THUẬT TOÁN ỨNG DỤNG

Tìm kiếm và Sắp xếp

Nội dung

1. Tìm kiếm
    1. Tuyến tính
    2. Nhị phân
2. Sắp xếp
    1. Nổi bọt / Chèn / Chọn
    2. Trộn / Nhanh / Vun đống
3. Các cấu trúc dữ liệu trừu tượng
    1. Stack
    2. Queue
    3. Heap
    4. Set
    5. Map

Phần 1

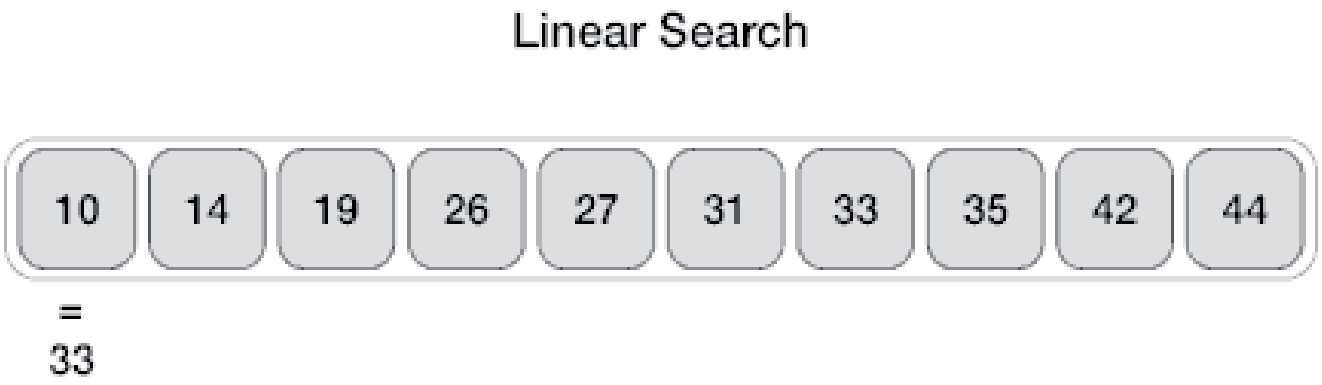
Tìm kiếm

Tìm kiếm

- Bài toán cơ bản nhất của máy tính
  - Tìm thành phần trên trang màn hình
  - Tìm tên trong danh bạ
  - Tìm kiếm web
- Câu trả lời
  - Có dữ liệu cần tìm hay không
  - Vị trí của dữ liệu cần tìm
- Tùy vào dữ liệu
  - Dữ liệu lộn xộn không có đặc trưng gì cụ thể
  - Dữ liệu được sắp xếp
  - Dữ liệu được tổ chức

Tìm kiếm tuyến tính (linear search)

- Giải thuật tìm kiếm cơ bản nhất
- Dữ liệu lộn xộn không có tính chất gì đặc biệt
- Duyệt mọi phần tử từ đầu cho đến khi tìm được dữ liệu mong muốn hoặc hết dữ liệu
- Có lẽ là cách giải duy nhất trong trường hợp bài toán không có ràng buộc về dữ liệu



Tìm kiếm nhị phân (binary search)

- Dữ liệu đã được sắp xếp (tăng dần)
- Chia đôi khoảng tìm kiếm, cho đến khi đủ nhỏ

```
// tìm kiếm nhị phân, cài đặt kiểu đệ quy
int binary_search(int arr[], int l, int r, int x) {
    if (r < l) return -1;

    int mid =  l + (r - l) / 2;
    // tìm thấy ở giữa
    if (arr[mid] == x) return mid;
    // tìm ở nửa trước
    if (arr[mid] > x)
        return binary_search(arr, l, mid - 1, x);
    // tìm ở nửa sau
    return binary_search(arr, mid + 1, r, x);
}
```

Tìm kiếm nhị phân (binary search)

