Tìm kiếm nhị phân – Binary Search

Thuật toán

Thuật toán tìm kiếm nhị phân hoạt động trên các mảng được sắp xếp. Thuật toán bắt đầu bằng việc so sánh một phần tử đứng chính giữa mảng với giá trị cần tìm. Nếu bằng nhau, vị trí của nó trong mảng sẽ được trả về. Nếu giá trị cần tìm nhỏ hơn phần tử này, quá trình tìm kiếm tiếp tục ở nửa nhỏ hơn của mảng. Nếu giá trị cần tìm lớn hơn phần tử ở giữa, quá trình tìm kiếm tiếp tục ở nửa lớn hơn của mảng. Bằng cách này, ở mỗi phép lặp thuật toán ta có thể loại bỏ nửa mảng giá trị cần tìm chắn chắn không xuất hiện.

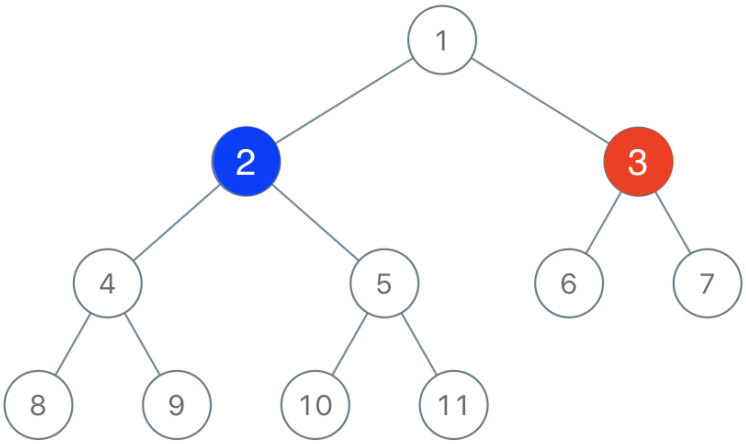
Quy trình

Cho 1 mảng A có n phần tử với giá trị hoặc bản ghi $A_0,A_1,A_2,\ldots,A_{n-1}$ đã được sắp xếp sao cho $A_0 \leq A_1 \leq A_2 \leq \ldots \leq A_{n-1}$ và giá trị cần tìm T, chương trình con sau đây sử dụng tìm kiếm nhị phân để tìm chỉ số T trong A.

- Gán L với giá trị 0 và R với giá trị n-1.
- Nếu L > R, tìm kiếm kết thúc (Không thành công)
- Gán m (vị trí của phần tử đứng giữa) với giá trị floor của $\frac{L+R}{2}$, tức là số nguyên lớn nhất nhỏ hơn hoặc bằng $\frac{L+R}{2}$
- Nếu A_m < T, gán L với m + 1 và quay lui lại bước 2.
- Nếu A_m > T, gán R với m - 1 và quay lui lại bước 2.
- Khi A_m = T , quá trình tìm kiếm nhị phân hoàn tất, trả về m

Ta có cây nhị phân được vẽ như sau:

Ví dụ: Cho A[] = {8,4,9,2,10,5,11,1,6,3,7}



Thuật toán tìm kiếm tuần tự

Code:

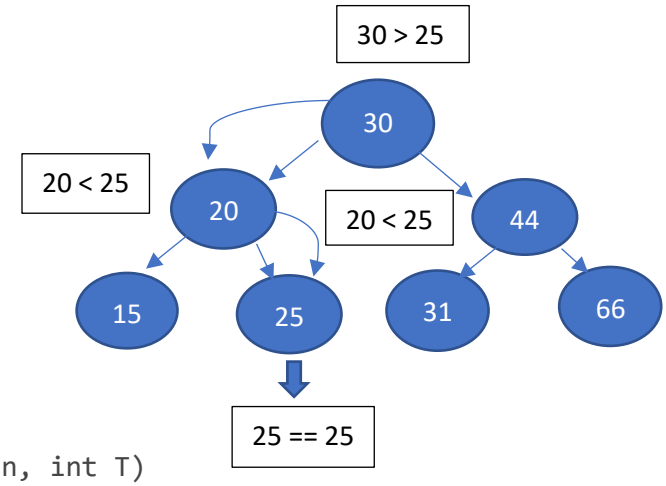
```
int n; // Số phần tử của mảng
int L = 0; // Khởi gán giá trị bên trái mảng
int R = n - 1; // Khởi gán giá trị phải của mảng

int binary_search(int A[], int n, int T)
{
	while(L <= R)
	{
		int m = (L + R) / 2;
		if(A[m] < T)
			L = m + 1;
		else if(A[m] > T)
			R = m - 1;
		else
			return m;
	}
	return -1;
}
```

Ví dụ 1: Tìm vị trí của số T = 25 trong mảng cho trước:
A[] = {15, 20, 25, 30, 31, 44, 66}.

