

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Ôn tập về lập trình

- 1 Kiến thức nền tảng cần thiết
- 2 Bài toán tính tổng hai số
- 3 Bài toán tính tích hai số lớn nhất
- 4 Tại sao cần học giải thuật
- 5 Bài toán Fibonacci
- 6 Bài toán Ước chung lớn nhất
- 7 Độ phức tạp tính toán, Ký hiệu Big-O

Giới thiệu về thuật toán

- Thuật toán xuất hiện ở khắp nơi trong thế giới của chúng ta:

Viết phần mềm, Dự đoán tắc đường, Đề xuất tự động, ...

- Các thuật toán phải hiệu quả, cho kết quả trong thời gian chờ đợi chấp nhận được.
- Các công ty về công nghệ thường xuyên đặt các câu hỏi về thuật toán tại các cuộc phỏng vấn.
- Thuật toán đóng vai trò quan trọng trong việc xây dựng các hệ thống đáp ứng các nhu cầu thực tế trong các lĩnh vực như công nghệ, y học, kinh doanh, ...

Kiến thức cơ bản về ít nhất một ngôn ngữ lập trình: C, C ++, Java, C

- Có thể triển khai một chương trình máy tính:
 - + Đọc dữ liệu vào từ đầu vào chuẩn,
 - + Tính toán kết quả,
 - + In kết quả theo yêu cầu.
- Thử nghiệm tự lập trình với các ví dụ trong chương này, trong trường hợp không thể tự triển khai xây dựng mã nguồn, sinh viên nên tự trau dồi thêm kiến thức về lập trình qua các môn học Tin học cơ sở 2, Ngôn ngữ lập trình C++.

Kiến thức cơ bản của toán rời rạc: chứng minh bằng quy nạp, chứng minh bằng phản chứng

- Kiến thức về toán học rời rạc là cần thiết để phân tích các thuật toán (chứng minh tính đúng đắn, ước tính thời gian chạy) và cho tư duy thuật toán nói chung.
- Trong lập trình tối ưu, các thuật toán có thể được xây dựng và chứng minh từ các công thức toán học. Vì vậy, việc nắm vững các kiến thức toán học có thể giúp tối ưu các kết quả tính toán và đưa ra các giải thuật nhanh và chính xác.

Yêu cầu

- Cho hai số nguyên $0 \leq A \leq 9$ và $0 \leq B \leq 9$.
- Dữ liệu vào: Một dòng chứa hai số A và B
- Dữ liệu ra: Kết quả $A + B$

Bài toán tính tổng hai số

Ví dụ

Input	Output
1 2	3

Input	Output
5 6	11

Bài toán tính tổng hai số

Mã nguồn mẫu C++

```
#include <iostream>
using namespace std;

int main() {
    int a, b;
    cin >> a >> b;
    int sum = a + b;
    cout << sum;
    return 0;
}
```


Mã nguồn mẫu Java

```
public class APlusB {  
    public static void main(String[] args) {  
        int a, b;  
        Scanner sc = new Scanner(System.in);  
        a = sc.nextInt();  
        b = sc.nextInt();  
        int sum = a + b;  
        System.out.println(sum);  
    }  
}
```

Yêu cầu

- Cho danh sách n các số nguyên $0 \leq A_0 A_1 \dots A_{n-1} \leq 10^5$, Tìm tích của cặp số lớn nhất sao cho $\max_{0 \leq i \neq j \leq n-1} a_i a_j$
- Dữ liệu vào: Dòng đầu tiên là số n sao cho $2 \leq n \leq 2 \cdot 10^5$. Dòng tiếp theo cho n số nguyên không âm $0 \leq A_0 A_1 \dots A_{n-1} \leq 10^5$
- Dữ liệu ra: Một số nguyên là tích của cặp số lớn nhất

Thời gian yêu cầu: 1s.

Bài toán tính tích hai số lớn nhất

Ví dụ

Input	Output
3 1 2 3	6

Input	Output
10 7 5 14 2 8 8 10 1 2 3	140

Giải thuật 1

- Giả sử tích lớn nhất là 0 (Dãy số nguyên không âm).
- Duyệt lần lượt tất cả các cặp số có thể được tạo bởi dãy
- Tính thử tích cặp số đang duyệt:
 - + Nếu tích lớn hơn, cập nhật kết quả
 - + Nếu tích nhỏ hơn hoặc bằng, bỏ qua

Mã nguồn giải thuật 1

```
int max_pairwise_product(vector<int> v) {  
    int result = 0;  
    for (int i = 0; i < v.size() - 1; i++) {  
        for (int j = i + 1; j < v.size(); j++) {  
            if (result < v[i] * v[j]) {  
                result = v[i] * v[j];  
            }  
        }  
    }  
    return result;  
}
```

Đánh giá giải thuật 1

- Có vẻ giải quyết đúng yêu cầu của bài toán.
- Tuy nhiên kết quả vẫn chưa đúng với một vài trường hợp.
- Xét trường hợp đầu vào như sau:

Input	Output
2	410065408
90000 100000	

Đánh giá giải thuật 1

- Dễ dàng thấy kết quả bị sai.
- Nguyên nhân do tràn số, kiểu dữ liệu int không chứa được giá trị 9000000000.
- Giải quyết bằng cách thay đổi kiểu dữ liệu thành long long:

Input	Output
2	9000000000
90000 100000	

Đánh giá giải thuật 1

- Chương trình đã giải quyết được vấn đề tràn số, tuy nhiên vẫn không đạt yêu cầu của bài toán do xử lý một số trường hợp quá thời gian.
- Nguyên nhân của việc chạy quá thời gian là gì?