- Cài đặt kiểu vòng lặp ổn hơn kiểu đệ quy ở chỗ nào?
- Cài đặt dưới đây có thể cải tiến ở điểm nào

```
// tìm kiếm nhị phân, cài đặt bằng vòng lặp
int binary_search2(int arr[], int l, int r, int x) {
    while (l <= r) {
        int m = l + (r - l) / 2;

        if (arr[m] == x) return m;

        if (arr[m] < x) l = m + 1;
        else r = m - 1;
    }

    return -1;
}
```

Tìm kiếm nội suy (interpolation search)

- Tìm kiếm khi dữ liệu cực lớn đã được sắp xếp
- Cải tiến từ tìm kiếm nhị phân: vẫn chia đôi, nhưng cân nhắc theo tương quan của dữ liệu
- Thích hợp với dữ liệu cực lớn và cân bằng

```
// tìm kiếm nội suy: nhị phân thông minh hơn
int interpolation_search(int a[], int l, int r, int x) {
    while (l <= r) {
        int m = l + (x - a[l]) * ((r - l) / (a[r] - a[l]))

        if (a[m] == x) return m;
        if (a[m] < x) l = m + 1;
        else r = m - 1;
    }
    return -1;
}
```

Cài đặt tìm kiếm ở thư viện STL C++

- Thư viện <algorithm>
- Tìm tuyến tính:
  - find: tìm giá trị trong đoạn
- Tìm nhị phân:
  - binary_search: kiểm tra xem có phần tử trong đoạn tăng dần hay không
  - lower_bound: trả về vị trí của phần tử đầu tiên không bé hơn phần tử cần tìm
  - upper_bound: trả về vị trí của phần tử đầu tiên lớn hơn phần tử cần tìm

Bài tập

1. Nhập 4 số thực A, B, C và D. Hãy tìm giá trị x với độ chính xác 5 số sau dấu phấy để phương trình sau đây đúng:

$$Ax^3 + Bx^2 + Cx + D = 0$$

2. Cho số nguyên dương k và một dãy A có N số nguyên. Hãy đếm xem có bao nhiêu cặp số trong A chênh lệch nhau đúng k đơn vị.

   Ví dụ:

   Với đầu vào k = 2 và A = (1, 5, 3, 4, 2)

   Kết quả trả về là 3.

