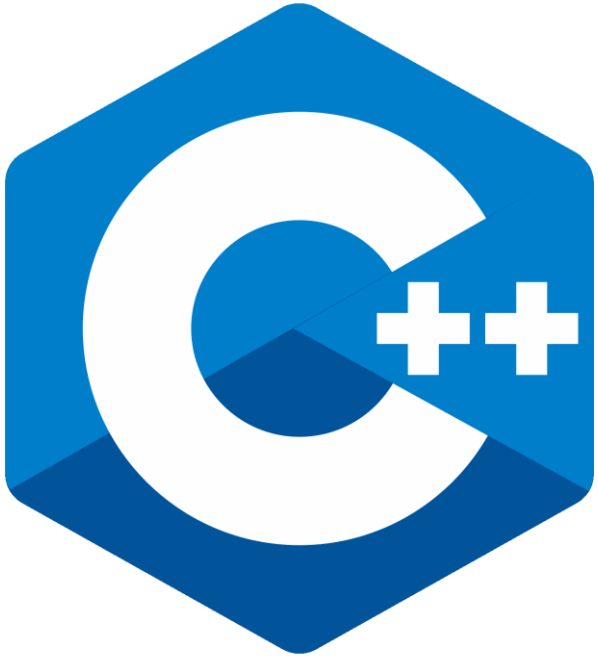
KỸ THUẬT LẬP TRÌNH

TRẦN ĐÌNH LUYỆN

Các thuật toán cơ bản

Giới thiệu một số thuật toán cơ bản trong lập trình



Định nghĩa

- Là danh sách các hướng dẫn và quy tắc mà máy tính cần thực hiện để hoàn thành một tác vụ. Về bản chất, các thuật toán là một loạt các chỉ dẫn được tuân theo từng bước để làm điều gì đó hữu ích hoặc giải quyết một vấn đề.
- Thuật toán là một tập hợp hữu hạn các lệnh được xác định rõ ràng, có thể thực hiện được bằng máy tính, thường để giải quyết một lớp vấn đề hoặc thực hiện một phép tính.

Tính chất

- **Chính xác**: Đối với một thuật toán về cơ bản cần tính chính xác ban đầu khá cao. Đây là yếu tố gần như quyết định độ thông dụng và khách quan của thuật toán đó.
- **Khách quan**: Tính khách quan được thể hiện ở chỗ chỉ có một kết quả duy nhất cho dù có giải bằng cách nào đi nữa. Trường hợp đưa ra kết quả khác nhau thì cần xem lại cách xử lý.
- **Thông dụng**: Tính thông dụng thì được thể hiện ở sự linh động. Ví dụ một thuật toán có thể sử dụng với nhiều bài toán tương tự.

Tính chất

- **Rõ ràng**: Tính rõ ràng được thể hiện ở việc thuật toán được sắp xếp theo quy trình, quy củ khiến nó hoạt động nhanh gọn và trơn tru hơn nhiều. Bên cạnh đó tình chất rõ ràng còn thể hiện ở các nguyên tắc lệnh.
- **Kết thúc**: Kết thúc một thuật toán là khi nó đưa ra kết quả cuối cùng. Đây cũng là tính kết thúc của thuật toán đó.

Một số thuật toán cơ bản

- Tìm kiếm
- Sắp xếp
- Đệ quy
- Quay lui
- Quy hoạch động
- ....

