không.



Bài tập

Tiếp theo...

9. Viết hàm trả về node thứ n trong danh sách.
10. Viết hàm tìm kiếm một node có giá trị X trong danh sách.
11. Viết hàm tìm kiếm các địa chỉ node có giá trị X trong danh sách.
12. Viết hàm đếm số lượng các node có giá trị âm, dương.
13. Viết hàm tìm node có giá trị âm lớn nhất, tìm node có giá trị lẻ nhỏ nhất trong danh sách.
14. Viết hàm cập nhật giá trị cho một node có giá trị là X bằng giá trị Y mới.
15. Viết hàm copy danh sách.



Bài tập

Viết chương trình cài đặt danh sách liên kết đơn quản lý sinh viên với các thao tác như sau:

1. Xây dựng cấu trúc dữ liệu có tên **SinhVien** lưu trữ các thông tin sau: maSV, hoTen, diemTB, xepLoai... (hoặc lưu trữ thêm các thông tin khác nếu muốn).
2. Xây dựng cấu trúc node để lưu trữ sinh viên và cấu trúc danh sách liên kết đơn quản lý sinh viên.
3. Viết hàm khởi tạo danh sách liên kết đơn và hàm khởi tạo địa chỉ node từ kiểu **SinhVien**.
4. Viết hàm thêm sinh viên vào đầu danh sách.
5. Viết hàm thêm sinh viên vào cuối danh sách.



Bài tập

Tiếp theo...

6. Viết hàm khởi tạo dữ liệu cho danh sách sinh viên (*Dữ liệu kiểm thử xem cuối bài*).
7. Viết hàm in danh sách sinh viên.
8. Viết hàm cập nhật xếp loại học lực.
9. Viết hàm tìm điểm trung bình cao nhất.
10. Viết hàm tìm kiếm sinh viên theo mã sinh viên.
11. Viết hàm tìm SV có trung bình cao nhất.
12. Viết hàm tìm SV có điểm trung bình thấp nhất.



Bài tập

Tiếp theo...

Dữ liệu SV dùng để kiểm thử

- {123, "Nguyen Van A", 9.1, ""};
- {124, "Nguyen Van B", 8.8, ""};
- {125, "Nguyen Van C", 9.1, ""};
- {126, "Nguyen Van D", 2.1, ""};
- {127, "Nguyen Van F", 9.7, ""};
- {128, "Nguyen Van G", 4.1, ""};
- {129, "Nguyen Van H", 8.3, ""};
- {130, "Nguyen Van K", 7.9, ""};



Hỏi - Đáp



CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

DANH SÁCH LIÊN KẾT ĐÔI



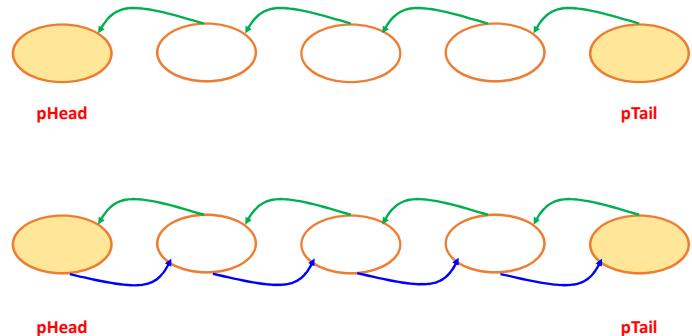


Nội dung

1. Kiến trúc danh sách liên kết đôi
2. Các bước xây dựng
3. Các thao tác
4. Bài tập



Kiến trúc danh sách liên kết đôi



Các bước xây dựng

Bước #1: Tạo cấu trúc Node



```
struct Node{
    int data;
    Node* pPre;
    Node* pNext;
};
```

Bước #2: Khởi tạo Node từ value
(Viết hàm chuyển value thành Node)

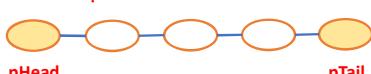
```
Node* initNode(int value)
{
    Node* p = new Node;

    p->data = value;
    p->pPre = NULL;
    p->pNext = NULL;
    return p;
}
```



Các bước xây dựng

Bước #3: Tạo cấu trúc LIST



```
struct List
{
    Node* pHead;
    Node* pTail;
};
```

Bước #4: Khởi tạo LIST

```
void initList(List& l)
{
    l.pHead = NULL;
    l.pTail = NULL;
}
```



Khai báo – Khởi tạo Node và List

- Khai báo danh sách và khởi tạo danh sách

```
List l;
initList(l);
```

NULL

 $pHead \Rightarrow \text{NULL}$
 $pTail \Rightarrow \text{NULL}$

- Khai báo Node và khởi tạo Node

```
Node* p1 = initNode(10);
Node* p2 = initNode(79);
Node* p3 = initNode(39);
```

$p \Rightarrow 0xd56$

 $pHead \Rightarrow \text{NULL}$
 $pTail \Rightarrow \text{NULL}$



Thêm Node tại vị trí đầu List

```
10 79 39
```

```
Node* p = initNode(...);
```

$p \Rightarrow 0xd56$

 $pHead \Rightarrow \text{NULL}$
 $pTail \Rightarrow \text{NULL}$

 $p \Rightarrow 0xd56$

```
void addHead(List& l, Node* p)
{
    if(l.pTail == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        p->pNext = l.pHead;
        l.pHead->pPre = p;
        l.pHead = p;
    }
}
```



Thêm Node tại vị trí đầu List

```
10 79 39
```

```
Node* p = initNode(...);
```

$p \Rightarrow 0xd56$

 $pHead \Rightarrow \text{NULL}$
 $pTail \Rightarrow \text{NULL}$

```
void addHead(List& l, Node* p)
{
    if(l.pTail == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        p->pNext = l.pHead;
        l.pHead->pPre = p;
        l.pHead = p;
    }
}
```



Thêm Node tại vị trí đầu List

```
10 79 39
```

```
Node* p = initNode(...);
```

$0xd56$

 $pHead \Rightarrow d56$
 $pTail \Rightarrow d56$

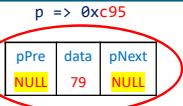
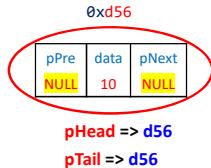
```
void addHead(List& l, Node* p)
{
    if(l.pTail == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        p->pNext = l.pHead;
        l.pHead->pPre = p;
        l.pHead = p;
    }
}
```



Thêm Node tại vị trí đầu List

10 79 39

Node* p = initNode(.....);



```
void addHead(List& l, Node* p)
{
    if(l.pTail == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        p->pNext = l.pHead;
        l.pHead->pPre = p;
        l.pHead = p;
    }
}
```

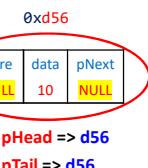
Cấu trúc Dữ liệu và Giải thuật

10

Thêm Node tại vị trí đầu List

10 79 39

Node* p = initNode(.....);



```
void addHead(List& l, Node* p)
{
    if(l.pTail == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        p->pNext = l.pHead;
        l.pHead->pPre = p;
        l.pHead = p;
    }
}
```

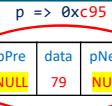
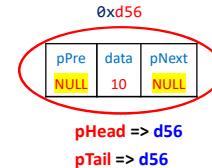
Cấu trúc Dữ liệu và Giải thuật

12

Thêm Node tại vị trí đầu List

10 79 39

Node* p = initNode(.....);



```
void addHead(List& l, Node* p)
{
    if(l.pTail == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        p->pNext = l.pHead;
        l.pHead->pPre = p;
        l.pHead = p;
    }
}
```

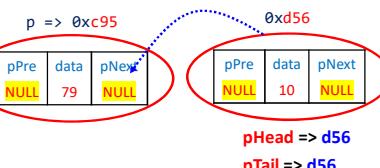
Cấu trúc Dữ liệu và Giải thuật

11

Thêm Node tại vị trí đầu List

10 79 39

Node* p = initNode(.....);



```
void addHead(List& l, Node* p)
{
    if(l.pTail == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        p->pNext = l.pHead;
        l.pHead->pPre = p;
        l.pHead = p;
    }
}
```

Cấu trúc Dữ liệu và Giải thuật

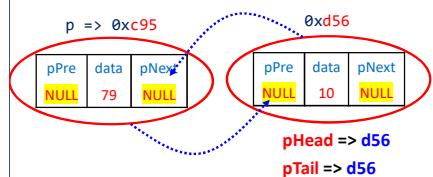
13



Thêm Node tại vị trí đầu List

10 79 39

```
Node* p = initNode(.....);
```



```
void addHead(List& l, Node* p)
{
    if(l.pTail == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        p->pNext = l.pHead;
        l.pHead->pPre = p;
        l.pHead = p;
    }
}
```

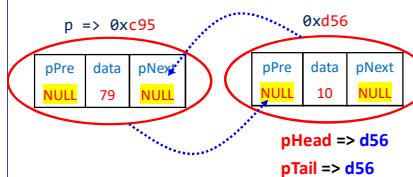
Cấu trúc Dữ liệu và Giải thuật

14

Thêm Node tại vị trí đầu List

10 79 39

```
Node* p = initNode(.....);
```



```
void addHead(List& l, Node* p)
{
    if(l.pTail == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        p->pNext = l.pHead;
        l.pHead->pPre = p;
        l.pHead = p;
    }
}
```

Cấu trúc Dữ liệu và Giải thuật

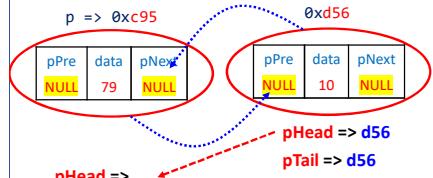
15



Thêm Node tại vị trí đầu List

10 79 39

```
Node* p = initNode(.....);
```



```
void addHead(List& l, Node* p)
{
    if(l.pTail == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        p->pNext = l.pHead;
        l.pHead->pPre = p;
        l.pHead = p;
    }
}
```

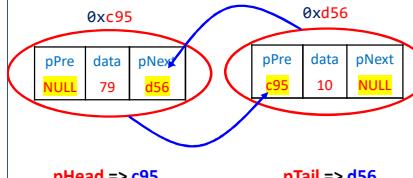
Cấu trúc Dữ liệu và Giải thuật

16

Thêm Node tại vị trí đầu List

10 79 39

```
Node* p = initNode(.....);
```



```
void addHead(List& l, Node* p)
{
    if(l.pTail == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        p->pNext = l.pHead;
        l.pHead->pPre = p;
        l.pHead = p;
    }
}
```

Cấu trúc Dữ liệu và Giải thuật

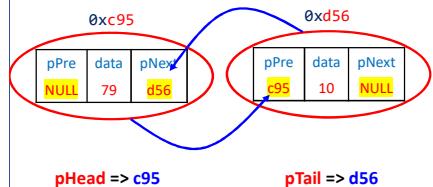
17



Thêm Node tại vị trí cuối List

10 79 39

Node* p = initNode(.....);



```
void addTail(List& l, Node* p)
{
    if(l.pHead == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        l.pTail->pNext = p;
        p->pPre = l.pTail;
        l.pTail = p;
    }
}
```

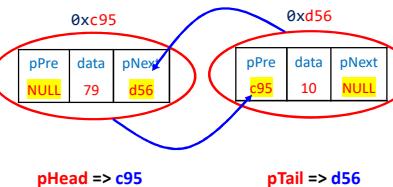
Cấu trúc Dữ liệu và Giải thuật

18

Thêm Node tại vị trí cuối List

10 79 39

Node* p = initNode(.....);



p => 0xa36

```
void addTail(List& l, Node* p)
{
    if(l.pHead == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        l.pTail->pNext = p;
        p->pPre = l.pTail;
        l.pTail = p;
    }
}
```

Cấu trúc Dữ liệu và Giải thuật

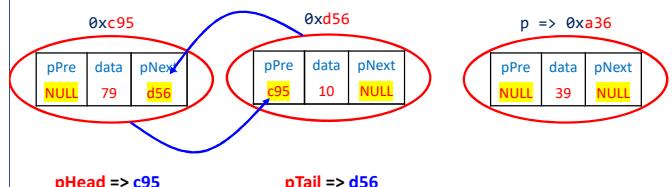
19



Thêm Node tại vị trí cuối List

10 79 39

Node* p = initNode(.....);



```
void addTail(List& l, Node* p)
{
    if(l.pHead == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        l.pTail->pNext = p;
        p->pPre = l.pTail;
        l.pTail = p;
    }
}
```

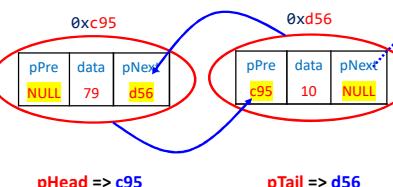
Cấu trúc Dữ liệu và Giải thuật

20

Thêm Node tại vị trí cuối List

10 79 39

Node* p = initNode(.....);



p => 0xa36

```
void addTail(List& l, Node* p)
{
    if(l.pHead == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        l.pTail->pNext = p;
        p->pPre = l.pTail;
        l.pTail = p;
    }
}
```

Cấu trúc Dữ liệu và Giải thuật

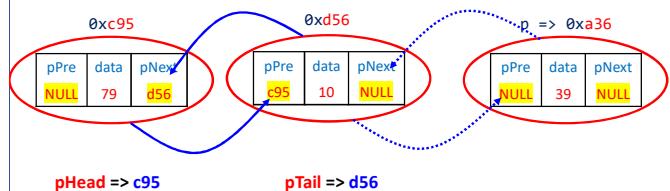
21



Thêm Node tại vị trí cuối List

10 79 39

Node* p = initNode(.....);



```
void addTail(List& l, Node* p)
{
    if(l.pHead == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        l.pTail->pNext = p;
        p->pPre = l.pTail;
        l.pTail = p;
    }
}
```

Cấu trúc Dữ liệu và Giải thuật

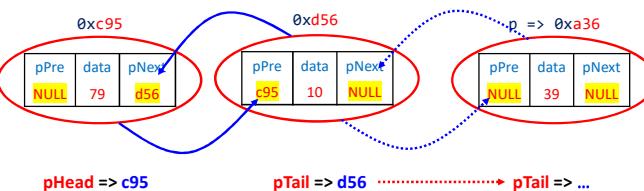
22



Thêm Node tại vị trí cuối List

10 79 39

Node* p = initNode(.....);



```
void addTail(List& l, Node* p)
{
    if(l.pHead == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        l.pTail->pNext = p;
        p->pPre = l.pTail;
        l.pTail = p;
    }
}
```

Cấu trúc Dữ liệu và Giải thuật

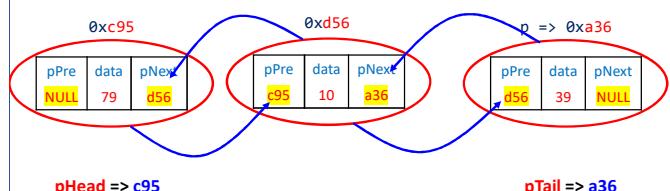
23



Thêm Node tại vị trí cuối List

10 79 39

Node* p = initNode(.....);



```
void addTail(List& l, Node* p)
{
    if(l.pHead == NULL)
    {
        l.pHead = p;
        l.pTail = p;
    }
    else
    {
        l.pTail->pNext = p;
        p->pPre = l.pTail;
        l.pTail = p;
    }
}
```

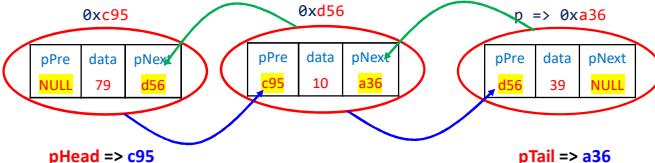
Cấu trúc Dữ liệu và Giải thuật

24



In danh sách

```
for(Node* p = l.pHead; p != NULL; p = p->pNext)
{
    cout << p->data << "\t";
}
```



```
for(Node* p = l.pTail; p != NULL; p = p->pPre)
{
    cout << p->data << "\t";
}
```

Cấu trúc Dữ liệu và Giải thuật

25



In danh sách

```
void printListFor(List l)
{
    for(Node* p = l.pHead; p != NULL; p = p->pNext)
    {
        cout << p->data << "\t";
    }

    cout << endl;
    for(Node* p = l.pTail; p != NULL; p = p->pPre)
    {
        cout << p->data << "\t";
    }
}
```

main

```
int main()
{
    List l;
    initList(l);

    addHead(l, initNode(10));
    addHead(l, initNode(79));
    addTail(l, initNode(39));

    printListFor(l);

    return 0;
}
```

79 10 39
39 10 79



Hỏi - Đáp



CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Stack và Queue



CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Ngăn Xếp - Stack



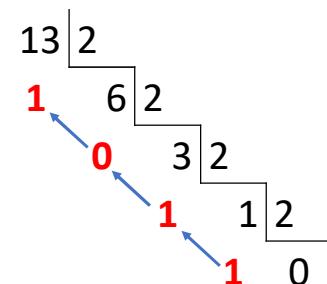
Nội dung

1. Ý tưởng
2. Giới thiệu ngăn xếp – Stack
3. Phương pháp
4. Kiến trúc
5. Các bước xây dựng
6. Các thao tác
 1. Thêm vào – Push
 2. Lấy ra – Pop
 3. In hàng đợi
 4. Kiểm tra rỗng
7. Bài tập ứng dụng



Ý tưởng

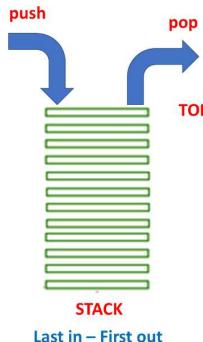
Vấn đề: Chuyển đổi số hệ 10 sang hệ 2





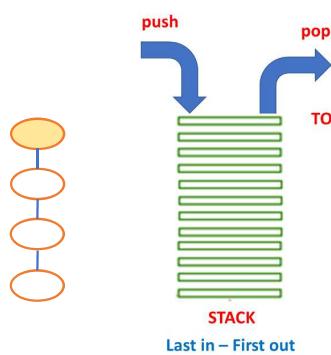
Giới thiệu

- Ngăn xếp: Stack
- Cấu trúc dữ liệu được sử dụng để quản lý các phần tử theo cơ chế "LIFO" (Last-In-First-Out)
- Vào sau – Ra trước
- Phần tử cuối cùng được đưa vào ngăn xếp thì là phần tử đầu tiên được lấy ra.

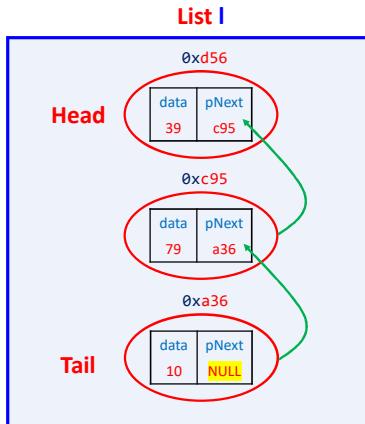
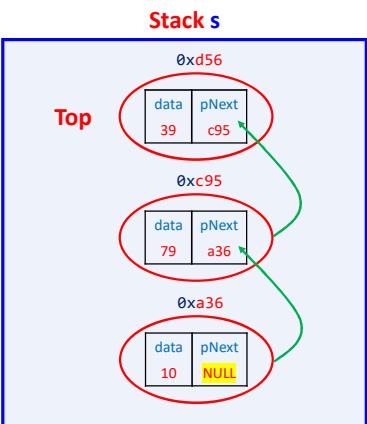


Phương pháp

- Dùng mảng
- Dùng danh sách

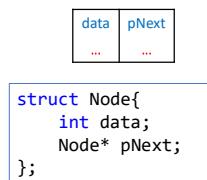


Kiến trúc



Các bước xây dựng

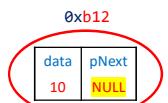
Bước #1: Tạo cấu trúc Node



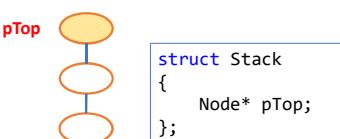
Bước #2: Khởi tạo Node từ value (Viết hàm chuyển value thành Node)

```
Node* initNode(int value)
{
    Node* p = new Node;

    p->data = value;
    p->pNext = NULL;
    return p;
}
```



Bước #3: Tạo cấu trúc Stack



```
struct Stack
{
    Node* pTop;
};
```

Bước #4: Khởi tạo Stack

```
void initStack(Stack& s)
{
    s.pTop = NULL;
}
```



Khai báo – Khởi tạo Node và Stack

- Khai báo stack và khởi tạo stack

```
Stack s;
initStack(s);
```

pTop => NULL



- Khai báo Node và khởi tạo Node

```
Node* p1 = initNode(10);
Node* p2 = initNode(79);
Node* p3 = initNode(39);
```

Các thao tác

- Push: Thêm phần tử vào Stack
- Pop: Lấy lấy phần tử ra khỏi Stack
- Kiểm tra Stack có rỗng hay không
- Kiểm tra Stack có đầy hay không
- Lấy giá trị phần tử trên cùng

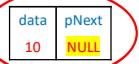


Push

10 79 39

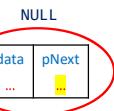
```
Node* p = initNode(...);
```

p => 0xa36



```
void push(Stack& s, Node* p)
{
    if(s.pTop == NULL)
    {
        s.pTop = p;
    }
    else
    {
        p->pNext = s.pTop;
        s.pTop = p;
    }
}
```

pTop => NULL



Push

10 79 39

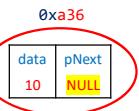
```
Node* p = initNode(...);
```

p => 0xa36



```
void push(Stack& s, Node* p)
{
    if(s.pTop == NULL)
    {
        s.pTop = p;
    }
    else
    {
        p->pNext = s.pTop;
        s.pTop = p;
    }
}
```

pTop => a36





Push

10 79 39

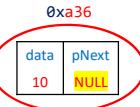
Node* p = initNode(.....);

```
void push(Stack& s, Node* p)
{
    if(s.pTop == NULL)
    {
        s.pTop = p;
    }
    else
    {
        p->pNext = s.pTop;
        s.pTop = p;
    }
}
```

$p \Rightarrow 0xc95$



$pTop \Rightarrow a36$



Cấu trúc Dữ liệu và Giải thuật



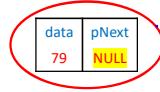
Push

10 79 39

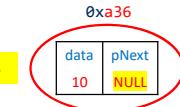
Node* p = initNode(.....);

```
void push(Stack& s, Node* p)
{
    if(s.pTop == NULL)
    {
        s.pTop = p;
    }
    else
    {
        p->pNext = s.pTop;
        s.pTop = p;
    }
}
```

$p \Rightarrow 0xc95$



$pTop \Rightarrow a36$



Cấu trúc Dữ liệu và Giải thuật

14



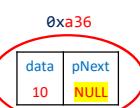
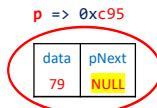
Push

10 79 39

Node* p = initNode(.....);

```
void push(Stack& s, Node* p)
{
    if(s.pTop == NULL)
    {
        s.pTop = p;
    }
    else
    {
        p->pNext = s.pTop;
        s.pTop = p;
    }
}
```

$p \Rightarrow 0xc95$



Cấu trúc Dữ liệu và Giải thuật

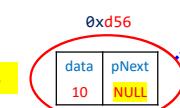


Push

10 79 39

Node* p = initNode(.....);

```
void push(Stack& s, Node* p)
{
    if(s.pTop == NULL)
    {
        s.pTop = p;
    }
    else
    {
        p->pNext = s.pTop;
        s.pTop = p;
    }
}
```



Cấu trúc Dữ liệu và Giải thuật

16

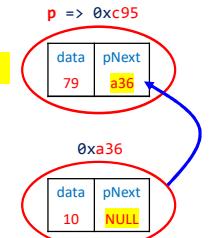


Push

10 79 39

Node* p = initNode(.....);

```
void push(Stack& s, Node* p)
{
    if(s.pTop == NULL)
    {
        s.pTop = p;
    }
    else
    {
        p->pNext = s.pTop;
        s.pTop = p;
    }
}
```