Phần 2

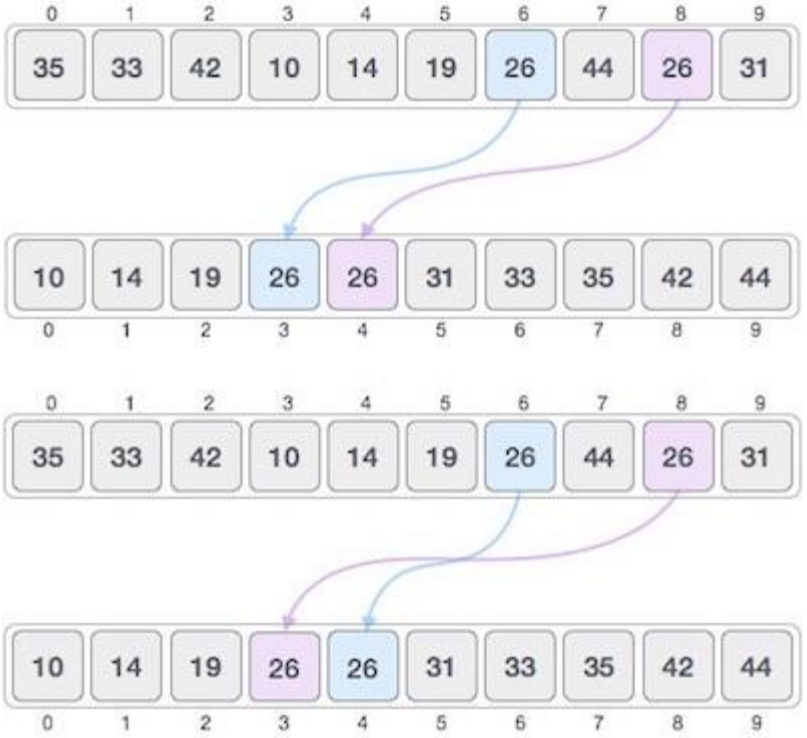
Sắp xếp

Sắp xếp

- Bài toán cơ bản của lập trình máy tính
  - Xếp tăng dần một danh sách
  - So sánh theo các khóa
- Được nghiên cứu từ rất sớm, hiện vẫn có vài cải tiến
- Rất nhiều thuật toán đã được phát triển, mỗi thuật toán có ưu / nhược điểm riêng
- Tính so sánh = thuật toán sắp xếp dựa trên việc so sánh các phần tử với nhau
  - Hầu hết các thuật toán sắp xếp đều thuộc loại này
  - Một vài thuật toán đặc biệt không cần so sánh
- Tính thích ứng (adaptive) = thuật toán tận dụng được đặc tính của dữ liệu, chạy nhanh hơn nếu dữ liệu đã sắp sẵn

Sắp xếp

- Phân loại theo cách làm việc với dữ liệu:
  - Sắp xếp tại chỗ (in-place): làm việc với chính dữ liệu sắp xếp
  - Sắp xếp ra ngoài (out-place): đẩy kết quả ra ngoài
- Phân loại theo mức độ xáo trộn dữ liệu:
  - Sắp xếp ổn định (stable): thứ tự tương đối (trước / sau) giữa các phần tử bằng nhau sẽ được giữ nguyên sau khi thực hiện thuật toán sắp xếp
  - Sắp xếp bất ổn (unstable): thứ tự tương đối của các phần tử bằng nhau có thể bị xáo trộn sau khi thực hiện thuật toán



Sắp xếp nổi bọt (bubble sort)

- Duyệt toàn bộ danh sách: nếu hai phần tử liên tiếp không đúng thứ tự (tăng dần) thì đổi chỗ chúng cho nhau
- Lặp lại bước duyệt cho đến khi không xảy ra đổi chỗ nữa
- Thuật toán có vẻ khá tệ, nhưng chạy tốt trong vài tình huống đặc biệt

6 5 3 1 8 7 2 4

Sắp xếp chèn (insertion sort)

- Giả sử phần đầu của dãy đã được sắp xếp gồm k phần tử
  - Giá trị k luôn tồn tại, ít nhất là k = 1
- Lặp lại cho đến khi k = n:
  - Lấy phần tử thứ k+1 chèn vào vị trí phù hợp của nó trong dãy ban đầu
  - Mở rộng dãy ban đầu thành gồm k+1 phần tử
- Hữu ích với những cấu trúc dữ liệu cho phép chèn nhanh

6 5 3 1 8 7 2 4

Sắp xếp chọn (selection sort)

- Chọn phần tử nhỏ nhất, đặt vào vị trí đầu tiên
- Chọn phần tử nhỏ thứ hai, đặt vào vị trí thứ hai
- Chọn phần tử nhỏ thứ ba, đặt vào vị trí thứ ba
- ...

5 3 4 1 2

Selection Sort

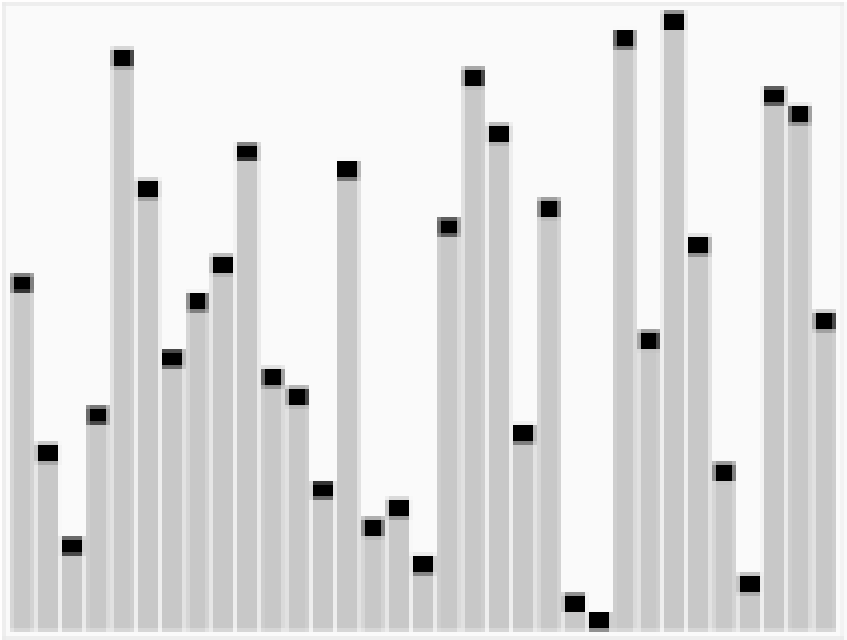
Sắp xếp trộn (merge sort)