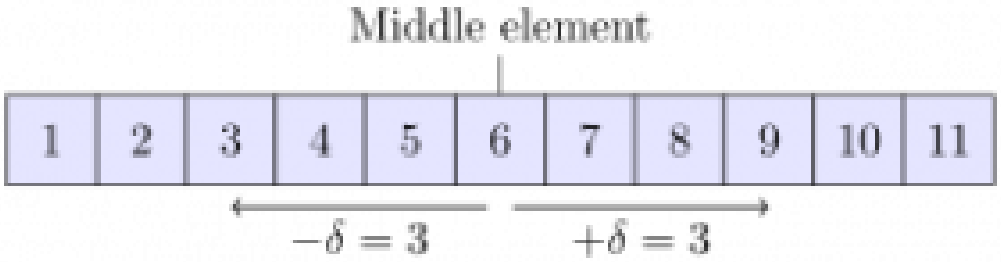
Search Algorithms

- Thuật toán tìm kiếm thường được áp dụng cho dãy cấu trúc dữ liệu tuyến tính hoặc cấu trúc dữ liệu đồ họa.
- Thuật toán tìm kiếm tuyến tính còn được gọi là tìm kiếm nhị phân, giúp nhà phát triển tiến hành tìm kiếm hiệu quả trên các tập dữ liệu được sắp xếp với hàm phức tạp thời gian của O (log N).
- Cơ chế của tìm kiếm nhị phân là chia danh sách thành hai nửa cho đến khi nó tìm thấy mục được yêu cầu và thường được sử dụng để gỡ những lỗi liên quan đến git bisection.

Search Algorithms

- Các thuật toán này còn được biết đến với chức năng là Chiều sâu/Chiều rộng Tìm kiếm Đầu tiên, nó cho ta cấu trúc dữ liệu là một biểu đồ tròn hoặc hình cây đã bật chức năng tìm kiếm, xác định các tập dữ liệu cần thiết trong mô hình cây ngang.
- BFS rất phổ biến trong các công cụ tìm kiếm, cũng được sử dụng để xây dựng các bot trong AI hay định vị các con đường ngắn nhất giữa hai thành phố.

Search Algorithms



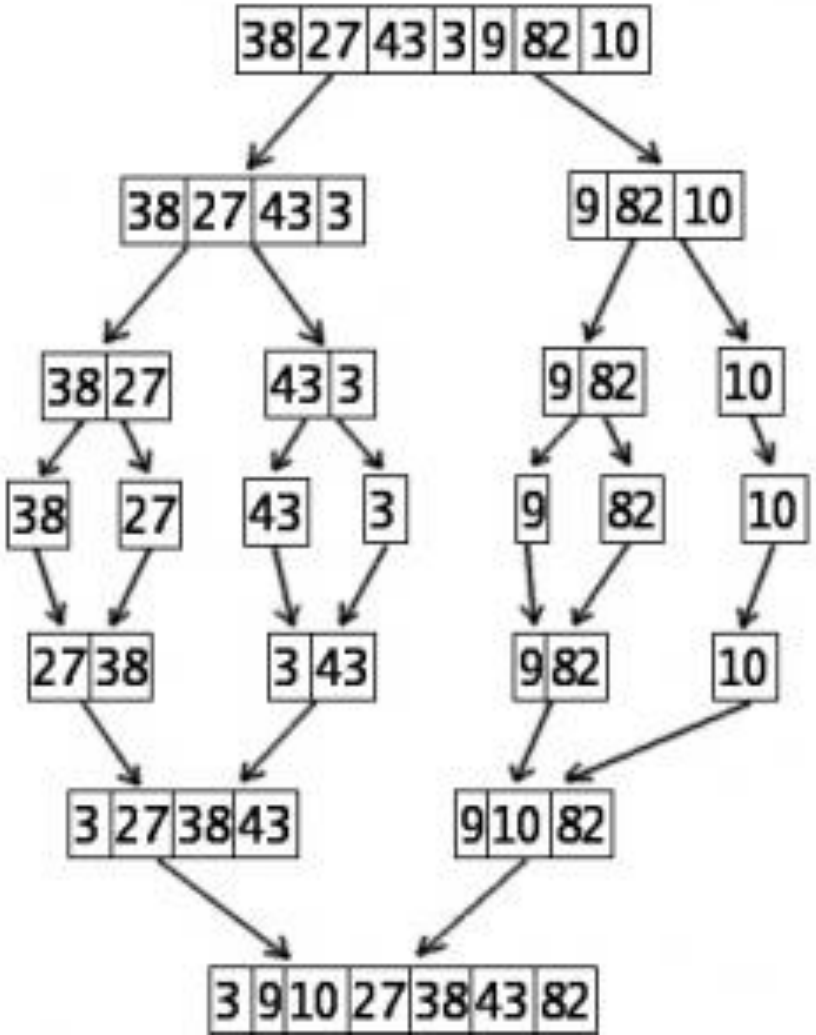
Sort Algorithms

- Các thuật toán sắp xếp thường được các nhà phát triển dùng để đặt dữ liệu theo cách có tổ chức.
  - Sắp xếp chọn
  - Sắp xếp nổi bọt
  - Sắp xếp trộn
  - Sắp xếp nhanh
  - Sắp xếp vun đống
  - ...