Mã nguồn giải thuật 1

```
int max_pairwise_product(vector<int> v) {  
    int result = 0;  
    for (int i = 0; i < v.size() - 1; i++) {  
        for (int j = i + 1; j < v.size(); j++) {  
            if (result < v[i] * v[j]) {  
                result = v[i] * v[j];  
            }  
        }  
    }  
    return result;  
}
```

Giải thuật 2

- Thử cải thiện số lần lặp bằng cách tìm hai số lớn nhất từ dãy và tính tích của hai số đó.
- Duyệt lần lượt tất cả các số trong dãy, tìm vị trí của số lớn nhất
- Lặp lại bước duyệt một lần nữa, tìm vị trí của số lớn nhất sau khi đã loại trừ số đã tìm ở bước trên.
- Tính tích của cặp số vừa tìm được

Mã nguồn giải thuật 2

```
long long max_pairwise_product2(vector<int> v) {
    int max_index1 = 0;
    for (int i = 1; i < v.size(); i++) {
        if (v[i] > v[max_index1]) {
            max_index1 = i;
        }
    }
    int max_index2 = -1;
    for (int i = 0; i < v.size(); i++) {
        if (v[i] != v[max_index1] && (max_index2 == -1 || v[i] > v[max_index2]))
            max_index2 = i;
    }
    long long result = (long long) v[max_index1] * v[max_index2];
    return result;
}
```

Đánh giá giải thuật 2

- Giải thuật vẫn cho kết quả chưa chính xác.
- Cần có phương án để tìm được lỗi sai của lời giải một cách tự động thay vì dự đoán

Kỹ thuật Stress Testing

- Lý do sử dụng: không được xem trực tiếp testcase ẩn khi giải bài.
- Tạo vòng lặp vô hạn để sinh ngẫu nhiên các giá trị đầu vào theo điều kiện đề bài
- Tính kết quả dựa trên hai giải thuật đã xây dựng.
- So sánh và tìm kết quả khác nhau giữa hai giải thuật để tìm nguyên nhân việc sai kết quả.
- Phương pháp này phù hợp với các bài toán đã tìm được phương án giải đúng nhưng chưa tối ưu

Triển khai stress testing

```
while (true) {
    int n = rand() % 10 + 2;
    cout << n << endl;
    vector<int> a;
    for (int i = 0; i < n; i++) {
        a.push_back(rand() % 100);
    }
    for (int i = 0; i < n; i++) {
        cout << a[i] << " ";
    }
    cout << endl;
    long long result1 = max_pairwise_product(a);
    long long result2 = max_pairwise_product2(a);
    if (result1 == result2) {
        cout << "OK" << endl;
    } else {
        cout << "WA r1: " << result1 << ", r2: " << result2 << endl;
        break;
    }
}
```

Kết quả kiểm tra

```
55 10 59 24
OK
9
48 83 95 41 2 50 91 36 74
OK
2
96 21
OK
10
99 68 84 81 34 53 99 18 38 0
WA r1: 9801, r2: 8316
```

Phân tích nguyên nhân

- Trường hợp khác nhau xuất hiện 2 giống nhau cùng có giá trị lớn nhất.
- Giải thuật 2 cho kết quả sai do tìm được cặp số lớn nhất là 99 và 84
- Xác định đoạn chương trình lỗi và tiến hành chỉnh sửa mã nguồn để có kết quả đúng hơn.
- Lặp lại quy trình stress testing một khoảng thời gian nhất định để đảm bảo kết quả có độ chính xác cao

Bài toán tính tích hai số lớn nhất

```
long long max_pairwise_product2(vector<int> v) {
    int max_index1 = 0;
    for (int i = 1; i < v.size(); i++) {
        if (v[i] > v[max_index1]) {
            max_index1 = i;
        }
    }
    int max_index2 = -1;
    for (int i = 0; i < v.size(); i++) {
        if (i != max_index1 && (max_index2 == -1 || v[i] > v[max_index2])) {
            max_index2 = i;
        }
    }
    long long result = (long long) v[max_index1] * v[max_index2];
    return result;
}
```

Mục tiêu học tập

- Tìm hiểu về các loại bài tập trong môn học.
- Nhận diện được các bài toán cần sử dụng các giải thuật phù hợp
- Giải quyết các vấn đề thường xuyên gặp phải trong lập trình một cách tối ưu

Một số ví dụ

- Chương trình sao chép tập tin.
- Chương trình tìm đường đi ngắn nhất giữa hai địa điểm.
- Các chương trình trên có thể lập trình để nó hoạt động đúng, tuy nhiên thời gian chờ đợi đối với người sử dụng phải đáp ứng được nhu cầu thực tế.
- Có thể chương trình đã đáp ứng được nhu cầu thực tế nhưng cần tối ưu hơn nữa để giảm kích thước lưu trữ, giảm bộ nhớ, ... để sẵn sàng cho việc mở rộng.

Một số ví dụ mở rộng

- Viết một chương trình máy tính hiểu được ngôn ngữ tự nhiên
- Xác định các đối tượng xuất hiện trong một bức ảnh bất kỳ một cách tự động
- Viết chương trình tìm kiếm lời giải hiệu quả cho các trò chơi như cờ Caro, cờ vua.

