N1

Tìm kiếm và sắp xếp

Đình Khôi - Nguyên Khang - Tuấn Kiệt

Nhu cầu

Thao tác tìm kiếm thường được thực hiện nhất để khai thác thông tin.
Việc xây dựng các giải thuật cho phép tìm kiếm nhanh sẽ có ý nghĩa rất lớn.

Bài toán tìm kiếm

là bài toán tìm một phần tử nằm trong một tập hợp rất nhiều phần tử dựa vào một yêu cầu nào đó.

Các giải thuật tìm kiếm

Có nhiều thuật toán tìm kiếm khác nhau được sử dụng để giải quyết bài toán này, tùy thuộc vào tính chất và yêu cầu của dữ liệu.

Tìm kiếm tuyến tính

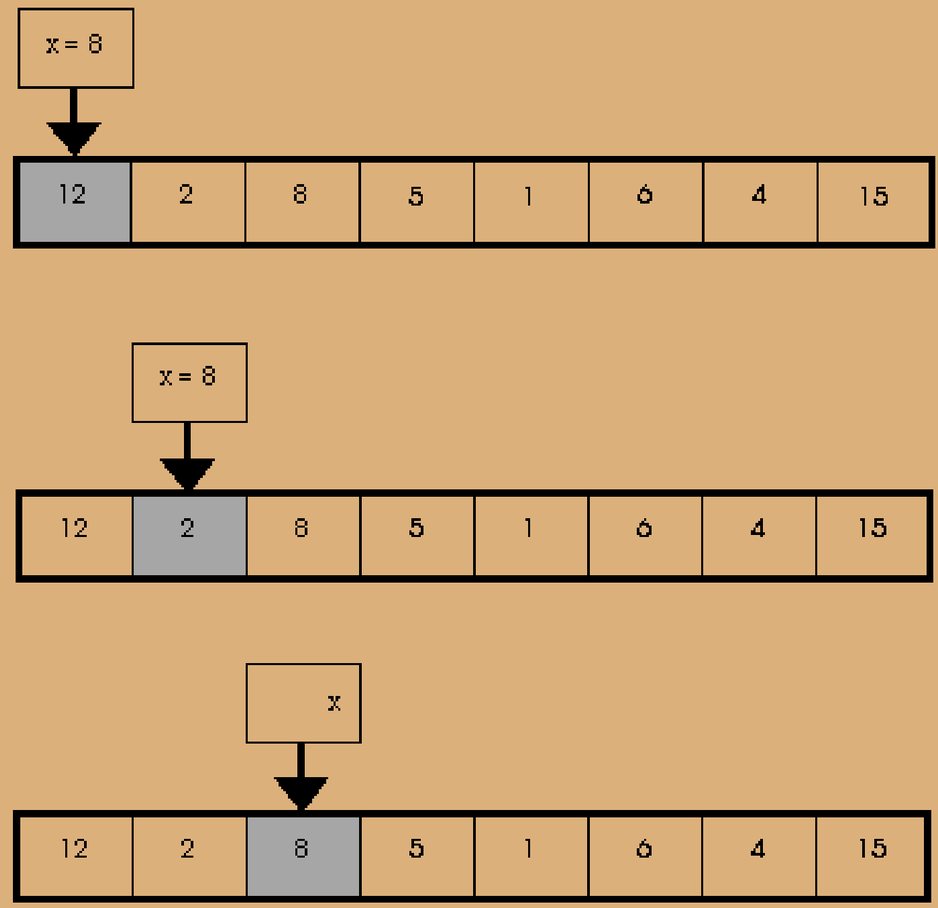
Tuyến tính cải tiến

Tìm kiếm nhị phân

Tìm kiếm tuyến tính

So sánh x lần lượt với phần tử thứ nhất, thứ hai, ... của mảng a cho đến khi gặp được phần tử có khóa cần tìm, hoặc đã tìm hết mảng mà không thấy x.

```
1  int Linear_search(float a[], int n, float& x){
2      std::cout << "- Nhap gia tri x muon tim kiem trong mang: ";
3      std::cin >> x;
4      for (int i = 0; i < n; ++i){
5          if (a[i] == x){
6              return i;
7          }
8      }
9      return -1;
10 }
```



Tìm kiếm tuyến tính cải tiến

Thay vì kiểm tra 2 điều kiện (i<n) và (a[i]==x), ta chỉ cần kiểm tra điều kiện chính (a[i]!=x) và đặt thêm một phần tử có giá trị x vào cuối mảng, như vậy bảo đảm luôn tìm thấy x trong mảng.