Cấu trúc Dữ liệu và Giải thuật

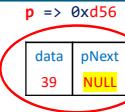
17

Push

10 79 39

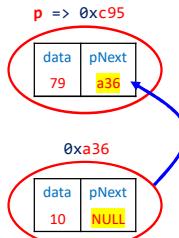
Node* p = initNode(.....);

```
void push(Stack& s, Node* p)
{
    if(s.pTop == NULL)
    {
        s.pTop = p;
    }
    else
    {
        p->pNext = s.pTop;
        s.pTop = p;
    }
}
```



p => 0xd56

pTop => c95



0xa36

p => 0xc95

Cấu trúc Dữ liệu và Giải thuật

18

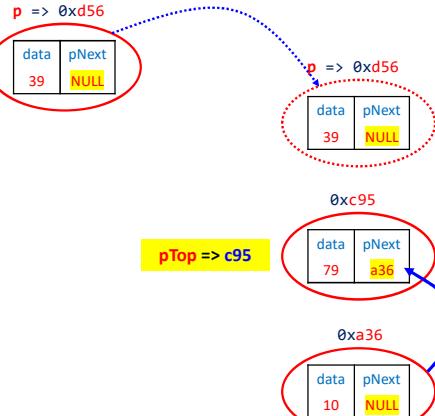


Push

10 79 39

Node* p = initNode(.....);

```
void push(Stack& s, Node* p)
{
    if(s.pTop == NULL)
    {
        s.pTop = p;
    }
    else
    {
        p->pNext = s.pTop;
        s.pTop = p;
    }
}
```



Cấu trúc Dữ liệu và Giải thuật

19

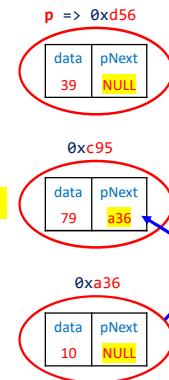


Push

10 79 39

Node* p = initNode(.....);

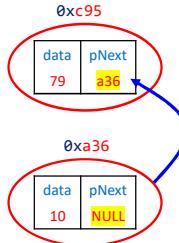
```
void push(Stack& s, Node* p)
{
    if(s.pTop == NULL)
    {
        s.pTop = p;
    }
    else
    {
        p->pNext = s.pTop;
        s.pTop = p;
    }
}
```



p => 0xd56

0xc95

pTop => c95



0xa36

p => 0xc95

Cấu trúc Dữ liệu và Giải thuật

20

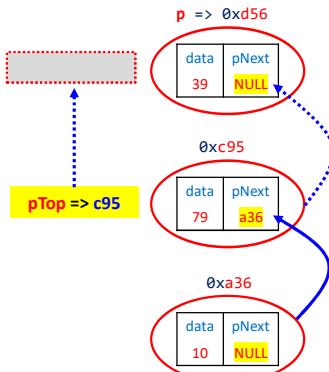


Push

10 79 39

```
Node* p = initNode(.....);
```

```
void push(Stack& s, Node* p)
{
    if(s.pTop == NULL)
    {
        s.pTop = p;
    }
    else
    {
        p->pNext = s.pTop;
        s.pTop = p;
    }
}
```



Cấu trúc Dữ liệu và Giải thuật

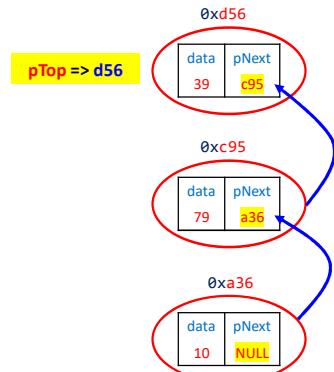
21

Push

10 79 39

```
Node* p = initNode(.....);
```

```
void push(Stack& s, Node* p)
{
    if(s.pTop == NULL)
    {
        s.pTop = p;
    }
    else
    {
        p->pNext = s.pTop;
        s.pTop = p;
    }
}
```



Cấu trúc Dữ liệu và Giải thuật

22

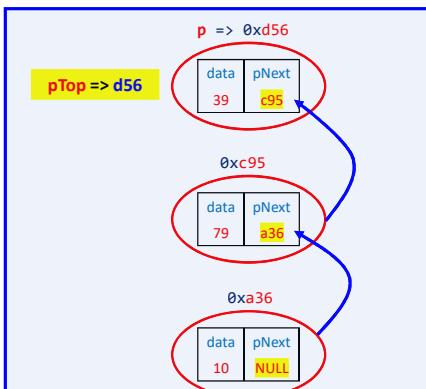


Push

10 79 39

```
Node* p = initNode(.....);
```

```
void push(Stack& s, Node* p)
{
    if(s.pTop == NULL)
    {
        s.pTop = p;
    }
    else
    {
        p->pNext = s.pTop;
        s.pTop = p;
    }
}
```



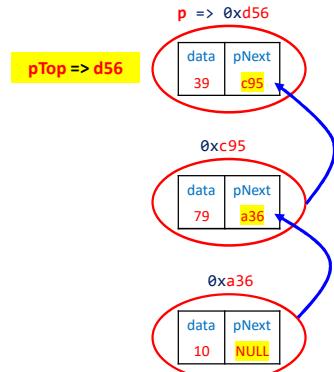
Cấu trúc Dữ liệu và Giải thuật

23

Print

```
void printStack(Stack s)
```

```
{
    cout << "Stack = Top < ";
    for(Node* p = s.pTop; p != NULL; p=p->pNext)
    {
        cout << p->data << " ";
    }
    cout << ">" << endl;
}
```



Cấu trúc Dữ liệu và Giải thuật

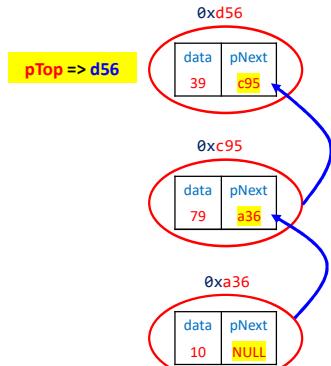
24



Pop

```
void pop(Stack& s, int& value)
{
    if(s.pTop==NULL)
        return;

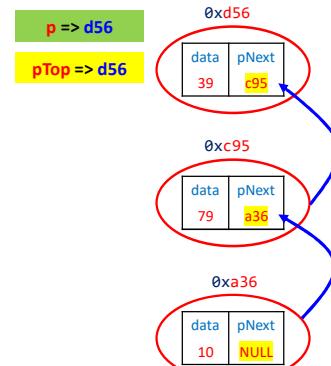
    Node* p = s.pTop;
    s.pTop = s.pTop->pNext;
    value = p->data;
    delete p;
}
```



Pop

```
void pop(Stack& s, int& value)
{
    if(s.pTop==NULL)
        return;

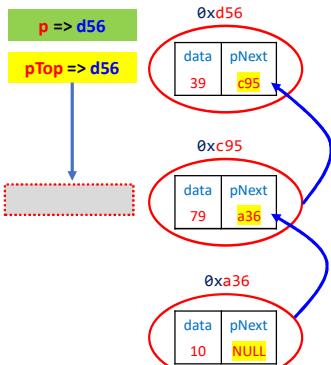
    Node* p = s.pTop;
    s.pTop = s.pTop->pNext;
    value = p->data;
    delete p;
}
```



Pop

```
void pop(Stack& s, int& value)
{
    if(s.pTop==NULL)
        return;

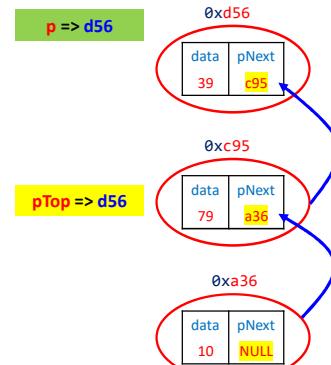
    Node* p = s.pTop;
    s.pTop = s.pTop->pNext;
    value = p->data;
    delete p;
}
```



Pop

```
void pop(Stack& s, int& value)
{
    if(s.pTop==NULL)
        return;

    Node* p = s.pTop;
    s.pTop = s.pTop->pNext;
    value = p->data;
    delete p;
}
```

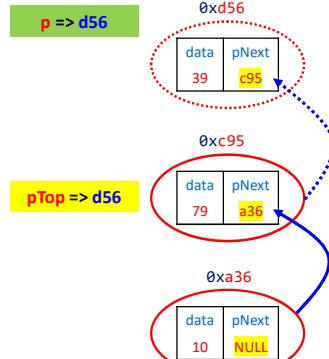




Pop

```
void pop(Stack& s, int& value)
{
    if(s.pTop==NULL)
        return;

    Node* p = s.pTop;
    s.pTop = s.pTop->pNext;
    value = p->data;
    delete p;
}
```



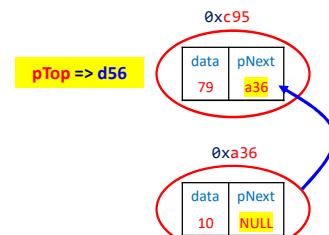
Cấu trúc Dữ liệu và Giải thuật

29

Pop

```
void pop(Stack& s, int& value)
{
    if(s.pTop==NULL)
        return;

    Node* p = s.pTop;
    s.pTop = s.pTop->pNext;
    value = p->data;
    delete p;
}
```



Cấu trúc Dữ liệu và Giải thuật

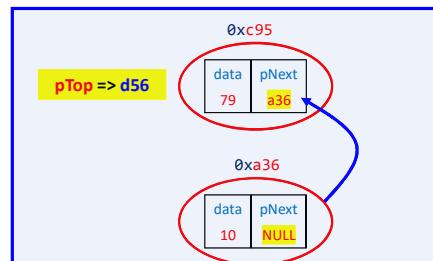
30



Pop

```
void pop(Stack& s, int& value)
{
    if(s.pTop==NULL)
        return;

    Node* p = s.pTop;
    s.pTop = s.pTop->pNext;
    value = p->data;
    delete p;
}
```



Cấu trúc Dữ liệu và Giải thuật

31



main

```
int main()
{
    Stack s;
    initStack(s);

    Node* p1 = initNode(10);

    push(s, p1);
    push(s, initNode(79));
    push(s, initNode(39));

    printStack(s);

    int value;
    pop(s, value);
    cout << "Pop => " << value << endl;

    printStack(s);
    return 0;
}
```

Stack = Top < 39 79 10 >
Pop => 39
Stack = Top < 79 10 >

Cấu trúc Dữ liệu và Giải thuật

32



Bài tập áp dụng

Viết chương trình cài đặt stack (cấp phát động) để lưu trữ tạm các số nguyên. Thực hiện các yêu cầu sau:

- Khai báo cấu trúc Node và Stack.
- Viết hàm khởi tạo Node và Stack.
- Viết hàm kiểm tra stack rỗng.
- Viết hàm kiểm tra stack có full hay không (*nếu có*).
- Viết hàm đẩy một phần tử vào stack.
- Viết hàm lấy một phần tử ra khỏi stack.
- Lấy giá trị phần tử TOP của stack.
- Trong hàm main, viết menu thể hiện các lựa chọn câu trên.



Bài tập áp dụng

Viết chương trình áp dụng stack để minh họa quá trình chuyển đổi số thập phân sang số nhị phân.

- Ví dụ:
 - Người dùng nhập: **13**
 - Kết quả nhị phân: **1101**



Bài tập áp dụng

Viết chương trình áp dụng stack để minh họa cách đảo một chuỗi nhập vào.

- Ví dụ:
 - Người dùng nhập: Dai Hoc
 - Kết quả đảo chuỗi: Hoc Dai

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Hàng đợi - Queue





Nội dung

1. Ý tưởng
2. Giới thiệu hàng đợi - Queue
3. Kiến trúc
4. Các bước xây dựng
5. Các thao tác
 1. Thêm vào – EnQueue
 2. Lấy ra – DeQueue
 3. In hàng đợi
 4. Kiểm tra rỗng
6. Bài tập ứng dụng



Ý tưởng

Printers & scanners

Add printers & scanners

Printer Document View					
Document Name	Status	Owner	Pages	Size	Submitted
GIỚI THIỆU.pdf	HP	15	1.40 MB	4:10:57 PM	10/21/2023
GIỚI THIỆU.pdf	HP	3	254 KB	4:10:49 PM	10/21/2023
GIỚI THIỆU.pdf	HP	12	1.24 MB	4:10:31 PM	10/21/2023

Printers & scanners

Canon G2010 series
Offline

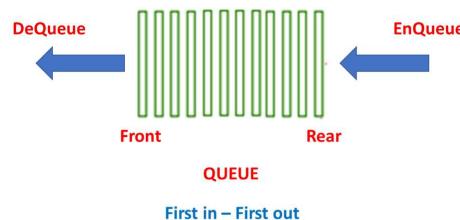
Canon LBP2900
Offline

Fax



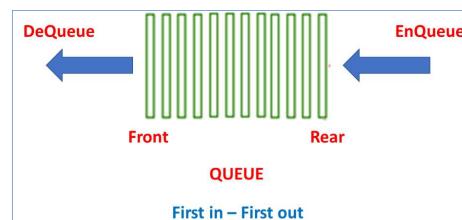
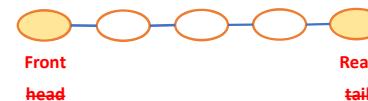
Giới thiệu

- Hàng đợi: Queue
- Là một cấu trúc dữ liệu được sử dụng để lưu trữ và quản lý các phần tử.
- Hoạt động theo nguyên tắc "FIFO" (First-In, First-Out)
- Vào trước – Ra trước.
- Phần tử được đưa vào hàng đợi trước sẽ được lấy ra trước.



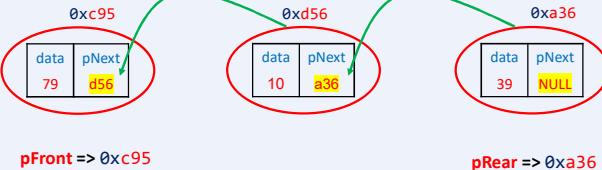
Phương pháp

- Dùng mảng
- Dùng danh sách





Kiến trúc



Các bước xây dựng

Bước #1: Tạo cấu trúc Node



```
struct Node{
    int data;
    Node* pNext;
};
```

0xb12

Bước #2: Khởi tạo Node từ value
(Viết hàm chuyển value thành Node)

Bước #3: Tạo cấu trúc Queue



Bước #4: Khởi tạo Queue

```
void initQueue(Queue& q)
{
    q.pFront = NULL;
    q.pRear = NULL;
}
```

```
Node* initNode(int value)
{
    Node* p = new Node;
    p->data = value;
    p->pNext = NULL;
    return p;
}
```

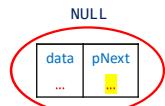
```
struct Queue
{
    Node* pFront;
    Node* pRear;
};
```



Khai báo – Khởi tạo Node và Queue

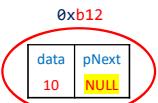
- Khai báo hàng đợi và khởi tạo hàng đợi

```
Queue q;
initQueue(q);
```



- Khai báo Node và khởi tạo Node

```
Node* p1 = initNode(10);
Node* p2 = initNode(79);
Node* p3 = initNode(39);
```



EnQueue

```
void enqueue(Queue& q, Node* p)
{
    if(q.pFront == NULL)
    {
        q.pFront = p;
        q.pRear = p;
    }
    else
    {
        q.pRear->pNext = p;
        q.pRear = p;
    }
}
```



DeQueue

```
int deQueue(Queue& q)
{
    if(q.pFront == NULL)
        return 0;

    Node* p = q.pFront;
    q.pFront = q.pFront->pNext;
    int value = p->data;
    delete p;

    return value;
}
```

In Queue

```
void printQueueWhile(Queue q)
{
    cout << "Queue Q = Front<\t";
    Node* p = q.pFront;
    while(p != NULL)
    {
        cout << p->data << "\t",
        p = p->pNext;
    }
    cout << ">Rear" << endl;
}
```

```
void printQueue(Queue q)
{
    cout << "Queue Q = Front<\t";
    for(Node* p = q.pFront; p != NULL; p = p->pNext)
    {
        cout << p->data << "\t";
    }
    cout << ">Rear" << endl;
}
```



main

```
int main()
{
    Queue q;
    initQueue(q);

    enqueue(q, initNode(10));
    enqueue(q, initNode(79));
    enqueue(q, initNode(39));

    printQueue(q);

    int value = deQueue(q);
    printQueueWhile(q);

    return 0;
}
```

Queue Q = Front< 10 79 39 >Rear
Queue Q = Front< 79 39 >Rear



Bài tập

Viết chương trình cài đặt queue bằng danh sách liên kết đơn để lưu trữ tạm các số nguyên (hoặc ký tự).

Thực hiện các yêu cầu sau:

Câu 1. Viết hàm khởi tạo queue.

Câu 2. Viết hàm kiểm tra queue rỗng.

Câu 3. Viết hàm đưa một phần tử vào queue.

Câu 4. Viết hàm lấy một phần tử ra khỏi queue.

Câu 5. Viết hàm in ra phần tử REAR và FRONT của queue.

Câu 6. Trong hàm main, viết menu thể hiện các lựa chọn câu trên.



Thank you

CẨU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

BẢNG BĂM *Hash Table*



Hỏi - Đáp



Câu trúc Dữ liệu và Giải thuật

51



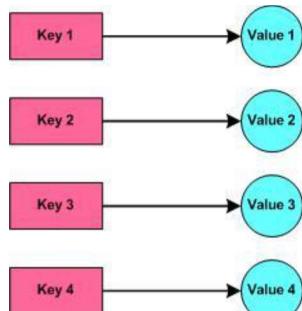
Nội dung

1. Giới thiệu bảng băm
2. Cơ chế hoạt động
3. Cài đặt cách 1 (6 bước)
4. Cài đặt cách 2 (4 bước)
5. Các thao tác
6. Bài tập



Giới thiệu

- Bảng băm: hash table
- Là một cấu trúc dữ liệu trong lập trình sử dụng để **lưu trữ và truy xuất** dữ liệu rất hiệu quả.
- Hiệu quả:
 - Truy xuất nhanh.
 - Lưu trữ lớn.
- Nguyên tắc hoạt động: ánh xạ một khóa (key) tới một giá trị (value) bằng cách sử dụng một hàm băm (hash function) để tính toán vị trí lưu trữ trong bảng băm.



Giới thiệu

Phân loại

- Có nhiều loại bảng băm
- Chained Hash Table – Băm liên kết
 - Linear Probing - Dò tuyến tính
 - Quadratic Probing - Dò tuyến tính bậc 2
 - Double hashing - Băm kép

Chained Hash Table được chọn minh họa cài đặt.



Chained Hash Table: Cơ chế hoạt động

50 73 35 36 64 28 90 21 53 13

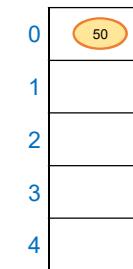
value % 5 => key



Cơ chế hoạt động

50 73 35 36 64 28 90 21 53 13

50 % 5 = 0



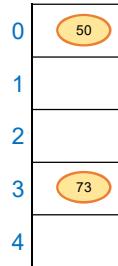


Cơ chế hoạt động

50 73 35 36 64 28 90 21 53 13

$$50 \% 5 = 0$$

$$73 \% 5 = 3$$



Cấu trúc Dữ liệu và Giải thuật



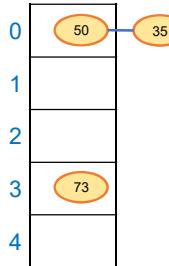
Cơ chế hoạt động

50 73 35 36 64 28 90 21 53 13

$$50 \% 5 = 0$$

$$73 \% 5 = 3$$

$$35 \% 5 = 0$$



Cấu trúc Dữ liệu và Giải thuật

8



Cơ chế hoạt động

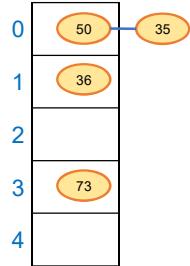
50 73 35 36 64 28 90 21 53 13

$$50 \% 5 = 0$$

$$73 \% 5 = 3$$

$$35 \% 5 = 0$$

$$36 \% 5 = 1$$



Cấu trúc Dữ liệu và Giải thuật



Cơ chế hoạt động

50 73 35 36 64 28 90 21 53 13

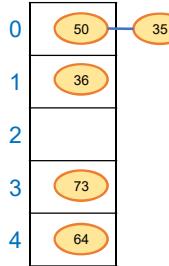
$$50 \% 5 = 0$$

$$73 \% 5 = 3$$

$$35 \% 5 = 0$$

$$36 \% 5 = 1$$

$$64 \% 5 = 4$$



Cấu trúc Dữ liệu và Giải thuật

10

9



Cơ chế hoạt động

50 73 35 36 64 28 90 21 53 13

$$50 \% 5 = 0$$

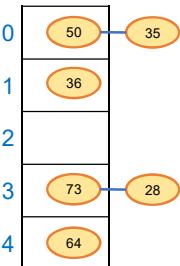
$$73 \% 5 = 3$$

$$35 \% 5 = 0$$

$$36 \% 5 = 1$$

$$64 \% 5 = 4$$

$$28 \% 5 = 3$$



Cấu trúc Dữ liệu và Giải thuật

11



Cơ chế hoạt động

50 73 35 36 64 28 90 21 53 13

$$50 \% 5 = 0$$

$$73 \% 5 = 3$$

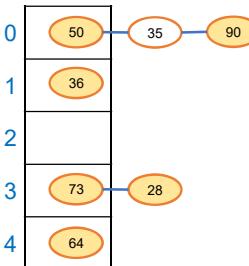
$$35 \% 5 = 0$$

$$36 \% 5 = 1$$

$$64 \% 5 = 4$$

$$28 \% 5 = 3$$

$$90 \% 5 = 0$$



Cấu trúc Dữ liệu và Giải thuật

12



Cơ chế hoạt động

50 73 35 36 64 28 90 21 53 13

$$50 \% 5 = 0$$

$$73 \% 5 = 3$$

$$35 \% 5 = 0$$

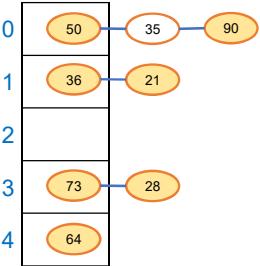
$$36 \% 5 = 1$$

$$64 \% 5 = 4$$

$$28 \% 5 = 3$$

$$90 \% 5 = 0$$

$$21 \% 5 = 1$$



Cấu trúc Dữ liệu và Giải thuật

13



Cơ chế hoạt động

50 73 35 36 64 28 90 21 53 13

$$50 \% 5 = 0$$

$$73 \% 5 = 3$$

$$35 \% 5 = 0$$

$$36 \% 5 = 1$$

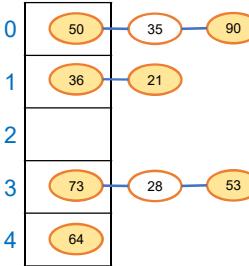
$$64 \% 5 = 4$$

$$28 \% 5 = 3$$

$$90 \% 5 = 0$$

$$21 \% 5 = 1$$

$$53 \% 5 = 3$$



Cấu trúc Dữ liệu và Giải thuật

14



Cơ chế hoạt động

50 73 35 36 64 28 90 21 53 13

$$50 \% 5 = 0$$

$$73 \% 5 = 3$$

$$35 \% 5 = 0$$

$$36 \% 5 = 1$$

$$64 \% 5 = 4$$

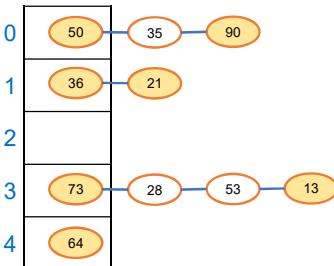
$$28 \% 5 = 3$$

$$90 \% 5 = 0$$

$$21 \% 5 = 1$$

$$53 \% 5 = 3$$

$$13 \% 5 = 3$$



Cơ chế hoạt động

50 73 35 36 64 28 90 21 53 13

$$50 \% 5 = 0$$

$$73 \% 5 = 3$$

$$35 \% 5 = 0$$

$$36 \% 5 = 1$$

$$64 \% 5 = 4$$

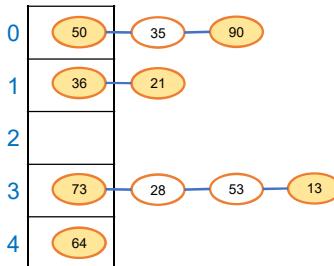
$$28 \% 5 = 3$$

$$90 \% 5 = 0$$

$$21 \% 5 = 1$$

$$53 \% 5 = 3$$

$$13 \% 5 = 3$$



Cơ chế hoạt động

hash function

$$50 \% 5 = 0$$

$$73 \% 5 = 3$$

$$35 \% 5 = 0$$

$$36 \% 5 = 1$$

$$64 \% 5 = 4$$

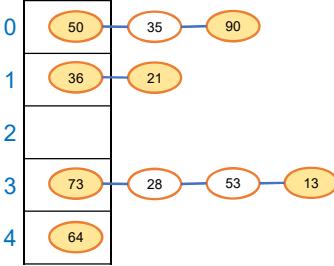
$$28 \% 5 = 3$$

$$90 \% 5 = 0$$

$$21 \% 5 = 1$$

$$53 \% 5 = 3$$

$$13 \% 5 = 3$$



Cơ chế hoạt động

$$50 \% 5 = 0$$

$$73 \% 5 = 3$$

$$35 \% 5 = 0$$

$$36 \% 5 = 1$$

$$64 \% 5 = 4$$

$$28 \% 5 = 3$$

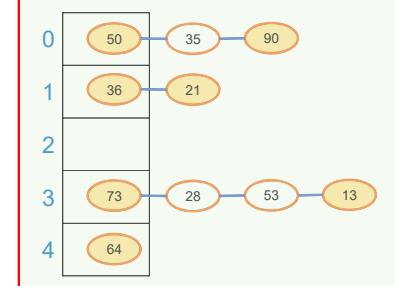
$$90 \% 5 = 0$$

$$21 \% 5 = 1$$

$$53 \% 5 = 3$$

$$13 \% 5 = 3$$

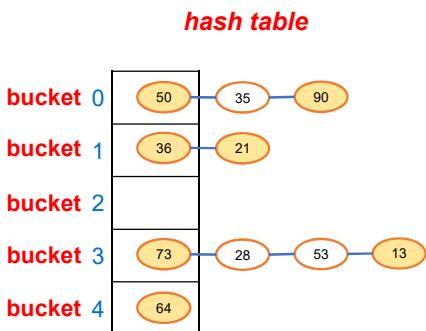
hash table





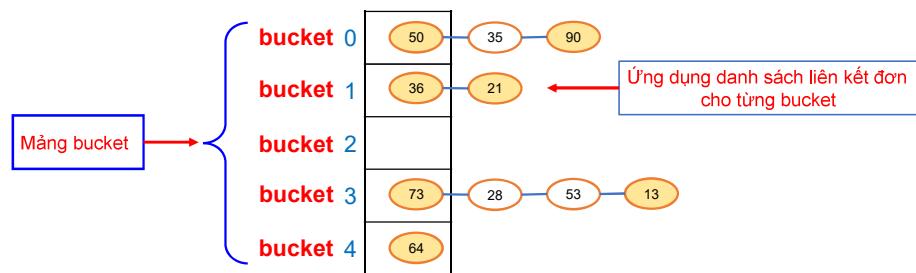
Cơ chế hoạt động

$50 \% 5 = 0$
 $73 \% 5 = 3$
 $35 \% 5 = 0$
 $36 \% 5 = 1$
 $64 \% 5 = 4$
 $28 \% 5 = 3$
 $90 \% 5 = 0$
 $21 \% 5 = 1$
 $53 \% 5 = 3$
 $13 \% 5 = 3$