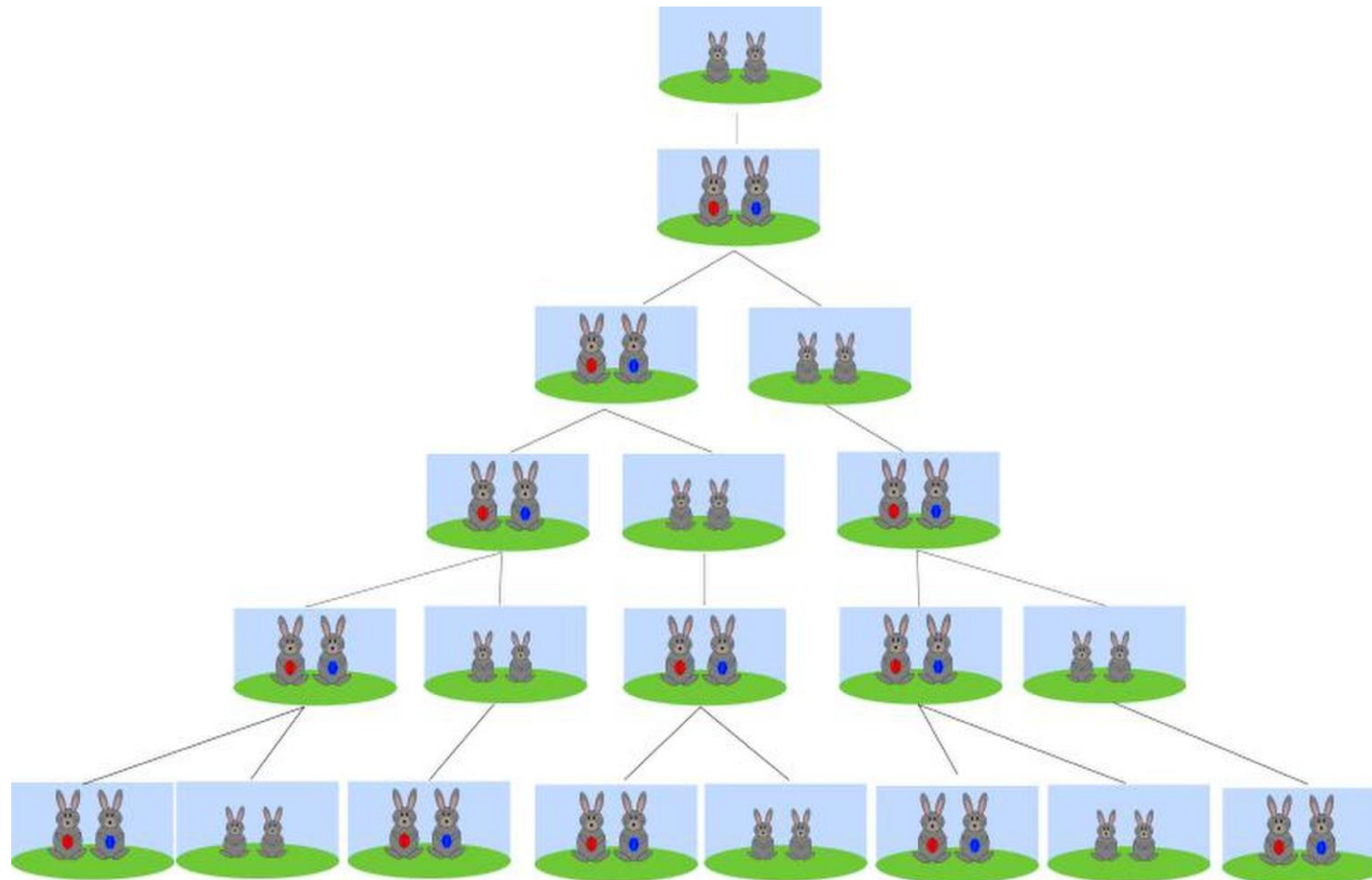
Nội dung môn học này không đi sâu vào các vấn đề liên quan đến AI nhưng có thể cung cấp các kiến thức cơ bản về thuật toán để có nền tảng vững chắc cho việc phát triển theo hướng này trong tương lai.

Định nghĩa về dãy số fibonacci

$$\bullet f(n) = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ F_{n-1} + F_{n-2}, & n > 1 \end{cases}$$

• 0, 1, 1, 2, 3, 5, 8, 13, ...

Bài toán cổ về nuôi thỏ



Công thức số hạng tổng quát Fibonacci

$$F_n = \frac{1}{\sqrt{5}} \left[\left(\frac{1 + \sqrt{5}}{2} \right)^n - \left(\frac{1 - \sqrt{5}}{2} \right)^n \right]$$

Ví dụ

$F_{20} = 6765$

$F_{50} = 12586269025$

$F_{100} = 354224848179261915075$

$F_{500} = 13942322456169788013972438287040$

72839500702565876973072641089629483255

71622863290691557658876222521294125

Yêu cầu

- Cho số nguyên không âm n
- Dữ liệu vào: Số n ($0 \leq n \leq 92$)
- Dữ liệu ra: Số Fibonacci thứ n

Thời gian yêu cầu: 1s.