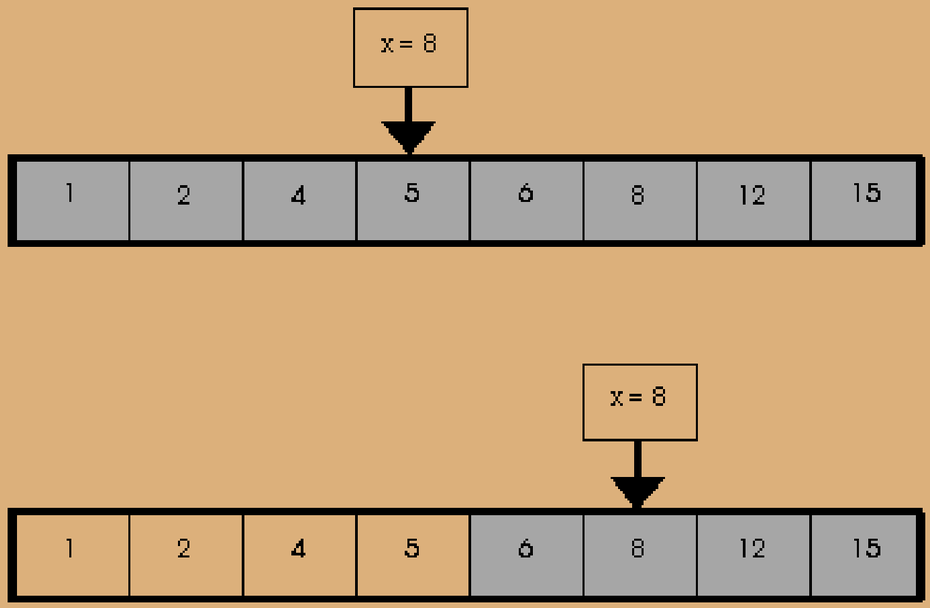
```
int Linear_search_but_better(float a[], int n, float& x){
    std::cout << "- Nhap gia tri x muon tim kiem trong mang: ";
    std::cin >> x;
    int i = 0;
    a[n] = x;
    while (a[i] != x){
        ++i;
    }
    if (i == n){
        return -1;
    }
    return i;
}
```

