CS112. PHÂN TÍCH VÀ THIẾT KẾ THUẬT TOÁN

Chương 1. Tổng Quan.

# 0. Bài tập đếm các phép gán, phép so sánh:

Quy ước:

* *n*: Kích thước của dữ liệu đầu vào (có thể là số phần tử của mảng, …).
* Gán: *g(n).*
* So sánh: *ss(n).*

## 0.1. Một số công thức cần nhớ:

**Cấp số cộng**: Dãy số thoả mãn điều kiện: 2 phần tử liên tiếp nhau sai khác nhau một hằng số (công sai).

* + Số hạng thứ n:
  + Tổng của n số hạng đầu tiên:

(: phần tử đầu, : công sai).

**Cấp số nhân:** dãy số thoả mãn điều kiện: tỷ số của hai phần tử liên tiếp nhau là hằng số (công bội).

* + Số hạng thứ n:
  + Tổng các phần tử của cấp số nhân.

(: phần tử đầu, : công bội).

* **Tổng hữu hạn (Summation Formulas):**

**Faulhaber Formulas:**

Trong đó:

* chỉ số *j* có giới hạn trên là *k*;
* là các số Bernouli.
* là tổ hợp chập j của (p + 1), còn được ký hiệu là .

Đây là 7 công thức đầu tiên:

**Cận trên (Ceiling), cận dưới (Floor):**

* + Cận dưới của một số thực , được ký hiệu là ,, *,*
  + Cận trên của một số thực , được ký hiệu là,, *,*

**Logarithms:**

**Tính chất của Logarithms:** Tất cả các hệ(cơ) số của logarithm đều mặc định lớn hơn 1 trong những công thức phía dưới; lg là ký hiệu cho logarithm cơ số 2, ln là ký hiệu cho logarithm cơ số ; là các số dương bất kỳ.

### Mẹo:

**Tách phân thức:**

Ví dụ:

## 0.2. Bài tập:

### Bài tập 1:

int sum = 0; **1 g**

int i = 1; **1 g**

while(i <= n) { **(n + 1) ss**

    int j = 1; **n g**

    while(j <= n) { **(inner while + 1) ss**

sum = sum + i \* j; **(inner while) g**

j = j + 1; **(inner while) g**

    }

    i = i + 1; **n g**

}

Vòng lặp inner while = số con j, với j chạy , bước tăng là 1.

**Cứ 1 vòng lặp inner while sẽ tốn chi phí là 2n phép gán và n + 1 phép so sánh.**

* Gán:
* So sánh:
* Thời gian chạy (Running time):

### Bài tập 2:

int sum = 0; **1 g**

int i = 1; **1 g**

while(i <= n) { **(n + 1) ss**

    int j = 1; **n g**

    while(j <= n) { **(inner while + 1) ss**

        sum = sum + i \* j; **(inner while) g**

        j = j + 1; **(inner while) g**

    }

    i = i + 1; **n g**

}

* Gán:
* So sánh:

### Bài tập 3:

int sum = 0; **1 g**

int i = 1; **1 g**

while(i <= n) { **(n + 1) ss**

    int j = n - 1; **n g**

    while(j <= i) { **(inner while + 1) ss**

        sum = sum + j; **(inner while) g**

        j = j + 1; **(inner while) g**

    }

    i = i + 1; **n g**

}

Gọi là số lần lặp của inner while (xét độc lập với outer while):

số con j, với j chạy từ , bước tăng = 1.

Chỉ thoả mãn khi:

### Bài tập 4:

int i = 1, cnt = 1; **2 g**

while(i <= 3 \* n) { **(3n + 1)ss**

    int x = i - 2 \* n; **3n g**

    int y = n - i; **3n g**

    int j = 1; **3n g**

    while(j <= x) { **(inner while + 1) ss**

        cnt--; **(inner while) g**

        j += 2; **(inner while)** **g**

    }

    if(y > 0) **3n ss**

        if(x > 0) **(n – 1) ss**

            cnt++; **0 g**

    i = i + 1; **3n g**

}

Đầu tiên, ta sẽ xét 2 trường hợp if trước:

Bảng biến thiên:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  | 0 |  |  |
|  |  |  |  |  |  |  |  |

Ta thấy dòng if(y > 0) sẽ thoả mãn khi .

dòng if(y > 0) và if(x > 0) sẽ không thoả mãn.

Gọi là số lần lặp của inner while (xét độc lập với outer while):

số con j, với j chạy từ 1 , bước tăng = 2.

Thoả mãn khi

hay là khi

Vậy ta có:

* Gán:
* So sánh:

ở đây là số lần lặp của if(y > 0), nếu thoả mãn thì sẽ có trường hợp xét if(x > ).

### Bài tập 5:

int sum = 0, i = 1; **2 g**

while(i <= n) { **(2n + 1) ss**

    int j = i; **n g**

    while(j > 0) { **(inner while + 1) ss**

        sum = sum + 1; **(inner while) g**

        j = j / 2; **(inner while) g**

    }

    i = i + 1; **n g**

}

Gọi là số lần lặp của inner while (xét độc lập với outer while):

số con j, với j chạy từ *i* , bước tăng = .

Số con *k* với

Vậy có lần lặp .

* Gán:
* So sánh:

## 0.3. Bài tập nhóm:

### Bài tập 1: Tính tổng hữu hạn.

**1.1. Yêu cầu: Tính chính xác, không cho phép sai số hay xấp xỉ.**

**a.**

là một dãy cấp số cộng có công sai là 2 với số số hạng là .

Tổng 500 số hạng:

**b.** .

là một dãy cấp số nhân có công bội (r) là 2, phần tử đầu (a) là 2, n là 9.

**c.**

**d.**

**e.**

Vậy, ta có:

**f.**

**g.**

**h.**

**i.**

**j.**

**1.2. Yêu cầu: Tính chính xác được thì càng tốt, không thì cho phép tính gần đúng/xấp xỉ.**

**a.**

Ta có:

**b.**

**c.**

**d.**

### Bài tập 2:

A screenshot of a computer

Description automatically generated

Gọi là số lần lặp của vòng lặp while trong. (Xét độc lập với vòng lặp while ngoài).

số con với chạy từ , bước tăng là 1.

Tổng số phép gán:

Tổng số phép so sánh:

### Bài tập 3:

A computer screen shot of a program

Description automatically generated

Gọi là số lần lặp của vòng lặp while trong. (Xét độc lập với vòng lặp while ngoài).

số con j với j chạy từ , bước tăng là .

Số phép gán:

Số phép so sánh:

### Bài tập 4:

A white background with black text

Description automatically generated

Gọi là số lần lặp của vòng lặp while trong thứ 1. (Xét độc lập với vòng lặp while ngoài và các vòng lặp while khác).

Gọi là số lần lặp vòng lặp while trong thứ 2. (Xét độc lập với vòng lặp while ngoài và các vòng lặp while khác).

* Xét :

= số con , với chạy từ , bước tăng là 2.

Vòng lặp while trong thứ 1 được thực hiện khi:

* Xét :

= số con , với chạy từ , bước giảm là .

Số phép gán:

Số phép so sánh:

### Bài tập 5:

    i = 1; **1 g**

    res = 0; **1 g**

    while i<=n do **(n + 1) ss**

        j = 1; **n g**

        k = 1; **n g**

        while j <= i do **(inner while + 1) ss**

            res = res + i \* j; **(inner while) g**

            k = k + 2; **(inner while) g**

            j = j + k; **(inner while) g**

        endw

        i = i + 1; **n g**

    endw

Gọi là số lần lặp của vòng lặp while trong (xét độc lập với vòng lặp while ngoài):

số con j, với j chạy từ , bước tăng = .

Ta thấy với việc khởi tạo j = 1, k = 1 và mỗi vòng lặp while trong, k tăng thêm 2 và j tăng thêm k, nghĩa là cứ qua mỗi vòng lặp j sẽ tăng thêm số lượng là số lẻ kế tiếp của k, ta có như sau:

Hay

Chỉ thoản mãn khi:

Vậy

Số phép gán:

Số phép so sánh:

### Bài tập 6:

### Bài tập 7:

### Bài tập 8:

### Bài tập 9:

    sum = 0; i = 1; idx = 1; **3 g**

    while(i<=n) **(n + 1) ss**

    {   j = 1; **n g**

        while(j<=n) **(n + 1) ss**

        {       if( (i == j) && (i + j == n + 1) **n ss**

                    idx = i  **g**

                sum = sum + a[i][j]; **n g**

                j++; **n g**

        }

        i++; **n g**

    }

    if(idx != -1) **1 ss**

        sum = sum - a[idx][idx];  **g**

Gọi là số lần lặp của vòng lặp while trong (xét độc lập với vòng lặp while ngoài):

số con j, với j chạy từ , bước tăng là 1.