Ví dụ

Input	Output
3	2

Input	Output
20	6765

Giải thuật 1

```
long long fibo(int n) {  
    if (n < 2) {  
        return n;  
    }  
    return fibo(n-1) + fibo(n-2);  
}
```

Đánh giá giải thuật 1

- Ngắn gọn
- Dễ triển khai, tư duy theo đúng yêu cầu và mô tả của bài toán
- Kết quả phản hồi trong dưới 1s với $n \leq 40$. Tuy nhiên, với n lớn, chương trình không có khả năng phản hồi ngay.

Ước tính thời gian chạy

- Gọi $T(n)$ biểu thị số dòng lệnh được thực thi bởi hàm $\text{fibonacci}(n)$
- Xét $n < 2$, $T = 2$

```
long long fibo(int n) {  
    if (n < 2) {  
        return n;  
    }  
    return fibo(n-1) + fibo(n-2);  
}
```

Ước tính thời gian chạy

- Xét $n \geq 2$, $T(2) = 3$?

```
long long fibo(int n) {  
    if (n < 2) {  
        return n;  
    }  
    return fibo(n-1) + fibo(n-2);  
}
```

$$T(n) = 2 + T(n-1) + T(n-2)$$

Ước tính thời gian chạy

$$T(n) = \begin{cases} 2, & n < 2 \\ T(n-1) + T(n-2) + 2, & n \geq 2 \end{cases}$$

Do đó: $T(n) \geq F(n)$

$$T(100) \geq 1,77 * 10^{21}$$

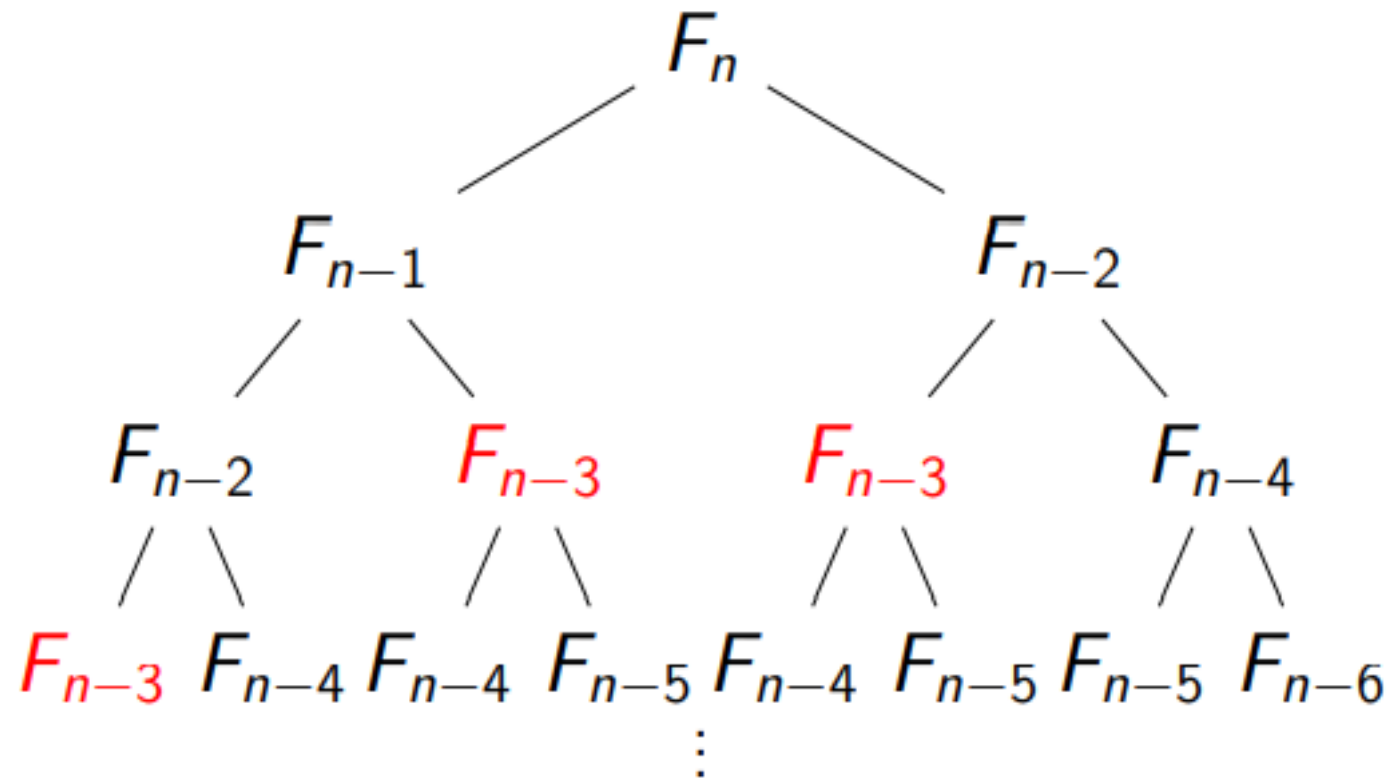
Đánh giá giải thuật 1

- Ngắn gọn
- Dễ triển khai, tư duy theo đúng yêu cầu và mô tả của bài toán
- Kết quả phản hồi trong dưới 1s với $n \leq 40$. Tuy nhiên, với n lớn, chương trình không có khả năng phản hồi ngay.
- Xét $N = 100$, máy tính có tốc độ (1Ghz) có khả năng thực hiện 1 tỷ phép tính / giây. Theo ước tính sẽ mất 56.000 năm để chương trình đưa ra kết quả \Rightarrow thời gian không chấp nhận được, cần thuật toán tốt hơn để xử lý.