Tìm kiếm nhị phân

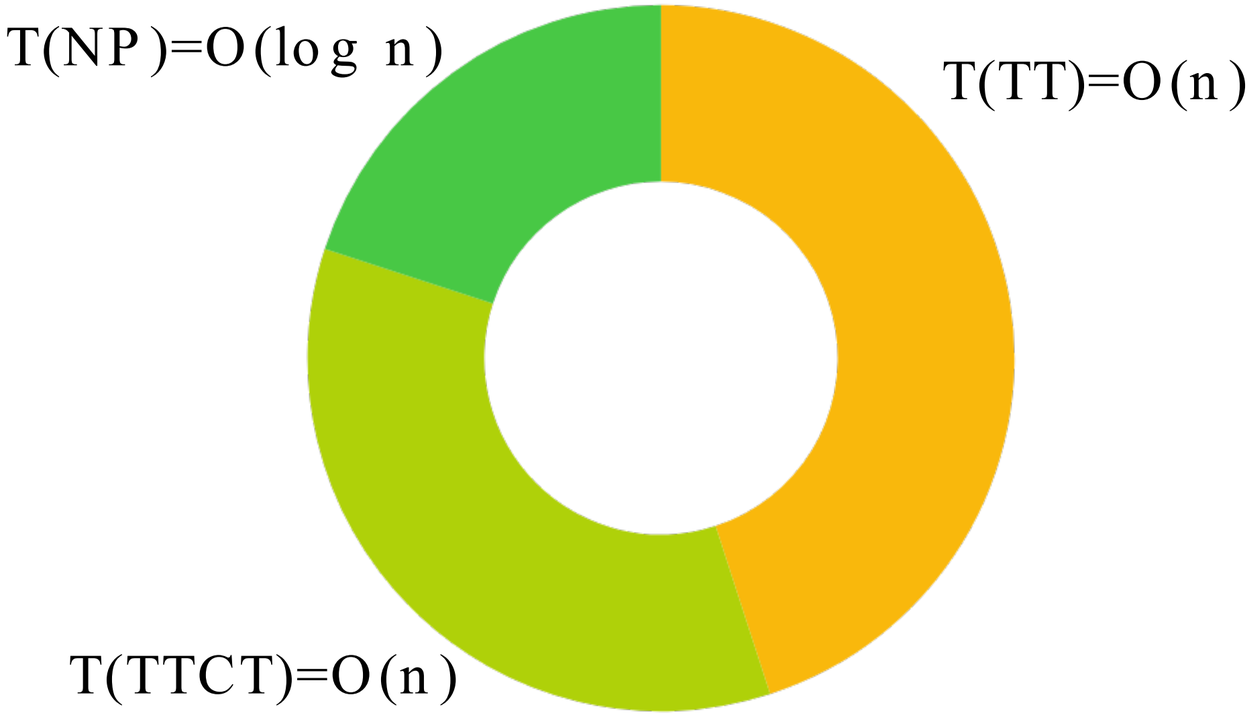
Kiểm tra x với phần tử giữa mảng(a[mid]), nếu x>a[mid] thì kiểm tra trên mảng con bên phải a[mid] hoặc x<a[i] thì kiểm tra lại trên mảng con bên trái a[mid]. Thực hiện cho đến khi tìm thấy x hoặc hết mảng.

```
int Binary_search(float a[], int n, float& x){
    std::cout << "- Nhap gia tri x muon tim kiem trong mang: ";
    std::cin >> x;
    int left = 0, right = n - 1, middle;
    while (left <= right) {
        middle = (right + left) / 2;
        if (a[middle] == x)
            return middle;
        if (a[middle] < x)
            left = middle + 1;

        else
            right = middle - 1;
    }
    return -1;
}
```