Vậy vòng lặp bên trong sẽ lặp *n* lần.

* Xét biểu thức so sánh (xét độc lặp với vòng lặp while ngoài cùng).
* Để biểu thức so sánh được thực hiện thì phải đúng.
* Xét biểu thức , biểu thức này sẽ có số lần thực hiện bằng với số lần lặp của vòng lặp while trong, ta thấy khi chạy từ , luôn tồn tại 1 và duy nhất 1 lần . Vì vậy biểu thức so sánh sẽ được lặp *n* lần và luôn đúng 1 lần.
* Xét biểu thức , biểu thức so sánh này chỉ được thực hiện khi đúng, đồng thời khi, để thì n phải là số **lẻ**. Vì vậy, biểu thức so sánh sẽ được lặp 1 lần và chỉ đúng khi .

Với mỗi vòng lặp while trong biểu thức so sánh được thực hiện n lần, biểu thức so sánh là 1 lần.

* Phép gán chỉ được thực hiện khi n là số **lẻ**.
* Gọi là số phép gán thực hiện dựa trên biểu thức so sánh . Ta có:
* Xét biểu thức so sánh :
  + Điều kiện chỉ đúng khi câu lệnh được thực hiện (hay ). Gọi là số phép gán . Ta có:

Vậy, ta có các kết quả như sau:

Số phép so sánh:

Số phép gán:

### Bài tập 10:

### Bài tập 11:

Chương 2. Phân tích thuật toán.

# 1. Phân tích thuật toán.

## 1.1. Thuật toán là gì?

Theo slide của cô: “Thuật toán là một dãy hữu hạn các bước không mập mờ và có thể thực thi được, quá trình hành động theo các bước này phải dừng và cho kết quả mong muốn.”

*“Two ideas lie gleaming on the jeweler’s velvet. The first is the calculus, the second, the algorithm. The calculus and the rich body of mathematical analysis to which it gave rise made modern science possible; but it has been the algorithm that has made possible the modern world.”*

—David Berlinski, *The Advent of the Algorithm,* 2000

Tạm dịch:

“Hai viên ngọc ý tưởng lung linh đang nằm trên những tấm nhung lụa của thợ kim hoàn. Viên thứ nhất là Calculus (Giải tích), và viên thứ nhì là Algorithm (thuật toán). Giải tích cùng với nền tảng phân tích toàn học phong phú mà nó tạo ra đã khiến cho nền khoa học hiện đại trở nên khả thi; nhưng chính Thuật toán mới là thứ tạo nên thế giới hiện đại.”

—David Berlinski, *The Advent of the Algorithm,* 2000

Mặc dù không có nhiều từ ngữ được công nhận rộng rãi để nói về khái niệm này, nhưng tổng quát người ta đồng ý với cách hiểu như sau:

Thuật toán là một “lời chỉ dẫn” theo từng bước để giải một bài toán hoặc một vấn đề nào đó, để nhận được một output theo yêu cầu với mọi input hợp lệ trong một khoảng thời gian hữu hạn.

Ví dụ để chiên một cái trứng thì ta sẽ có một thuật toán cơ bản như sau:

1. Tìm xem trong tủ lạnh còn trứng không?

2. Nếu có thì tới bước 3.

2.1 Nếu không có thì có đi mua một vỉ trứng mới không?

2.1.2. Nếu có thì tới bước 3.

2.1.2. Nếu không thì thoát.

3. Đập quả trứng ra…

Một định nghĩa nghe có vẻ hơi “trang trọng” quá của thuật toán:

**An algorithm is the step-by-step unambiguous instructions to solve a given problem**

* Data Structure and Algorithms Made Easy.

**“**Instruction” hay “lời chỉ dẫn” mà được đề cập ở trên ngụ ý là có một thứ gì đó, hoặc một ai đó có khả năng hiểu và thực hiện được theo hướng dẫn. Chúng ta gọi đây là một “computer”, nhớ rằng trước khi máy tính điện tử được phát minh, từ “computer” được dùng để chỉ một con người có khả năng tính toán số học. Sau này, với sự bùng nổ về độ phổ biến thì nó mới dần chuyển nghĩa sang thành một cỗ máy. Lưu ý rằng, cho dù hiện nay đa số thuật toán được thiết kế cho việc triển khai bằng máy tính, tuy nhiên định nghĩa của thuật toán không phụ thuộc vào yếu tố nói trên.

A diagram of a problem

Description automatically generated

Để xác định thuật toán “thật sự” cần có các yếu tố như sau:

* Tính đúng đắn (Correctness).
* Tính hữu hạn/có điểm dừng (Complete).
* Tính xác định, tường minh (Explicit).

## 1.2. Phân tích thuật toán.

**1. Tại sao phải phân tích thuật toán?**

* Để có thể so sánh các thuật toán với nhau, từ đó chọn thuật toán phù hợp nhất.
* Biết được điểm mạnh, điểm yếu của các thuật toán. Để từ đó có thể cải tiến các thuật toán.
  + Ví dụ: Heap Sort là thuật toán cải tiến từ Selection Sort. Shell Sort là thuật toán cải tiến từ Insertion Sort. Interpolation Sort là thuật toán cải tiến từ Binary Search.

**2. Vậy thì ta sẽ phân tích những gì trong thuật toán ?**

Ta sẽ phân tích các yếu tố cần thiết của thuật toán:

* Tính đúng đắn/chính xác (Correctness).
* Tính hữu hạn/dừng (Complete).
* Tính xác định, tường minh (Explicit).
* Tính hiệu quả: không gian và thời gian (Time/Space Complexity).
* Tính đơn giản (Simplicity).
* Tính khách quan.
* Tính tổng quát/phổ dùng (Generality).

Trong số những yếu tố kể trên, độ hiệu quả thời gian – Time Efficency – là thứ được đánh giá cao hơn cả và là yếu tố ta nên tập trung vào.

**3. Vậy ta sẽ phân tích thuật toán bằng cách nào?**

**Measuring Running Time:** là công việc đo lường **thời gian thực hiện của thuật toán** để so sánh tương đối các thuật toán với nhau.

Có 2 phương pháp đánh giá tính hiệu quả về th, **thực nghiệm (Empirical Analysis)** và **dùng toán học**(Mathematical Analysis).

Đầu tiên là thực nghiệm:

* Đây là một cách khá “thực tế”, nghĩa là, chúng ta sẽ so sánh thời gian chạy của 2 thuật toán bằng máy tính. Tuy nhiên, cách này có nhiều vấn đề do phụ thuộc nhiều yếu tố khách quan:
  + - Cấu hình của máy tính.
    - Compiler/Interpreter.
    - Programming Language.
    - Code skill.
* Các bước thực hiện:
  + - Viết chương trình, cài đặt.
    - Chọn các bộ dữ liệu thực nghiệm.
    - Thực thi chương trình với nhiều bộ dữ liệu đã chọn.
    - Đo và thống kê thời gian, thông số.
    - Xấp xỉ biểu đồ.
* Đơn vị đo đạc: giây, phút, giờ.
* Ưu điểm: Dễ thực hiện.
* Nhược điểm:
  + - Cách này lại khá hữu dụng cho những người không giỏi “toán” (gà). Ta có thể sử dụng cách này với các điều kiện là những yếu tố kể trên là giống nhau đối với các thuật toán được dùng để phân tích
    - Và kết quả phân tích được của kết quả này cũng là kết quả “cục bộ”, hay nói cách khác, nếu công bố rộng rãi ra thì có thể sẽ không được công nhận.

Tiếp theo là cách phân tích dùng toán học:

* Đơn vị đo đạc: Số phép toán. Ký hiệu là .
* Ai giỏi toán thì làm, tại đỉnh 😉.
* Đánh giá tính hiệu quả về mặt thời gian của thuật toán bằng phương pháp toán học = Đếm tất cả số phép toán hoặc số phép toán cơ bản được thực hiện trong thuật toán đó.
  + - Phép toán cơ bản (Basic operation) là phép toán quan trọng nhất, “tích cực nhất”, tốn nhiều chi phí thực hiện nhất hoặc có thể hiểu là phép toán mà số lần thực hiện không ít hơn các phép toán khác.