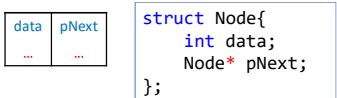
Thiết kế

- Mỗi bucket là 1 List
- Dùng mảng có kích thước bằng với kích thước bảng băm để quản lý các bucket



Các bước xây dựng – Cách 1

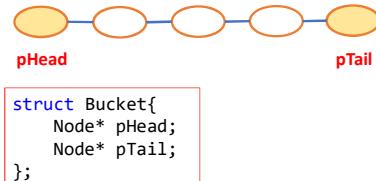
Bước #1: Tạo cấu trúc Node



Bước #2: Khởi tạo Node từ value (Viết hàm chuyển value thành Node)

```
Node* initNode(int value)
{
    Node* p = new Node;
    p->data = value;
    p->pNext = NULL;
    return p;
}
```

Bước #3: Tạo cấu trúc Bucket



Bước #4: Khởi tạo Bucket

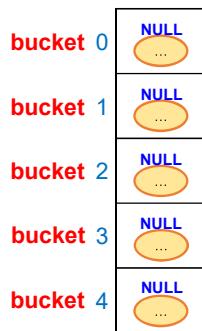
```
void initBucket(Bucket& bk)
{
    bk.pHead = bk.pTail = NULL;
}
```

Bước #5: Tạo cấu trúc HashTable

```
struct HashTable
{
    Bucket bucket[Size];
};
```

Bước #6: Khởi tạo HashTable (Viết hàm gọi hàm khởi tạo các bucket)

```
void initHashTable(HashTable& h)
{
    for(int i = 0; i < Size; i++)
    {
        initBucket(h.bucket[i]);
    }
}
```



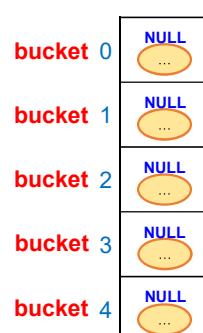


Khai báo – Khởi tạo Node và HashTable

Khai báo – Khởi tạo Node

```
initNode(50)
initNode(73)
initNode(35)
```

HashTable *h*



Khai báo – Khởi tạo HashTable

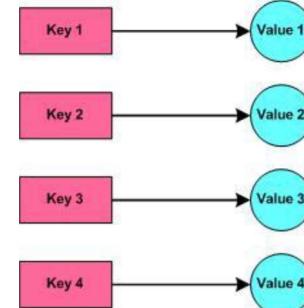
```
HashTable h;
initHashTable(h);
```

Cấu trúc Dữ liệu và Giải thuật

23



Hàm băm theo phương pháp chia lấy dư



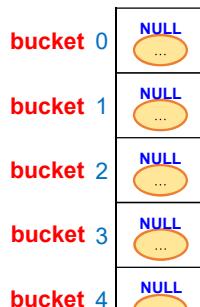
```
int hashFunc(int value)
{
    return value % Size;
}
```

Cấu trúc Dữ liệu và Giải thuật

24



Thêm Node vào HashTable



HashTable *h*

```
void addTail(Bucket& bk, Node* p)
{
    if(bk.pHead == NULL)
    {
        bk.pHead = bk.pTail = p;
    }
    else
    {
        bk.pTail->pNext = p;
        bk.pTail = p;
    }
}

void add(HashTable& h, Node* p)
{
    int i = hashFunc(p->data);

    addTail(h.bucket[i], p);
}
```

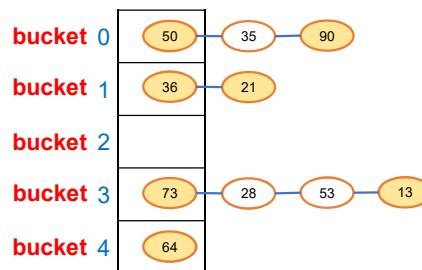
Cấu trúc Dữ liệu và Giải thuật

25



In bảng băm

HashTable *h*



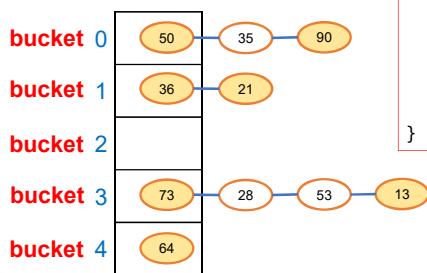
Cấu trúc Dữ liệu và Giải thuật

26



In bảng băm

HashTable h



```

void print(HashTable h)
{
    for(int i = 0; i < Size; i++)
    {
        Bucket bk = h.bucket[i];
        printf("Bucket[%d] = ", i);
        for(Node* p = bk.pHead; p != NULL; p=p->pNext)
        {
            cout << p->data << " ";
        }
        cout << endl;
    }
}
  
```

Cấu trúc Dữ liệu và Giải thuật

27

Hàm main

```

int main()
{
    HashTable h;
    initHashTable(h);

    add(h, initNode(50));
    add(h, initNode(73));
    add(h, initNode(35));
    add(h, initNode(36));
    add(h, initNode(64));
    add(h, initNode(28));
    add(h, initNode(90));
    add(h, initNode(21));
    add(h, initNode(53));
    add(h, initNode(13));

    print(h);
    return 0;
}
  
```

```

E:\Code_IT003\hash_table_2\bin\Debug\h...
Bucket[0] = 50 35 90
Bucket[1] = 36 21
Bucket[2] =
Bucket[3] = 73 28 53 13
Bucket[4] = 64

Process returned 0 (0x0) execution time : 0.358 s
Press any key to continue.
  
```

Cấu trúc Dữ liệu và Giải thuật

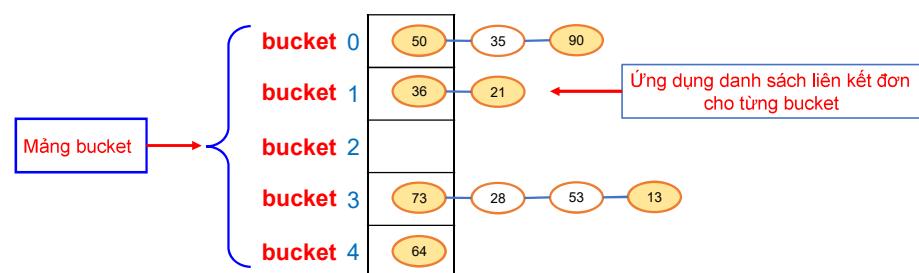
28

CÀI ĐẶT CÁCH 2



Thiết kế

- Mỗi bucket là 1 List
- Dùng mảng có kích thước bằng với kích thước bảng băm để quản lý các bucket



29

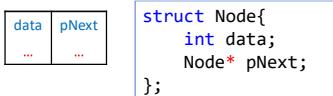
Cấu trúc Dữ liệu và Giải thuật

30



Các bước xây dựng – Cách 2: 4 Bước

Bước #1: Tạo cấu trúc Node



Bước #2: Khởi tạo Node từ value
(Viết hàm chuyển value thành Node)

```
Node* initNode(int value)
{
    Node* p = new Node;
    p->data = value;
    p->pNext = NULL;
    return p;
}
```

Cấu trúc Dữ liệu và Giải thuật



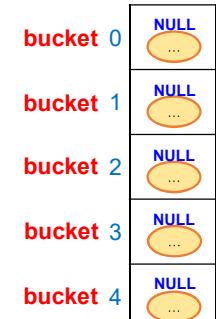
Các bước xây dựng – Cách 2: 4 Bước

Bước #3: Tạo cấu trúc HashTable

```
struct HashTable
{
    struct Bucket
    {
        Node* pHead;
        Node* pTail;
    };
    Bucket bucket[Size];
};
```

Bước #4: Khởi tạo HashTable (Tức là khởi tạo từng Bucket trong HashTable)

```
void initHashTable(HashTable& h) {
    for(int i = 0; i < Size; i++)
    {
        h.bucket[i].pHead = NULL;
        h.bucket[i].pTail = NULL;
    }
}
```



HashTable h

Cấu trúc Dữ liệu và Giải thuật

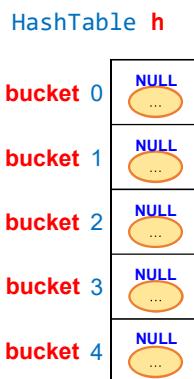
32



Khai báo – Khởi tạo Node và HashTable

Khai báo – Khởi tạo Node

```
initNode(50)
initNode(73)
initNode(35)
...
```



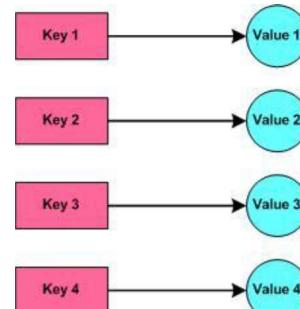
Khai báo – Khởi tạo HashTable

```
HashTable h;
initHashTable(h);
```

Cấu trúc Dữ liệu và Giải thuật



Hàm băm theo phương pháp chia lấy dư



```
int hashFunc(int value)
{
    return value % Size;
}
```

Cấu trúc Dữ liệu và Giải thuật

34



Thêm Node vào HashTable

```
void add(HashTable& h, Node* p)
{
    int i = hashFunc(p->data);

    if(h.bucket[i].pHead == NULL)
    {
        h.bucket[i].pHead = p;
        h.bucket[i].pTail = p;
    }
    else
    {
        h.bucket[i].pTail->pNext = p;
        h.bucket[i].pTail = p;
    }
}
```

```
void add(HashTable& h, Node* p)
{
    int i = hashFunc(p->data);

    addHead(h.bucket[i], p);
}

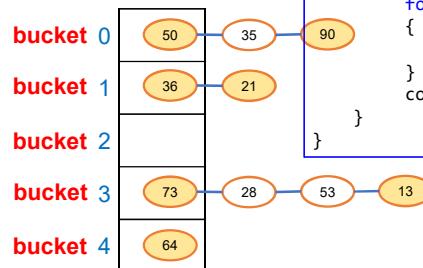
void addHead(Bucket& bk, Node* p)
{
    if(bk.pHead == NULL)
    {
        bk.pHead = bk.pTail = p;
    }
    else
    {
        bk.pTail->pNext = p;
        bk.pTail = p;
    }
}
```

Cấu trúc Dữ liệu và Giải thuật

35

In bảng băm

HashTable h



```
void print(HashTable h)
{
    for(int i = 0; i < Size; i++)
    {
        printf("Bucket[%d] = ", i);
        for(Node* p = h.bucket[i].pHead; p != NULL; p=p->pNext)
        {
            cout << p->data << "   ";
        }
        cout << endl;
    }
}
```

Cấu trúc Dữ liệu và Giải thuật

36



Hàm main

```
int main()
{
    HashTable h;
    initHashTable(h);

    add(h, initNode(50));
    add(h, initNode(73));
    add(h, initNode(35));
    add(h, initNode(36));
    add(h, initNode(64));
    add(h, initNode(28));
    add(h, initNode(21));
    add(h, initNode(53));
    add(h, initNode(13));

    print(h);
    return 0;
}
```

```

E:\Code_IT003\hash_table\bin... - □ ×
Bucket[0] = 50 35 90
Bucket[1] = 36 21
Bucket[2] =
Bucket[3] = 73 28 53 13
Bucket[4] = 64

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.

```

Cấu trúc Dữ liệu và Giải thuật

37



Bài tập

Viết chương trình cài đặt bảng băm liên kết để lưu các số thực

1. Viết hàm tạo dữ liệu dạng tự động cho bảng băm. Các giá trị được chọn ngẫu nhiên trong đoạn [856; 988]. Số lượng giá trị cần nhập [45; 95].
2. Viết hàm tạo dữ liệu cho bảng băm từ mảng 1 chiều.
3. Viết hàm tạo dữ liệu dạng thủ công nhập từ bàn phím.
4. Viết hàm in bảng băm.
5. Viết hàm tìm một giá trị trong bảng băm.
6. Viết hàm xóa giá trị trong bảng băm.
7. Viết hàm tính tổng các giá trị lẻ trong bảng băm.
8. Viết hàm kiểm tra bảng băm có rỗng không.
9. Viết hàm xử lý bảng băm theo một điều kiện nào đó (ví dụ: chẵn/lẻ, số âm/dương, nguyên tố...) theo yêu cầu người dùng (GV hướng dẫn trên lớp).
10. Trong hàm main thể hiện các menu lựa chọn các hàm câu 1-9.

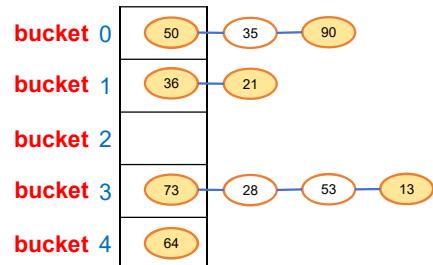
Cấu trúc Dữ liệu và Giải thuật

38



In bảng băm

HashTable **h**



Hỏi - Đáp



CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

CÂY NHỊ PHÂN TÌM KIẾM
Binary Search Tree - BST





Nội dung

1. Ý tưởng – Giải pháp
2. Kiến trúc cây nhị phân tìm kiếm
3. Các bước xây dựng
4. Các thao tác
5. Bài tập



Ý tưởng

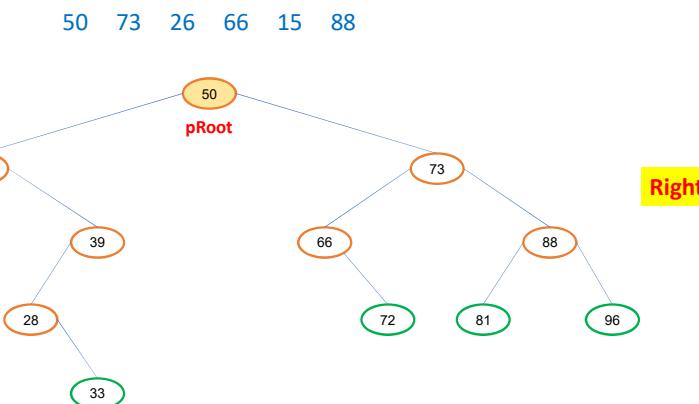
Vấn đề: Muốn lưu trữ nhiều số: 50 73 26 66 15 88

- Chúng ta làm sao?
- Xem xét dùng các cấu trúc dữ liệu sau:
 1. Mảng 1 chiều, 2 chiều
 2. Danh sách liên kết đơn, đôi, vòng...
 3. Bảng băm
 4. ...

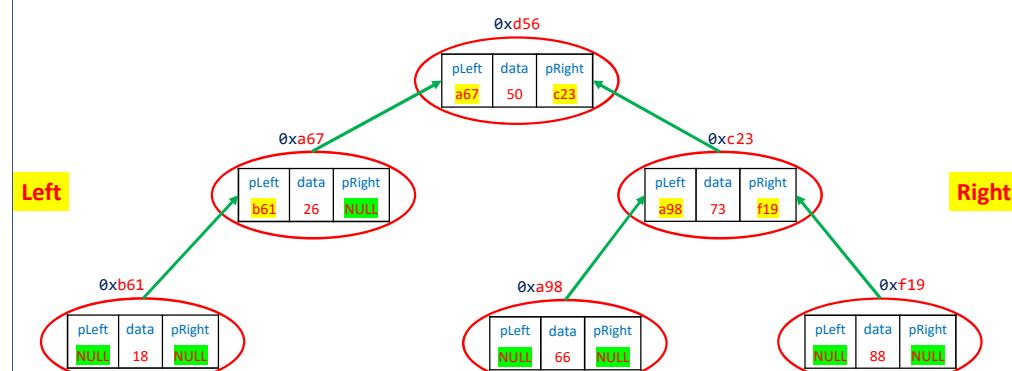


Kiến trúc

- Giá trị cây con bên **trái** của một nút **nhỏ hơn** giá trị của nút đó.
- Giá trị cây con bên **phải** của một nút **lớn hơn** giá trị của nút đó.



Kiến trúc





Các bước xây dựng

Bước #1: Tạo cấu trúc Node



```
struct Node{
    int data;
    Node* pLeft;
    Node* pRight;
};
```

Bước #2: Khởi tạo Node từ value
(Viết hàm chuyển value thành Node)

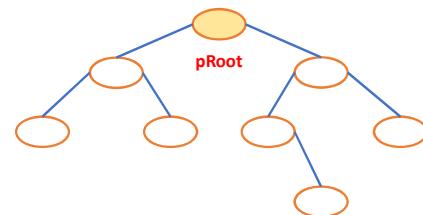
```
Node* initNode(int value)
{
    Node* p = new Node;

    p->data = value;
    p->pLeft = NULL;
    p->pRight = NULL;
    return p;
}
```

Các bước xây dựng

Bước #3: Tạo cấu trúc Tree

```
struct Tree
{
    Node* pRoot;
};
```



Bước #4: Khởi tạo Tree

```
void initTree(Tree& t)
{
    t.pRoot = NULL;
}
```



Khai báo – Khởi tạo Node và List

- Khai báo danh sách và khởi tạo danh sách

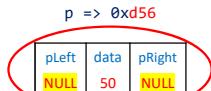
```
Tree tree;
initTree(tree);

NULL
pRoot => NULL
```

- Khai báo Node và khởi tạo Node

```
Node* p1 = initNode(50);
Node* p2 = initNode(26);
Node* p3 = initNode(73);

p => 0xd56
```



Thêm Node vào Tree

50 26 73 66 88 61

Node* p = initNode(...);

pRoot => NULL



addNode(tree, p);

1. Kiểm tra cây rỗng: Nếu cây rỗng, nút p thành nút gốc.

2. Tìm vị trí thích hợp để chèn

- Bắt đầu từ nút gốc của cây.
- So sánh giá trị của nút p với giá trị nút hiện tại
 - Nếu p **nhỏ** hơn, di chuyển qua nút con bên **trái**
 - Nếu p **lớn** hơn, di chuyển qua nút con bên **phải**
- Lặp lại quá trình này cho đến khi tìm được **vị trí trống** (nút con trái hoặc phải không tồn tại) để chèn nút mới.

3. Chèn nút mới

- Nếu giá trị của nút p < nút hiện tại thì chèn p vào vị trí bên **trái**.
- Nếu giá trị của nút p > nút hiện tại thì chèn p vào vị trí bên **phải**.



Thêm Node vào Tree

50 26 73 66 88 61

```
Node* p = initNode(...);
```

pRoot

```
addNode(tree, p);
```

1. Kiểm tra cây rỗng: Nếu cây rỗng, nút p thành nút gốc.
2. Tìm vị trí thích hợp để chèn
 - Bắt đầu từ nút gốc của cây.
 - So sánh giá trị của nút p với giá trị nút hiện tại
 - Nếu p **nhỏ** hơn, di chuyển qua nút con bên **trái**
 - Nếu p **lớn** hơn, di chuyển qua nút con bên **phải**
 - Lặp lại quá trình này cho đến khi tìm được vị trí trống (nút con trái hoặc phải không tồn tại) để chèn nút mới.
3. Chèn nút mới
 - Nếu giá trị của nút p < nút hiện tại thì chèn p vào vị trí bên **trái**.
 - Nếu giá trị của nút p > nút hiện tại thì chèn p vào vị trí bên **phải**.

Cấu trúc Dữ liệu và Giải thuật

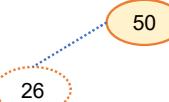
10



Thêm Node vào Tree

50 26 73 66 88 61

```
Node* p = initNode(...);
```



```
addNode(tree, p);
```

1. Kiểm tra cây rỗng: Nếu cây rỗng, nút p thành nút gốc.
2. Tìm vị trí thích hợp để chèn
 - Bắt đầu từ nút gốc của cây.
 - So sánh giá trị của nút p với giá trị nút hiện tại
 - Nếu p **nhỏ** hơn, di chuyển qua nút con bên **trái**
 - Nếu p **lớn** hơn, di chuyển qua nút con bên **phải**
 - Lặp lại quá trình này cho đến khi tìm được vị trí trống (nút con trái hoặc phải không tồn tại) để chèn nút mới.
3. Chèn nút mới
 - Nếu giá trị của nút p < nút hiện tại thì chèn p vào vị trí bên **trái**.
 - Nếu giá trị của nút p > nút hiện tại thì chèn p vào vị trí bên **phải**.

Cấu trúc Dữ liệu và Giải thuật

11



Thêm Node vào Tree

50 26 73 66 88 61

```
Node* p = initNode(...);
```



```
addNode(tree, p);
```

1. Kiểm tra cây rỗng: Nếu cây rỗng, nút p thành nút gốc.
2. Tìm vị trí thích hợp để chèn
 - Bắt đầu từ nút gốc của cây.
 - So sánh giá trị của nút p với giá trị nút hiện tại
 - Nếu p **nhỏ** hơn, di chuyển qua nút con bên **trái**
 - Nếu p **lớn** hơn, di chuyển qua nút con bên **phải**
 - Lặp lại quá trình này cho đến khi tìm được vị trí trống (nút con trái hoặc phải không tồn tại) để chèn nút mới.
3. Chèn nút mới
 - Nếu giá trị của nút p < nút hiện tại thì chèn p vào vị trí bên **trái**.
 - Nếu giá trị của nút p > nút hiện tại thì chèn p vào vị trí bên **phải**.

