THUẬT TOÁN TÌM KIẾM

NỘI DUNG

/01 Tìm kiếm tuyến tính (Linear search)

/02 Tìm kiếm nhị phân (Binary search)

/03 Vị trí đầu tiên trong mảng tăng dần

/04 Vị trí cuối cùng trong mảng tăng dần

/05 Vị trí đầu tiên lớn hơn hoặc bằng X trong mảng tăng dần

/06 Vị trí cuối cùng lớn hơn hoặc bằng X trong mảng tăng dần

/07 LOWER_BOUND

/08 UPPER_BOUND

1.Tìm kiếm tuyến tính (Linear Search):

Ý tưởng: Duyệt tuần tự các phần tử trong mảng và so sánh giá trị cần tìm kiếm với từng phần tử trong mảng.

Các bài toán như tìm kiếm vị trí đầu tiên, cuối cùng, đếm số lần xuất hiện của phần tử trong mảng đều là biến đổi của thuật toán tìm kiếm tuyến tính.

Code

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

Độ phức tạp: O(N)

2.Tìm kiếm nhị phân (Binary Search):

Ý tưởng: Tìm kiếm trong đoạn từ [left, right] của mảng, ở mỗi bước thuật toán tìm vị trí mid ở giữa đoạn left, right. Nếu phần tử cần tìm kiếm bằng phần tử ở vị trí mid thì kết luận là tìm thấy, nếu không ta có thể giảm 1 nửa đoạn tìm kiếm xuống và tiếp tục tìm kiếm ở bên trái hay bên phải của mid.

Đây là một thuật toán cực kì hiệu quả và quan trọng, khi học lập trình bạn cần nắm rõ thuật toán này.

Điều kiện áp dụng: Mảng đã được sắp xếp.

Code

```
bool binarySearch(int a[], int n, int x){
    int left = 0, right = n - 1;
    while(left <= right){
        int mid = (left + right) / 2;
        if(a[mid] == x){
            return true;
        }
        else if(a[mid] < x){
            // Tìm kiếm ở bên phải
            left = mid + 1;
        }
        else{
            //Tìm kiếm ở bên trái
            right = mid - 1;
        }
    }
    return false;
}
```

Độ phức tạp: O(logN)

3. Vị trí đầu tiên trong mảng tăng dần:

Bài toán: Tìm vị trí đầu tiên của phần tử X trong mảng đã được sắp xếp

```
int firstPos(int a[], int n, int x){
    int res = -1, left = 0, right = n - 1;
    while(left <= right){
        int mid = (left + right) / 2;
        if(a[mid] == x){
            res = mid; // cập nhật
            //Tìm thêm đáp án tốt hơn
            right = mid - 1;
        }
        else if(a[mid] < x){
            left = mid + 1;
        }
        else{
            right = mid - 1;
        }
    }
    return res;
}
```

Độ phức tạp: O(logN)

4. Vị trí cuối cùng trong mảng tăng dần:

Bài toán: Tìm vị trí cuối cùng của phần tử X trong mảng đã được sắp xếp

```
int lastPos(int a[], int n, int x){
    int res = -1, left = 0, right = n - 1;
    while(left <= right){
        int mid = (left + right) / 2;
        if(a[mid] == x){
            res = mid; // cập nhật
            //Tìm thêm đáp án tốt hơn
            left = mid + 1;
        }
        else if(a[mid] < x){
            left = mid + 1;
        }
        else{
            right = mid - 1;
        }
    }
    return res;
}
```

Độ phức tạp: O(logN)

5. Vị trí đầu tiên lớn hơn hoặc bằng X trong mảng tăng dần:

Bài toán: Tìm vị trí đầu tiên của phần tử lớn hơn hoặc bằng X trong mảng đã được sắp xếp

```
int lower(int a[], int n, int x){
    int res = -1;
    int left = 0, right = n - 1;
    while(left <= right){
        int mid = (left + right) / 2;
        if(a[mid] >= x){
            res = mid; // cập nhật
            //Tìm thêm đáp án tốt hơn
            right = mid - 1;
        }
        else{
            left = mid + 1;
        }
    }
    return res;
}
```

Độ phức tạp: O(logN)

6. Vị trí cuối cùng nhỏ hơn hoặc bằng X trong mảng tăng dần:

Bài toán: Tìm vị trí cuối cùng của phần tử nhỏ hơn hoặc bằng X trong mảng đã được sắp xếp

```
int upper(int a[], int n, int x){
    int res = -1;
    int left = 0, right = n - 1;
    while(left <= right){
        int mid = (left + right) / 2;
        if(a[mid] <= x){
            res = mid; // cập nhật
            //Tìm thêm đáp án tốt hơn
            left = mid + 1;
        }
        else{
            right = mid - 1;
        }
    }
    return res;
}
```

Độ phức tạp: O(logN)

7. LOWER_BOUND:

CÚ PHÁP

```
lower_bound(first_iter, last_iter, X);
```

SỬ DỤNG LOWER_BOUND

Lower_bound là một hàm tương tự như mục 5 ở trên, nó có thể áp dụng cho mảng, vector, set, map.

Điều kiện áp dụng: Mảng, vector đã được sắp xếp tăng dần

Tương tự như hàm sort, lower_bound cũng trả về iterator chứ không trả về giá trị.