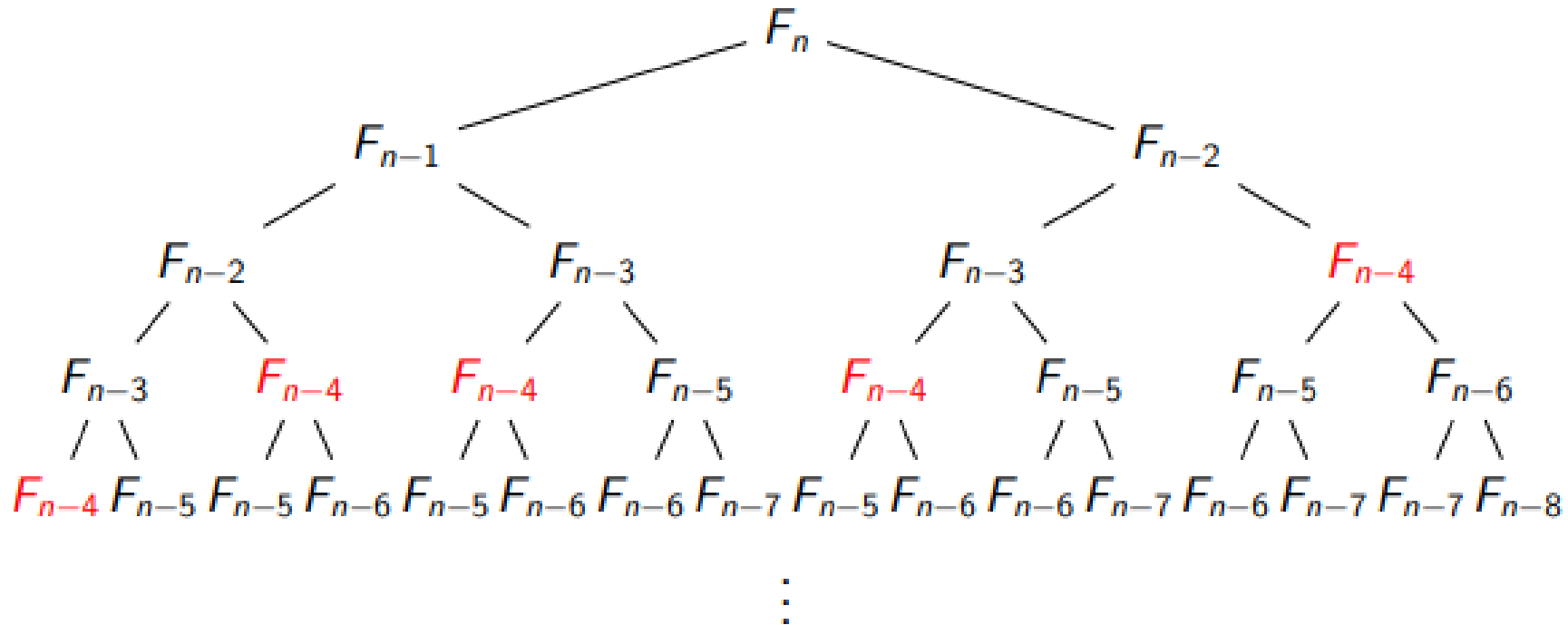
Lý do thuật toán 1 chạy chậm

- Số lượng lời gọi hàm quá lớn và trùng lặp
- Để tính $f(n)$, cần phải tính $f(n-1)$ và $f(n-2)$. Để tính $f(n-1)$ cần tính $f(n-2)$ và $f(n-3)$, ... \Rightarrow cây đệ quy

Lý do thuật toán 1 chạy chậm



Lý do thuật toán 1 chạy chậm



Giải thuật 2

- Tính toán bằng cách liệt kê từng phần tử của dãy số

0, 1, 1, 2, ...

- Phần tử tiếp theo là $1 + 2 = 3$

Giải thuật 2

```
long long fibo_array(int n) {  
    vector<long long> f;  
    f.push_back(0);  
    f.push_back(1);  
    for (int i = 2; i <= n; i++) {  
        long long next_number = f[i - 1] + f[i - 2];  
        f.push_back(next_number);  
    }  
    return f[n];  
}
```

Giải thuật 2

Một cách triển khai khác

```
long long fib(int n)
{
    if (n < 2)
        return n;
    long long a = 0, b = 1, c;
    for (int i = 2; i <= n; i++) {
        c = a + b;
        a = b;
        b = c;
    }
    return c;
}
```

Ước tính thời gian chạy

```
long long fibo_array(int n) {  
    vector<long long> f;  
    f.push_back(0);  
    f.push_back(1);  
    for (int i = 2; i <= n; i++) {  
        long long next_number = f[i - 1] + f[i - 2];  
        f.push_back(next_number);  
    }  
    return f[n];  
}
```

$$T(n) = 4 + 2 * (n - 1) \quad T(100) = 202$$

Ước tính thời gian chạy

$$T(n) = 4 + 3 * (n-1)$$

$$T(100) = 301$$

```
long long fib(int n)
{
    if (n < 2)
        return n;
    long long a = 0, b = 1, c;
    for (int i = 2; i <= n; i++) {
        c = a + b;
        a = b;
        b = c;
    }
    return c;
}
```


Ước chung lớn nhất

- Ước chung lớn nhất (greatest common divisor) của hai số là số lớn nhất mà cả hai số đều chia hết cho nó.
- Liên quan trực tiếp đến các lý thuyết về số học như số nguyên tố, phân số, các thuật toán mã hóa.