Cấu trúc Dữ liệu và Giải thuật

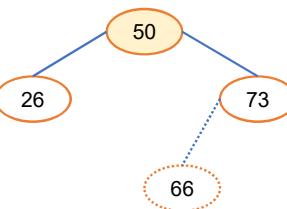
12



Thêm Node vào Tree

50 26 73 66 88 61

```
Node* p = initNode(...);
```



```
addNode(tree, p);
```

1. Kiểm tra cây rỗng: Nếu cây rỗng, nút p thành nút gốc.
2. Tìm vị trí thích hợp để chèn
 - Bắt đầu từ nút gốc của cây.
 - So sánh giá trị của nút p với giá trị nút hiện tại
 - Nếu p **nhỏ** hơn, di chuyển qua nút con bên **trái**
 - Nếu p **lớn** hơn, di chuyển qua nút con bên **phải**
 - Lặp lại quá trình này cho đến khi tìm được vị trí trống (nút con trái hoặc phải không tồn tại) để chèn nút mới.
3. Chèn nút mới
 - Nếu giá trị của nút p < nút hiện tại thì chèn p vào vị trí bên **trái**.
 - Nếu giá trị của nút p > nút hiện tại thì chèn p vào vị trí bên **phải**.

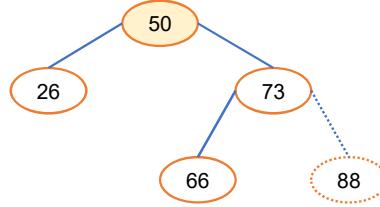
Cấu trúc Dữ liệu và Giải thuật

13



Thêm Node vào Tree

```
50 26 73 66 88 61
Node* p = initNode(...);
```



`addNode(tree, p);`

1. Kiểm tra cây rỗng: Nếu cây rỗng, nút **p** thành **nút gốc**.

2. Tìm vị trí thích hợp để chèn

- Bắt đầu từ **nút gốc** của cây.
- So sánh giá trị của nút **p** với giá trị **nút hiện tại**
 - Nếu **p** **nhỏ** hơn, di chuyển qua nút con bên **trái**
 - Nếu **p** **lớn** hơn, di chuyển qua nút con bên **phải**
- Lặp lại quá trình này cho đến khi tìm được vị trí trống (nút con trái hoặc phải không tồn tại) để chèn nút mới.

3. Chèn nút mới

- Nếu giá trị của nút **p < nút hiện tại** thì chèn **p** vào vị trí bên **trái**.
- Nếu giá trị của nút **p > nút hiện tại** thì chèn **p** vào vị trí bên **phải**.

Cấu trúc Dữ liệu và Giải thuật

```
50 26 73 66 88 61
Node* p = initNode(...);
```



Thêm Node vào Tree

`addNode(tree, p);`

1. Kiểm tra cây rỗng: Nếu cây rỗng, nút **p** thành **nút gốc**.

2. Tìm vị trí thích hợp để chèn

- Bắt đầu từ **nút gốc** của cây.
- So sánh giá trị của nút **p** với giá trị **nút hiện tại**
 - Nếu **p** **nhỏ** hơn, di chuyển qua nút con bên **trái**
 - Nếu **p** **lớn** hơn, di chuyển qua nút con bên **phải**
- Lặp lại quá trình này cho đến khi tìm được vị trí trống (nút con trái hoặc phải không tồn tại) để chèn nút mới.

3. Chèn nút mới

- Nếu giá trị của nút **p < nút hiện tại** thì chèn **p** vào vị trí bên **trái**.
- Nếu giá trị của nút **p > nút hiện tại** thì chèn **p** vào vị trí bên **phải**.

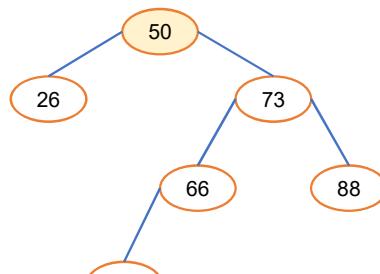
Cấu trúc Dữ liệu và Giải thuật

15



Thêm Node vào Tree

```
50 26 73 66 88 61
Node* p = initNode(...);
```



`addNode(tree, p);`

1. Kiểm tra cây rỗng: Nếu cây rỗng, nút **p** thành **nút gốc**.

2. Tìm vị trí thích hợp để chèn

- Bắt đầu từ **nút gốc** của cây.
- So sánh giá trị của nút **p** với giá trị **nút hiện tại**
 - Nếu **p** **nhỏ** hơn, di chuyển qua nút con bên **trái**
 - Nếu **p** **lớn** hơn, di chuyển qua nút con bên **phải**
- Lặp lại quá trình này cho đến khi tìm được vị trí trống (nút con trái hoặc phải không tồn tại) để chèn nút mới.

3. Chèn nút mới

- Nếu giá trị của nút **p < nút hiện tại** thì chèn **p** vào vị trí bên **trái**.
- Nếu giá trị của nút **p > nút hiện tại** thì chèn **p** vào vị trí bên **phải**.

Cấu trúc Dữ liệu và Giải thuật

16

```
void addNode(Tree& tree, Node* p)
{
    if(tree.pRoot == NULL)
    {
        tree.pRoot = p;
        return;
    }

    Node* pGoto = tree.pRoot;
    Node* pLoca = NULL;

    while(pGoto != NULL)
    {
        pLoca = pGoto;
        if(p->data < pGoto->data)
            pGoto = pGoto->pLeft;
        else if(p->data > pGoto->data)
            pGoto = pGoto->pRight;
    }

    if(p->data < pLoca->data)
        pLoca->pLeft = p;
    else if(p->data > pLoca->data)
        pLoca->pRight = p;
}
```

17

```
addNode(tree, p);
```

1. Kiểm tra cây rỗng: Nếu cây rỗng, nút **p** thành nút gốc.

2. Tìm vị trí thích hợp để chèn

- Bắt đầu từ nút gốc của cây.
- So sánh giá trị của nút **p** với giá trị nút hiện tại
 - Nếu **p** nhỏ hơn, di chuyển qua nút con bên trái
 - Nếu **p** lớn hơn, di chuyển qua nút con bên phải
- Lặp lại quá trình này cho đến khi tìm được vị trí trống (nút con trái hoặc phải không tồn tại) để chèn nút mới.

3. Chèn nút mới

- Nếu giá trị của nút **p < nút hiện tại** thì chèn **p** vào vị trí bên trái.
- Nếu giá trị của nút **p > nút hiện tại** thì chèn **p** vào vị trí bên phải.

```
void addNode(Tree& tree, Node* p)
{
    if(tree.pRoot == NULL)
    {
        tree.pRoot = p;
        return;
    }

    Node* pGoto = tree.pRoot;
    Node* pLoca = NULL;

    while(pGoto != NULL)
    {
        pLoca = pGoto;
        if(p->data < pGoto->data)
            pGoto = pGoto->pLeft;
        else if(p->data > pGoto->data)
            pGoto = pGoto->pRight;
    }

    if(p->data < pLoca->data)
        pLoca->pLeft = p;
    else if(p->data > pLoca->data)
        pLoca->pRight = p;
}
```

```
addNode(tree, p);
```

1. Kiểm tra cây rỗng: Nếu cây rỗng, nút **p** thành nút gốc.

2. Tìm vị trí thích hợp để chèn

- Bắt đầu từ nút gốc của cây.
- So sánh giá trị của nút **p** với giá trị nút hiện tại
 - Nếu **p** nhỏ hơn, di chuyển qua nút con bên trái
 - Nếu **p** lớn hơn, di chuyển qua nút con bên phải
- Lặp lại quá trình này cho đến khi tìm được vị trí trống (nút con trái hoặc phải không tồn tại) để chèn nút mới.

3. Chèn nút mới

- Nếu giá trị của nút **p < nút hiện tại** thì chèn **p** vào vị trí bên trái.
- Nếu giá trị của nút **p > nút hiện tại** thì chèn **p** vào vị trí bên phải.

```
void addNode(Tree& tree, Node* p)
{
    if(tree.pRoot == NULL)
    {
        tree.pRoot = p;
        return;
    }

    Node* pGoto = tree.pRoot;
    Node* pLoca = NULL;

    while(pGoto != NULL)
    {
        pLoca = pGoto;
        if(p->data < pGoto->data)
            pGoto = pGoto->pLeft;
        else if(p->data > pGoto->data)
            pGoto = pGoto->pRight;
    }

    if(p->data < pLoca->data)
        pLoca->pLeft = p;
    else if(p->data > pLoca->data)
        pLoca->pRight = p;
}
```

```
addNode(tree, p);
```

1. Kiểm tra cây rỗng: Nếu cây rỗng, nút **p** thành nút gốc.

2. Tìm vị trí thích hợp để chèn

- Bắt đầu từ nút gốc của cây.
- So sánh giá trị của nút **p** với giá trị nút hiện tại
 - Nếu **p** nhỏ hơn, di chuyển qua nút con bên trái
 - Nếu **p** lớn hơn, di chuyển qua nút con bên phải
- Lặp lại quá trình này cho đến khi tìm được vị trí trống (nút con trái hoặc phải không tồn tại) để chèn nút mới.

3. Chèn nút mới

- Nếu giá trị của nút **p < nút hiện tại** thì chèn **p** vào vị trí bên trái.
- Nếu giá trị của nút **p > nút hiện tại** thì chèn **p** vào vị trí bên phải.

```
void addNode(Tree& tree, Node* p)
{
    if(tree.pRoot == NULL)
    {
        tree.pRoot = p;
        return;
    }

    Node* pGoto = tree.pRoot;
    Node* pLoca = NULL;

    while(pGoto != NULL)
    {
        pLoca = pGoto;
        if(p->data < pGoto->data)
            pGoto = pGoto->pLeft;
        else if(p->data > pGoto->data)
            pGoto = pGoto->pRight;
    }

    if(p->data < pLoca->data)
        pLoca->pLeft = p;
    else if(p->data > pLoca->data)
        pLoca->pRight = p;
}
```

Chú ý: Code này chưa xử lý trường hợp thêm trùng giá trị.

```
addNode(tree, p);
```

1. Kiểm tra cây rỗng: Nếu cây rỗng, nút **p** thành nút gốc.

2. Tìm vị trí thích hợp để chèn

- Bắt đầu từ nút gốc của cây.
- So sánh giá trị của nút **p** với giá trị nút hiện tại
 - Nếu **p** nhỏ hơn, di chuyển qua nút con bên trái
 - Nếu **p** lớn hơn, di chuyển qua nút con bên phải
- Lặp lại quá trình này cho đến khi tìm được vị trí trống (nút con trái hoặc phải không tồn tại) để chèn nút mới.

3. Chèn nút mới

- Nếu giá trị của nút **p < nút hiện tại** thì chèn **p** vào vị trí bên trái.
- Nếu giá trị của nút **p > nút hiện tại** thì chèn **p** vào vị trí bên phải.

```
void addNode(Tree& tree, Node* p)
{
    if(tree.pRoot == NULL)
    {
        tree.pRoot = p;
        return;
    }

    Node* pGoto = tree.pRoot;
    Node* pLoca = NULL;

    while(pGoto != NULL)
    {
        pLoca = pGoto;
        if(p->data < pGoto->data)
            pGoto = pGoto->pLeft;
        else if(p->data > pGoto->data)
            pGoto = pGoto->pRight;
    }

    if(p->data < pLoca->data)
        pLoca->pLeft = p;
    else if(p->data > pLoca->data)
        pLoca->pRight = p;
}
```



Duyệt cây

1. Cách duyệt InOrder: còn gọi là Left - Node - Right (**LNR**)

- Duyệt cây con bên left
- **Xử lý node**
- Duyệt cây con bên right

=> Với cách duyệt InOrder (LNR) các giá trị node trong cây sẽ duyệt theo thứ tự tăng dần.

2. Cách duyệt PreOrder: còn gọi là Node - Left - Right (**NLR**)

- **Xử lý node**
- Duyệt cây con bên left
- Duyệt cây con bên right

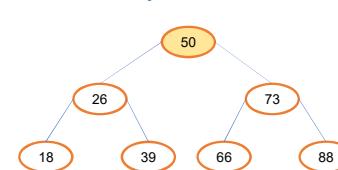
3. Cách duyệt PostOrder: còn gọi là Left - Right - Node (**LRN**)

- Duyệt cây con bên left
- Duyệt cây con bên right
- **Xử lý node**



Hàm duyệt Node

Minh họa duyệt InOrder - LNR



Kết quả:

...

1 – Tạo một stack rỗng.

- Bắt đầu đi từ node gốc.

2 – Duyệt trái

- Đưa các node đi qua vào stack.
- Di chuyển hết node con bên trái.

3 – Xử lý node

- Lấy node trong stack để xử lý.
- In ra giá trị hoặc xử lý khác.

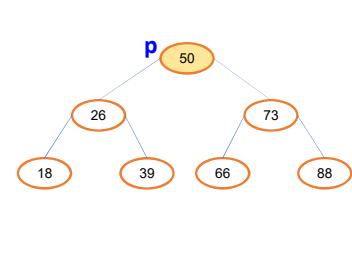
4 – Di chuyển qua node con bên phải.

5 – Lặp #2

- Dừng khi stack rỗng.
- Không còn node để đi.



Hàm duyệt Node



1 – Tạo một stack rỗng.

- Bắt đầu đi từ node gốc.

2 – Duyệt trái

- Đưa các node đi qua vào stack.
- Di chuyển hết node con bên trái.

3 – Xử lý node

- Lấy node trong stack để xử lý.
- In ra giá trị hoặc xử lý khác.

4 – Di chuyển qua node con bên phải.

5 – Lặp #2

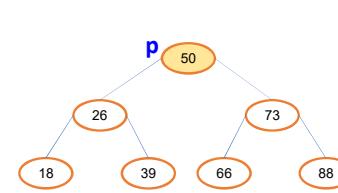
- Dừng khi stack rỗng.
- Không còn node để đi.

Kết quả:

...



Hàm duyệt Node



1 – Tạo một stack rỗng.

- Bắt đầu đi từ node gốc.

2 – Duyệt trái

- Đưa các node đi qua vào stack.
- Di chuyển hết node con bên trái.

3 – Xử lý node

- Lấy node trong stack để xử lý.
- In ra giá trị hoặc xử lý khác.

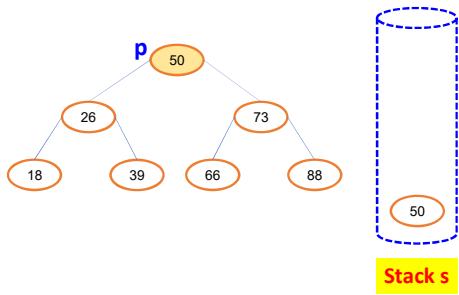
4 – Di chuyển qua node con bên phải.

5 – Lặp #2

- Dừng khi stack rỗng.
- Không còn node để đi.



Hàm duyệt Node



Kết quả:

...

- 1 – Tạo một stack rỗng.**
 - Bắt đầu đi từ node gốc.
- 2 – Duyệt trái**
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node**
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.**
- 5 – Lặp #2**
 - Dừng khi stack rỗng.
 - Không còn node để đi.

Cấu trúc Dữ liệu và Giải thuật

26

27

Hàm duyệt Node



Kết quả:

...

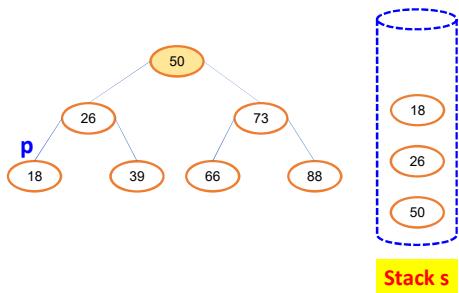
- 1 – Tạo một stack rỗng.**
 - Bắt đầu đi từ node gốc.
- 2 – Duyệt trái**
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node**
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.**
- 5 – Lặp #2**
 - Dừng khi stack rỗng.
 - Không còn node để đi.

Cấu trúc Dữ liệu và Giải thuật

27



Hàm duyệt Node



Kết quả:

...

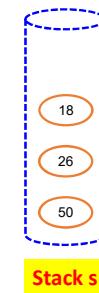
- 1 – Tạo một stack rỗng.**
 - Bắt đầu đi từ node gốc.
- 2 – Duyệt trái**
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node**
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.**
- 5 – Lặp #2**
 - Dừng khi stack rỗng.
 - Không còn node để đi.

Cấu trúc Dữ liệu và Giải thuật

28



Hàm duyệt Node



Kết quả:

...

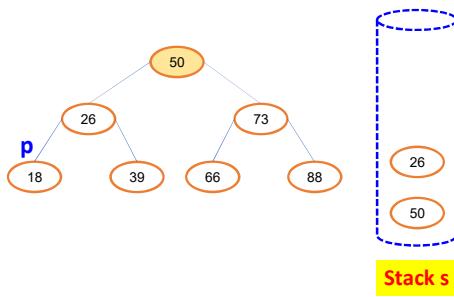
- 1 – Tạo một stack rỗng.**
 - Bắt đầu đi từ node gốc.
- 2 – Duyệt trái**
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node**
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.**
- 5 – Lặp #2**
 - Dừng khi stack rỗng.
 - Không còn node để đi.

Cấu trúc Dữ liệu và Giải thuật

29



Hàm duyệt Node

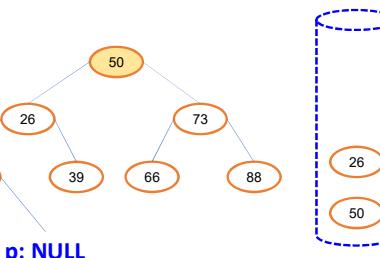


Kết quả:
18

- 1 – Tạo một stack rỗng.
- Bắt đầu đi từ node gốc.
- 2 – Duyệt trái
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.
- 5 – Lặp #2
 - Dừng khi stack rỗng.
 - Không còn node để đi.



Hàm duyệt Node

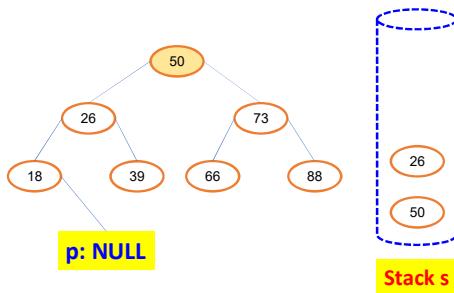


Kết quả:
18

- 1 – Tạo một stack rỗng.
- Bắt đầu đi từ node gốc.
- 2 – Duyệt trái
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.
- 5 – Lặp #2
 - Dừng khi stack rỗng.
 - Không còn node để đi.



Hàm duyệt Node

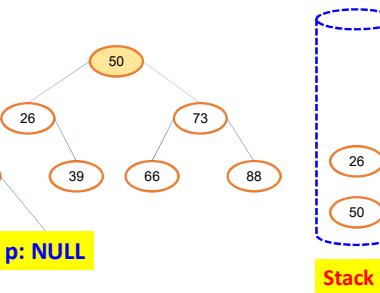


Kết quả:
18

- 1 – Tạo một stack rỗng.
- Bắt đầu đi từ node gốc.
- 2 – Duyệt trái
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.
- 5 – Lặp #2
 - Dừng khi stack rỗng.
 - Không còn node để đi.



Hàm duyệt Node

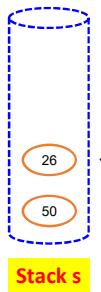
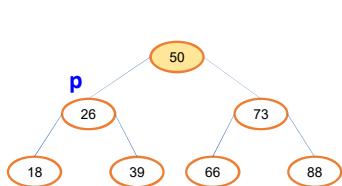


Kết quả:
18

- 1 – Tạo một stack rỗng.
- Bắt đầu đi từ node gốc.
- 2 – Duyệt trái
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.
- 5 – Lặp #2
 - Dừng khi stack rỗng.
 - Không còn node để đi.



Hàm duyệt Node

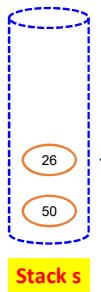
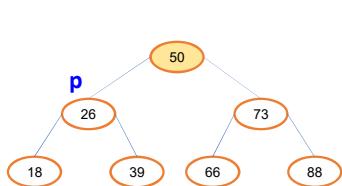


- 1 – Tạo một stack rỗng.**
 - Bắt đầu đi từ node gốc.
- 2 – Duyệt trái**
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node**
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.**
- 5 – Lặp #2**
 - Dừng khi stack rỗng.
 - Không còn node để đi.

Kết quả:
18



Hàm duyệt Node

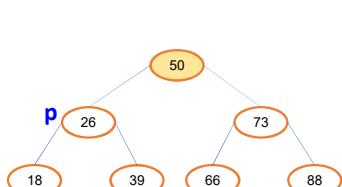


- 1 – Tạo một stack rỗng.**
 - Bắt đầu đi từ node gốc.
- 2 – Duyệt trái**
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node**
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.**
- 5 – Lặp #2**
 - Dừng khi stack rỗng.
 - Không còn node để đi.

Kết quả:
18



Hàm duyệt Node

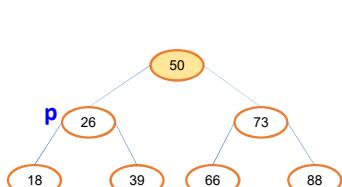


- 1 – Tạo một stack rỗng.**
 - Bắt đầu đi từ node gốc.
- 2 – Duyệt trái**
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node**
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.**
- 5 – Lặp #2**
 - Dừng khi stack rỗng.
 - Không còn node để đi.

Kết quả:
18 26



Hàm duyệt Node

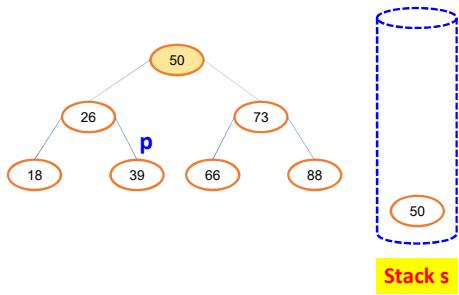


- 1 – Tạo một stack rỗng.**
 - Bắt đầu đi từ node gốc.
- 2 – Duyệt trái**
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node**
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.**
- 5 – Lặp #2**
 - Dừng khi stack rỗng.
 - Không còn node để đi.

Kết quả:
18 26



Hàm duyệt Node



Kết quả:

18 26

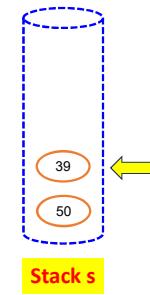
- 1 – Tạo một stack rỗng.**
 - Bắt đầu đi từ node gốc.
- 2 – Duyệt trái**
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node**
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.**
- 5 – Lặp #2**
 - Dừng khi stack rỗng.
 - Không còn node để đi.

Cấu trúc Dữ liệu và Giải thuật

38



Hàm duyệt Node



Kết quả:

18 26

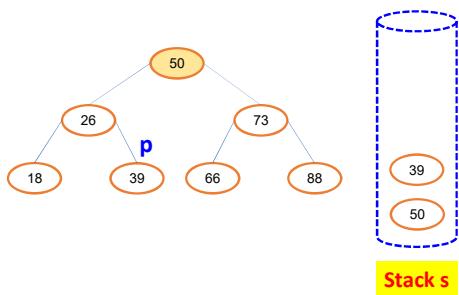
- 1 – Tạo một stack rỗng.**
 - Bắt đầu đi từ node gốc.
- 2 – Duyệt trái**
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node**
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.**
- 5 – Lặp #2**
 - Dừng khi stack rỗng.
 - Không còn node để đi.

Cấu trúc Dữ liệu và Giải thuật

39



Hàm duyệt Node



Kết quả:

18 26

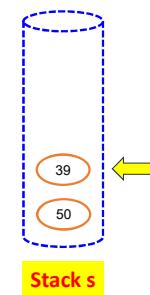
- 1 – Tạo một stack rỗng.**
 - Bắt đầu đi từ node gốc.
- 2 – Duyệt trái**
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node**
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.**
- 5 – Lặp #2**
 - Dừng khi stack rỗng.
 - Không còn node để đi.