# 2. Phân tích thuật toán đệ quy.

**Cách 1:**

* Thành lập phương trình đệ quy.
* Giải phương trình đệ quy
  + Thời gian thực hiện chương trình = nghiệm của phương trình đệ quy.

**Cách 2 (gà thì làm):**

* Khử đệ quy, phân tích Thuật toán không đệ quy.

## 2.1. Recurrence Relation. (Phương trình đệ quy)

Phương trình đệ quy tổng quát:

Trong đó:

* C(n): Thời gian thực hiện chương trình trong trường hợp dừng.
* g(T(k)): đa thức của T(k).
* f(n): là thời gian phân chia/kết hợp của các kết quả.

Ví dụ:

Cứ mỗi 1 giờ số lượng vi trùng sẽ nhân đôi, hỏi sau *n* giờ thì số lượng vi trùng là bao nhiêu? Biết lúc đầu có 2 vi trùng.

long long vitrung(int n) {

    if(n == 0) *return* 2;

*return* vitrung(n - 1) \* 2;

}

Reccurence relation:

## 2.2. Backward Substitution. (Phương pháp Truy hồi/Thay thế)

* Dùng đệ quy để **thay thế** T() () vào phía phải của phương trình.
* Lặp lại cho đến khi tất cả T() được thay thế bởi các biểu thức T() hoặc T(0).
* T(1) và T(0) luôn là các hằng số công thức T() chứa các số hạng chỉ liên quan tới và hằng số.

**Ví dụ 1:**

Nhớ phải có phần giải thích ở đây:

Gợi ý giải thích:

* Khi , phương trình đệ quy gặp điều kiện dừng, và do chi phí của các phép toán trong trường hợp này là không đáng kể, ta sẽ đặt chi phí đó là một hằng số .
* Khi , ta thấy có 3 lời gọi hàm đệ quy với kích thước bài toán là . Ngoài ra, còn có một số phép toán khác lời gọi đệ quy, như: phép cộng, … nhưng những chi phí của các phép toán nói trên là không đáng kể, do đó ta đặt chi phí kể trên là một hằng số .

**Ví dụ 2:**

Nhớ phải có phần giải thích ở đây:

Gợi ý giải thích:

* Khi n = 0, phương trình đệ quy gặp điều kiện dừng, và do chi phí của các phép toán trong trường hợp này là không đáng kể, ta sẽ đặt chi phí đó là một hằng số .
* Khi n > 0, ta thấy có 3 lời gọi hàm đệ quy với kích thước bài toán là . Ngoài ra, còn có một số phép toán khác lời gọi đệ quy, như: phép cộng, … nhưng những chi phí của các phép toán nói trên là không đáng kể, do đó ta đặt chi phí kể trên là một hằng số . Phép toán kể trên được đặt trong các vòng lặp và sẽ lặp số lần lặp là xấp xỉ .

**Ví dụ 3:**

Quá trình dừng lại khi   
**Ví dụ 4:**

Quá trình dừng lại khi tiến tới

Sử dụng công thức “Tổng của N cấp số nhân đầu tiên”:

### Bài tập lấy điểm:

**Yêu cầu:** Giải bằng Phương pháp Truy hồi/Thay thế (Backward Substitution).

**Bài tập 1:**

Quá trình dừng lại khi tiến tới :

**Bài tập 2:**

Quá trình dừng lại khi tiến tới :

## **2.3. Characteristic Equation. (Phương trình đặc trưng)**

Xét phương trình dạng:

Đặt , đưa về dạng phương trình ẩn :

**TH1:**  có nghiệm đơn, gọi các nghiệm đơn đó là thì

, với c là các hằng

**TH2:**  có nghiệm bội thì

Ví dụ:

**Ví dụ:**

**Giải:**

Ta có:

Với

## 2.4. Guessing and Induction. (Phương pháp Đoán nghiệm)

Phương pháp được sử dụng thông dụng nhất.

Các bước thực hiện như sau:

1. **Guess** the form of the solution (**Đoán** dạng của lời giải).
2. **Verify** by induction (**Chứng** **minh** bằng quy nạp) .
3. **Solve** for constants (**Giải** các hằng số).

Ta đoán 1 nghiệm và dùng **chứng minh quy nạp** để chứng tỏ rằng:, với mọi .

thường là một trong các hàm quen thuộc như .

Đôi khi ta chỉ đoán dạng của trong đó có vài tham số chưa xác định và trong quá trình quy nạp, ta sẽ tìm ra giá trị thích hợp cho các tham số.   
**Ví dụ:**

**Đoán** .

**Bước 1:**

Chứng minh: *.*

Nếu chọn thì ta có đccm ()

**Bước 2:**

Giả sử

**Bước 3:**

Chứng minh tại

**Bước 4:**

Ta có

Nếu thì

Chọn

Ta có:

**Ví dụ:**

Thuật toán MergeSort.

**Đoán**

**Bước 1:**

Chứng minh: .

Để thì phải chọn

**Bước 2:**

Giả sử

**Bước 3:**

Chứng minh tại

**Bước 4:** Chọn a, b sao cho

(Chứng minh lại:

)

Ta có:

## 2.5. Generating Function. (Phương pháp Hàm sinh)

### 2.5.1. Định nghĩa:

Hàm sinh của dãy vô hạn là tổng hình thức:

Hàm sinh có dạng biểu diễn là 1 **chuỗi luỹ thừa**.

Ký hiệu:

### 2.5.2. Một số hàm sinh cơ bản và dãy số tương ứng:

Ví dụ:

**Giai đoạn 1:**

Hàm sinh của dãy vô hạn là:

Đặt , .

Khử

**Giai đoạn 2 (Cố gắng đưa về các hàm sinh đáng nhớ):**

Nháp (tiến hành đồng nhất thức):

### Bài tập lấy điểm:

**Giai đoạn 1:**

Hàm sinh của dạy vô hạn là:

Đặt , .

Khử

**Giai đoạn 2:**

# 3. Độ phức tạp.

## 3.1. So sánh 2 thuật toán.

* **Cách tốt nhất**: xác định chính xác thời gian thực hiện (the exact running time) của 2 giải thuật rồi so sánh chúng.
* **Thực tế**:
  + Rất khó để tính chính xác.
  + Nếu tính được rồi thì có thể gặp khó khăn khác khi so sánh 2 hàm số với nhau.

A graph with lines and dots

Description automatically generated

Ví dụ khó hơn thì sao?

**Mục tiêu mới:**

1. So sánh tương đối: hàm không sai biệt nhiều thì xem như xấp xỉ nhau về độ lớn.
2. Chỉ quan tâm đến những giá trị *n* đủ lớn lớn như thế nào ?
3. Xét *n* tiến tới .

Ví dụ:

So sánh “độ lớn” của và bằng cách nào?

Độ lớn của là cỡ khoảng độ lớn của là cỡ khoảng .

* Khi *n* càng lớn (tiến tới ) thì hàm nào sẽ lớn hơn ?

A diagram of a function

Description automatically generated

* “Nếu tăng nhanh hơn thì luôn lớn hơn ở ”.

Nhưng làm sao để biết hàm số nào tăng nhanh hơn?

* Tốc độ tăng trưởng được chia thành các cấp độ từ nhỏ đến lớn: tăng tuyến tính, tăng bậc 2, tăng phi mã, …
* Dựa vào “order of growth” ( cấp độ tăng = bậc tăng trưởng).
* Hàm có bậc tăng trưởng lớn hơn hàm đó tăng nhanh hơn sẽ lớn hơn với các giá trị *n* đủ lớn.
* Những hàm cùng bậc tăng trưởng cùng độ lớn.
  + Ví dụ: có bậc tăng trưởng là *n*.

## 3.2. Asymptotic Notation.

Ký hiệu/ký pháp tiệm:

* Cho ta 1 phương pháp để phân loại các hàm “chặt chẽ hơn” dựa theo **bậc tăng trưởng** của chúng.