Yêu cầu

- Cho hai số nguyên không âm a và b .
- Tìm ước chung lớn nhất của a và b .

Ví dụ

Input	Output
3 10	1

Input	Output
20 6	2

Giải thuật 1

```
long long naive_gcd(long long a, long long b) {  
    long long best = 0;  
    for (long long i = 1; i <= a + b; i++) {  
        if (a % i == 0 && b % i == 0) {  
            best = i;  
        }  
    }  
    return best;  
}
```

Đánh giá giải thuật 1

- Đã áp dụng đúng định nghĩa toán học để tính toán.
- Chương trình có thể hoạt động và cho kết quả nhanh với các số khoảng 10^8 trở lại trong vòng dưới 1s. Tuy nhiên chương trình hoạt động rất chậm cho các số từ 10^9 trở lên.
- Nguyên nhân giải thuật tính toán chậm là do phải thực hiện vòng lặp với $a + b$ bước.
- Nếu áp dụng giải thuật này cho các số a, b có 20 chữ số, có thể máy tính phải mất hàng nghìn năm để trả về kết quả

Giải thuật 2

- Sử dụng kiến thức toán học để giải quyết yêu cầu.

Gọi a' là phần dư của phép chia a cho b , khi đó:

$$\gcd(a, b) = \gcd(a', b) = \gcd(b, a')$$

Giải thuật 2

Chứng minh

- Vì a' là phần dư của phép chia a cho b nên $a = a' + b \cdot q$
- d là ước chung của a và b khi và chỉ khi nó là ước của a' và b

(a' chia hết cho d , $b \cdot q$ chia hết cho $d \iff a' + b \cdot q$ chia hết cho d)

Mã nguồn giải thuật 2

```
long long euclid_gcd(long long a, long long b) {  
    if (b == 0) {  
        return a;  
    }  
    return euclid_gcd(b, a % b);  
}
```


Giải thuật 2

Áp dụng

$$\begin{aligned} &\text{gcd}(3918848, 1653264) \\ &= \text{gcd}(1653264, 612320) \\ &= \text{gcd}(612320, 428624) \\ &= \text{gcd}(428624, 183696) \\ &= \text{gcd}(183696, 61232) \\ &= \text{gcd}(61232, 0) = 61232. \end{aligned}$$

Đánh giá giải thuật 2

- Số bước thực hiện giảm mạnh
- Tốn khoảng $\log(ab)$ bước
- Theo tính toán tính GCD của hai số có 100 chữ số tốn khoảng 600 bước.

Kết luận: Giải thuật 1 quá chậm so với giải thuật 2

Việc chọn đúng giải thuật cho bài toán là rất quan trọng

Xem lại giải thuật tính số fibonacci

```
long long fibo_array(int n) {  
    vector<long long> f;  
    f.push_back(0);  
    f.push_back(1);  
    for (int i = 2; i <= n; i++) {  
        long long next_number = f[i - 1] + f[i - 2];  
        f.push_back(next_number);  
    }  
    return f[n];  
}
```

Xem lại giải thuật tính số fibonacci

$$T(n) = 4 + 2 * (n - 1) \quad T(100) = 202$$

Số dòng mã nguồn cần thực hiện liệu có phản ánh đúng tốc độ tính toán?

Một số bước liên quan đến xử lý vector phụ thuộc vào trình quản lý bộ nhớ của máy tính.

Các phép toán gán, so sánh, rẽ nhánh, tăng giá trị có tốc độ thực hiện có giống nhau không?

Các phép toán truy vấn phần tử, return được xử lý như thế nào?

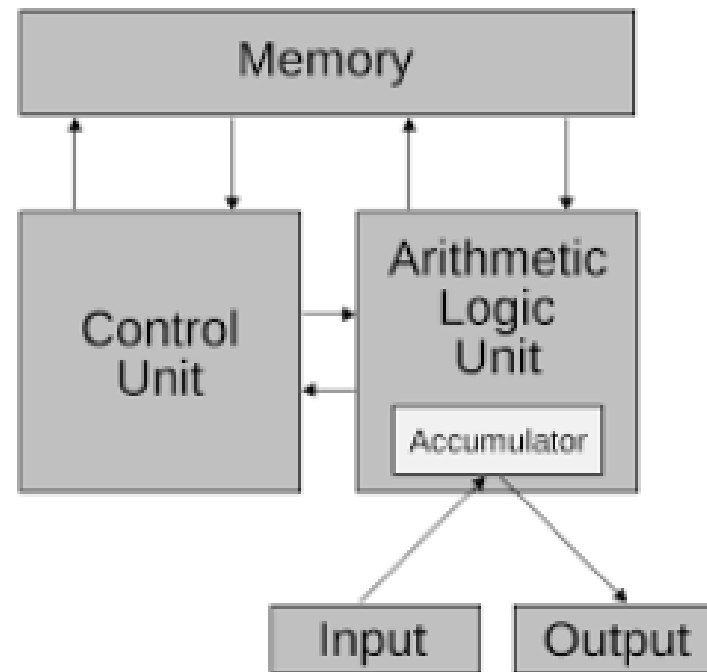
Một số yếu tố khác ảnh hưởng đến tốc độ tính toán