- Dãy có 1 phần tử thì không cần làm gì thêm
- Nếu dãy có từ 2 phần tử thì chia dãy làm đôi
  - Sắp xếp từng dãy con (gọi đệ quy)
  - Trộn hai dãy con đã sắp xếp lại làm một

6 5 3 1 8 7 2 4

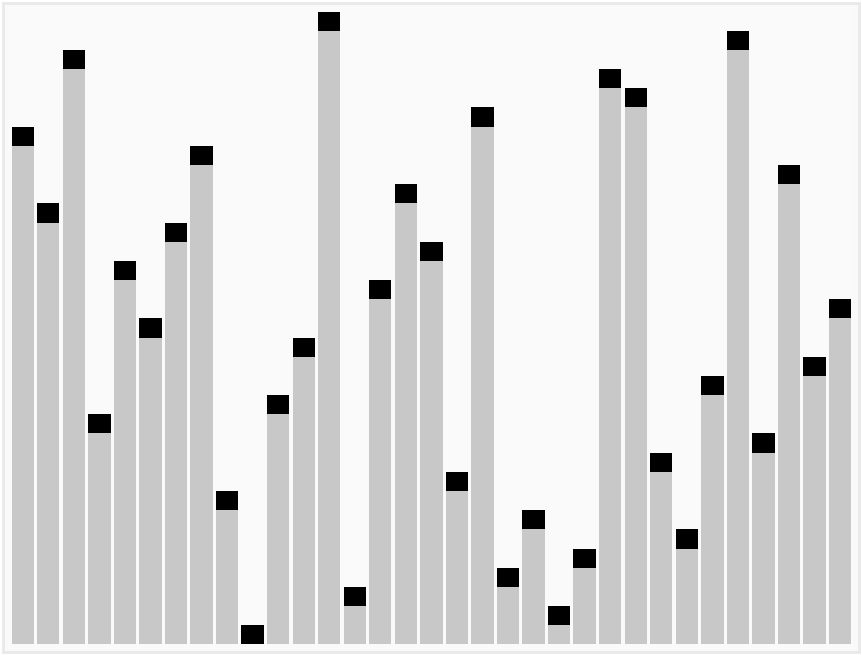
Sắp xếp nhanh (quick sort)

- Dãy độ dài 1 thì không cần sắp xếp
- Dãy độ dài 2 trở lên:
  - Chọn ngẫu nhiên một giá trị M trong dãy
  - Dồn những giá trị nhỏ hơn M về đầu dãy, cuối dãy là những giá trị lớn hơn M
  - Sắp xếp hai dãy con (đệ quy)



Sắp xếp vun đống (heap sort)

- Bước 1: tạo cấu trúc "đống" (heap) từ dữ liệu đã có
  - Heap = Dãy A $(a_1,...,a_n)$ mà $a_k > \max(a_{2k}, a_{2k+1})$
- Bước 2: lần lượt lấy phần tử lớn nhất ra khỏi đống và chuyển xuống cuối dãy