## 3.3. Độ phức tạp:

Ý nghĩa chung:

* Là 1 phương tiện, công cụ giúp ta đánh giá tính hiệu quả về mặt thời gian của thuật toán.
* Giúp phân lớp “cấp độ lớn” hay “bậc tăng trưởng” của hàm khi *n* đủ lớn. (Nhớ có mấy chữ như: gom nhóm, gom lớp, phân nhóm, phân lớp).
* Là thuật ngữ giúp chúng ta nói về các ký hiệu tiệm cận.
* (Giải thích về ít nhất 1 ký hiệu tiệm cận ).
* Độ phức tạp của giải thuật: được xác định thông qua các ký hiệu tiệm cận.có 3 loại độ phức tạp phổ biến*.*

### 1. Big – O.

Khi tăng, tăng không nhanh hơn là chặn trên (asymptotic upper bound) của

A graph of a person with a curve

Description automatically generated with medium confidence

Định nghĩa (phân tích giải thuật):

* nếu và chỉ nếu tồn tại hằng số dương và sao cho:

**Ta nói:**  có bậc tăng trưởng là có độ phức tạp là của .

Ký hiệu: **.** Bản chất: **.**

**Chú ý:**

* Dấu = chỉ là ký hiệu hình thức.
* là tập hợp.

**Lưu ý:**

* Có thể đánh giá độ phức tạp bằng ký hiệu tiệm cận khác như nên kèm theo ký hiệu khi nói đến độ phức tạp. VD: “Giải thuật có độ phức tạp ”
* chỉ là 1 hàm chặn trên của , vẫn có thể có cách ước lượng **chặt hơn**.
* Luôn tìm được và cần tìm nhỏ nhất có thể.

**Ví dụ:**

Ta thấy: với .

* . Hỏi dùng ký hiệu gì thay cho ? là đúng. Trả lời, ký hiệu .

**Ví dụ 1:** Xét Tìm để ?

Ta có:

, sao cho:

Chọn , theo định nghĩa của Big – O, ta có điều cần chứng minh,

### 2. Big – .

Khi tăng, tăng không chậm hơn là chặn dưới (asymptotic lower bound) của A graph of a graph

Description automatically generated with medium confidence

Định nghĩa (phân tích giải thuật):

* nếu và chỉ nếu tồn tại hằng số dương và sao cho:

**Tính chất:**

Cho , ta có:

### 3. Big - .

Khi tăng, tăng không nhanh, cũng không chậm hơn là cả chặn trên và chặn dưới (asymptotic upper and lower bound) của A graph of a function

Description automatically generated

Định nghĩa (phân tích giải thuật):

* nếu và chỉ nếu tồn tại hằng số dương và sao cho:

nếu và chỉ nếu và g

### 4. Ví dụ.

Chứng minh:

, sao cho:

Chọn , theo định nghĩa của :

Chứng minh:

Ta sẽ sử dụng phương pháp **phản chứng** để giải quyết bài này.

Giả sử .

Suy ra sao cho:

Dưới đây là phần giải thích:

TH1:

c





Trường hợp này mâu thuẫn giả thiết.

TH2:







Trường hợp này cũng mâu thuẫn giả thiết.

Hết phần giải thích:

Giả sử ban đầu là sai Giả thiết ban đầu là đúng.

Chứng minh:

Ta có:

Xét hàm bất kỳ, sao cho .

Xét hàm bất kỳ, sao cho .

Từ và, ta có:

### 5. Bài tập lấy điểm.

a) Phép suy ra bên dưới là đúng hay sai và vì sao?

Ta có thể hiểu, dấu “=” ở hai biểu thức đầu tiên không thực sự là dấu “đẳng thức”, mà là dấu “”. Do đó hai biểu thức đầu tiên được hiểu như sau:

Còn dấu “=” ở phần kết luận là dấu đẳng thức, hai phần tử của một tập hợp không chắc chắn là có thể bằng nhau. Do đó phép suy lận trên là sai.

b) Xét

Chứng minh (dùng định nghĩa, không dùng lim):

Ta có:

Chọn , ta có:

Vậy , theo định nghĩa của Big – O, ta có: (đccm).

Ta có:

Chọn , ta có:

Vậy, theo định nghĩa của Big – O, ta có: (đccm).

Chứng minh bằng phản chứng:

Giả sử

, sao cho:

Dưới đây là phần giải thích:

TH1:

*7c*





Trường hợp này mâu thuẫn giả thiết, do .

TH2:







Trường hợp này cũng mâu thuẫn giả thiết, do .

Do đó, mâu thuẫn vậy giả thiết ban đầu sai.

Ta có:

Ta chỉ cần chỉ ra sao cho:

Thử với .

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | *n* |  | 640 |  |  |
|  |  | 0 |  | 0 |  |  |

Chọn

c) Nếu thì .

Ta có

, sao cho:

Ta có

, sao cho:

Vậy:

, ta có:

Vậy, theo định nghĩa của big O, ta có: “Nếu thì ”

### 6. Câu hỏi ôn tập:

**Câu 1:**

Với mỗi nhóm hàm bên dưới, hãy sắp xếp các hàm số theo thứ tự tăng dần của “order of growth”. Ký hiệu log là log cơ số 2, là tổ hợp chập k của n.

**Group 1:**

**Group 2:**

Cách làm cơ bản:

* Xác định Big – O của hàm số .
* Khi xác định chặn trên thì đó phải là chặn trên nhỏ nhất, khi xác định chặn dưới thì đó phải là chặn dưới lớn nhất.

Chú ý:

* với *c* là một số rất nhỏ, có thể là 0.00000000000000000001 > 0
* Với *a* là hằng số:
* Khi làm việc với các số rất lớn (hàm mũ, …) thì ta sẽ chuyển các hàm số về dạng có cơ số chung, quen nhất là cơ số 2.

**Group 1:**

Với c rất bé.

**Group 2:**

Với c rất bé.

**Câu 2:**

1. Chứng minh: *C* là hằng số.
   * Chứng minh

Xét 1 hàm bất kì

, sao cho:

, sao cho:

* + Chứng minh
* Xét 1 hàm bất kì

, sao cho:

, sao cho:

Từ (1) và (2), ta có điều cần chứng minh.

Ta cần tìm 1 con sao cho:

* + Thử chọn :
  + Thử chọn :

Vậy, sao cho:

1. Hãy sắp xếp 07 hàm số bên dưới theo thứ tự tăng dần của “order of growth”, và có giải thích ngắn gọn cách thực hiện. Ký hiệu: log là log cơ số 2.

Ta sẽ chia ra làm 2 nhóm để xử lý:

Group 1:

Group 2:

Từ (1) và (2), ta có:

## 3.4. Chiến lược cho Big – O.

* Đôi khi cách đơn giản nhất để chứng minh là **chọn c là tổng của các hệ số dương** của Thường lờ đi các hệ số âm.

Ví dụ:

Để chứng minh

Chọn

Khi đó, nếu

Cách làm trên không phải luôn dễ dàng.

Ví dụ: chứng minh.

* + Chỉ quan tâm đến các luỹ thừa cao nhất của . Bỏ qua các hằng số.

**Note:** Vai trò của hằng số trong phân tích:

**Lý thuyết**: do khá lớn nên không đáng kể, không tạo ra sự khác biệt trọng yếu nên có thể bỏ qua.

**Thực nghiệm**: Đôi khi rất quan trọng cẩn thận.

## 3.5. Các tính chất.

### 3.5.1. Big – O notation (upper bounds).

* + với là hằng số
  + .
  + và

### 3.5.2. Big - notation (lower bounds).

### 3.5.3. Big - notation (tight bounds).

Ví dụ:

## 3.6. Chiến lược cho Big -

Chứng minh

* Thường chọn .
* Chiến lược tốt nhất là thử chọn c trước rồi xác định .

Chứng minh

* Dùng định nghĩa để tìm .
* nếu và chỉ nếu và .

## 3.7. o – notation and – notation

for any constant , there is a constant such that

Ví dụ:

for any constant , there is a constant such that

Ví dụ:

## 3.8. Các lớp độ phức tạp.

|  |  |
| --- | --- |
| Dạng O | Tên phân loại |
|  | Hằng |
|  | Logarithmic |
|  | Căn thức |
|  |
|  |
|  |
|  | Tuyến tính | Đa thức |
|  | Bình phương |
|  | Bậc ba |
|  |  |
|  | Đa thức |
| , với | Mũ | Độ phức tạp lớn |
|  | Giai thừa |

**Common Rates of Growth.**

* **Constant:** , for example .
* **Linear:** .
* **Logarithm**: .
* :().
* **Quadratic:**
* **Polynomial**: .
* **Exponential:** .