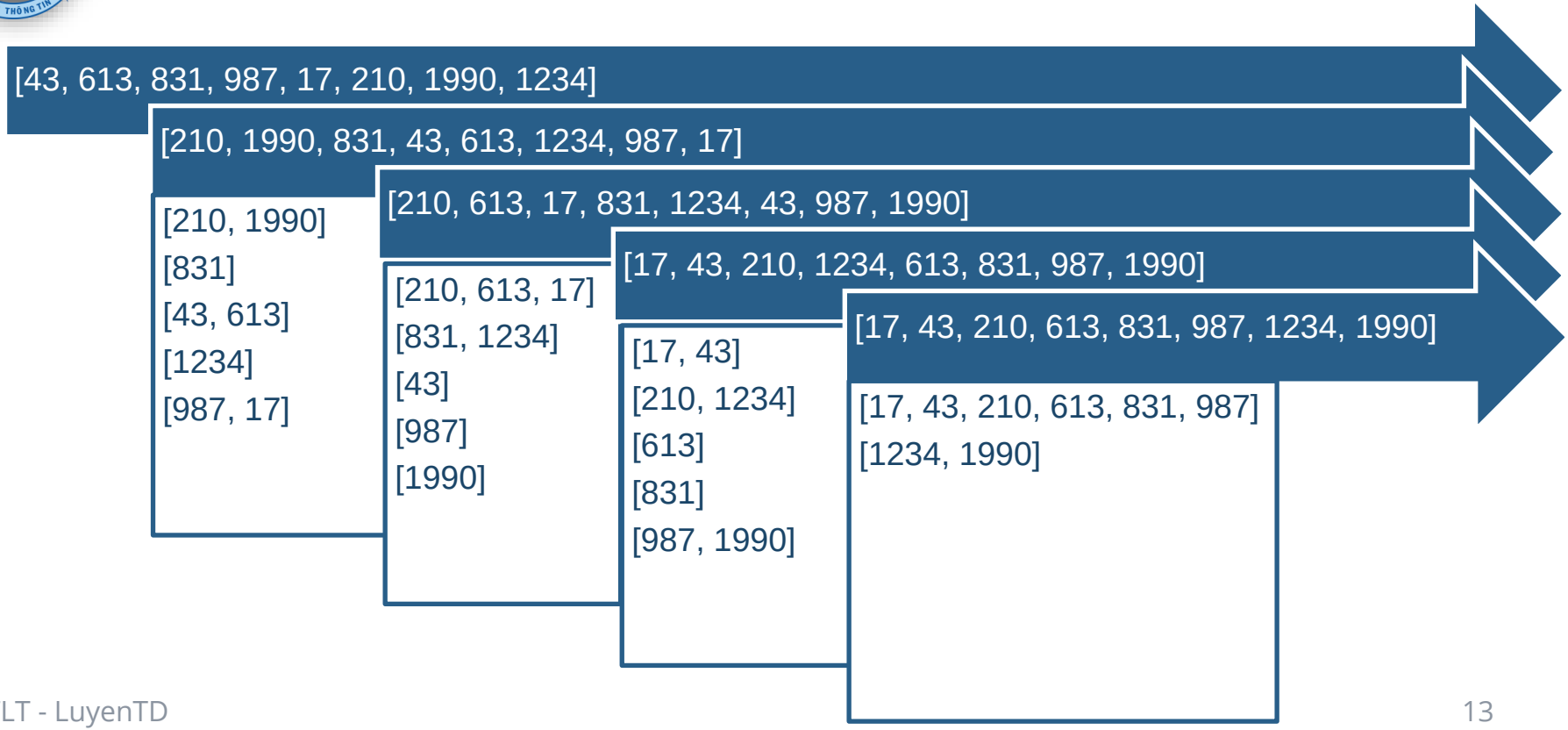
Sort Algorithms

- Trong thuật toán QuickSort, các thành phần dữ liệu được so sánh với nhau để xác định thứ tự tương ứng của chúng.
- Độ phức tạp thời gian của O(nLogn) để thực hiện so sánh.
- Tuy nhiên, Radix Sort là một kỹ thuật nhanh hơn QuickSort vì nó sắp xếp các phần tử trong một mô hình tuyến tính với độ phức tạp thời gian O(n). Tính đơn giản của thuật toán này làm cho nó được ưa chuộng.