Cài đặt sắp xếp ở thư viện STL C++

- Thư viện <algorithm>
- sort: sắp xếp (tăng dần) một đoạn, sử dụng introsort
- stable_sort: sắp xếp ổn định (tăng dần) một đoạn, sử dụng mergesort
- partial_sort: sắp xếp phần đầu của đoạn theo thứ tự tăng dần, sử dụng khi ta chỉ cần lấy vài phần tử nhỏ nhất

Bài tập

3. Cho một dãy số tự nhiên A có nhiều hơn một phần tử, hãy tìm các cặp phần tử gần nhau nhất trong dãy.
   - Các cặp phần tử mà chênh lệch giữa chúng là nhỏ nhất trong dãy, nếu có nhiều cặp như vậy thì in ra tất cả
   - A = (-20, 737481, -73301, 30, -61594, 26854, -520, -470)
   - Kết quả in ra 2 cặp: (-20, 30) (-520, -470)
4. Cho một dãy số nguyên A có n phần tử và số nguyên dương k < n. Hãy chọn ra k số nguyên trong A tạo thành dãy B sao cho chênh lệch giữa số lớn và nhỏ nhất trong B là tối thiểu. In ra chênh lệch đó.
   - A = (1, 2, 3, 4, 10, 20, 30, 40, 100, 200) và k = 4
   - Kết quả in ra: 3

Bài tập

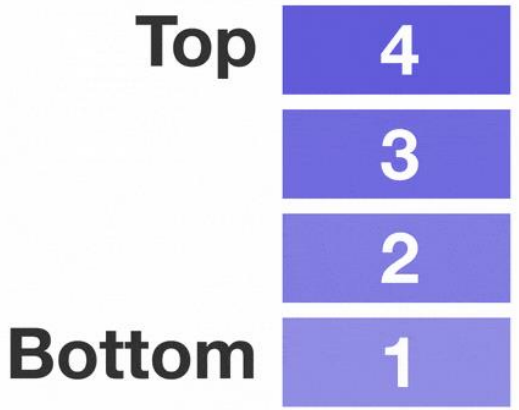
5. Cho số m và dãy số tự nhiên A có n phần tử, hãy đếm xem có bao nhiêu cách chọn hai phần tử trong A mà tích của chúng lớn hơn m.
   - A = (1, 2, 3, 4, 5), m = 15
   - Kết quả là 1 (4 x 5), không tính 4 x 4 vì đó là hai phần tử trùng nhau
6. Cho dãy số A = (a_1, a_2,..., a_n). Với mỗi cặp chỉ số p và q (p < q), bạn cần chỉ ra cách chia dãy con (a_p, a_{p+1},..., a_q) thành hai nửa sao cho chênh lệch giữa tổng các số thuộc hai nửa là nhỏ nhất.
   - A = (3, 1, 4, 2, 5), p = 2, q = 5
   - Kết quả là 2 (chia thành 2 nửa [1, 4] và [2, 5])
   - Chú ý: cần thiết kế thuật toán hợp lý với nhiều cặp (p, q)