**So sánh:**

A diagram of different colors

Description automatically generated

A table of numbers with a number on it

Description automatically generated with medium confidence

## 3.9. Sử dụng các giới hạn.

Nếu

nghĩa là và (hoặc )

Nếu

Nếu

Nghĩa là .

**Tóm tắt:**

Thuật toán có độ phức tạp có thể không mất nhiều thời gian như vậy có thể thật sự thời gian chạy là tuyến tính

Thuật toán có thể thực sự là .

Thuật toán thì chính xác thời gian chạy là bậc 2.

**Limits:**

## 3.10. Master Theorem.

### 3.10.1. Master Theorem (1): Dạng đơn giản.

Định lý Master cho phép ước lượng nghiệm của các phương trình đệ quy có dạng (đơn giản):

Ví dụ:

,

Vì TH1 được áp dụng.

**Nhược điểm:**

* Chỉ áp dụng cho phương trình đệ quy có dạng như trên.
* Không thể dùng định lý Master (dạng 1) nếu:
  + không đơn điệu. Ví dụ:
  + không phải hàm đa thức. Ví dụ:
  + không thể biểu diễn như một hằng số.

### 3.10.2. Master Theorem (2): Dạng tổng quát.

The Mastery Theorem applies to recurrence of the following form (**general – tổng quát hơn 1**):

where and are constants,

is an asymptotically positive function

Có 3 trường hợp:

1. If for some constant , then

2. If with , then

3. If with >0, and f(n) satisfies the regularity condition, then

Regularity condition:

for some constant and all sufficiently large .

Ví dụ:

và

Ta thấy,

và

Vì ()

ta được:

# 4. Thiết kế thuật toán.

Q&A:

* Mục tiêu của lập trình: Xây dựng chương trình máy tính để **giải quyết vấn đề** thay cho con người.
* Giải quyết vấn đề: tìm một giải pháp (solution).
* Diễn đạt bằng cách nào:
  + Con người: bằng ngôn ngữ tự nhiên.
  + Máy tính: bằng thuật toán.

## 4.1. Problem.

Problem (nói chung, cách hiểu của con người): a matter of situation regarded as unwelcome or harmful and needing to be dealt with and overcome.

2 Tình huống thông dụng của problem:

* A thing that is difficult to achieve.
* An inquiry starting from given conditions to investigate or demonstrate a fact, result, or law.

A colorful staircase with arrows pointing up

Description automatically generated

**Giai đoạn trọng yếu:**

Xác định vấn đề và biểu diễn (mô hình hoá) vấn đề:

* Mục đích (kết quả): mô hình của vấn đề.
* Xây dựng mô hình vấn đề xong mới nói tới giải pháp.
* Trước khi mô hình hoá phải hiểu bài toán thực tế và phát biểu ở góc độ tự nhiên.
* Trả lời rõ ràng các câu hỏi: What ? Why ? How ? … trên cơ sở đó xây dựng mô hình cho vấn đề.

## 4.2. Define a problem.

Phải hiểu cặn kẽ bài toán này, bằng cách trả lời rõ ràng các câu hỏi:

1. Thông tin/dữ liệu của bài toán bao gồm những gì?
2. Yêu cầu xử lý ra sao, nghĩa là ta phải làm gì?
3. Có điều kiện ràng buộc gì hay không?
4. **Các mẫu ví dụ và đáp án.**

Cần phát biểu rõ ràng để thiết kế giải thuật trên máy tính:

1. Tình huống, ngữ cảnh, cơ sở dữ liệu – thông tin – tri thức của vấn đề (Base).
2. Giả thiết (Input).
3. Mục tiêu, yêu cầu (Output).
4. Các điều kiện, ràng buộc, phạm vi.

### 4.2.1. Mô hình hoá (biểu diễn vấn đề).

Vấn đề phải được đặt tả hay mô hình hoá dựa trên **công cụ toán học** hay các **ngôn ngữ đặc tả** (hình thức) cho máy tính.

Mô hình hoá cho từng thành phần và tổng hợp lại mô hình cho vấn đề.

**Ví dụ 1: Giếng ma thuật.**

**Mô tả bài toán:**

* Bạn đang đứng trước một cái giếng ma thuật, trên đó có ghi 2 số nguyên dương *a* và *b*.
* Mỗi lần ném một viên sỏi xuống giếng, nó sẽ trả về cho bạn  đồng tiền vàng, sau đó *a* và *b* sẽ tăng lên 1.
* Vậy nếu bạn có n viên sỏi thì sẽ kiếm được bao nhiêu đồng tiền vàng?

**Input:**

* Hai số nguyên dương a và b ghi trên giếng với .
* Một số nguyên không âm n với là số viên sỏi mà bạn có được.

**Output:**

* Một số nguyên cho biết số đồng tiền vàng kiếm được.

**Ví dụ:** Cho và , output là 8 đồng.

\*\* Nên đặt điều kiện ràng buộc cho các biến \*\*

**Ví dụ 2: Knapsack Problem.**

**Mô tả bài toán:**

Một kẻ trộm đột nhập vào nhà, hắn tìm thấy đồ vật có trọng lượng và giá trị khác nhau. Nhưng hắn chỉ mang theo 1 cái balo/túi có sức chứa về trọng lượng tối đa là . Kẻ trộm cố bỏ các đồ vật vào túi sao cho đạt 1 giá trị cao nhất khi mang đi.

**Input:**

* Một số nguyên với là số đồ vật có trong ngôi nhà.
* Mảng gồm có số thực, là giá trị của đồ vật thứ .
* Mảng gồm có số thực, là trọng lượng của đồ vật thứ , .
* Số thực cho biết sức chứa về trọng lượng tối đa của balo, .

\*\* Nên đặt điều kiện ràng buộc cho các biến \*\*

**Output:**

* Một số thực cho biết tổng giá trị của balo.
* Một phương án bỏ đồ vật vào trong túi.
* Gọi là số lượng đồ vật thứ i được chọn bỏ vào trong túi, thì đồ vật thứ không được chọn, nghĩa là đồ vật thứ i được chọn bỏ vào balo.
* Một phương án sẽ được biểu diễn như sau:

Mảng

Điều kiện ràng buộc của output:

**phương án tối ưu = phương án chấp nhận được + tổng giá trị của balo là lớn nhất.**

và

### 4.2.2. Bài tập trên lớp.

**Bài 1: Bài toán sản xuất (Production planning problem).**

Công ty X sản xuất sơn nội thất và sơn ngoài trời. Nguyên liệu gồm 2 loại A và B với trữ lượng là 6 tấn và 8 tấn tương ứng. Để sản xuất 1 tấn sơn nội thất cần 2 tấn nguyên liệu A và 1 tấn nguyên liệu B. Hai số tương ứng của sơn ngoài trời là 1 tấn và 2 tấn. Qua tiếp thị được biết thì nhu cầu thị trường là như sau: (cho 1 ngày):

* Nhu cầu sơn nội thất không lớn hơn nhu cầu sơn ngoài trời quá 1 tấn.
* Nhu cầu cực đại của sơn nội thất là 2 tấn.
* Giá bán sỉ là 2000 USD 1 tấn sơn nội thất và 3000 USD tấn sơn ngoài trời. Vấn đề là cần sản xuất mỗi ngày như thế nào để doanh thu là lớn nhất.

**Input:**

Các số nguyên dương lần lượt là trữ lượng của loại nguyên liệu A và B.

Các số nguyên dương lần lượt là giá bán sỉ của sơn ngoại thất và sơn ngoài trời.

**Output:**

* 1 Phương án sản xuất trong ngày có dạng ( là số lượng sơn nội thất (tấn )phải sản xuất trong 1 ngày, là số lượng sơn ngoài trời (tấn) phải sản xuất trong 1 ngày).
* **Hàm mục tiêu:**
* **Ràng buộc**

**Bài 2: Bài toán vận tải (Transportation problem):**

Hàng hoá được vận chuyển từ kho đến cửa hiệu bán lẻ. Lượng hàng ở kho là (tấn), và cửa hiệu có nhu cầu (tấn. Cước vận chuyển 1 tấn hàng từ kho đến cửa hiệu là đồng. Giả sử tổng hàng ở các kho và tổng nhu cầu bằng nhau. Bài toán đặt ra là lập kế hoạch vận chuyển để tiền cước là nhỏ nhất, với điều kiện là mỗi cửa hàng đều nhận đủ và mỗi kho hàng đều trao hết hàng.