Tốc độ của máy tính



Một số yếu tố khác ảnh hưởng đến tốc độ tính toán

Kiến trúc của máy tính



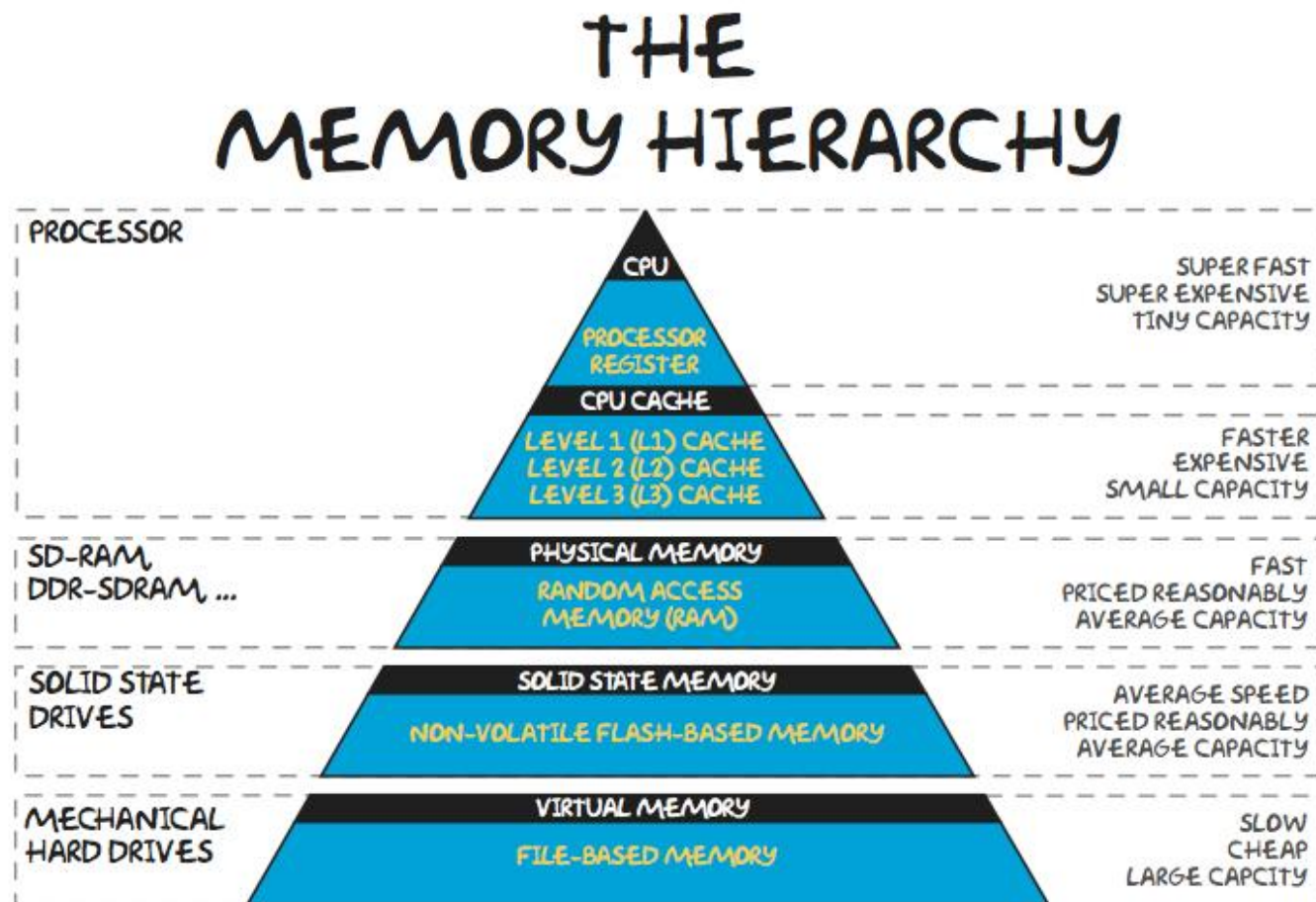
Một số yếu tố khác ảnh hưởng đến tốc độ tính toán

Trình biên dịch được sử dụng



Một số yếu tố khác ảnh hưởng đến tốc độ tính toán

Phân cấp bộ nhớ



Vấn đề

Việc tính toán thời gian chạy của chương trình là khó khăn

Đôi khi phụ thuộc vào những yếu tố ngay cả người lập trình không biết hoặc không kiểm soát được

Làm thế nào để đo được thời gian chạy mà không bị phụ thuộc những yếu tố này?

Tính toán xấp xỉ

- Việc tính toán chính xác thời gian chạy của thuật toán là khó khăn vì phụ thuộc nhiều yếu tố
- Cần tìm giải pháp để tính toán hơn, có thể không chính xác tuyệt đối nhưng lại dễ tính toán hơn mà vẫn phản ánh được tốc độ tính toán.

Tính toán xấp xỉ - Ý tưởng

- Bỏ qua các hằng số, ví dụ do cấu hình phần cứng của các máy tính khác nhau, máy A nhanh hơn máy B 100 lần thì thời gian thực hiện n phép tính hay $100n$ phép tính được coi là không đáng kể.
- Tất nhiên ý tưởng trên có thể áp dụng thực tế được nếu như thời gian hoàn thành n phép tính là tương đối nhỏ. Nếu thời gian chạy là 1s hay lớn hơn thì khác biệt giữa 100s cũng là quá lớn.

