Các thuật toán tìm kiếm

Linear Search

Ý tưởng

Duyệt toàn bộ danh sách A để xác định a_i trả về i nếu tồn tại a_i.

Đầu vào – Đầu ra

- **Input** : Danh sách A có n phần tử, giá trị khóa x cần tìm.
- **Output** : Chỉ số i của phần tử a_i trong A có giá trị khóa là x.

Độ phức tạp thuật toán

- **Best case** : a_0 chứa khóa x - số lần lặp lại là 1 - độ phức tạp hằng số $O(1)$.
- **Worst case** : A không chứa phần tử có khóa x - số lần lặp là n - độ phức tạp tuyến tính $O(n)$.
- **Average case** : Độ phức tạp tuyến tính $O(n)$.

Lưu ý

Linear Search sẽ duyệt toàn bộ không gian tìm kiếm nên các đối tượng tìm kiếm không cần được sắp xếp, có thể nói là dữ liệu không cần được tổ chức.

Giải thuật mẫu

```
int linearSearch(int *a,int n,int x)
{
    for(int i = 0;i < n;i++){

        if(a[i] == x){

            return i;
        }
    }
    return -1;
}
```

Không gian tìm kiếm

Là đối tượng mà thuật toán tìm kiếm thực hiện lên, thường là mảng, đoạn giá trị nào đó, danh sách liên kết, cây,...

Điều kiện tìm kiếm

Là tiêu chuẩn tìm kiếm trình bày dưới dạng phát biểu không hình thức và lập trình viên cần hình thức hóa nó thành một biểu thức logic.

Binary Search

Ý tưởng

Chọn a_m ở giữa A để so sánh với khóa x. A được chia thành hai phần trước và sau a_m. Chỉ số bắt đầu và kết thúc của A là L và R.

Nếu x = a_m tức là đã tìm thấy và dừng thuật toán.

Xét thứ tự x, a_m:

- Nếu thứ tự này là *P* (tức là bé hơn hoặc lớn hơn tùy thuộc vào cách sắp xếp của dữ liệu cho trước), thì tìm x trong đoạn trước a_m([L,R]) với R = M – 1 (Tức là đi từ L đến M).
- Ngược lại thì tìm x trong đoạn sau a_m([L,R]) với L = M + 1 (Đi từ M đến R)

Đầu vào – Đầu ra

- **Input** : Danh sách A có n phần tử đã sắp xếp theo thứ tự *P* (tăng dần hoặc giảm dần), giá trị khóa x cần tìm.
- **Output** : Chỉ số i của phần tử a_i trong A có giá trị khóa là x. Không tìm thấy trả về i = -1.

Độ phức tạp thuật toán

- **Best case** : Phần tử cần tìm nằm ở vị trí (L + R)/2 – Số lần lặp là 1 – Độ phức tạp hằng số $O(1)$.
- **Worst case** : Số lần tìm là số lần chia đôi đến khi dãy tìm kiếm còn 1 phần tử - Lặp khoảng log2(n) + 1 – Độ phức tạp logarithm $O(log(n))$.
- **Average case** : Độ phức tạp là $O(log(n))$.

Lưu ý

Điều kiện tìm kiếm là không gian tìm kiếm phải được sắp xếp theo một thứ tự nào đó và ta sẽ dựa theo thứ tự này để tìm kiếm.

Giải thuật mẫu

```
int binarySearch(int *a,int n,int x)
{
    int L = 0,R = n-1;
    while(L <= R)
    {
        int M = (L + R) / 2;
        if(x == a[M])

            return M;
        if(x > a[M]){
```

```
            R = M - 1;
        }
        else{
            L = M + 1;
        }
    }
    return -1;
}
```

Interpolation Search

Ý tưởng

Thay vì xác định điểm M = (L+R)/2 thì xác định M như sau:

$$M = L + \frac{R - L.(x - AL)}{AR - A[L]}$$

Các bước còn lại như Binary Search.

Đầu vào – Đầu ra

Input : Danh sách A có n phần tử đã có thứ tự *P* (tăng dần hoặc giảm dần), giá trị x cần tìm.

Output : Chỉ số i của phần tử a_i trong A có giá trị khóa là x. Không tìm thấy trả về i= -1.

Độ phức tạp thuật toán

- **Best case** : Tương tự Binary Search khi mà $a[M] = x$ thì số lần lặp là 1, độ phức tạp sẽ là hằng số $O(1)$.
- **Worst case** : Trong trường hợp giá trị số của khóa tăng theo cấp số nhân thì độ phức tạp là $O(n)$.
- **Average case** : Nếu như dữ liệu được phân bố liên tục thì độ phức tạp sẽ là $log_2(log_2(n))$.

Lưu ý

Thuật toán Interpolation Search áp dụng trong trường hợp không gian tìm kiếm đã được tổ chức theo một thứ tự (tăng dần hoặc giảm dần) và khóa cần tìm được phân bố liên tục trong không gian tìm kiếm (rải đều).

Giải thuật mẫu

```
int interpolationSearch(int *a,int n,int x)
{
    int L = 0, R = n-1;
    while(L <= R) {
        int M = L + (R - L)*((x - a[L])/(a[R] - a[L]));
        if(x == a[M]){
            return M;
        }
        if(x > a[M]){
```

```
            R = M - 1;
        }
        else{
            L = M + 1;
        }
    }
    return -1;
}
```