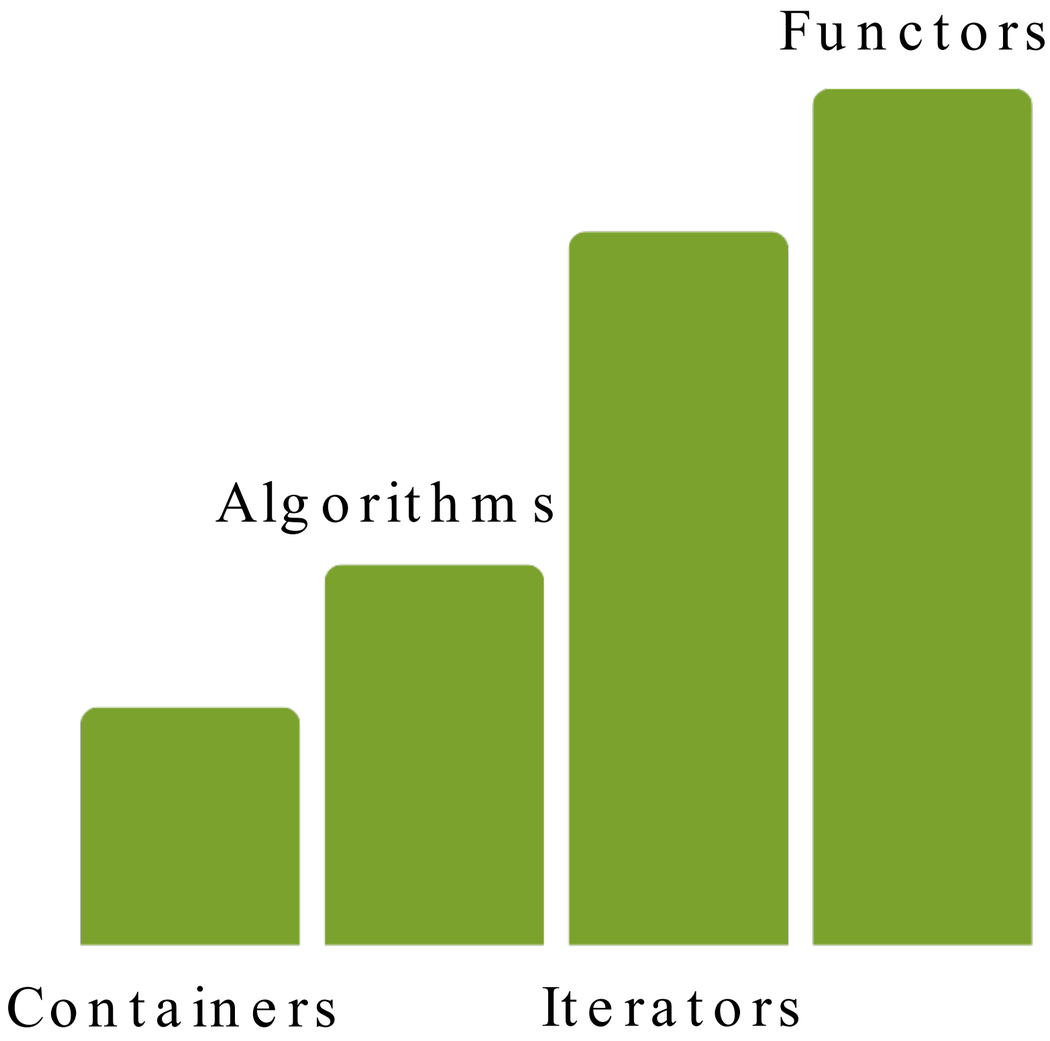
Đánh giá độ phức tạp



Giải thuật tuyến tính là phương pháp tổng quát nhất.
Tuyến tính cải tiến giảm được phép so sánh của điều kiện dừng vòng lặp.
Tìm kiếm nhị phân rút được rất nhiều thời gian thực hiện.

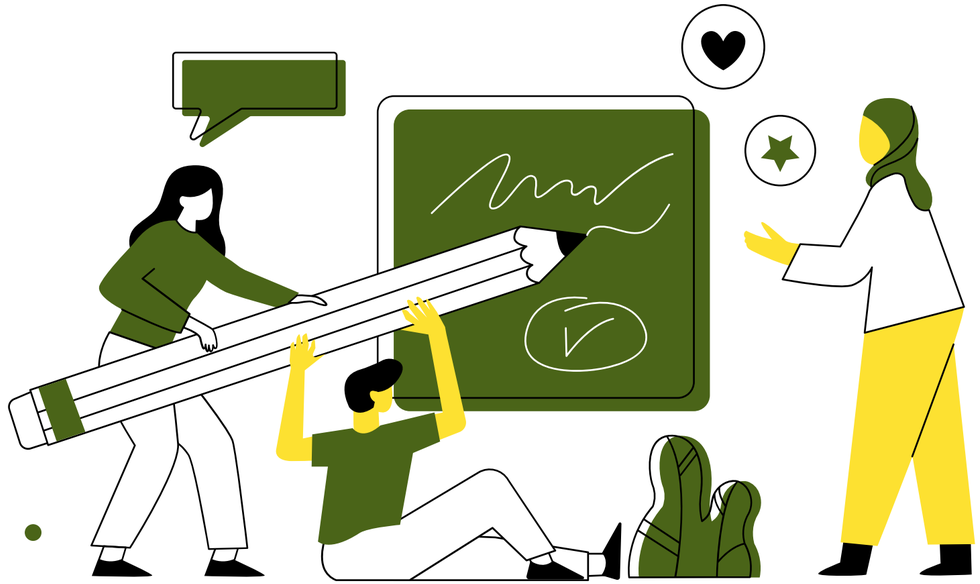
Standard Template Library

Để lập trình viên nhanh chóng khai thác các cấu trúc dữ liệu cũng như các thuật toán trở nên dễ dàng hơn, tổng quát hơn.



Lớp vector

Nhìn như array nhưng thật ra không phải như vậy



Thuận lợi của vector

Với vector trong C++, các developer không còn phải thực hiện nhiều thao tác thừa thải, lặp đi lặp lại khi phải làm việc với mảng ở trong C++.