**Input**:

Một số nguyên dương thể hiện số lượng kho.

Một số nguyên dương n thể hiện số lượng của hiệu bán lẻ.

Một mảng số nguyên không âm cho biết số lượng hàng ở kho thứ (tấn)

Một mảng số nguyên không âm cho biết nhu cầu ở cửa hiệu bán lẻ thứ (tấn).

Một ma trận số nguyên không âm cho biết số tiền cước vận chuyển ở kho thứ đến cửa hiệu thứ (đồng).

**Output**:

* Tiền cước vận chuyển.
* 1 phương án vận chuyển tối ưu:

Gọi là lượng hàng vận chuyển từ kho đến cửa hiệu (tấn).

1 Phương án biểu diễn là một ma trận

* **Hàm mục tiêu:**
* **Các ràng buộc:**

**Nhận xét:**

Các bài toán trên thuộc các lĩnh vực khác nhau nhưng cùng có chung 1 dạng:

“Xác định các biến hay tìm 1 phương án chấp nhận được (tức là thoả mọi ràng buộc) sao cho làm cực đại (hoặc cực tiểu) hàm mục tiêu”.

### 4.3.3. Bài toán tối ưu tổ hợp.

Có dạng tổng quát:

Cho hàm là hàm mục tiêu xác định trên 1 tập hữu hạn các phần tử *D* (tập các phương án).

Mỗi có dạng gọi là 1 phương án, (tập các biến).

Cần tìm 1 phương án X “chấp nhận được” (thoả mọi ràng buộc) sao cho đạt **phương án tối ưu**.

# 5. Các chiến lược xây dựng thuật toán.

## 5.0. Bài toán toán ưu có các chiến lược:

1. Thô sơ, trâu bò: Brute Force (duyệt trâu) Tuần tự xét tất cả các khả năng (phương án) có thể có cho đến khi gặp giải pháp cho vấn đề cần giải quyết (nghĩa là thoả mãn tất cả các ràng buộc). thời gian mũ.
2. Toán học: ngành tối ưu (Quy hoạch tuyến tính).
3. Tối ưu cục bộ: phương pháp tham lam.
4. Các phương pháp đơn giản:
   1. Giải bằng hình học.
   2. Thuật toán Fourier – Motzkin.
5. Phức tạp hơn: Phương pháp đơn hình, phương pháp trọng tâm, …

### 5.0.1. Phương pháp 1: Đơn giản, hình học.

Cho các ràng buộc:

A red triangle with black text

Description automatically generated

Điểm cuối cùng mà đường mức còn cắt miền chấp nhận được là điểm Optimal Point.

Optimal Point là giao điểm của () và (), giải hệ phương trình ra được Optimal Point có toạ độ là .

Vậy phương án tối ưu là và vói giá trị tối ưu là

### 5.0.2. Phương pháp 2: Thuật toán Fourier – Motzkin.

Cho các ràng buộc:

**Bước 1: Khử .**

**Bước 1.2: .**

* Ghép (1, 5):
* Ghép (2, 5):
* Ghép (3, 5):
* Ghép (4, 5):
* Ghép (1, 7):
* Ghép (2, 7):
* Ghép (3, 7):
* Ghép (4, 7):
* (6):

**Bước 1.3: Rút gọn.**

**Bước 2: Khử .**

* Ghép (2, 4):
* Ghép (2, 6):
* Ghép (2, 7):
* Ghép (2, 8):
* Ghép (5, 4):
* Ghép (5, 6):
* Ghép (5, 7):
* Ghép (5, 8):

**Vậy, ta có:**

**Vậy**

Để tìm , ta cần tìm một cặp số từ các bất phương trình các bước 2 và bước 1 sao cho:

Giải ra được:

**Bài tập lấy điểm:**

Một công ty điện tử sản xuất 2 kiểu radio trên 2 dây chuyền độc lập. Công suất của dây chuyền 1 là 60 radio/ngày và dây chuyền 2 là 75 radio/ngày. Để sản xuất 1 chiếc radio kiểu 1 cần 10 linh kiện điện tử E và 1 chiếc radio kiểu 2 cần 8 linh kiện này. Số linh kiện này được cung cấp mỗi ngày không quá 800. Tiền lãi khi bán 1 radio kiểu 1 là 30 USD và kiểu 2 là 20 USD. Xác định phương án sản xuất cho lãi nhiều nhất trong này.

**Hãy mô hình hoá bài toán và giải bằng phương pháp hình học và bằng phương pháp Fourier – Motzkin.**

* **Giải bằng hình học:**

Đường mức:

Các ràng buộc được đánh số thứ tự:

A red square with blue dots

Description automatically generated

Điểm cuối cùng mà đường mức còn cắt miền chấp nhận được là điểm D.

D là giao điểm của (1) và (3), giải hệ phương trình ta được toạ độ điểm D là (60, 25).

Vậy phương án tối ưu là và với lãi suất cao nhất là 2300 USD.

* **Giải bằng thuật toán Fourier – Motzkin.**

Viết lại các ràng buộc:

Khử trước:

Bước 1.1. Chuyển về 1 vế và viết lại các bất đẳng thức sao cho cùng chiều.

Bước 1.2. Ghép từng BĐT dạng với các BĐT dạng

* Ghép (1, 4): (Đúng)
* (2):
* Ghép (4, 6):
* (5):
* Ghép (1, 6):
* Ghép (3, 6):

Tương tự: Khử .

Bước 2.1. Đưa về 1 vế.

Bước 2.2. Ghép từng BĐT dạng với các BĐT dạng

* Ghép (1, 3):
* Ghép (3, 5):
* Ghép (1, 4):
* Ghép (4, 5):

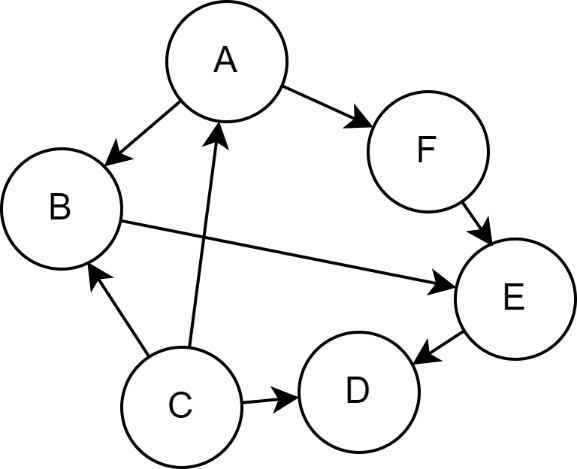
Rút gọn:

Kết luận: nên

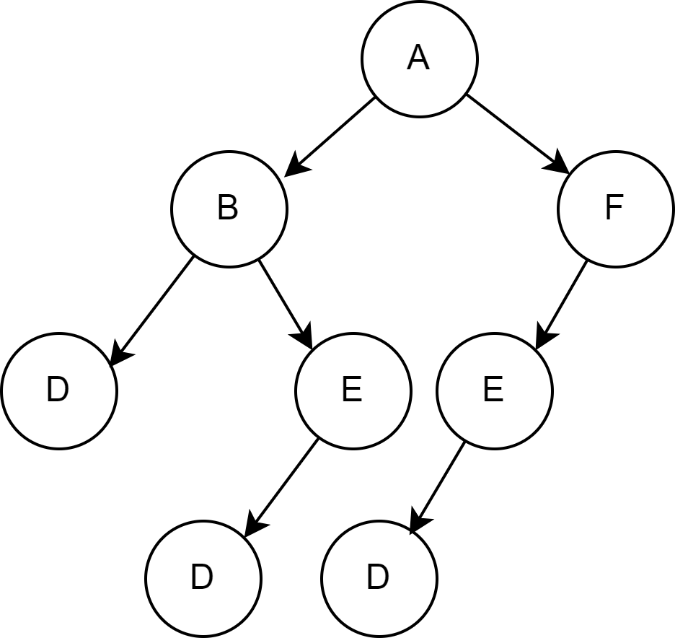
Thế lại vào các bất phương trình ở trên, ta được:

## 5.1. Backtracking (Quay lui).

Ví dụ: Tìm kiếm **tất cả** các đường đi từ A đến D.



Một thuật toán đơn giản có thể sử dụng là DFS:



Ta có thể gọi đây là cây đệ quy quay lui.