Sort Algorithms



Radix sort


[43, 613, 831, 987, 17, 210, 1990, 1234]
[210, 1990, 831, 43, 613, 1234, 987, 17]
[210, 1990]
[831]
[43, 613]
[1234]
[987, 17]
[210, 613, 17, 831, 1234, 43, 987, 1990]
[210, 613, 17]
[831, 1234]
[43]
[987]
[1990]
[17, 43, 210, 1234, 613, 831, 987, 1990]
[17, 43]
[210, 1234]
[613]
[831]
[987, 1990]
[17, 43, 210, 613, 831, 987, 1234, 1990]
[17, 43, 210, 613, 831, 987]
[1234, 1990]

Đệ quy (RECURSIVE)

- Đệ quy: có nghĩa là một hàm tự gọi lại chính nó.
- Thành phần: 2 phần:
  - Phần cơ sở: Điều kiện để thoát khỏi đệ quy. Nếu như không có phần này, hàm đệ quy sẽ thực hiện mãi mãi gây ra tràn bộ nhớ Stack.
  - Phần đệ quy: Thân hàm có chứa phần gọi đệ quy, thực hiện cho đến khi thỏa mãn điều kiện ở phần cơ sở.

Đệ quy (RECURSIVE)

- Thuật toán đệ quy được dùng hiệu quả trong việc giải các bài toán mà có thể biểu diễn dưới dạng các đối tượng, các hàm mà có thể gọi lại chính nó. Sử dụng các thuật toán đệ quy sẽ giúp cho cách giải bài toán trở nên ngắn gọn và dễ hiểu hơn.
- Ví dụ:
  - Tính n! sẽ viết f(n+1) = f(n) * (n+1)
  - Tính tổng n số nguyên đầu tiên s(n+1) = s(n) + (n+1)
  - Tính số tiếp theo của dãy fibonaci: f(n+1) = f(n) + f(n-1)

Đệ quy (RECURSIVE)

- Việc sử dụng các thuật toán đệ quy sẽ giúp cho việc giải các bài toán trở nên ngắn gọn và dễ hiểu.
- Tuy nhiên, các thuật toán đệ quy ngốn khá nhiều bộ nhớ đệm khi thực hiện nên thỉnh thoảng gây ra lỗi tràn ngăn xếp (stack) rất khó chịu nên một số ngôn ngữ không khuyến khích dùng đệ qui hay nói cách khác nó không thiết kế tối ưu cho các thuật toán đệ quy.
- Tuy vậy, một số ngôn ngữ lập trình xem việc xử lý đệ quy như thế mạnh của mình như Python chẳng hạn. Do vậy, việc sử dụng đệ quy hay không còn tùy thuộc vào bài toán và ngôn ngữ lập trình mà bạn sử dụng.

Đệ quy (RECURSIVE)

- Ví dụ 1: Tính tổng các số từ 1 đến n.

```
#include <stdio.h>
int sum(int n){
    if(n == 0)
        return 0;
    return n + sum(n-1);
}
int main(){
    int sum = sum(5);
    cout << "Sum = " << sum <<  endl;
    return 0;
}
```

Vấn đề gì xảy ra nếu n lớn?

Đệ quy (RECURSIVE)

- Ví dụ: Dãy Fibonacci

```
#include <stdio.h>
int fibonacci(int n){
    if ((n == 1) || (n == 2))
        return 1;
    return fibonacci(n-1) + fibonacci(n-2);
}
int main(){
    cout << fibonacci(30) << endl;
    return 0;
}
```

Bài tập luyện tập

- Cài đặt thuật toán sắp xếp radix: chia các số thành nhóm có chung chữ số ở hang tương ứng
  - Hàng đơn vị,
  - Hàng chục,
  - Hàng trăm,
  - Hàng nghìn,
  - Hàng chục nghìn,
  - ...

Q&A