Cấu trúc Dữ liệu và Giải thuật

40



Hàm duyệt Node



Kết quả:

18 26

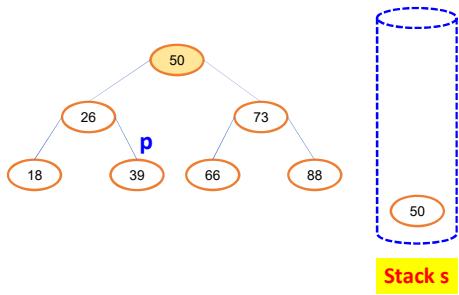
- 1 – Tạo một stack rỗng.**
 - Bắt đầu đi từ node gốc.
- 2 – Duyệt trái**
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node**
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.**
- 5 – Lặp #2**
 - Dừng khi stack rỗng.
 - Không còn node để đi.

Cấu trúc Dữ liệu và Giải thuật

41



Hàm duyệt Node



Kết quả:
18 26 39

- 1 – Tạo một stack rỗng.
- Bắt đầu đi từ node gốc.
- 2 – Duyệt trái
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.
- 5 – Lặp #2
 - Dừng khi stack rỗng.
 - Không còn node để đi.



Hàm duyệt Node

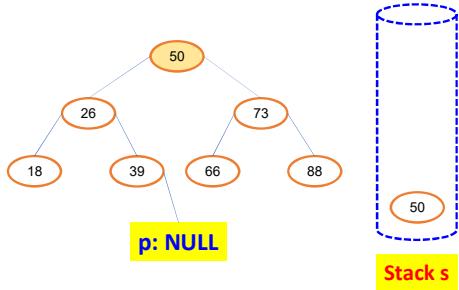


Kết quả:
18 26 39

- 1 – Tạo một stack rỗng.
- Bắt đầu đi từ node gốc.
- 2 – Duyệt trái
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.
- 5 – Lặp #2
 - Dừng khi stack rỗng.
 - Không còn node để đi.



Hàm duyệt Node

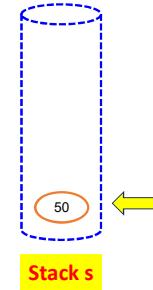


Kết quả:
18 26 39

- 1 – Tạo một stack rỗng.
- Bắt đầu đi từ node gốc.
- 2 – Duyệt trái
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.
- 5 – Lặp #2
 - Dừng khi stack rỗng.
 - Không còn node để đi.



Hàm duyệt Node

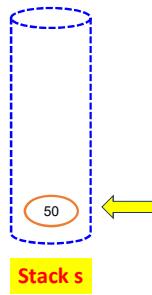
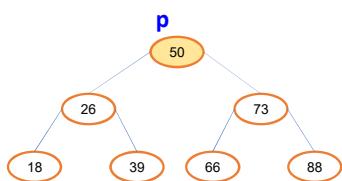


Kết quả:
18 26 39

- 1 – Tạo một stack rỗng.
- Bắt đầu đi từ node gốc.
- 2 – Duyệt trái
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.
- 5 – Lặp #2
 - Dừng khi stack rỗng.
 - Không còn node để đi.



Hàm duyệt Node

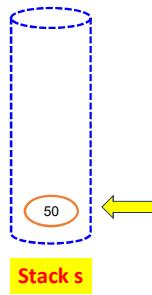
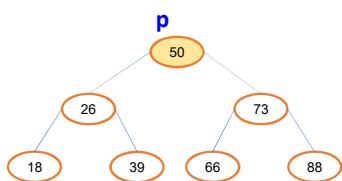


- 1 – Tạo một stack rỗng.**
 - Bắt đầu đi từ node gốc.
- 2 – Duyệt trái**
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node**
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.**
- 5 – Lặp #2**
 - Dừng khi stack rỗng.
 - Không còn node để đi.

Kết quả:
18 26 39



Hàm duyệt Node

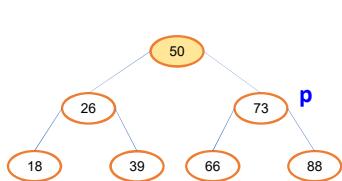


- 1 – Tạo một stack rỗng.**
 - Bắt đầu đi từ node gốc.
- 2 – Duyệt trái**
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node**
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.**
- 5 – Lặp #2**
 - Dừng khi stack rỗng.
 - Không còn node để đi.

Kết quả:
18 26 39 50



Hàm duyệt Node

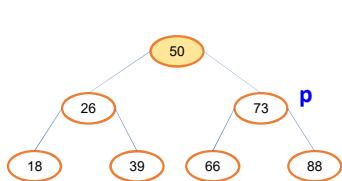


- 1 – Tạo một stack rỗng.**
 - Bắt đầu đi từ node gốc.
- 2 – Duyệt trái**
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node**
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.**
- 5 – Lặp #2**
 - Dừng khi stack rỗng.
 - Không còn node để đi.

Kết quả:
18 26 39 50



Hàm duyệt Node

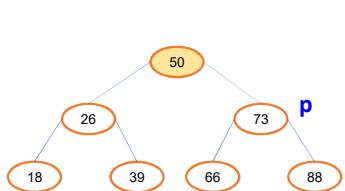


- 1 – Tạo một stack rỗng.**
 - Bắt đầu đi từ node gốc.
- 2 – Duyệt trái**
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node**
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.**
- 5 – Lặp #2**
 - Dừng khi stack rỗng.
 - Không còn node để đi.

Kết quả:
18 26 39 50



Hàm duyệt Node



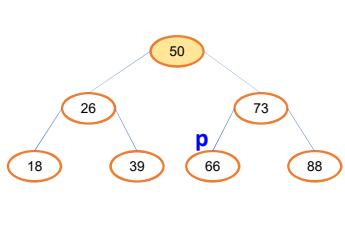
Kết quả:
18 26 39 50

Cấu trúc Dữ liệu và Giải thuật

50

- 1 – Tạo một stack rỗng.
 - Bắt đầu đi từ node gốc.
- 2 – Duyệt trái
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.
- 5 – Lặp #2
 - Dừng khi stack rỗng.
 - Không còn node để đi.

Hàm duyệt Node



Kết quả:
18 26 39 50

Cấu trúc Dữ liệu và Giải thuật

52

- 1 – Tạo một stack rỗng.
 - Bắt đầu đi từ node gốc.
- 2 – Duyệt trái
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.
- 5 – Lặp #2
 - Dừng khi stack rỗng.
 - Không còn node để đi.

Hàm duyệt Node



Kết quả:
18 26 39 50

Cấu trúc Dữ liệu và Giải thuật

51

- 1 – Tạo một stack rỗng.
 - Bắt đầu đi từ node gốc.
- 2 – Duyệt trái
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.
- 5 – Lặp #2
 - Dừng khi stack rỗng.
 - Không còn node để đi.

Hàm duyệt Node



Kết quả:
18 26 39 50

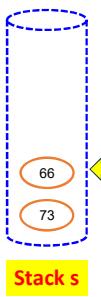
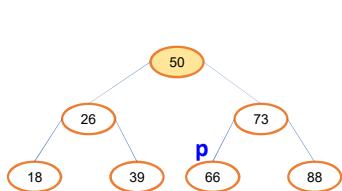
Cấu trúc Dữ liệu và Giải thuật

53

- 1 – Tạo một stack rỗng.
 - Bắt đầu đi từ node gốc.
- 2 – Duyệt trái
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.
- 5 – Lặp #2
 - Dừng khi stack rỗng.
 - Không còn node để đi.



Hàm duyệt Node



Kết quả:
18 26 39 50

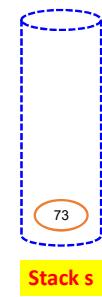
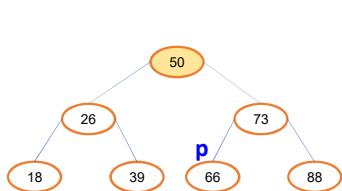
Cấu trúc Dữ liệu và Giải thuật

54

- 1 – Tạo một stack rỗng.
 - Bắt đầu đi từ node gốc.
- 2 – Duyệt trái
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.
- 5 – Lặp #2
 - Dừng khi stack rỗng.
 - Không còn node để đi.



Hàm duyệt Node



Kết quả:
18 26 39 50 66

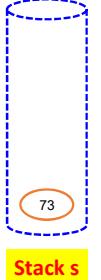
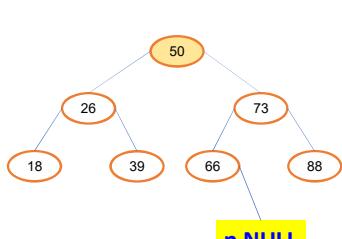
Cấu trúc Dữ liệu và Giải thuật

55

- 1 – Tạo một stack rỗng.
 - Bắt đầu đi từ node gốc.
- 2 – Duyệt trái
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.
- 5 – Lặp #2
 - Dừng khi stack rỗng.
 - Không còn node để đi.



Hàm duyệt Node



Kết quả:
18 26 39 50 66

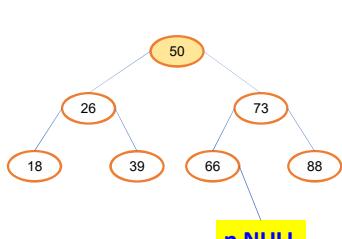
Cấu trúc Dữ liệu và Giải thuật

56

- 1 – Tạo một stack rỗng.
 - Bắt đầu đi từ node gốc.
- 2 – Duyệt trái
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.
- 5 – Lặp #2
 - Dừng khi stack rỗng.
 - Không còn node để đi.



Hàm duyệt Node



Kết quả:
18 26 39 50 66

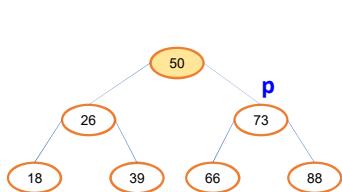
Cấu trúc Dữ liệu và Giải thuật

57

- 1 – Tạo một stack rỗng.
 - Bắt đầu đi từ node gốc.
- 2 – Duyệt trái
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.
- 5 – Lặp #2
 - Dừng khi stack rỗng.
 - Không còn node để đi.



Hàm duyệt Node



Stack s

Kết quả:

18 26 39 50 66

Cấu trúc Dữ liệu và Giải thuật

58

- 1 – Tạo một stack rỗng.**
 - Bắt đầu đi từ node gốc.
- 2 – Duyệt trái**
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node**
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.**
- 5 – Lặp #2**
 - Dừng khi stack rỗng.
 - Không còn node để đi.



Hàm duyệt Node



Stack s

Kết quả:

18 26 39 50 66 73

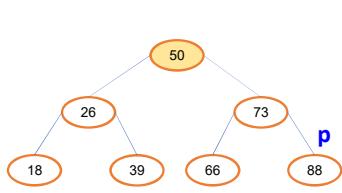
Cấu trúc Dữ liệu và Giải thuật

59

- 1 – Tạo một stack rỗng.**
 - Bắt đầu đi từ node gốc.
- 2 – Duyệt trái**
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node**
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.**
- 5 – Lặp #2**
 - Dừng khi stack rỗng.
 - Không còn node để đi.



Hàm duyệt Node



Stack s

Kết quả:

18 26 39 50 66 73

Cấu trúc Dữ liệu và Giải thuật

60

- 1 – Tạo một stack rỗng.**
 - Bắt đầu đi từ node gốc.
- 2 – Duyệt trái**
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node**
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.**
- 5 – Lặp #2**
 - Dừng khi stack rỗng.
 - Không còn node để đi.



Hàm duyệt Node



Stack s

Kết quả:

18 26 39 50 66 73

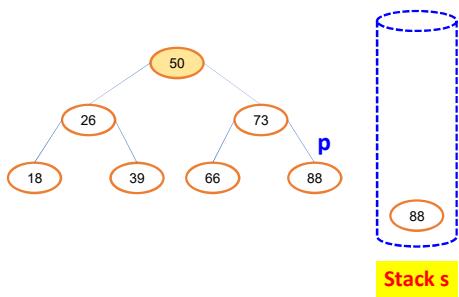
Cấu trúc Dữ liệu và Giải thuật

61

- 1 – Tạo một stack rỗng.**
 - Bắt đầu đi từ node gốc.
- 2 – Duyệt trái**
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node**
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.**
- 5 – Lặp #2**
 - Dừng khi stack rỗng.
 - Không còn node để đi.



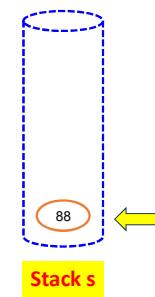
Hàm duyệt Node



Kết quả:
18 26 39 50 66 73

- 1 – Tạo một stack rỗng.
 - Bắt đầu đi từ node gốc.
- 2 – Duyệt trái
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.
- 5 – Lặp #2
 - Dừng khi stack rỗng.
 - Không còn node để đi.

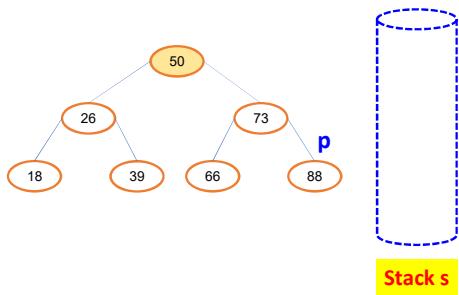
Hàm duyệt Node



Kết quả:
18 26 39 50 66 73



Hàm duyệt Node



Kết quả:
18 26 39 50 66 73 88

- 1 – Tạo một stack rỗng.
 - Bắt đầu đi từ node gốc.
- 2 – Duyệt trái
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.
- 5 – Lặp #2
 - Dừng khi stack rỗng.
 - Không còn node để đi.

Hàm duyệt Node

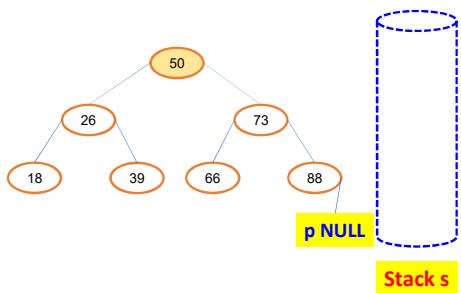


Kết quả:
18 26 39 50 66 73 88

- 1 – Tạo một stack rỗng.
 - Bắt đầu đi từ node gốc.
- 2 – Duyệt trái
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.
- 5 – Lặp #2
 - Dừng khi stack rỗng.
 - Không còn node để đi.



Hàm duyệt Node

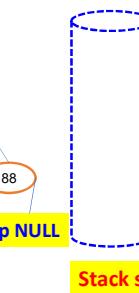


Kết quả:

18 26 39 50 66 73 88



Hàm duyệt Node



Đến đây dừng vì không còn node để đi và stack không còn node.

Kết quả:

18 26 39 50 66 73 88



Hàm duyệt Node

Không dùng đệ quy

```
void inOrder(Tree tree) {
    stack<Node*> s;
    Node* p = tree.pRoot;

    while (p != NULL || !s.empty()) {
        while (p != NULL) {
            s.push(p);
            p = p->pLeft;
        }

        p = s.top();
        s.pop();
        cout << p->data << " ";
        p = p->pRight;
    }
}
```

1 – Tạo một stack rỗng.

- Bắt đầu đi từ node gốc.

2 – Duyệt trái

- Đưa các node đi qua vào stack.
- Di chuyển hết node con bên trái.

3 – Xử lý node

- Lấy node trong stack để xử lý.
- In ra giá trị hoặc xử lý khác.

4 – Di chuyển qua node con bên phải.

5 – Lặp #2

- Dừng khi stack rỗng.
- Không còn node để đi.



Hàm duyệt Node

Không dùng đệ quy

```
void inOrder(Tree tree) {
    stack<Node*> s;
    Node* p = tree.pRoot;

    while (p != NULL || !s.empty()) {
        while (p != NULL) {
            s.push(p);
            p = p->pLeft;
        }

        p = s.top();
        s.pop();
        cout << p->data << " ";
        p = p->pRight;
    }
}
```

1 – Tạo một stack rỗng.

- Bắt đầu đi từ node gốc.

2 – Duyệt trái

- Đưa các node đi qua vào stack.
- Di chuyển hết node con bên trái.

3 – Xử lý node

- Lấy node trong stack để xử lý.
- In ra giá trị hoặc xử lý khác.

4 – Di chuyển qua node con bên phải.

5 – Lặp #2

- Dừng khi stack rỗng.
- Không còn node để đi.



Hàm duyệt Node

Không dùng đệ quy

```
void inOrder(Tree tree) {
    stack<Node*> s;
    Node* p = tree.pRoot;

    while (p != NULL || s.empty() == NULL) {
        while (p != NULL) {
            s.push(p);
            p = p->pLeft;
        }

        p = s.top();
        s.pop();
        cout << p->data << " ";
        p = p->pRight;
    }
}
```

- 1 – Tạo một stack rỗng.
 - Bắt đầu đi từ node gốc.
- 2 – Duyệt trái
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.
- 5 – Lặp #2
 - Dừng khi stack rỗng.
 - Không còn node để đi.

Cấu trúc Dữ liệu và Giải thuật

70

Hàm duyệt Node

Không dùng đệ quy

```
void inOrder(Tree tree) {
    stack<Node*> s;
    Node* p = tree.pRoot;

    while (p != NULL || s.empty() == NULL) {
        while (p != NULL) {
            s.push(p);
            p = p->pLeft;
        }

        p = s.top();
        s.pop();
        cout << p->data << " ";
        p = p->pRight;
    }
}
```

- 1 – Tạo một stack rỗng.
 - Bắt đầu đi từ node gốc.
- 2 – Duyệt trái
 - Đưa các node đi qua vào stack.
 - Di chuyển hết node con bên trái.
- 3 – Xử lý node
 - Lấy node trong stack để xử lý.
 - In ra giá trị hoặc xử lý khác.
- 4 – Di chuyển qua node con bên phải.
- 5 – Lặp #2
 - Dừng khi stack rỗng.
 - Không còn node để đi.

Cấu trúc Dữ liệu và Giải thuật

71



Hàm duyệt Node

Không dùng đệ quy

```
void inOrder(Tree tree) {
    stack<Node*> s;
    Node* p = tree.pRoot;

    while (p != NULL || s.empty() == NULL) {
        while (p != NULL) {
            s.push(p);
            p = p->pLeft;
        }

        p = s.top();
        s.pop();
        cout << p->data << " ";
        p = p->pRight;
    }
}
```



Hàm duyệt Node

Không dùng đệ quy

```
void inOrder(Tree tree) {
    stack<Node*> s;
    Node* p = tree.pRoot;

    while (p != NULL || s.empty() == NULL) {
        while (p != NULL) {
            s.push(p);
            p = p->pLeft;
        }

        p = s.top();
        s.pop();
        cout << p->data << " ";
        p = p->pRight;
    }
}
```

Hoán vị các khối code
==> cách duyệt khác.

Cấu trúc Dữ liệu và Giải thuật

72

73



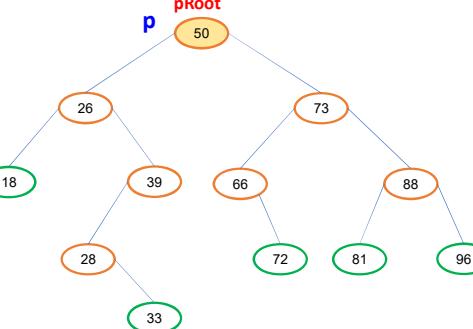
Hàm duyệt Node

Dùng đệ quy

```
void LNR(Node* p)
{
    if(p != NULL)
    {
        LNR(p->pLeft);
        cout << p->data << " ";
        LNR(p->pRight);
    }
}
```

Tìm kiếm

Tìm $x = 28$



Bước 1: Bắt đầu đi từ nút gốc

Bước 2: So sánh giá trị tìm kiếm với giá trị nút hiện tại

- Nếu giá trị tìm == giá trị nút hiện tại => tìm thấy (dừng)
- Nếu giá trị tìm < giá trị nút hiện tại => di tiếp sang Trái
- Nếu giá trị tìm > giá trị nút hiện tại => di tiếp sang Phải

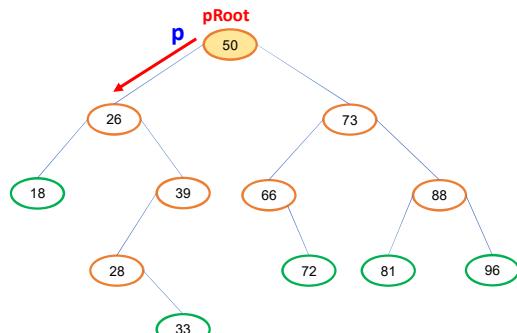
Bước 3: Lặp lại **Bước 2**. Kết thúc việc tìm kiếm khi:

- Tìm thấy giá trị
- Hoặc không còn node để so sánh



Tìm kiếm

Tìm $x = 28$



Bước 1: Bắt đầu đi từ nút gốc

Bước 2: So sánh giá trị tìm kiếm với giá trị nút hiện tại

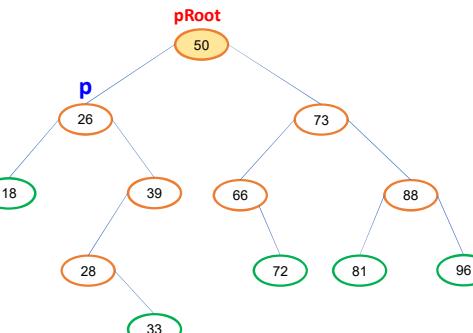
- Nếu giá trị tìm == giá trị nút hiện tại => tìm thấy (dừng)
- Nếu giá trị tìm < giá trị nút hiện tại => di tiếp sang Trái
- Nếu giá trị tìm > giá trị nút hiện tại => di tiếp sang Phải

Bước 3: Lặp lại **Bước 2**. Kết thúc việc tìm kiếm khi:

- Tìm thấy giá trị
- Hoặc không còn node để so sánh

Tìm kiếm

Tìm $x = 28$



Bước 1: Bắt đầu đi từ nút gốc

Bước 2: So sánh giá trị tìm kiếm với giá trị nút hiện tại

- Nếu giá trị tìm == giá trị nút hiện tại => tìm thấy (dừng)
- Nếu giá trị tìm < giá trị nút hiện tại => di tiếp sang Trái
- Nếu giá trị tìm > giá trị nút hiện tại => di tiếp sang Phải

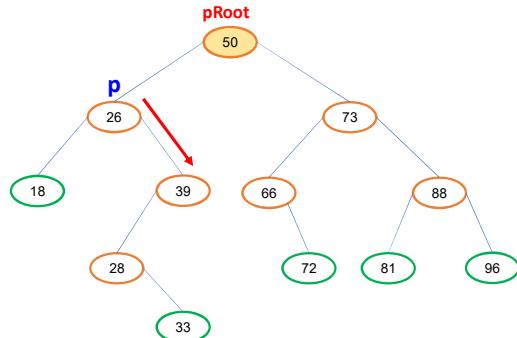
Bước 3: Lặp lại **Bước 2**. Kết thúc việc tìm kiếm khi:

- Tìm thấy giá trị
- Hoặc không còn node để so sánh



Tìm kiếm

Tìm $x = 28$



Bước 1: Bắt đầu đi từ nút gốc

Bước 2: So sánh giá trị tìm kiếm với giá trị nút hiện tại

- Nếu giá trị tìm == giá trị nút hiện tại => tìm thấy (dừng)
- Nếu giá trị tìm < giá trị nút hiện tại => di tiếp sang Trái
- Nếu giá trị tìm > giá trị nút hiện tại => di tiếp sang Phải

Bước 3: Lặp lại Bước 2. Kết thúc việc tìm kiếm khi:

- Tìm thấy giá trị
- Hoặc không còn node để so sánh

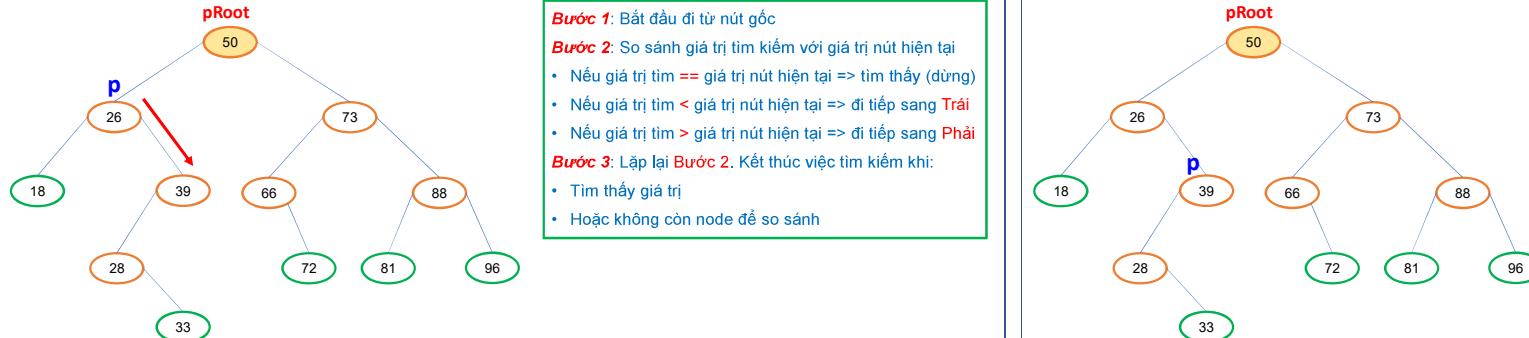
Tìm kiếm

Tìm $x = 28$



Tìm kiếm

Tìm $x = 28$



Bước 1: Bắt đầu đi từ nút gốc

Bước 2: So sánh giá trị tìm kiếm với giá trị nút hiện tại

- Nếu giá trị tìm == giá trị nút hiện tại => tìm thấy (dừng)
- Nếu giá trị tìm < giá trị nút hiện tại => di tiếp sang Trái
- Nếu giá trị tìm > giá trị nút hiện tại => di tiếp sang Phải

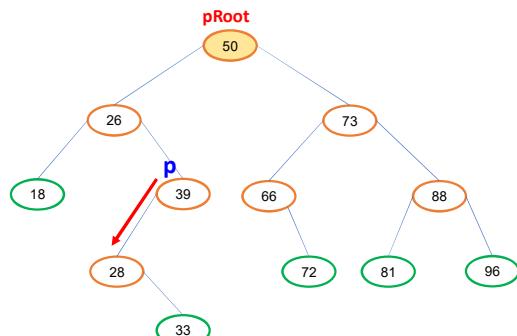
Bước 3: Lặp lại Bước 2. Kết thúc việc tìm kiếm khi:

- Tìm thấy giá trị
- Hoặc không còn node để so sánh



Tìm kiếm

Tìm $x = 28$



Bước 1: Bắt đầu đi từ nút gốc

Bước 2: So sánh giá trị tìm kiếm với giá trị nút hiện tại

- Nếu giá trị tìm == giá trị nút hiện tại => tìm thấy (dừng)
- Nếu giá trị tìm < giá trị nút hiện tại => di tiếp sang Trái
- Nếu giá trị tìm > giá trị nút hiện tại => di tiếp sang Phải

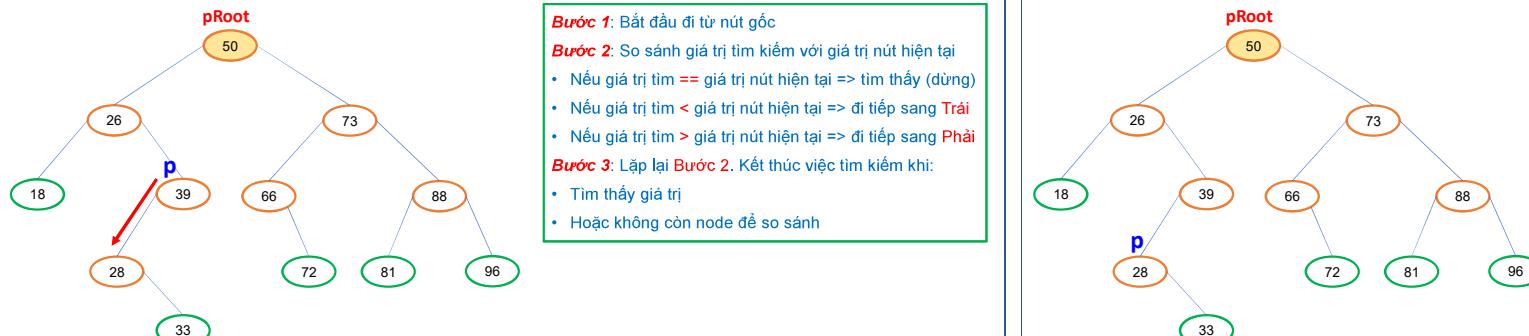
Bước 3: Lặp lại Bước 2. Kết thúc việc tìm kiếm khi:

- Tìm thấy giá trị
- Hoặc không còn node để so sánh



Tìm kiếm

Tìm $x = 28$



Bước 1: Bắt đầu đi từ nút gốc

Bước 2: So sánh giá trị tìm kiếm với giá trị nút hiện tại

- Nếu giá trị tìm == giá trị nút hiện tại => tìm thấy (dừng)
- Nếu giá trị tìm < giá trị nút hiện tại => di tiếp sang Trái
- Nếu giá trị tìm > giá trị nút hiện tại => di tiếp sang Phải

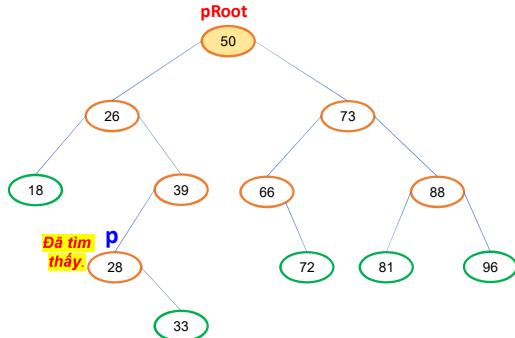
Bước 3: Lặp lại Bước 2. Kết thúc việc tìm kiếm khi:

- Tìm thấy giá trị
- Hoặc không còn node để so sánh



Tìm kiếm

Tìm $x = 28$



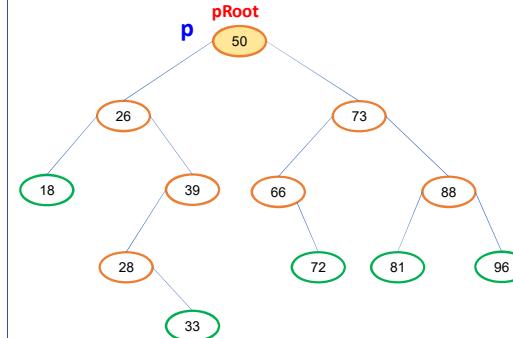
Bước 1: Bắt đầu đi từ nút gốc

Bước 2: So sánh giá trị tìm kiếm với giá trị nút hiện tại

- Nếu giá trị tìm == giá trị nút hiện tại => tìm thấy (dừng)
 - Nếu giá trị tìm < giá trị nút hiện tại => di tiếp sang Trái
 - Nếu giá trị tìm > giá trị nút hiện tại => di tiếp sang Phải
- Bước 3:** Lặp lại Bước 2. Kết thúc việc tìm kiếm khi:
- Tìm thấy giá trị
 - Hoặc không còn node để so sánh

Tìm kiếm

Tìm $x = 46$



Bước 1: Bắt đầu đi từ nút gốc

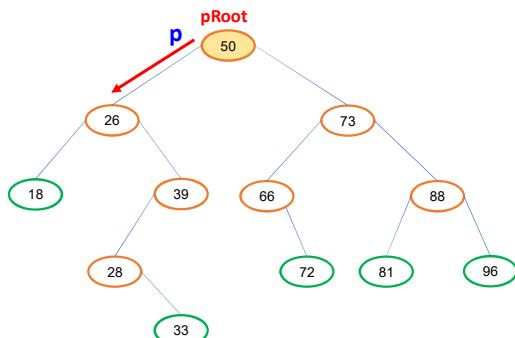
Bước 2: So sánh giá trị tìm kiếm với giá trị nút hiện tại

- Nếu giá trị tìm == giá trị nút hiện tại => tìm thấy (dừng)
 - Nếu giá trị tìm < giá trị nút hiện tại => di tiếp sang Trái
 - Nếu giá trị tìm > giá trị nút hiện tại => di tiếp sang Phải
- Bước 3:** Lặp lại Bước 2. Kết thúc việc tìm kiếm khi:
- Tìm thấy giá trị
 - Hoặc không còn node để so sánh



Tìm kiếm

Tìm $x = 46$



Bước 1: Bắt đầu đi từ nút gốc

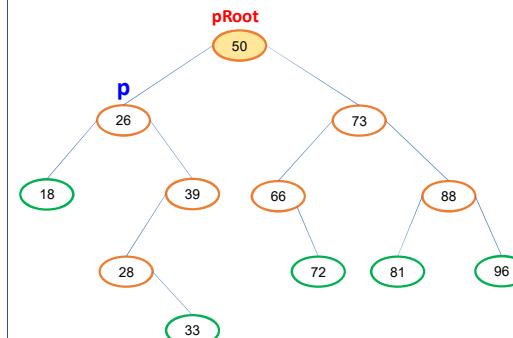
Bước 2: So sánh giá trị tìm kiếm với giá trị nút hiện tại

- Nếu giá trị tìm == giá trị nút hiện tại => tìm thấy (dừng)
 - Nếu giá trị tìm < giá trị nút hiện tại => di tiếp sang Trái
 - Nếu giá trị tìm > giá trị nút hiện tại => di tiếp sang Phải
- Bước 3:** Lặp lại Bước 2. Kết thúc việc tìm kiếm khi:
- Tìm thấy giá trị
 - Hoặc không còn node để so sánh



Tìm kiếm

Tìm $x = 46$



Bước 1: Bắt đầu đi từ nút gốc

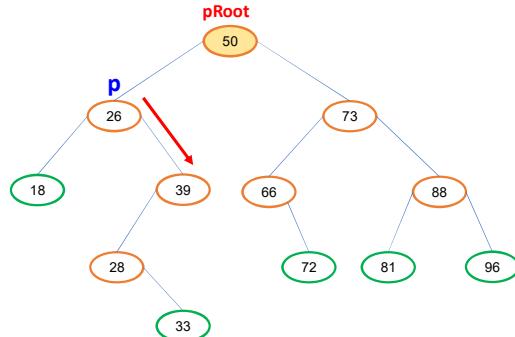
Bước 2: So sánh giá trị tìm kiếm với giá trị nút hiện tại

- Nếu giá trị tìm == giá trị nút hiện tại => tìm thấy (dừng)
 - Nếu giá trị tìm < giá trị nút hiện tại => di tiếp sang Trái
 - Nếu giá trị tìm > giá trị nút hiện tại => di tiếp sang Phải
- Bước 3:** Lặp lại Bước 2. Kết thúc việc tìm kiếm khi:
- Tìm thấy giá trị
 - Hoặc không còn node để so sánh



Tìm kiếm

Tìm $x = 46$



Bước 1: Bắt đầu đi từ nút gốc

Bước 2: So sánh giá trị tìm kiếm với giá trị nút hiện tại

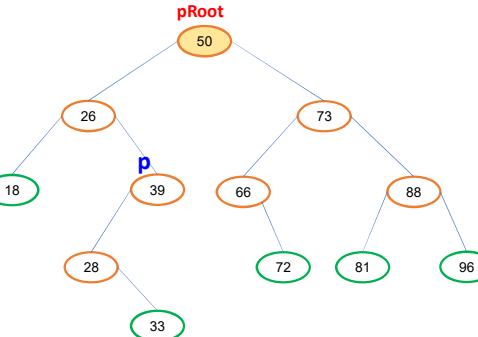
- Nếu giá trị tìm == giá trị nút hiện tại => tìm thấy (dừng)
- Nếu giá trị tìm < giá trị nút hiện tại => di tiếp sang **Trái**
- Nếu giá trị tìm > giá trị nút hiện tại => di tiếp sang **Phải**

Bước 3: Lặp lại **Bước 2**. Kết thúc việc tìm kiếm khi:

- Tìm thấy giá trị
- Hoặc không còn node để so sánh

Tìm kiếm

Tìm $x = 46$



Bước 1: Bắt đầu đi từ nút gốc

Bước 2: So sánh giá trị tìm kiếm với giá trị nút hiện tại

- Nếu giá trị tìm == giá trị nút hiện tại => tìm thấy (dừng)
- Nếu giá trị tìm < giá trị nút hiện tại => di tiếp sang **Trái**
- Nếu giá trị tìm > giá trị nút hiện tại => di tiếp sang **Phải**

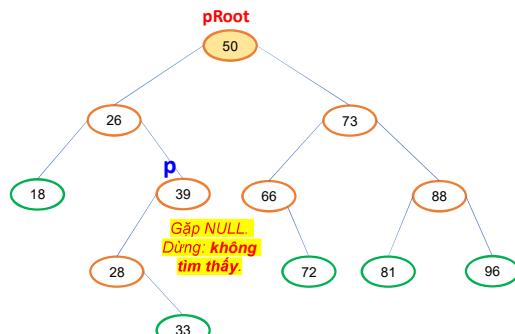
Bước 3: Lặp lại **Bước 2**. Kết thúc việc tìm kiếm khi:

- Tìm thấy giá trị
- Hoặc không còn node để so sánh



Tìm kiếm

Tìm $x = 46$



Bước 1: Bắt đầu đi từ nút gốc

Bước 2: So sánh giá trị tìm kiếm với giá trị nút hiện tại

- Nếu giá trị tìm == giá trị nút hiện tại => tìm thấy (dừng)
- Nếu giá trị tìm < giá trị nút hiện tại => di tiếp sang **Trái**
- Nếu giá trị tìm > giá trị nút hiện tại => di tiếp sang **Phải**

Bước 3: Lặp lại **Bước 2**. Kết thúc việc tìm kiếm khi:

- Tìm thấy giá trị
- Hoặc không còn node để so sánh



Tìm kiếm

Viết hàm

Phân tích:

- Input:**
 - Cây
 - Giá trị cần tìm \Rightarrow Tree tree
 \Rightarrow int value
- Output:**
 - Có hay không \Rightarrow return bool

Bước 1: Bắt đầu đi từ nút gốc

Bước 2: So sánh giá trị tìm kiếm với giá trị nút hiện tại

- Nếu giá trị tìm == giá trị nút hiện tại => tìm thấy (dừng)
- Nếu giá trị tìm < giá trị nút hiện tại => di tiếp sang **Trái**
- Nếu giá trị tìm > giá trị nút hiện tại => di tiếp sang **Phải**

Bước 3: Lặp lại **Bước 2**. Kết thúc việc tìm kiếm khi:

- Tìm thấy giá trị
- Hoặc không còn node để so sánh



Tìm kiếm

Viết hàm

```
bool tim(Tree tree, int value)
{
    Node* p = tree.pRoot;
    while(p != NULL)
    {
        if(p->data == value)
            return true;
        if(value < p->data)
            p = p->pLeft;
        else if(value > p->data)
            p = p->pRight;
    }
    return false;
}
```

Cấu trúc Dữ liệu và Giải thuật

90

Bước 1: Bắt đầu đi từ nút gốc

Bước 2: So sánh giá trị tìm kiếm với giá trị nút hiện tại

- Nếu giá trị tìm == giá trị nút hiện tại => tìm thấy (đừng)
- Nếu giá trị tìm < giá trị nút hiện tại => di tiếp sang Trái
- Nếu giá trị tìm > giá trị nút hiện tại => di tiếp sang Phải

Bước 3: Lặp lại Bước 2. Kết thúc việc tìm kiếm khi:

- Tìm thấy giá trị
- Hoặc không còn node để so sánh

Tìm kiếm

Viết hàm

```
bool tim(Tree tree, int value)
{
    Node* p = tree.pRoot;
    while(p != NULL)
    {
        if(p->data == value)
            return true;
        if(value < p->data)
            p = p->pLeft;
        else if(value > p->data)
            p = p->pRight;
    }
    return false;
}
```

Cấu trúc Dữ liệu và Giải thuật

91

Bước 1: Bắt đầu đi từ nút gốc

Bước 2: So sánh giá trị tìm kiếm với giá trị nút hiện tại

- Nếu giá trị tìm == giá trị nút hiện tại => tìm thấy (đừng)
- Nếu giá trị tìm < giá trị nút hiện tại => di tiếp sang Trái
- Nếu giá trị tìm > giá trị nút hiện tại => di tiếp sang Phải

Bước 3: Lặp lại Bước 2. Kết thúc việc tìm kiếm khi:

- Tìm thấy giá trị
- Hoặc không còn node để so sánh



Tìm kiếm

Viết hàm

```
bool tim(Tree tree, int value)
{
    Node* p = tree.pRoot;
    while(p != NULL)
    {
        if(p->data == value)
            return true;
        if(value < p->data)
            p = p->pLeft;
        else if(value > p->data)
            p = p->pRight;
    }
    return false;
}
```

Cấu trúc Dữ liệu và Giải thuật

92

Bước 1: Bắt đầu đi từ nút gốc

Bước 2: So sánh giá trị tìm kiếm với giá trị nút hiện tại

- Nếu giá trị tìm == giá trị nút hiện tại => tìm thấy (đừng)
- Nếu giá trị tìm < giá trị nút hiện tại => di tiếp sang Trái
- Nếu giá trị tìm > giá trị nút hiện tại => di tiếp sang Phải

Bước 3: Lặp lại Bước 2. Kết thúc việc tìm kiếm khi:

- Tìm thấy giá trị
- Hoặc không còn node để so sánh

Tìm kiếm

Viết hàm

```
bool tim(Tree tree, int value)
{
    Node* p = tree.pRoot;
    while(p != NULL)
    {
        if(p->data == value)
            return true;
        if(value < p->data)
            p = p->pLeft;
        else if(value > p->data)
            p = p->pRight;
    }
    return false;
}
```

Cấu trúc Dữ liệu và Giải thuật

93

Bước 1: Bắt đầu đi từ nút gốc

Bước 2: So sánh giá trị tìm kiếm với giá trị nút hiện tại

- Nếu giá trị tìm == giá trị nút hiện tại => tìm thấy (đừng)
- Nếu giá trị tìm < giá trị nút hiện tại => di tiếp sang Trái
- Nếu giá trị tìm > giá trị nút hiện tại => di tiếp sang Phải

Bước 3: Lặp lại Bước 2. Kết thúc việc tìm kiếm khi:

- Tìm thấy giá trị
- Hoặc không còn node để so sánh



Nhắc lại duyệt cây

```
void inOrder(Tree tree) {
    stack<Node*> s;
    Node* p = tree.pRoot;

    while (p != NULL || !s.empty()) {
        while (p != NULL) {
            s.push(p);
            p = p->pLeft;
        }

        p = s.top();
        s.pop();
        cout << p->data << " ";
        p = p->pRight;
    }
}
```



Nhắc lại duyệt cây

```
void traversal(Node* p)
{
    if(p != NULL)
    {
        //Xử lý node

        traversal(p->pLeft);
        traversal(p->pRight);
    }
}
```



Tính tổng

Yêu cầu: Viết hàm tính tổng các giá trị trong cây?

Phân tích:

- Input:
 - Cây => Tree tree => nút gốc
- Output:
 - Tổng các giá trị => ref

- Mở rộng thêm:
 - Tổng các node âm
 - Tổng các node lẻ
 - ...

```
void tinh(Node* p, int& tong)
{
    if(p != NULL)
    {
        tong = tong + p->data;

        tinh(p->pLeft, tong);
        tinh(p->pRight, tong);
    }
}
```



Đếm

Yêu cầu: Viết hàm đếm số lượng node có trong cây?

Phân tích:

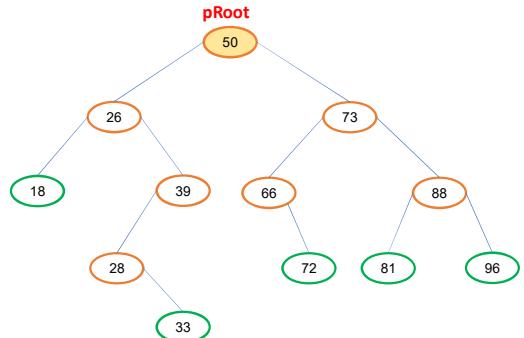
- Input:
 - Cây => Tree tree => nút gốc
- Output:
 - Số lượng các nút => ref

- Mở rộng thêm:
 - Đếm các node dương
 - Đếm các node chẵn
 - ...

```
void dem(Node* p, int& s)
{
    if(p != NULL)
    {
        s = s + p->data;

        dem(p->pLeft, tong);
        dem(p->pRight, tong);
    }
}
```

Node lá



- Điều kiện node lá: địa chỉ pLeft và pRight đều NULL.
- Áp dụng:
 - In các node lá
 - Tìm các node lá
 - Tính tổng các node lá
 - Số lượng node lá
 - Số lượng các node không phải là node lá.

Node lá

Yêu cầu: Viết hàm in các node lá trong cây?

Phân tích:

- Input:
 - Cây => Tree tree => nút gốc
- Output:
 - In các node

```

void printLeaf(Node* p)
{
    if(p != NULL)
    {
        printLeaf(p->pLeft);

        if(p->pLeft == NULL && p->pRight == NULL)
            cout << p->data << " ";

        printLeaf(p->pRight);
    }
}
  
```

```

void inOrder(Tree tree) {
    stack<Node*> s;
    Node* p = tree.pRoot;

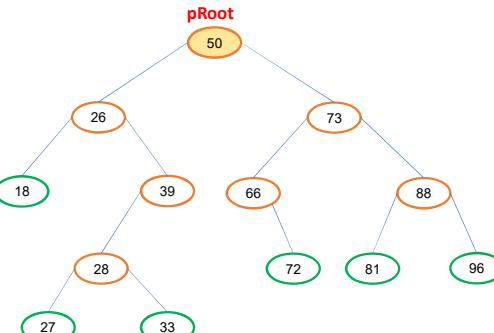
    while (p != NULL || !s.empty()) {
        while (p != NULL) {
            s.push(p);
            p = p->pLeft;
        }

        p = s.top();
        s.pop();
        cout << p->data << " ";

        p = p->pRight;
    }
}
  
```

Node có 1 nhánh con

- Điều kiện node có 1 nhánh con:
 - địa chỉ pLeft hoặc pRight bằng NULL.
- Áp dụng:
 - In các node có 1 nhánh
 - Tìm các node có 1 nhánh
 - Tính tổng các node có 1 nhánh
 - Số lượng node có 1 nhánh
 - Tính tổng các node có 2 nhánh





Chú ý

- Trong cây nhị phân tìm kiếm, các giá trị có thể trùng nhau hoặc không trùng nhau phụ thuộc vào thiết kế của người dùng.
- Bài học này, các giá trị trong cây nhị phân tìm kiếm được xử lý không trùng nhau.
- Anh/chị thiết kế cây nhị phân tìm kiếm giải quyết trường hợp cây trùng nhau.

Bài tập

Viết chương trình cài đặt cây tìm kiếm nhị phân lưu trữ các số nguyên (hoặc số thực). Sau đó, viết hàm thực hiện các yêu cầu sau:

- Tạo cấu trúc node và cây.
- Viết hàm chuyển một giá trị nguyên sang node.
- Viết hàm khởi tạo cây.
- Viết hàm chèn node chứa giá trị (dữ liệu) vào cây (không dùng đệ quy).
- Viết hàm khởi tạo giá trị tự động cho cây. Các giá trị được chọn ngẫu nhiên trong đoạn [-38; 68]. Số lượng cho một lần tạo tự động [10; 20].
- Viết hàm khởi tạo giá trị cho cây từ mảng 1 chiều.
- Viết hàm duyệt cây theo NLR, LNR, LRN.
- Viết hàm tìm kiếm giá trị trong cây.



Bài tập

Viết chương trình cài đặt cây tìm kiếm nhị phân lưu trữ các số nguyên (hoặc số thực). Sau đó, viết hàm thực hiện các yêu cầu sau (*tiếp theo*):

- Viết hàm đếm toàn bộ số node của cây.
- Viết hàm đếm số node lá của cây.
- Viết hàm in ra nhánh (NLR) của một node.
- Viết hàm đếm số node có giá trị nhỏ hơn X.
- Viết hàm tính tổng các node trong cây.
- Viết hàm tính tổng các node chẵn trong cây.
- Viết hàm tìm node có giá trị lớn nhất và nhỏ nhất trong cây (viết 1 hàm).