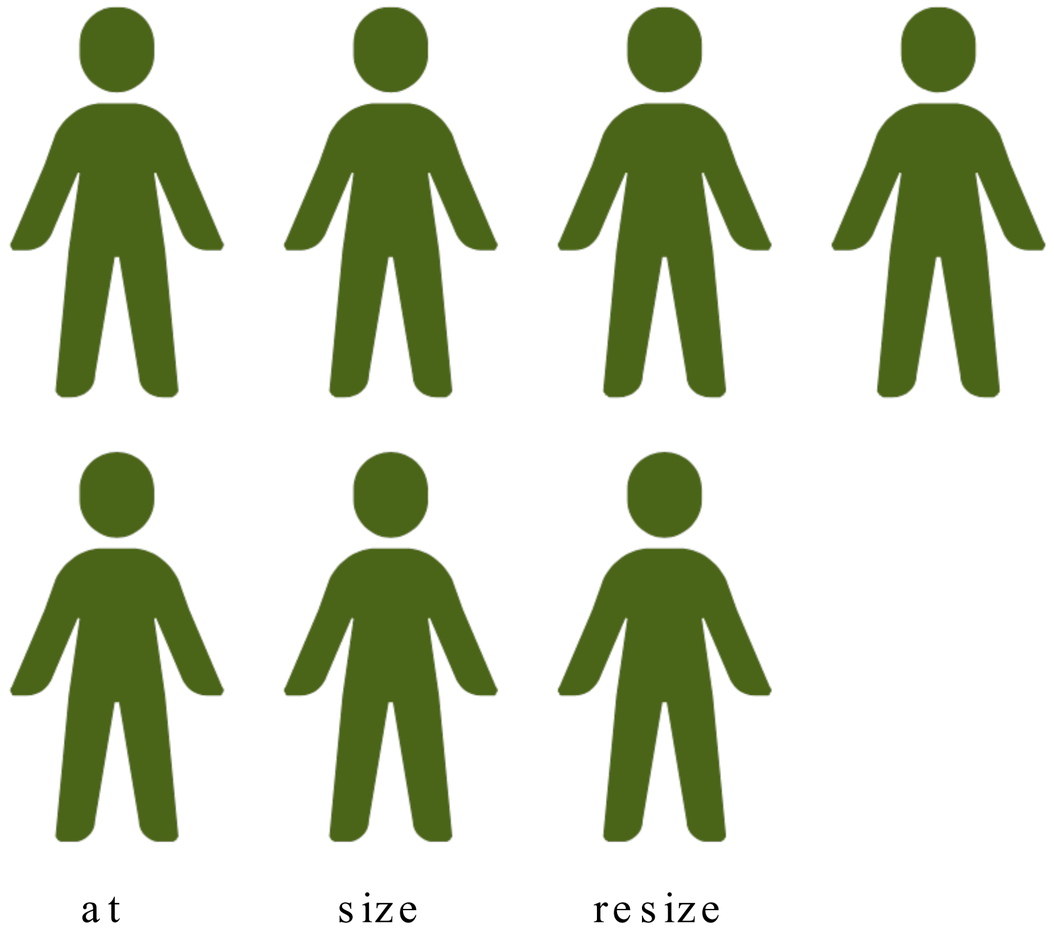
Khai báo thư viện

- #include<vector>
- using namespace std;

Khai báo vector

- std::vector<int> vec;

Một số hàm trong lớp vector



Tìm kiếm tuyến tính

```
1  int Linear_search(std::vector<float> v, float& x){
2      std::cout << "- Nhap gia tri x muon tim trong vector: ";
3      std::cin >> x;
4      for (int i = 0; i < v.size(); ++i){
5          if (v.at(i) == x){
6              return i;
7          }
8      }
9      return -1;
10 }
11
```

Tìm kiếm tuyến tính cải tiến

```
int Linear_search_but_better(std::vector<float> v, float& x){
    std::cout << "- Nhap gia tri x muon tim trong vector: ";
    std::cin >> x;
    int i = 0;
    v.push_back(x);
    while (v.at(i) != x){
        ++i;
    }
    if (i == v.size()){
        return -1;
    }
    return i;
}
```

Tìm kiếm nhị phân

```cpp
int Binary_search(std::vector<float> v, float& x){
    std::cout << "- Nhap gia tri x muon tim trong vector: ";
    std::cin >> x;
    int left = 0, right = v.size() - 1, middle;
    while (left <= right) {
        middle = (right + left) / 2;
        if (v.at(middle) == x)
            return middle;
        if (v.at(middle) < x)
            left = middle + 1;

        else
            right = middle - 1;
    }
    return -1;
}

```

THANK YOU