Input: T = 25

Output: 2



Code:

```
#include <iostream>
#include <conio.h>

using namespace std;

int n = 7;
int L = 0;
int R = n - 1;

int binary_search(int A[], int n, int T)
{
	while(L <= R)
	{
		int m = (L + R) / 2;
		if(A[m] < T)
			L = m + 1;
		else if(A[m] > T)
			R = m - 1;
		else
			return m;
	}
```

```
        return -1;
    }

    int main()
    {
        int A[] = {15, 20, 25, 30, 31, 44, 66};
        int T = 25;
        cout << binary_search(A,n,T); // trả về vị trí của số T là 2

        return 0;
    }
```

Giải thích:

[0]	[1]	[2]	[3]	[4]	[5]	[6]
15	20	25	30	31	44	66
↓						↓
L = 0						R = n-1

Bước 1: Bắt đầu khởi gán

L = 0;

L = n-1;

Bước 2: middle = $\frac{L+R}{2}$ Tìm kiếm vị trí số chính giữa

middle = 6 / 2 = 3 (Lấy phần nguyên)

Bước 3:

A[3] = 30

Nếu A[m] > T -> 30 > 25 -> L = m – 1 = 3 – 1 = 2

Quay lại B2 -> middle = 3 / 2 = 1 (Lấy phần nguyên)

Nếu A[m] < T -> 20 < 25 -> R = m + 1 = 1 + 1 = 2

Quay lại B2 -> middle = 4 / 2 = 2 (Lấy phần nguyên)

➔ A[m] == T -> 25 == 25 -> m = 2

Vậy vị trí số 25 là VT m = 2;

Ví dụ 2: Tìm vị trí của số T = 25 trong mảng cho trước:
A[] = {1, 2, 3, 4, 5, 6}.

Input: 6

Output: 5

Code:

```
    #include <iostream>
    #include <conio.h>

    using namespace std;

    int n = 6;
    int L = 0;
    int R = n - 1;

    int binary_search(int A[], int n, int T)
```

```
{
	while(L <= R)
	{
		int m = (L + R) / 2;
		if(A[m] < T)
			L = m + 1;
		else if(A[m] > T)
			R = m - 1;
		else
			return m;
	}
	return -1;
}

int main()
{
	int A[] = {1,2,3,4,5,6};
	int T = 6;
	cout << binary_search(A,n,T);
	return 0;
}
```

Giải thích:

[0]	[1]	[2]	[3]	[4]	[5]
1	2	3	4	5	6
↓					↓
L = 0					R = n-1

Bước 1: Bắt đầu khởi gán

L = 0

R = n-1

Bước 2: middle = $\frac{L+R}{2}$ Tìm kiếm vị trí số chính giữa

middle = 5 / 2 = 2 (Lấy phần nguyên)

A[2] = 3

Bước 3:

Nếu A[m] < T -> 3 < 6 -> L = m + 1 = 2 + 1 = 3

Quay lại B2 -> middle = 8 / 2 = 4 (Lấy phần nguyên)

Nếu A[m] < T -> 5 < 6 -> L = m + 1 = 4 + 1 = 5

Quay lại B2 -> middle = 10 / 2 = 5 (Lấy phần nguyên)

➔ A[m] == T -> 6 == 6 -> m = 5

Vậy vị trí số 6 là VT m = 5;

Ví dụ 3:Sử dụng mảng của Ví dụ 2

Input: 10

Output: -1

Code:

```
#include <bits/stdc++.h>
#include <iostream>
#include <conio.h>
```

```
using namespace std;

int n = 6;
int L = 0;
int R = n - 1;

int binary_search(int A[], int n, int T)
{
	while(L <= R)
	{
		int m = (L + R) / 2;
		if(A[m] < T)
			L = m + 1;
		else if(A[m] > T)
			R = m - 1;
		else
			return m;
	}
	return -1;
}

int main()
{
	int A[] = {1,2,3,4,5,6};
	int T = 10;
	cout << binary_search(A,n,T);
	return 0;
}
```

Giải thích:

[0]	[1]	[2]	[3]	[4]	[5]
1	2	3	4	5	6
↓					↓
L = 0					R = n-1

Bước 1: Bắt đầu khởi gán

L = 0

R = n-1

Bước 2: middle = $\frac{L+R}{2}$ Tìm kiếm vị trí số chính giữa

middle = 5 / 2 = 2 (Lấy phần nguyên)

A[2] = 3

Bước 3:

Nếu A[m] < T -> 3 < 10 -> L = m + 1 = 2 + 1 = 3

Quay lại B2 -> middle = 8 / 2 = 4 (Lấy phần nguyên)

Nếu A[m] < T -> 4 < 10 -> L = m + 1 = 4 + 1 = 5

Quay lại B2 -> middle = 10 / 2 = 5 (Lấy phần nguyên)

Nếu A[m] < T -> 5 < 10 -> L = m + 1 = 5 + 1 = 6

➔ L = 6 > R = 5 -> return -1

Ví dụ 4: Sử dụng mảng của Ví dụ 2

Input: 0

Output: -1

Code:

```
#include <bits/stdc++.h>
#include <iostream>
#include <conio.h>

using namespace std;

int n = 6;
int L = 0;
int R = n - 1;

int binary_search(int A[], int n, int T)
{
	while(L <= R)
	{
		int m = (L + R) / 2;
		if(A[m] < T)
			L = m + 1;
		else if(A[m] > T)
			R = m - 1;
		else
			return m;
	}
	return -1;
}

int main()
{
	int A[] = {1,2,3,4,5,6};
	int T = 0;
	cout << binary_search(A,n,T);
	return 0;
}
```

Giải thích:

[0]	[1]	[2]	[3]	[4]	[5]
1	2	3	4	5	6
↓					↓
L = 0					R = n-1

Bước 1: Bắt đầu khởi gán

L = 0

R = n-1

Bước 2: middle = $\frac{L+R}{2}$ Tìm kiếm vị trí số chính giữa

middle = 5 / 2 = 2 (Lấy phần nguyên)

A[2] = 3

Bước 3:

Nếu A[m] > T -> 3 > 0 -> R = m - 1 = 2 - 1 = 1

Quay lại B2 -> middle = 1 / 2 = 0 (Lấy phần nguyên)

Nếu A[m] > T -> 1 > 0 -> R = m - 1 = 1 - 1 = 0

➔ L == R = 0 -> return -1