Thank you





Hỏi - Đáp



Cấu trúc Dữ liệu và Giải thuật

106

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

TÌM KIẾM



Nội dung

- 1. Giới thiệu
- 2. Tìm kiếm tuyến tính - Linear search
- 3. Tìm kiếm nhị phân - Binary search
- 4. Tìm kiếm tuyến tính nội suy - Interpolation search
- 5. Bài tập



Cấu trúc Dữ liệu và Giải thuật

2



Tìm kiếm tuyến tính - Linear search

Giới thiệu

- Thực hiện tìm kiếm bằng cách kiểm tra tuần tự từng phần tử trong **dãy** (mảng, danh sách...) cho đến khi tìm thấy phần tử cần tìm.
- Không hiệu quả nếu dãy lớn.
- Dãy không cần sắp xếp.
- Thời gian thực hiện phụ thuộc vào số lượng phần tử và vị trí cần tìm. Trường hợp xấu nhất thì phải tìm toàn bộ dãy.

Cấu trúc Dữ liệu và Giải thuật

3



Tìm kiếm tuyến tính

Phương pháp

value = 18

10	35	12	18	16	20	50	32
----	----	----	----	----	----	----	----

Ý tưởng:

- Duyệt qua từng phần tử, sau đó so sánh với giá trị cần tìm.

Tìm kiếm tuyến tính

Phương pháp

10	35	12	18	16	20	50	32
----	----	----	----	----	----	----	----

value = 18

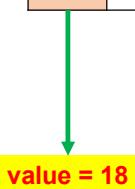


Tìm kiếm tuyến tính

Phương pháp

10	35	12	18	16	20	50	32
----	----	----	----	----	----	----	----

value = 18



Tìm kiếm tuyến tính

Phương pháp

10	35	12	18	16	20	50	32
----	----	----	----	----	----	----	----

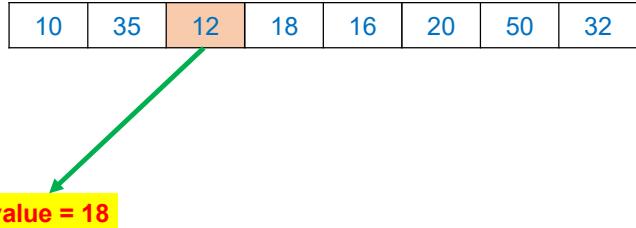
value = 18





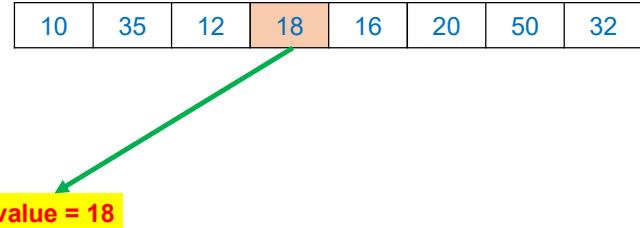
Tìm kiếm tuyến tính

Phương pháp



Tìm kiếm tuyến tính

Phương pháp



Tìm kiếm tuyến tính

Cài đặt

1. Bắt đầu từ phần tử đầu tiên.
2. So sánh phần tử hiện tại với giá trị cần tìm
 - Nếu = thì tìm thấy => Dừng.
3. Di chuyển tuần tự
 - Nếu giá trị hiện tại != giá trị cần tìm thì sẽ di chuyển sang phần tử kế tiếp.
4. Lặp lại #2.

```
bool linearSearch(int a[], int n, int value)
{
    for (int i = 0; i < n; i++)
    {
        if (a[i] == value)
        {
            return true;
        }
    }
    return false;
}
```



Tìm kiếm tuyến tính

Cài đặt

1. Bắt đầu từ phần tử đầu tiên.
2. So sánh phần tử hiện tại với giá trị cần tìm
 - Nếu = thì tìm thấy => Dừng.
3. Di chuyển tuần tự
 - Nếu giá trị hiện tại != giá trị cần tìm thì sẽ di chuyển sang phần tử kế tiếp.
4. Lặp lại #2.

```
bool linearSearch(int a[], int n, int value)
{
    int i = 0;
    while(i < n)
    {
        if(a[i] == value)
            return true;
        i++;
    }
    return false;
}
```



Tìm kiếm tuyển tính

Cài đặt

1. Bắt đầu từ phần tử đầu tiên.
 2. So sánh phần tử hiện tại với giá trị cần tìm
 - Nếu $=$ thì tìm thấy \Rightarrow Dừng.
 3. Di chuyển tuần tự
 - Nếu giá trị hiện tại \neq giá trị cần tìm thì sẽ di chuyển sang phần tử kế tiếp.
 4. Lặp lại #2.

```
bool linearSearch(int a[], int n, int value)
{
    for (int i = 0; i < n; i++)
    {
        if (a[i] == value)
        {
            return true;
        }
    }
    return false;
}
```



Tìm kiếm nhị phân - Binary search

Giới thiệu

- Tìm kiếm nhị phân được áp dụng để tìm kiếm một phần tử trong một dãy đã được **sắp xếp** (hay dãy có thứ tự).
 - Ý tưởng chính:
 - Tìm vị trí giữa.
 - Lấy giá trị tại vị trí giữa so sánh với giá trị cần tìm.
 - Hoạt động hiệu quả vì mỗi lần so sánh sẽ loại bỏ một nửa phần tử khỏi phạm vi tìm kiếm.



Tìm kiếm nhị phân

Phương pháp

0	1	2	3	4	5
10	15	18	25	27	35

Ý tưởng:

- Tìm vị trí giữa, sau đó lấy giá trị tại vị trí giữa so sánh với giá trị cần tìm.



Tìm kiếm nhị phân

Phương pháp

n = 6

0	1	2	3	4	5
10	15	18	25	27	35

Bước 1: Xác định phạm vi tìm kiếm L và R

Bước 2: Tính vị trí giữa $M = (L + R)/2$

Bước 3: So sánh giá trị giữa và giá trị cần tìm

- Nếu bằng thì tìm thấy.

- Nếu khác thì cập nhật phạm vi tìm L hoặc R

Bước 4: Cập nhật lại L hoặc R

- Nếu giá trị tìm < giá trị giữa thì cập nhật $R = M -$
- Nếu giá trị tìm > giá trị giữa thì cập nhật $L = M +$

Bước 5: Lặp, bắt đầu lại **Bước 2** nếu $L \leq R$



Tìm kiếm nhị phân

Phương pháp

$n = 6$

0	1	2	3	4	5
10	15	18	25	27	35

$L = 0$

$R = 5$

Bước 1: Xác định phạm vi tìm kiếm L và R

Bước 2: Tính vị trí giữa $M = (L + R)/2$

Bước 3: So sánh giá trị giữa và giá trị cần tìm

- Nếu **bằng** thì tìm thấy.
- Nếu **khác** thì cập nhật phạm vi tìm L hoặc R

Bước 4: Cập nhật lại L hoặc R

- Nếu giá trị tìm < giá trị giữa thì cập nhật $R = M - 1$
- Nếu giá trị tìm > giá trị giữa thì cập nhật $L = M + 1$

Bước 5: Lặp, bắt đầu lại **Bước 2** nếu $L \leq R$

Tìm kiếm nhị phân

Phương pháp

$n = 6$

0	1	2	3	4	5
10	15	18	25	27	35

$L = 0$ $M = (0+5)/2$
 = 2 $R = 5$

Bước 1: Xác định phạm vi tìm kiếm L và R

Bước 2: Tính vị trí giữa $M = (L + R)/2$

Bước 3: So sánh giá trị giữa và giá trị cần tìm

- Nếu **bằng** thì tìm thấy.
- Nếu **khác** thì cập nhật phạm vi tìm L hoặc R

Bước 4: Cập nhật lại L hoặc R

- Nếu giá trị tìm < giá trị giữa thì cập nhật $R = M - 1$
- Nếu giá trị tìm > giá trị giữa thì cập nhật $L = M + 1$

Bước 5: Lặp, bắt đầu lại **Bước 2** nếu $L \leq R$



Tìm kiếm nhị phân

Phương pháp

$n = 6$

0	1	2	3	4	5
10	15	18	25	27	35

$L = 0$ $M = (0+5)/2$
 = 2 $R = 5$

$a[M] ?? value$

Bước 1: Xác định phạm vi tìm kiếm L và R

Bước 2: Tính vị trí giữa $M = (L + R)/2$

Bước 3: So sánh giá trị giữa và giá trị cần tìm

- Nếu **bằng** thì tìm thấy.
- Nếu **khác** thì cập nhật phạm vi tìm L hoặc R

Bước 4: Cập nhật lại L hoặc R

- Nếu giá trị tìm < giá trị giữa thì cập nhật $R = M - 1$
- Nếu giá trị tìm > giá trị giữa thì cập nhật $L = M + 1$

Bước 5: Lặp, bắt đầu lại **Bước 2** nếu $L \leq R$



Tìm kiếm nhị phân

Phương pháp

$n = 6$

0	1	2	3	4	5
10	15	18	25	27	35

$L = 0$ $M = (0+5)/2$
 = 2 $R = 5$

$a[M] ?? value$

$18 < 27 \Rightarrow$ cập nhật $L = M + 1$

Bước 1: Xác định phạm vi tìm kiếm L và R

Bước 2: Tính vị trí giữa $M = (L + R)/2$

Bước 3: So sánh giá trị giữa và giá trị cần tìm

- Nếu **bằng** thì tìm thấy.
- Nếu **khác** thì cập nhật phạm vi tìm L hoặc R

Bước 4: Cập nhật lại L hoặc R

- Nếu giá trị tìm < giá trị giữa thì cập nhật $R = M - 1$
- Nếu giá trị tìm > giá trị giữa thì cập nhật $L = M + 1$

Bước 5: Lặp, bắt đầu lại **Bước 2** nếu $L \leq R$



Tìm kiếm nhị phân

Phương pháp

$n = 6$

0	1	2	3	4	5
10	15	18	25	27	35

$L = 3 \quad R = 5$

Bước 1: Xác định phạm vi tìm kiếm L và R

Bước 2: Tính vị trí giữa $M = (L + R)/2$

Bước 3: So sánh giá trị giữa và giá trị cần tìm

- Nếu bằng thì tìm thấy.
- Nếu khác thì cập nhật phạm vi tìm L hoặc R

Bước 4: Cập nhật lại L hoặc R

- Nếu giá trị tìm < giá trị giữa thì cập nhật $R = M - 1$
- Nếu giá trị tìm > giá trị giữa thì cập nhật $L = M + 1$

Bước 5: Lặp, bắt đầu lại Bước 2 nếu $L \leq R$

Tìm kiếm nhị phân

Phương pháp

$n = 6$

0	1	2	3	4	5
10	15	18	25	27	35

$L = 3 \quad R = 5$

$$M = (3+5)/2 \\ = 4$$

Bước 1: Xác định phạm vi tìm kiếm L và R

Bước 2: Tính vị trí giữa $M = (L + R)/2$

Bước 3: So sánh giá trị giữa và giá trị cần tìm

- Nếu bằng thì tìm thấy.
- Nếu khác thì cập nhật phạm vi tìm L hoặc R

Bước 4: Cập nhật lại L hoặc R

- Nếu giá trị tìm < giá trị giữa thì cập nhật $R = M - 1$
- Nếu giá trị tìm > giá trị giữa thì cập nhật $L = M + 1$

Bước 5: Lặp, bắt đầu lại Bước 2 nếu $L \leq R$



Tìm kiếm nhị phân

Phương pháp

$n = 6$

0	1	2	3	4	5
10	15	18	25	27	35

$L = 3 \quad R = 5$

$$M = (3+5)/2 \\ = 4$$

$a[M] ?? value$

$27 = 27 \Rightarrow True$

Bước 1: Xác định phạm vi tìm kiếm L và R

Bước 2: Tính vị trí giữa $M = (L + R)/2$

Bước 3: So sánh giá trị giữa và giá trị cần tìm

- Nếu bằng thì tìm thấy.
- Nếu khác thì cập nhật phạm vi tìm L hoặc R

Bước 4: Cập nhật lại L hoặc R

- Nếu giá trị tìm < giá trị giữa thì cập nhật $R = M - 1$
- Nếu giá trị tìm > giá trị giữa thì cập nhật $L = M + 1$

Bước 5: Lặp, bắt đầu lại Bước 2 nếu $L \leq R$

Tìm kiếm nhị phân

Phương pháp

$n = 6$

0	1	2	3	4	5
10	15	18	25	27	35

$L = 3 \quad R = 5$

$$M = (3+5)/2 \\ = 4$$

$a[M] ?? value$

$27 = 27 \Rightarrow True$

Bước 1: Xác định phạm vi tìm kiếm L và R

Bước 2: Tính vị trí giữa $M = (L + R)/2$

Bước 3: So sánh giá trị giữa và giá trị cần tìm

- Nếu bằng thì tìm thấy.
- Nếu khác thì cập nhật phạm vi tìm L hoặc R

Bước 4: Cập nhật lại L hoặc R

- Nếu giá trị tìm < giá trị giữa thì cập nhật $R = M - 1$
- Nếu giá trị tìm > giá trị giữa thì cập nhật $L = M + 1$

Bước 5: Lặp, bắt đầu lại Bước 2 nếu $L \leq R$

=> Đến đây dừng thuật toán vì đã tìm thấy.



Tìm kiếm nhị phân

Phương pháp

$n = 6$

0	1	2	3	4	5
10	15	18	25	27	35

$$L = 3 \quad R = 5$$

$$M = (3+5)/2 \\ = 4$$

a[M] ?? value

27 > 26

Bước 1: Xác định phạm vi tìm kiếm L và R

Bước 2: Tính vị trí giữa $M = (L + R)/2$

Bước 3: So sánh giá trị giữa và giá trị cần tìm

- Nếu bằng thì tìm thấy.
- Nếu khác thì cập nhật phạm vi tìm L hoặc R

Bước 4: Cập nhật lại L hoặc R

- Nếu giá trị tìm < giá trị giữa thì cập nhật $R = M - 1$
- Nếu giá trị tìm > giá trị giữa thì cập nhật $L = M + 1$

Bước 5: Lặp, bắt đầu lại Bước 2 nếu $L \leq R$

Cấu trúc Dữ liệu và Giải thuật

24



Tìm kiếm nhị phân

Phương pháp

$n = 6$

0	1	2	3	4	5
10	15	18	25	27	35

$$L = 3 \quad R = 5$$

$$M = (3+5)/2 \\ = 4$$

a[M] ?? value

27 > 26
cập nhật $R = M - 1$

Bước 1: Xác định phạm vi tìm kiếm L và R

Bước 2: Tính vị trí giữa $M = (L + R)/2$

Bước 3: So sánh giá trị giữa và giá trị cần tìm

- Nếu bằng thì tìm thấy.
- Nếu khác thì cập nhật phạm vi tìm L hoặc R

Bước 4: Cập nhật lại L hoặc R

- Nếu giá trị tìm < giá trị giữa thì cập nhật $R = M - 1$
- Nếu giá trị tìm > giá trị giữa thì cập nhật $L = M + 1$

Bước 5: Lặp, bắt đầu lại Bước 2 nếu $L \leq R$

Cấu trúc Dữ liệu và Giải thuật

25



Tìm kiếm nhị phân

Phương pháp

$n = 6$

0	1	2	3	4	5
10	15	18	25	27	35

$$L = 3 \quad R = 5$$

$$M = (3+5)/2 \\ = 4$$

a[M] ?? value

27 > 26
cập nhật $R = M - 1$
 $R = 3$

Bước 1: Xác định phạm vi tìm kiếm L và R

Bước 2: Tính vị trí giữa $M = (L + R)/2$

Bước 3: So sánh giá trị giữa và giá trị cần tìm

- Nếu bằng thì tìm thấy.
- Nếu khác thì cập nhật phạm vi tìm L hoặc R

Bước 4: Cập nhật lại L hoặc R

- Nếu giá trị tìm < giá trị giữa thì cập nhật $R = M - 1$
- Nếu giá trị tìm > giá trị giữa thì cập nhật $L = M + 1$

Bước 5: Lặp, bắt đầu lại Bước 2 nếu $L \leq R$

Cấu trúc Dữ liệu và Giải thuật

26



Tìm kiếm nhị phân

Phương pháp

$n = 6$

0	1	2	3	4	5
10	15	18	25	27	35

$$L = 3$$

$$R = 3$$

Bước 1: Xác định phạm vi tìm kiếm L và R

Bước 2: Tính vị trí giữa $M = (L + R)/2$

Bước 3: So sánh giá trị giữa và giá trị cần tìm

- Nếu bằng thì tìm thấy.
- Nếu khác thì cập nhật phạm vi tìm L hoặc R

Bước 4: Cập nhật lại L hoặc R

- Nếu giá trị tìm < giá trị giữa thì cập nhật $R = M - 1$
- Nếu giá trị tìm > giá trị giữa thì cập nhật $L = M + 1$

Bước 5: Lặp, bắt đầu lại Bước 2 nếu $L \leq R$

Cấu trúc Dữ liệu và Giải thuật

27



Tìm kiếm nhị phân

Phương pháp

$n = 6$

value = 26

0	1	2	3	4	5
10	15	18	25	27	35

$L = 3$

$R = 3$

$M = 3$

Bước 1: Xác định phạm vi tìm kiếm L và R

Bước 2: Tính vị trí giữa $M = (L + R)/2$

Bước 3: So sánh giá trị giữa và giá trị cần tìm

- Nếu **bằng** thì tìm thấy.
- Nếu **khác** thì cập nhật phạm vi tìm L hoặc R

Bước 4: Cập nhật lại L hoặc R

- Nếu giá trị tìm < giá trị giữa thì cập nhật $R = M - 1$
- Nếu giá trị tìm > giá trị giữa thì cập nhật $L = M + 1$

Bước 5: Lặp, bắt đầu lại **Bước 2** nếu $L \leq R$

Cấu trúc Dữ liệu và Giải thuật

28

Tìm kiếm nhị phân

Phương pháp

$n = 6$

value = 26

0	1	2	3	4	5
10	15	18	25	27	35

$L = 3$

$R = 3$

$M = 3$

a[M] ?? value

Bước 1: Xác định phạm vi tìm kiếm L và R

Bước 2: Tính vị trí giữa $M = (L + R)/2$

Bước 3: So sánh giá trị giữa và giá trị cần tìm

- Nếu **bằng** thì tìm thấy.
- Nếu **khác** thì cập nhật phạm vi tìm L hoặc R

Bước 4: Cập nhật lại L hoặc R

- Nếu giá trị tìm < giá trị giữa thì cập nhật $R = M - 1$
- Nếu giá trị tìm > giá trị giữa thì cập nhật $L = M + 1$

Bước 5: Lặp, bắt đầu lại **Bước 2** nếu $L \leq R$

Cấu trúc Dữ liệu và Giải thuật

29



Tìm kiếm nhị phân

Phương pháp

$n = 6$

value = 26

0	1	2	3	4	5
10	15	18	25	27	35

$L = 3$

$R = 3$

$M = 3$

a[M] ?? value

$25 < 26$

cập nhật $L = M + 1$

$L = 4$

Bước 1: Xác định phạm vi tìm kiếm L và R

Bước 2: Tính vị trí giữa $M = (L + R)/2$

Bước 3: So sánh giá trị giữa và giá trị cần tìm

- Nếu **bằng** thì tìm thấy.
- Nếu **khác** thì cập nhật phạm vi tìm L hoặc R

Bước 4: Cập nhật lại L hoặc R

- Nếu giá trị tìm < giá trị giữa thì cập nhật $R = M - 1$
- Nếu giá trị tìm > giá trị giữa thì cập nhật $L = M + 1$

Bước 5: Lặp, bắt đầu lại **Bước 2** nếu $L \leq R$

Cấu trúc Dữ liệu và Giải thuật

30

Tìm kiếm nhị phân

Phương pháp

$n = 6$

value = 26

0	1	2	3	4	5
10	15	18	25	27	35

$L = 3$

$R = 3$

$M = 3$

a[M] ?? value

$25 < 26$

cập nhật $L = M + 1$

$L = 4 \Rightarrow$ Dừng vì sai qui tắc $L \leq R$

Bước 1: Xác định phạm vi tìm kiếm L và R

Bước 2: Tính vị trí giữa $M = (L + R)/2$

Bước 3: So sánh giá trị giữa và giá trị cần tìm

- Nếu **bằng** thì tìm thấy.
- Nếu **khác** thì cập nhật phạm vi tìm L hoặc R

Bước 4: Cập nhật lại L hoặc R

- Nếu giá trị tìm < giá trị giữa thì cập nhật $R = M - 1$
- Nếu giá trị tìm > giá trị giữa thì cập nhật $L = M + 1$

Bước 5: Lặp, bắt đầu lại **Bước 2** nếu $L \leq R$

Cấu trúc Dữ liệu và Giải thuật

31



Tìm kiếm nhị phân

Phương pháp

$n = 6$

value = 26

0	1	2	3	4	5
10	15	18	25	27	35

L = 3

R = 3

M = 3

a[M] ?? value

25 < 26

cập nhật L = M + 1

L = 4 => Dừng vì sai qui tắc L <= R

Đây là trường hợp
KHÔNG tìm thấy giá trị trong dãy.

Tìm kiếm nhị phân

Cài đặt

```
bool binarySearch(int a[], int n, int value)
{
    int left = 0, right = n - 1;

    while(left <= right)
    {
        int m = left + (right - left) / 2;

        if (a[m] == value)
            return true;

        if (a[m] < value)
            left = m + 1;
        else if (a[m] > value)
            right = m - 1;
    }
    return false;
}
```

Bước 1: Xác định phạm vi tìm kiếm L và R

Bước 2: Tính vị trí giữa $M = (L + R)/2$

Bước 3: So sánh giá trị giữa và giá trị cần tìm

- Nếu bằng thì tìm thấy.

- Nếu khác thì cập nhật phạm vi tìm L hoặc R

- Nếu giá trị tìm < giá trị giữa thì cập nhật $R = M - 1$

- Nếu giá trị tìm > giá trị giữa thì cập nhật $L = M + 1$

Bước 5: Lặp, bắt đầu lại Bước 2 nếu $L <= R$



Tìm kiếm nhị phân

Cài đặt

```
bool binarySearch(int a[], int n, int value)
{
    int left = 0, right = n - 1;

    while (left <= right)
    {
        int m = left + (right - left) / 2;

        if (a[m] == value)
            return true;

        if (a[m] < value)
            left = m + 1;
        else if (a[m] > value)
            right = m - 1;
    }
    return false;
}
```

Bước 1: Xác định phạm vi tìm kiếm L và R

Bước 2: Tính vị trí giữa $M = (L + R)/2$

Bước 3: So sánh giá trị giữa và giá trị cần tìm

- Nếu bằng thì tìm thấy.

- Nếu khác thì cập nhật phạm vi tìm L hoặc R

Bước 4: Cập nhật lại L hoặc R

- Nếu giá trị tìm < giá trị giữa thì cập nhật $R = M - 1$

- Nếu giá trị tìm > giá trị giữa thì cập nhật $L = M + 1$

Bước 5: Lặp, bắt đầu lại Bước 2 nếu $L <= R$

Tìm kiếm nội suy - Interpolation search

Giới thiệu

- Tương tự tìm kiếm nhị phân, tìm kiếm nội suy cũng áp dụng cho dãy có thứ tự.
- Nâng cao từ ý tưởng của tìm kiếm nhị phân.
- Tính toán vị trí giữa bằng cách kết hợp thêm giá trị tìm.

$$L + ((R - L) / (a[R] - a[L])) * (value - a[L])$$

$$L + \frac{R - L}{a[R] - a[L]} * (value - a[L])$$



Tìm kiếm nội suy

Cài đặt

```
bool ....Search(int a[], int n, int value)
{
    int left = 0, right = n - 1;

    while (left <= right)
    {
        int m = left + (right - left) / 2;

        if (a[m] == value)
            return true;

        if (a[m] < value)
            left = m + 1;
        else if (a[m] > value)
            right = m - 1;
    }
    return false;
}
```

Bước 1: Xác định phạm vi tìm kiếm L và R

Bước 2: Tính vị trí giữa

$$L + ((R - L) / (a[R] - a[L])) * (value - a[L])$$

Bước 3: So sánh giá trị giữa và giá trị cần tìm

- Nếu bằng thì tìm thấy.
 - Nếu khác thì cập nhật phạm vi tìm L hoặc R
- Bước 4:** Cập nhật lại L hoặc R
- Nếu giá trị tìm < giá trị giữa thì cập nhật $R = M - 1$
 - Nếu giá trị tìm > giá trị giữa thì cập nhật $L = M + 1$
- Bước 5:** Lặp, bắt đầu lại **Bước 2** nếu $L \leq R$



Bài tập

- Viết hàm nhập giá trị tự động cho mảng, các giá trị thuộc [100; 999], số lượng phần tử thuộc [30; 50].
- Viết hàm nhập giá trị tự động **tăng dần** cho mảng, phần tử đầu tiên ≤ 130 , các phần tử kề nhau không quá 15, các giá trị thuộc [100; 999], số lượng phần tử thuộc [30; 50].
- Viết hàm nhập giá trị tự động **tăng dần** cho mảng, phần tử đầu tiên ≤ 130 , các phần tử kề nhau không quá 15, các giá trị thuộc [100; 250], số lượng phần tử thuộc [30; 50].
- Viết hàm xuất mảng.
- Viết hàm tìm kiếm một phần tử trong mảng bằng tìm kiếm tuyến tính.
- Viết hàm tìm kiếm một phần tử trong mảng bằng tìm kiếm nhị phân.
- Viết hàm tìm kiếm một phần tử trong mảng bằng tìm kiếm nội suy.
- Viết hàm so sánh số lần thực hiện tìm kiếm cùng một giá trị cho 3 thuật toán tìm kiếm trên.