Tính toán xấp xỉ

- Để giải quyết vấn đề này, ta không quan tâm đến thời gian chạy của một input cụ thể mà chỉ quan tâm đến thời gian chạy tương đối.
- Thời gian chạy là n so với $1000n$ có vẻ chênh lệch nhưng nếu như so sánh với n^2 với input lớn thì $1000n$ vẫn cho thời gian tốt hơn n^2 .

Tính toán xấp xỉ

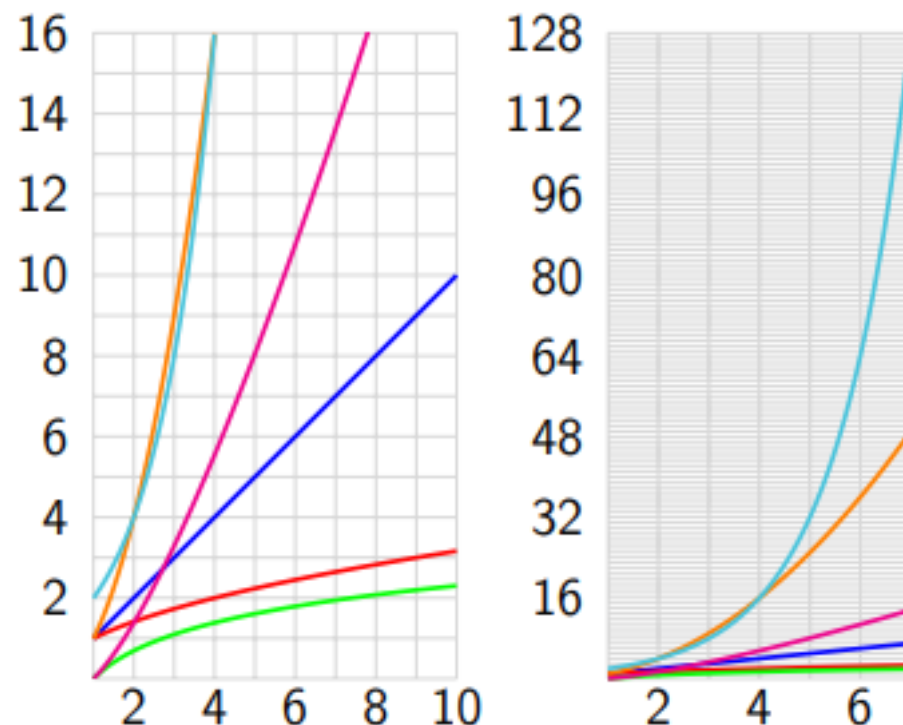
Thời gian ước tính với máy tính có tốc độ 1Ghz

	n	$n \log n$	n^2	2^n
$n = 20$	1 sec	1 sec	1 sec	1 sec
$n = 50$	1 sec	1 sec	1 sec	13 day
$n = 10^2$	1 sec	1 sec	1 sec	$4 \cdot 10^{13}$ year
$n = 10^6$	1 sec	1 sec	17 min	
$n = 10^9$	1 sec	30 sec	30 year	
max n	10^9	$10^{7.5}$	$10^{4.5}$	30

Tính toán xấp xỉ

Đồ thị các hàm số theo n

$$\log n \prec \sqrt{n} \prec n \prec n \log n \prec n^2 \prec 2^n$$



Ký hiệu Big-O

$f(n) = O(g(n))$ (f là Big-O của g) hay $f \leq g$ nếu tồn tại các hằng số N và c sao cho với mọi $n \geq N$, $f(n) \leq c \cdot g(n)$.

f được giới hạn ở trên bởi bội số không đổi của g .

Ví dụ:

$$3n^2 + 5n + 2 = O(n^2) \text{ với } n \geq 1, 3n^2 + 5n + 2 \leq 3n^2 + 5n^2 + 2n^2 = 10n^2$$

Nhận xét

- Ký hiệu BigO không cho chính xác thời gian chạy của chương trình nhưng có thể làm đơn giản quá trình đánh giá hiệu quả của thuật toán.
- Sử dụng BigO sẽ không quan tâm đến các yếu tố làm ảnh hưởng tới tốc độ tính toán giữa các máy tính khác nhau, do đó nó phù hợp để ước lượng thời gian chạy với dữ liệu đầu vào lớn

Một số quy tắc tính BigO

- Quy tắc tổng: Nếu độ phức tạp của $f_1(x)$, $f_2(x)$, ..., $f_m(x)$ lần lượt là $O(g_1(x))$, $O(g_2(x))$, ..., $O(g_m(x))$ thì độ phức tạp của $f_1(x) + f_2(x) + \dots + f_m(x)$ là $O(\max(g_1(x), g_2(x), \dots, g_m(x)))$.
- Quy tắc nhân: Nếu $f(x)$ có độ phức tạp là $O(g(x))$ thì độ phức tạp của $f^n(x)$ là $O(g^n(x))$.

Trong đó: $f^n(x) = f(x).f(x) \dots f(x)$. //n lần $f(x)$.

$g^n(x) = g(x).g(x) \dots g(x)$. //n lần $g(x)$

Ví dụ xác định BigO của giải thuật tìm số Fibonacci

```
long long fibo_array(int n) {  
    vector<long long> f;  
    f.push_back(0);  
    f.push_back(1);  
    for (int i = 2; i <= n; i++) {  
        long long next_number = f[i - 1] + f[i - 2];  
        f.push_back(next_number);  
    }  
    return f[n];  
}
```



THANK YOU FOR YOUR ATTENTION!