### 5.1.1. Các đặc điểm của một bài toán quay lui.

* Yêu cầu vét cạn.
* Solution form:
* Quá trình xây dựng lời giải là một quá trình tuần tự xây dựng từng bước giải thành phần.
* …

### 5.1.2. Mô hình giải quyết bài toán bằng quay lui.

A white background with black text

Description automatically generated

### 5.1.3. Ví dụ.

**Ví dụ 1**: Liệt kê tất cả hoán vị của dãy

Ta có:

Tập lời giải trong đó là một mảng chứa lời giải (một hoán vị) cho bài toán.

là một mảng chứa các chữ số của dãy .

là một mảng phụ biểu thị các số ở vị trí đã được chọn hay chưa.

A black and white text

Description automatically generated

**Ví dụ 2**. Quay lại ví dụ đầu tiên, trình bày cách sử dụng DFS.

Ta có:

Tập lời giải trong đó là một mảng chứa lời giải (đường đi) cho bài toán.

là một mảng 2 chiều, trong đó cho biết đỉnh hiện tại và cho biết hai đỉnh có kề nhau không, ( nếu có, nếu không)

là một mảng phụ biểu thị các đỉnh kề với ở đã được đi qua hay chưa.

A screenshot of a computer program

Description automatically generated

**Ví dụ 3**. Liệt kê tất cả các dãy nhị phân có độ dài n.

Tập lời giải trong đó là một mảng chứa lời giải (một dãy nhị phân) cho bài toán.

A white paper with black text

Description automatically generated

**Bài tập lấy điểm.**

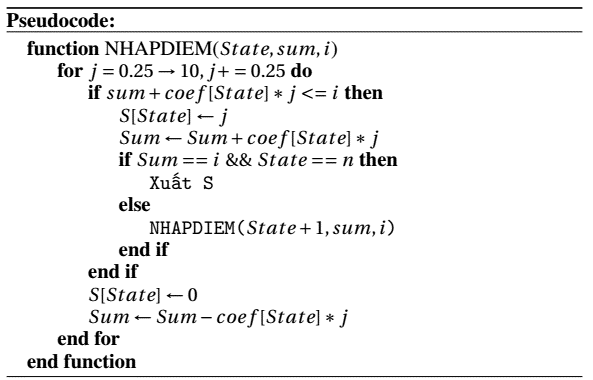
Môn học X có cột điểm. Hệ số (tỷ trọng) của mỗi cột điểm đã được phòng đào tạo (PĐT) quy định trước. Tuy nhiên, sinh viên (SV) đã thoả thuận với giảng viên (GV) chỉ làm một đồ án duy nhất lấy điểm cho cả môn. Sau khi chấm điểm đồ án xong bây giờ GV phải nhập đủ cột điểm cho PĐT nhưng vẫn phải đảm bảo điểm trung bình môn (ĐTB) theo cách tính của PĐT sẽ vẫn bằng với điểm, là điểm đồ án của SV. Hãy phát sinh tất cả cách ghi điểm mà GV có thể ghi cho SV. Biét rằng, điểm mỗi cột sẽ được làm tròn đến 0.25 và điểm mỗi cột , ĐTB được làm tròn đến 0.1.

* INPUT: Một số nguyên tương ứng với số cột điểm của môn học; Một mảng chứa các hệ số của cột điểm (hệ số của một cột là tỷ lệ % của cột đó trong ĐTB của môn học); Số thực là điểm đồ án và cũng là ĐTB của SV.
* OUTPUT: Tất cả các cách ghi điểm có thể có của GV. Mỗi cách ghi là một bộ gồm điểm số tương ứng với cột điểm.

Ví dụ: Môn học X có 3 cột điểm, hệ số của 3 cột điểm lần lượt là điểm quá trình 20%, điểm thực hành 30% và điểm thi cuối kỳ 50%. Nếu SV A có điểm đồ án là 9.5 thì một số cách điểm hợp lệ là (7.25 10 10), (7.75 9.75 10), (8.75 9 10),…

Ta có:

* Tập lời giải trong đó là một bộ lời giải (một bộ gồm số điểm) cho bài toán.
* Một mảng gồm phần tử là hệ số (tỷ trọng) của mỗi cột điểm được quy định trước. là hệ số của cột điểm thứ
* Một số thực là điểm đồ án và cũng là ĐTB của SV.
* Một số nguyên State, cho biết chúng ta đang xét đến cột điểm nào.
* Một số thực Sum, cho biết tổng các cột điểm tính đến cột điểm thứ Score.



## 5.2. Greedy (Tham lam).

Cách giải quyết **Tối ưu cục bộ**:

* Đi qua 1 số bước với 1 tập các khả năng lựa chọn cho mỗi bước.
* Tại mỗi bước, chọn một khả năng được xem là tốt nhất tại lúc đó.

Lấy tiêu chuẩn **tối ưu** (trên phạm vi **toàn cục**) của bài toán, dựa vào đó chọn lựa hành động tốt nhất của **từng bước** (hay từng giai đoạn) trong quá trình tìm kiếm lời giải.

### 5.2.1. Các đặc điểm của một bài toán tham lam.

Các bài toán tham lam có thể giải được bằng phương pháp tham lam có 2 tính chất/đặc trưng sau đây (**dấu hiệu nhận biết**):

* Optimal Substructure.
* Greedy choice property thiết kế tiêu chuẩn tối ưu cục bộ.

### 5.2.2. Mô hình giải quyết bài toán bằng tham lam.

* Phải được tiến hành qua nhiều bước. Ở mỗi bước xác định một thành phần của lời giải S.
* Có một vòng lặp for/while.
* Lời giải tối ưu toàn cục là tổ hợp các lời giải tối ưu cục bộ.
* **Quan trọng nhất: Tập hợp các lời giải tối ưu cục bộ hi vọng sẽ tạo thành lời giải tối ưu toàn cục.**

A black and white text

Description automatically generated

**Ưu điểm:**

* Đơn giản, dễ cài đặt.
* Tốc độ nhanh (thời gian đa thức).

**Khuyết điểm**:

* Chưa chắc cho lời giải chính xác (**thường cho phương án tốt chứ chưa hẳn là tối ưu**).
* Không phải luôn chấp nhận được (có thể cho lời giải tệ )[trong trường hợp đã chọn những thành phần lời giải tốt và chỉ còn lại những thành phần làm cho lời giải trở nên tệ hơn ở cuối cùng].
* Khó chứng minh tính đúng, nếu chứng minh được thì nó là thuật toán (còn không thì là thuật giải).

### 5.2.3. Ví dụ

**Ví dụ 1.**

Cho là một đồ thị vô hướng liên thông, trong đó là tập đỉnh, là tập cạnh và các cạnh đều có trọng số. Cây với được gọi là cây khung của . Cây không có chu trình và có cạnh. Cây khung ngắn nhất hay còn gọi là cây khung tối tiểu là cây khung của có tổng độ dài (trọng số) các cạnh nhỏ nhất. Tìm cây khung tối tiểu của .

**Yêu cầu**:

* Hãy thiết kế một thuật toán “quay lui” để giải bài toán trên (trình bày dưới dạng mã giả và có chú thích cho người đọc dễ hiểu).
* Trình bày một cách giải khác (chỉ cần nêu ý tưởng) có độ phức tạp thấp hơn thuật toán quay lui ở trên và minh hoạ cách giải này cho ví dụ sau:

A black and white background with white dots

Description automatically generated

**Lưu ý: Các thuật toán phải được trình bày dưới dạng mã giả, nên có chú thích và minh hoạ qua ví dụ cho người đọc dễ hiểu.**

Lời giải 1 cây khung

Ta có:

A white sheet with black text

Description automatically generated

**Prim’s Algorithm.**

* Bắt đầu bằng việc chọn một đỉnh bất kỳ, đặt nó vào cây khung T.
* Trong khi cây khung T có ít hơn n đỉnh
  + Ghép vào T cạnh có trọng số nhỏ nhất liên thuộc với một đỉnh của T và không tạo ra chu trình trong T.

**Chú ý**: - Thuật toán dừng lại khi T có đủ n đỉnh hay (n-1) cạnh.

- Có nhiều hơn một cây khung nhỏ nhất ứng với một đồ thị liên thông có trọng số.