Phần 3

Các cấu trúc dữ liệu trừu tượng

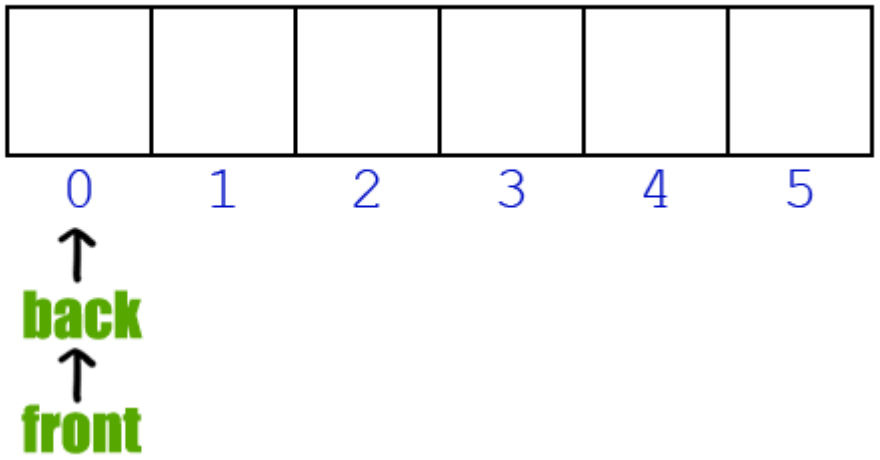
Stack

- Ngăn xếp
- LIFO: last-in, first-out
- Thường được cài đặt dựa trên list, vector, array
- Thao tác cơ bản:
  - Thêm vào (push): đặt vào cuối
  - Lấy ra (pop): lấy ra phần tử ở cuối
  - Đọc ở đầu (top)
  - Lấy số phần tử (size)
  - Kiểm tra rỗng (empty)



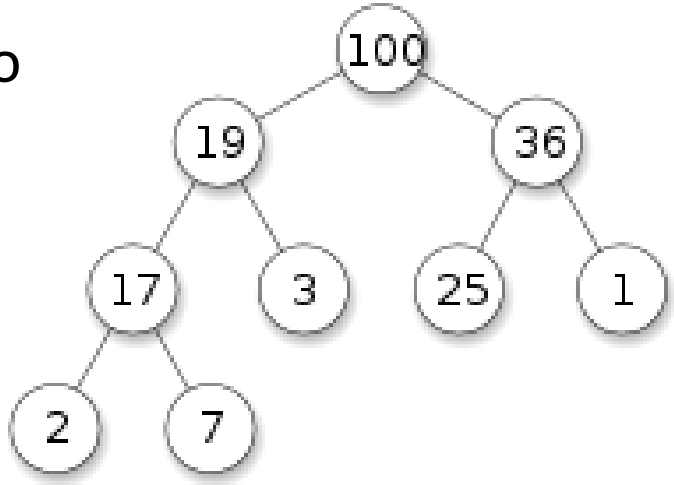
Queue

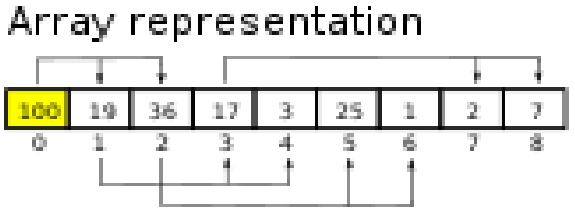
- Hàng đợi
- FIFO: first-in, first-out
- Dạng hai đầu: Deque
- Thao tác cơ bản: enqueue / dequeue
  - Thêm vào (push): thêm vào cuối
  - Lấy ra (pop): lấy phần tử ở đầu
  - Đầu (front)
  - Cuối (back)
  - Cỡ (size)
  - Rỗng? (empty)



Heap

- Đống
- Cấu trúc sử dụng trong heap sort
- Còn gọi là priority queue (hàng đợi ưu tiên)
  - Dữ liệu tổ chức dạng heap, thứ tự giảm dần
  - Thêm vào (push): tự đặt phần tử vào vị trí phù hợp trong heap
  - Lấy ra (pop): lấy phần tử lớn nhất
  - Đầu (top)
  - Cỡ (size)
  - Rỗng? (empty)





Set

- Tập hợp: các phần tử phải khác nhau
- Thường cài đặt trên red-black tree
- Phương thức: empty / size / insert / erase / clear / find
- Vài kiểu dữ liệu cùng loại:
  - multiset: cho phép các phần tử có thể bằng nhau
  - unordered_set: set nhưng sử dụng hash table
  - unordered_multiset: sử dụng hash table và các phần tử có thể bằng nhau

Map

- Ánh xạ / Từ điển
- Phương thức: empty / size / [] / at / insert / erase / clear / find / count / begin / end
- Cho phép ánh xạ từ một khóa (key) tới giá trị (value)
- Vài kiểu dữ liệu cùng loại:
  - multimap : các key có thể trùng nhau
  - unordered_map : sử dụng hash table
  - unordered_multimap: sử dụng hash table và các key có thể trùng nhau