Nó trả về vị trí đầu tiên của phần tử lớn hơn hoặc bằng X, nếu trong mảng, vector bạn tìm kiếm không có phần tử lớn hơn hoặc bằng X (tất cả đều nhỏ hơn X) thì lower_bound trả về last_iter

Ví dụ lower_bound với mảng:

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    int a[7] = {1, 2, 3, 3, 3, 4, 6};
    auto it = lower_bound(a, a + 7, 3);
    cout << *it << endl;
    cout << (it - a) << endl;
}
```

OUTPUT: 3
2

Giải thích: Phần tử đầu tiên lớn hơn hoặc bằng 3 trong mảng là 3 ở chỉ số 2

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    int a[7] = {1, 2, 3, 3, 3, 4, 6};
    auto it = lower_bound(a, a + 7, 8);
    if(it == a + 7){
        cout << "NOT FOUND";
    }
    else{
        cout << *it << endl;
    }
    cout << (it - a) << endl;
}
```

OUTPUT: NOT FOUND
7

Ví dụ lower_bound với vector:

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    vector<int> a = {1, 2, 3, 3, 3, 4, 6};
    auto it = lower_bound(a.begin(), a.end(), 3);
    if(it == a.end()){
        cout << "NOT FOUND\n";
    }
    else{
        cout << *it << endl;
    }
    cout << (it - a.begin()) << endl;
}
```

OUTPUT: 3
2

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    vector<int> a = {1, 2, 3, 3, 3, 4, 6};
    auto it = lower_bound(a.begin(), a.end(), 8);
    if(it == a.end()){
        cout << "NOT FOUND\n";
    }
    else{
        cout << *it << endl;
    }
    cout << (it - a.begin()) << endl;
}
```

OUTPUT: NOT FOUND
7

Dự đoán Output của các chương trình sau

Quiz 1

```
#include <bits/stdc++.h>
using namespace std;

int main(){
    vector<int> a = {1, 1, 3, 3, 3, 4, 6};
    auto it = lower_bound(a.begin(), a.end(), 2);
    --it;
    cout << *it << endl;
}
```

Quiz 2

```
#include <bits/stdc++.h>
using namespace std;

int main(){
    vector<int> a = {1, 1, 3, 3, 3, 4, 6};
    auto it = lower_bound(a.begin(), a.end(), 7);
    --it;
    cout << *it << endl;
}
```

Dự đoán Output của các chương trình sau

Quiz 3

```
#include <bits/stdc++.h>
using namespace std;

int main(){
    vector<int> a = {1, 1, 3, 3, 3, 4, 6};
    auto it = lower_bound(a.begin(), a.end(), 7);
    it -= 2;
    cout << *it << endl;
}
```

Quiz 4

```
#include <bits/stdc++.h>
using namespace std;

int main(){
    vector<int> a = {1, 1, 3, 3, 3, 4, 6};
    auto it = lower_bound(a.begin(), a.end(), 0);
    it--;
    cout << *it << endl;
}
```

8. UPPER_BOUND:

CÚ PHÁP

```
upper_bound(first_iter, last_iter, X);
```

SỬ DỤNG UPPER_BOUND

Điều kiện áp dụng: Mảng, vector đã được sắp xếp tăng dần

Upper_bound trả về vị trí đầu tiên của phần tử lớn X, nếu trong mảng, vector bạn tìm kiếm không có phần tử lớn hơn hoặc bằng X (tất cả đều nhỏ hơn hoặc bằng X) thì upper_bound trả về last_iter.

Ví dụ upper_bound với mảng:

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    int a[7] = {1, 2, 3, 3, 3, 4, 6};
    auto it = upper_bound(a, a + 7, 3);
    cout << *it << endl;
    cout << (it - a) << endl;
}
```

OUTPUT: 4
5

Giải thích: Phần tử đầu tiên lớn hơn 3 trong mảng là 4 ở chỉ số 5

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    int a[7] = {1, 2, 3, 3, 3, 4, 6};
    auto it = upper_bound(a, a + 7, 8);
    if(it == a + 7){
        cout << "NOT FOUND";
    }
    else{
        cout << *it << endl;
    }
    cout << (it - a) << endl;
}
```

OUTPUT: NOT FOUND
7

Ví dụ upper_bound với vector:

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    vector<int> a = {1, 2, 3, 3, 3, 4, 6};
    auto it = upper_bound(a.begin(), a.end(), 3);
    if(it == a.end()){
        cout << "NOT FOUND\n";
    }
    else{
        cout << *it << endl;
    }
    cout << (it - a.begin()) << endl;
}
```

OUTPUT: 4
5

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    vector<int> a = {1, 2, 3, 3, 3, 4, 6};
    auto it = upper_bound(a.begin(), a.end(), 6);
    if(it == a.end()){
        cout << "NOT FOUND\n";
    }
    else{
        cout << *it << endl;
    }
    cout << (it - a.begin()) << endl;
}
```

OUTPUT: NOT FOUND
7

Dự đoán Output của các chương trình sau

Quiz 1

```
#include <bits/stdc++.h>
using namespace std;

int main(){
    vector<int> a = {1, 1, 3, 3, 3, 4, 6};
    auto it = upper_bound(a.begin(), a.end(), 2);
    --it;
    cout << *it << endl;
}
```

Quiz 2

```
#include <bits/stdc++.h>
using namespace std;

int main(){
    vector<int> a = {1, 1, 3, 3, 3, 4, 6};
    auto it = upper_bound(a.begin(), a.end(), 6);
    --it;
    cout << *it << endl;
}
```

Dự đoán Output của các chương trình sau

Quiz 3

```
#include <bits/stdc++.h>
using namespace std;

int main(){
   vector<int> a = {1, 1, 3, 3, 3, 4, 6};
   auto it = upper_bound(a.begin(), a.end(), 6);
   it -= 2;
   cout << *it << endl;
}
```

Quiz 4

```
#include <bits/stdc++.h>
using namespace std;

int main(){
   vector<int> a = {1, 1, 3, 3, 3, 4, 6};
   auto it = upper_bound(a.begin(), a.end(), 0);
   it--;
   cout << *it << endl;
}
```