Hỏi - Đáp



Sắp xếp



Nội dung

1. Giới thiệu
2. Selection Sort
3. Insertion Sort
4. Tính thời gian sắp xếp
5. Bài tập



Selection Sort

Giới thiệu

- Selection sort: Sắp xếp chọn trực tiếp.
- Dãy: mảng, danh sách...
- Thực hiện bằng cách lặp qua dãy, chọn phần tử:
 - **nhỏ nhất** (sắp xếp tăng)
 - hoặc **lớn nhất** (sắp xếp giảm)từ dãy chưa được sắp xếp. Sau đó, hoán đổi phần tử đó với phần tử đầu tiên của dãy chưa được sắp xếp.
- Quá trình lặp lại cho đến khi toàn bộ dãy được sắp xếp.



Selection Sort

Phương pháp

79	39	26	66	55	20
----	----	----	----	----	----



Selection Sort

Phương pháp

Đầu vào

79	39	26	66	55	20
----	----	----	----	----	----

Lần #1

79	39	26	66	55	20
----	----	----	----	----	----



Cấu trúc Dữ liệu và Giải thuật

5

Selection Sort

Phương pháp

Lần #1

79	39	26	66	55	20
----	----	----	----	----	----



Cấu trúc Dữ liệu và Giải thuật

6



Selection Sort

Phương pháp

Lần #1

79	39	26	66	55	20
----	----	----	----	----	----



79	39	26	66	55	20
----	----	----	----	----	----

Cấu trúc Dữ liệu và Giải thuật

7

Selection Sort

Phương pháp

Lần #1

79	39	26	66	55	20
----	----	----	----	----	----



Cấu trúc Dữ liệu và Giải thuật

8

Tiếp theo, hoán vị hai phần tử



Selection Sort

Phương pháp

79	39	26	66	55	20
Lần #1	20	39	26	66	55

Đã hoán vị hai phần tử


Selection Sort

Phương pháp

79	39	26	66	55	20
Lần #1	20	39	26	66	55



Selection Sort

Phương pháp

79	39	26	66	55	20
Lần #1	20	39	26	66	55
Lần #2	20	39	26	66	55



Selection Sort

Phương pháp

79	39	26	66	55	20
Lần #1	20	39	26	66	55
Lần #2	20	39	26	66	55



Selection Sort

Phương pháp

79	39	26	66	55	20
Lần #1	20	39	26	66	55
Lần #2	20	39	26	66	55



Selection Sort

Phương pháp

79	39	26	66	55	20
Lần #1	20	39	26	66	55
Lần #2	20	39	26	66	55



Selection Sort

Phương pháp

79	39	26	66	55	20
Lần #1	20	39	26	66	55
Lần #2	20	39	26	66	55



Tiếp theo, hoán vị hai phần tử



Selection Sort

Phương pháp

79	39	26	66	55	20
Lần #1	20	39	26	66	55
Lần #2	20	26	39	66	55



Đã hoán vị hai phần tử



Selection Sort

Phương pháp

79	39	26	66	55	20
Lần #1	20	39	26	66	55
Lần #2	20	26	39	66	55



Selection Sort

Phương pháp

79	39	26	66	55	20
Lần #1	20	39	26	66	55
Lần #2	20	26	39	66	55
Lần #3	20	26	39	66	55



Selection Sort

Phương pháp

79	39	26	66	55	20
Lần #1	20	39	26	66	55
Lần #2	20	26	39	66	55
Lần #3	20	26	39	66	55



Selection Sort

Phương pháp

79	39	26	66	55	20
Lần #1	20	39	26	66	55
Lần #2	20	26	39	66	55
Lần #3	20	26	39	66	55





Selection Sort

Phương pháp

79	39	26	66	55	20
----	----	----	----	----	----

Lần #1	20	39	26	66	55	79
--------	----	----	----	----	----	----

Lần #2	20	26	39	66	55	79
--------	----	----	----	----	----	----

Lần #3	20	26	39	66	55	79
--------	----	----	----	----	----	----



Không có phần tử nào nhỏ hơn 39



Selection Sort

Phương pháp

79	39	26	66	55	20
----	----	----	----	----	----

Lần #1	20	39	26	66	55	79
--------	----	----	----	----	----	----

Lần #2	20	26	39	66	55	79
--------	----	----	----	----	----	----

Lần #3	20	26	39	66	55	79
--------	----	----	----	----	----	----



KHÔNG tìm thấy giá trị nhỏ hơn.



Selection Sort

Phương pháp

79	39	26	66	55	20
----	----	----	----	----	----

Lần #1	20	39	26	66	55	79
--------	----	----	----	----	----	----

Lần #2	20	26	39	66	55	79
--------	----	----	----	----	----	----

Lần #3	20	26	39	66	55	79
--------	----	----	----	----	----	----



Selection Sort

Phương pháp

79	39	26	66	55	20
----	----	----	----	----	----

Lần #1	20	39	26	66	55	79
--------	----	----	----	----	----	----

Lần #2	20	26	39	66	55	79
--------	----	----	----	----	----	----

Lần #3	20	26	39	66	55	79
--------	----	----	----	----	----	----

Lần #4	20	26	39	66	55	79
--------	----	----	----	----	----	----



Selection Sort

Phương pháp

79	39	26	66	55	20
Lần #1	20	39	26	66	55
Lần #2	20	26	39	66	55
Lần #3	20	26	39	66	55
Lần #4	20	26	39	66	55



Selection Sort

Phương pháp

79	39	26	66	55	20
Lần #1	20	39	26	66	55
Lần #2	20	26	39	66	55
Lần #3	20	26	39	66	55
Lần #4	20	26	39	66	55



Selection Sort

Phương pháp

79	39	26	66	55	20
Lần #1	20	39	26	66	55
Lần #2	20	26	39	66	55
Lần #3	20	26	39	66	55
Lần #4	20	26	39	66	55



Selection Sort

Phương pháp

79	39	26	66	55	20
Lần #1	20	39	26	66	55
Lần #2	20	26	39	66	55
Lần #3	20	26	39	66	55
Lần #4	20	26	39	66	55



Tiếp theo, hoán vị hai phần tử



Selection Sort

Phương pháp

79	39	26	66	55	20
----	----	----	----	----	----

Lần #1	20	39	26	66	55	79
--------	----	----	----	----	----	----

Lần #2	20	26	39	66	55	79
--------	----	----	----	----	----	----

Lần #3	20	26	39	66	55	79
--------	----	----	----	----	----	----

Lần #4	20	26	39	55	66	79
--------	----	----	----	----	----	----



Đã hoán vị hai phần tử



Selection Sort

Phương pháp

79	39	26	66	55	20
----	----	----	----	----	----

Lần #1	20	39	26	66	55	79
--------	----	----	----	----	----	----

Lần #2	20	26	39	66	55	79
--------	----	----	----	----	----	----

Lần #3	20	26	39	66	55	79
--------	----	----	----	----	----	----

Lần #4	20	26	39	55	66	79
--------	----	----	----	----	----	----



Selection Sort

Phương pháp

79	39	26	66	55	20
----	----	----	----	----	----

Lần #1	20	39	26	66	55	79
--------	----	----	----	----	----	----

Lần #2	20	26	39	66	55	79
--------	----	----	----	----	----	----

Lần #3	20	26	39	66	55	79
--------	----	----	----	----	----	----

Lần #4	20	26	39	55	66	79
--------	----	----	----	----	----	----

Lần #5	20	26	39	55	66	79
--------	----	----	----	----	----	----



Selection Sort

Phương pháp

79	39	26	66	55	20
----	----	----	----	----	----

Lần #1	20	39	26	66	55	79
--------	----	----	----	----	----	----

Lần #2	20	26	39	66	55	79
--------	----	----	----	----	----	----

Lần #3	20	26	39	66	55	79
--------	----	----	----	----	----	----

Lần #4	20	26	39	55	66	79
--------	----	----	----	----	----	----

Lần #5	20	26	39	55	66	79
--------	----	----	----	----	----	----



Selection Sort

Phương pháp

	79	39	26	66	55	20
Lần #1	20	39	26	66	55	79
Lần #2	20	26	39	66	55	79
Lần #3	20	26	39	66	55	79
Lần #4	20	26	39	55	66	79
Lần #5	20	26	39	55	66	79



Cấu trúc Dữ liệu và Giải thuật

33



Selection Sort

Phương pháp

	79	39	26	66	55	20
Lần #1	20	39	26	66	55	79
Lần #2	20	26	39	66	55	79
Lần #3	20	26	39	66	55	79
Lần #4	20	26	39	55	66	79
Lần #5	20	26	39	55	66	79

Cấu trúc Dữ liệu và Giải thuật

34



Selection Sort

Thuật toán

79	39	26	66	55	20
20	39	26	66	55	79
20	26	39	66	55	79
20	26	39	66	55	79
20	26	39	55	66	79
20	26	39	55	66	79
20	26	39	55	66	79

1. Bắt đầu: Đánh dấu toàn bộ là **dãy chưa sắp xếp** (màu trắng)
2. Tìm phần tử nhỏ nhất trong dãy chưa sắp xếp
3. Hoán vị phần tử nhỏ nhất và phần tử đầu tiên dãy chưa sắp xếp
4. Sau khi hoán vị xong thì đánh dấu phần tử đó thuộc **dãy đã sắp xếp** (màu vàng cam)
5. Lặp lại từ bước 2 cho dãy chưa sắp xếp



Selection Sort

Thuật toán

i	0	1	2	3	4	5
	79	39	26	66	55	20
i = 0	20	39	26	66	55	79
i = 1	20	26	39	66	55	79
i = 2	20	26	39	66	55	79
i = 3	20	26	39	55	66	79
i = 4	20	26	39	55	66	79
	20	26	39	55	66	79

```

hàm selectionSort(mảng A, n)
min: vị trí có giá trị nhỏ nhất
for i = 0 ---> n - 1
    min = i
    for j = i+1 ---> n
        if A[j] < A[min]
            min = j
    if min khác i
        hoán vị A[i] và A[min]
    
```

Cấu trúc Dữ liệu và Giải thuật

35

Cấu trúc Dữ liệu và Giải thuật

36



Selection Sort

Cài đặt

```
void selectionSort(int a[], int n)
{
    int m;
    for (int i = 0; i < n-1; i++)
    {
        m = i;
        for (int j = i+1; j < n; j++)
        {
            if (a[j] < a[m])
                m = j;
        }
        if(m != i)
            swap(a[i], a[m]);
    }
}
```

Cấu trúc Dữ liệu và Giải thuật

37



Insertion Sort

Giới thiệu

- Insertion sort: Sắp xếp chèn trực tiếp.
- Dãy: mảng, danh sách...
- Thực hiện bằng cách
 - Bắt đầu từ phần tử thứ hai.
 - Sau đó, chèn phần tử vào vị trí thích hợp.
 - Lặp lại cho phần tử tiếp theo để hết dãy.

Cấu trúc Dữ liệu và Giải thuật

38



Insertion Sort

Phương pháp

Đầu vào:

79	39	26	66	55	20
----	----	----	----	----	----

Cấu trúc Dữ liệu và Giải thuật

39



Insertion Sort

Phương pháp

79	39	26	66	55	20
Lần #1					
79	39	26	66	55	20

Cấu trúc Dữ liệu và Giải thuật

40



Insertion Sort

Phương pháp

Lần #1	79	39	26	66	55	20
	79	39	26	66	55	20

Cấu trúc Dữ liệu và Giải thuật

41



Insertion Sort

Phương pháp

Lần #1	79	39	26	66	55	20
	79	39	26	66	55	20



Cấu trúc Dữ liệu và Giải thuật

42



Insertion Sort

Phương pháp

Lần #1	79	39	26	66	55	20
	79		26	66	55	20
39						

Cấu trúc Dữ liệu và Giải thuật

43



Insertion Sort

Phương pháp

Lần #1	79	39	26	66	55	20
	79	26	66	55	20	
39						

Cấu trúc Dữ liệu và Giải thuật

44



Insertion Sort

Phương pháp

79	39	26	66	55	20	
Lần #1	39	79	26	66	55	20



Insertion Sort

Phương pháp

79	39	26	66	55	20	
Lần #1	39	79	26	66	55	20
Lần #2	39	79	26	66	55	20



Insertion Sort

Phương pháp

79	39	26	66	55	20	
Lần #1	39	79	26	66	55	20
Lần #2	39	79	26	66	55	20



Insertion Sort

Phương pháp

79	39	26	66	55	20	
Lần #1	39	79	26	66	55	20
Lần #2	39	79	26	66	55	20





Insertion Sort

Phương pháp

79	39	26	66	55	20
----	----	----	----	----	----

Lần #1	39	79	26	66	55	20
--------	----	----	----	----	----	----

Lần #2	39	79		66	55	20
--------	----	----	--	----	----	----

26

Insertion Sort

Phương pháp

79	39	26	66	55	20
----	----	----	----	----	----

Lần #1	39	79	26	66	55	20
--------	----	----	----	----	----	----

Lần #2	39		79	66	55	20
--------	----	--	----	----	----	----

26



Insertion Sort

Phương pháp

79	39	26	66	55	20
----	----	----	----	----	----

Lần #1	39	79	26	66	55	20
--------	----	----	----	----	----	----

Lần #2		39	79	66	55	20
--------	--	----	----	----	----	----

26



Insertion Sort

Phương pháp

79	39	26	66	55	20
----	----	----	----	----	----

Lần #1	39	79	26	66	55	20
--------	----	----	----	----	----	----

Lần #2	26	39	79	66	55	20
--------	----	----	----	----	----	----



Insertion Sort

Phương pháp

79	39	26	66	55	20	
Lần #1	39	79	26	66	55	20
Lần #2	26	39	79	66	55	20
Lần #3	26	39	79	66	55	20

Cấu trúc Dữ liệu và Giải thuật

53



Insertion Sort

Phương pháp

79	39	26	66	55	20	
Lần #1	39	79	26	66	55	20
Lần #2	26	39	79	66	55	20
Lần #3	26	39	79	66	55	20

Cấu trúc Dữ liệu và Giải thuật

54



Insertion Sort

Phương pháp

79	39	26	66	55	20	
Lần #1	39	79	26	66	55	20
Lần #2	26	39	79	66	55	20
Lần #3	26	39	79	66	55	20



Cấu trúc Dữ liệu và Giải thuật

55



Insertion Sort

Phương pháp

79	39	26	66	55	20	
Lần #1	39	79	26	66	55	20
Lần #2	26	39	79	66	55	20
Lần #3	26	39	79	66	55	20

66

Cấu trúc Dữ liệu và Giải thuật

56



Insertion Sort

Phương pháp

79	39	26	66	55	20	
Lần #1	39	79	26	66	55	20
Lần #2	26	39	79	66	55	20
Lần #3	26	39		79	55	20

66

Insertion Sort

Phương pháp

79	39	26	66	55	20	
Lần #1	39	79	26	66	55	20
Lần #2	26	39	79	66	55	20
Lần #3	26	39	66	79	55	20



Insertion Sort

Phương pháp

79	39	26	66	55	20	
Lần #1	39	79	26	66	55	20
Lần #2	26	39	79	66	55	20
Lần #3	26	39	66	79	55	20
Lần #4	26	39	66	79	55	20



Insertion Sort

Phương pháp

79	39	26	66	55	20	
Lần #1	39	79	26	66	55	20
Lần #2	26	39	79	66	55	20
Lần #3	26	39	66	79	55	20
Lần #4	26	39	66	79	55	20



Insertion Sort

Phương pháp

79	39	26	66	55	20	
Lần #1	39	79	26	66	55	20
Lần #2	26	39	79	66	55	20
Lần #3	26	39	66	79	55	20
Lần #4	26	39	66	79	55	20



Insertion Sort

Phương pháp

79	39	26	66	55	20	
Lần #1	39	79	26	66	55	20
Lần #2	26	39	79	66	55	20
Lần #3	26	39	66	79	55	20
Lần #4	26	39	66	79		20

55



Insertion Sort

Phương pháp

79	39	26	66	55	20	
Lần #1	39	79	26	66	55	20
Lần #2	26	39	79	66	55	20
Lần #3	26	39	66	79	55	20
Lần #4	26	39	66		79	20



Insertion Sort

Phương pháp

79	39	26	66	55	20	
Lần #1	39	79	26	66	55	20
Lần #2	26	39	79	66	55	20
Lần #3	26	39	66	79	55	20
Lần #4	26	39		66	79	20

55



Insertion Sort

Phương pháp

79	39	26	66	55	20	
Lần #1	39	79	26	66	55	20
Lần #2	26	39	79	66	55	20
Lần #3	26	39	66	79	55	20
Lần #4	26	39	55	66	79	20



Insertion Sort

Phương pháp

79	39	26	66	55	20	
Lần #1	39	79	26	66	55	20
Lần #2	26	39	79	66	55	20
Lần #3	26	39	66	79	55	20
Lần #4	26	39	55	66	79	20
Lần #5	26	39	55	66	79	20



Insertion Sort

Phương pháp

79	39	26	66	55	20	
Lần #1	39	79	26	66	55	20
Lần #2	26	39	79	66	55	20
Lần #3	26	39	66	79	55	20
Lần #4	26	39	55	66	79	20
Lần #5	26	39	55	66	79	20



Insertion Sort

Phương pháp

79	39	26	66	55	20	
Lần #1	39	79	26	66	55	20
Lần #2	26	39	79	66	55	20
Lần #3	26	39	66	79	55	20
Lần #4	26	39	55	66	79	20
Lần #5	26	39	55	66	79	



Insertion Sort

Phương pháp

79	39	26	66	55	20	
Lần #1	39	79	26	66	55	20
Lần #2	26	39	79	66	55	20
Lần #3	26	39	66	79	55	20
Lần #4	26	39	55	66	79	20
Lần #5	26	39	55	66	79	20

20



Insertion Sort

Phương pháp

79	39	26	66	55	20	
Lần #1	39	79	26	66	55	20
Lần #2	26	39	79	66	55	20
Lần #3	26	39	66	79	55	20
Lần #4	26	39	55	66	79	20
Lần #5	26	39	55		66	79

20



Insertion Sort

Phương pháp

79	39	26	66	55	20	
Lần #1	39	79	26	66	55	20
Lần #2	26	39	79	66	55	20
Lần #3	26	39	66	79	55	20
Lần #4	26	39	55	66	79	20
Lần #5	26	39		55	66	79

20



Insertion Sort

Phương pháp

79	39	26	66	55	20	
Lần #1	39	79	26	66	55	20
Lần #2	26	39	79	66	55	20
Lần #3	26	39	66	79	55	20
Lần #4	26	39	55	66	79	20
Lần #5	26		39	55	66	79

20



Insertion Sort

Phương pháp

79	39	26	66	55	20	
Lần #1	39	79	26	66	55	20
Lần #2	26	39	79	66	55	20
Lần #3	26	39	66	79	55	20
Lần #4	26	39	55	66	79	20
Lần #5	20	26	39	55	66	79

20



Insertion Sort

Phương pháp

79	39	26	66	55	20	
Lần #1	39	79	26	66	55	20
Lần #2	26	39	79	66	55	20
Lần #3	26	39	66	79	55	20
Lần #4	26	39	55	66	79	20
Lần #5	20	26	39	55	66	79



Insertion Sort

Phương pháp

79	39	26	66	55	20	
Lần #1	39	79	26	66	55	20
Lần #2	26	39	79	66	55	20
Lần #3	26	39	66	79	55	20
Lần #4	26	39	55	66	79	20
Lần #5	20	26	39	55	66	79
Kết quả:	20	26	39	55	66	79



Insertion Sort

Thuật toán

79	39	26	66	55	20
20	39	26	66	55	79
20	26	39	66	55	79
20	26	39	66	55	79
20	26	39	55	66	79
20	26	39	55	66	79
20	26	39	55	66	79

1. Quy định phần tử đầu tiên của dãy thuộc dãy đã sắp xếp.
2. Chọn phần tử tiếp theo: Phần tử này sẽ thuộc dãy chưa sắp xếp.
3. So sánh phần được chọn với các phần tử trong dãy đã sắp xếp để tìm vị trí thích hợp chèn vào:
 - o Dịch chuyển tất cả các phần tử lớn hơn phần tử hiện tại trong dãy đã sắp xếp sang phải 1 vị trí.
4. Chèn phần tử vào vị trí thích hợp.
5. Lặp lại bước 2 cho mỗi phần tử tiếp theo cho đến khi toàn bộ dãy được sắp xếp.



Insertion Sort

Cài đặt

```
void insertionSort(int a[], int n)
{
    for (int i = 1; i < n; i++)
    {
        int item = a[i];
        int j;
        for (j = i-1; j >= 0; j--)
        {
            if (a[j] < item)
                break;

            a[j+1] = a[j];
        }
        a[j+1] = item;
    }
}
```

- Quy định phần tử đầu tiên của dãy thuộc dãy đã sắp xếp.
- Chọn phần tử tiếp theo: Phần tử này sẽ thuộc dãy chưa sắp xếp.
- So sánh phần được chọn với các phần tử trong dãy đã sắp xếp để tìm vị trí thích hợp chèn vào:
 - Dịch chuyển tất cả các phần tử lớn hơn phần tử hiện tại trong dãy đã sắp xếp sang phải 1 vị trí.
- Chèn phần tử vào vị trí thích hợp.
- Lặp lại bước 2 cho mỗi phần tử tiếp theo cho đến khi toàn bộ dãy được sắp xếp.

Tính thời gian sắp xếp

Phương pháp

- Để lấy thời gian CPU của một tiến trình dùng hàm `clock()`.
- Khai báo thư viện `#include <time.h>` để dùng.

```
clock_t start, end;
double cpu_time_used;

start = clock();

/* Do the work. */

end = clock();
cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
```

- Hằng số `CLOCKS_PER_SEC`: số lần tích tắc mỗi giây.



Tính thời gian sắp xếp

Áp dụng

```
double sort_time(int a[], int n)
{
    clock_t start, end;
    double cpu_time_used;

    start = clock();

    selectionSort(a, n);

    end = clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;

    return cpu_time_used;
}
```



Bài tập

Khởi tạo mảng arrayA với kích thước 1e6 phần tử kiểu số thực. Viết hàm thực hiện hàm theo các yêu cầu sau:

- Viết hàm khởi tạo tự động 100000 phần tử kiểu số thực (lấy 3 số sau dấu thập phân) cho mảng.
- Viết hàm xuất các giá trị trong mảng.
- Viết hàm sắp xếp mảng tăng/giảm dần bằng thuật toán chọn trực tiếp.
- Viết hàm tính thời gian thực hiện sắp xếp mảng tăng/giảm dần bằng thuật toán chọn trực tiếp.
- Viết hàm sắp xếp mảng tăng/giảm dần bằng thuật toán chèn trực tiếp.
- Viết hàm tính thời gian thực hiện sắp xếp mảng tăng/giảm dần bằng thuật toán chèn trực tiếp.



Bài tập

(Tiếp theo...)

- Viết hàm so sánh thời gian thực hiện sắp xếp mảng tăng/giảm dần của hai thuật toán chọn trực tiếp và chèn trực tiếp trên cùng một bộ dữ liệu mảng. Sau đó, áp dụng minh họa thực nghiệm cho 15 bộ dữ liệu được tạo tự động ngẫu nhiên (hướng dẫn chi tiết trên lớp).



Hỏi - Đáp