* **Bước 1:** Khởi tạo
  + VT = {s}; ET = ∅; (VT – tập đỉnh; ET – tập cạnh)
  + ds = 0; v ∉ VT dv = w(s, v), nếu s và v liền kề
  + dv = ∞, nếu s và v không liền kề
* **Bước 2:** Tìm cạnh
  + Tìm u mà du = min {dv | v ∉ VT}
  + VT = VT ∪ {u};
  + ET = ET ∪ {e} , e – cạnh nối u với một đỉnh của cây có trọng số du
  + Nếu VT ≡ V thì dừng.
* **Bước 3**: Cập nhật nhãn
  + dv = min {dv, w(u, v)} với v∉ VT

**Kruskal’s Algorithm.**

* Bắt đầu bằng việc chọn một cạnh có trọng số nhỏ nhất, đặt nó vào cây khung T.
* Trong khi cây khung T có ít hơn (n-1) cạnh
  + Ghép vào T cạnh có trọng số nhỏ nhất và không tạo ra chu trình trong T.
    - * **Bước 1:**
        + Sắp xếp các cạnh của đồ thị G theo thứ tự có trọng số không giảm: w(e1) ≤ w(e2) ≤ … ≤ w(em)
        + ET = {e1} , i =1
      * **Bước 2**: Tìm k = min { j | ET ∪ {ej} không có chu trình}

* + - * **Bước 3**: i = i +1

Nếu i = n-1 thì dừng

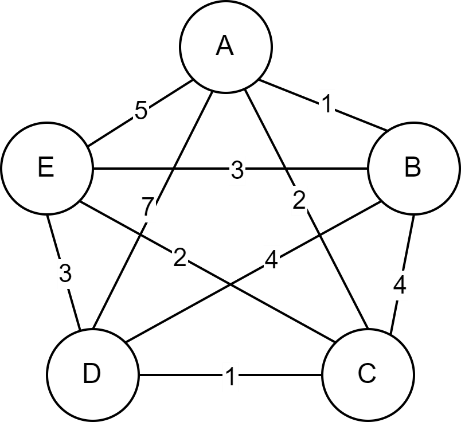
Nếu i < n-1 thì quay lại bước 2

**Comparison between Kruskal’s and Prim’s Algorithm.**

* Prim chọn cạnh có trọng số nhỏ nhất liên thuộc với một đỉnh đã thuộc cây và không tạo ra chu trình
* Kruskal chọn cạnh có trọng số nhỏ nhất miễn là không tạo ra chu trình
* Thuật toán Prim hiệu quả hơn đối với các đồ thị dày (số cạnh nhiều)

**Ví dụ 2: Bài toán người giao hàng (Traveling Salesman Problem - TSP).**

Giải thiết: Đồ thị đủ (nếu thực tế không có đường đi thì cho trọng số là ).



Thuật giải: Cách 1.

{các đỉnh trên đồ thị} / {}

; // đỉnh hiện hành

While ( )

- Chọn đỉnh trong có khoảng cách tới N là nhỏ nhất.

- Cập nhật các đối tượng để chọn: - Cập nhật vào

- ;

**Ví dụ 3: Bài toán chọn hoạt động (Activity Selection Problem/ Interval Scheduling).**

* Cho một tập các hoạt động
* Một hoạt động có thời điểm bắt đầu là và thời điểm chấm dứt là , .
* A diagram of a number

  Description automatically generatedNếu hoạt động được chọn thì i tiến hành trong thời gian thời gian
* Hai hoạt động và là “**tương thích nhau**” (compatible) nếu và không chạm nhau i.e. hoặc .
* **Yêu cầu**: Tìm tập hợp lớn nhất các hoạt động tương thích nhau?

A diagram of a graph

Description automatically generated

**Brute Force:**

* Xem xét tất cả các phương án (tập con các hoạt động) chấp nhận được.
* Chọn tập con (chấp nhận được) lớn nhất
* Độ phức tạp:

**Greedy:**

1. Dùng 1 **quy tắc đơn giản** để chọn 1 hoạt động .
2. Bỏ qua tất cả hoạt động không tương thích với
3. Lặp lại cho đến khi tất cả hoạt động đều được xem xét.

Những quy tắc có thể xem xét:

* Cách 1: Chọn hoạt động bắt đầu sớm nhất ( nhỏ nhất).

A black line with colorful dots

Description automatically generated with medium confidence

* Cách 2: Chọn hoạt động ngắn nhất ( nhỏ nhất).

A black line with a long black line

Description automatically generated with medium confidence

* Cách 3: Với mỗi hoạt động, tìm số hoạt động tương thích với nó và chọn hoạt động nào có số tương thích lớn nhất.

A close-up of a person's face

Description automatically generated

* Cách 4: Chọn hoạt động hoàn thành sớm nhất ( nhỏ nhất).

A diagram of a graph

Description automatically generated

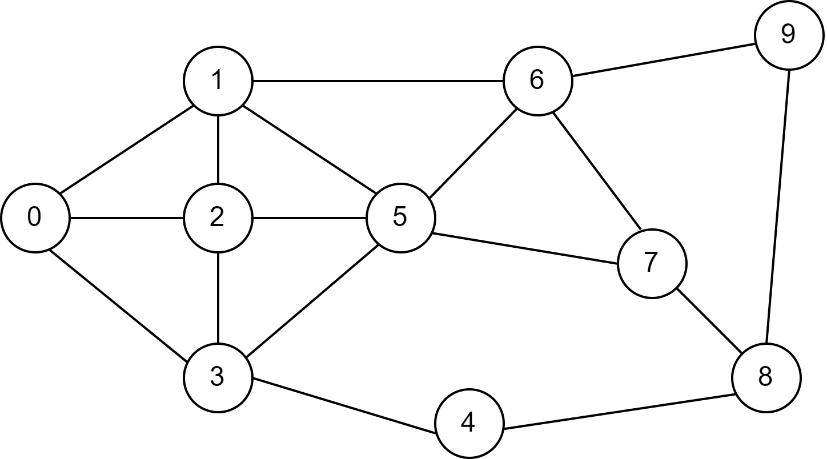
A white paper with black text

Description automatically generated

**Ví dụ 4: Bài toán tô màu.**

Giả sử ta có bản đồ các quốc gia trên thế giới, ta muốn tô màu các quốc gia này sao cho 2 nước có cùng ranh giới được tô khác màu nhau.

Yêu cầu tìm cách tô sao cho số màu sử dụng là ít nhất.



|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Đỉnh: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Bậc: | 3 | 4 | 4 | 4 | 2 | 5 | 4 | 3 | 3 | 2 |

Ta sẽ tô các đỉnh có bậc cao nhất trước, sau đó sẽ giảm số bậc của các đỉnh kề với bậc đó.

Sau đó chọn một đỉnh khác có bậc cao nhất để tô và chọn màu được sử dụng nhiều nhất để tô.

**Lý do**:

* + Chọn đỉnh có bậc cao nhất (Minimum Remaining Values): Choose the variable with the fewest legal left values in its domain. Because we need to use all the values, so we choose to deal with the hardest first.
  + Chọn màu được sử dụng nhiều nhất (Least Constraining Value): Given a choice of variable, choose the *least constraining value   
    (i.e.,* the one that rules out the fewest values in the remaining variables). I.e., the one that rules out the fewest values in the remaining variables.

## 5.3. Heuristic Algorithm.

### 5.3.1. Thuật giải.

Giải pháp được diễn đạt tương tự như thuật toán nhưng **không đòi hỏi như thuật toán.**

* **Tính xác định**
* **Tính đúng** chấp nhận các thuật giải đơn giản có thể cho **kết quả đúng hay gần đúng** nhưng có **khả năng thành công cao hơn**.

### 5.3.2. Thuật giải heuristic.

Thuật giải heuristic phải có những đặc trưng:

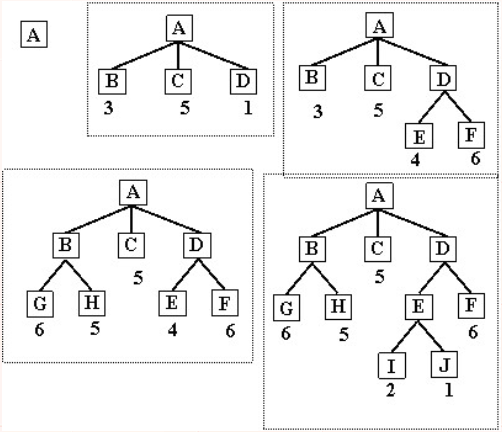
* Thường tìm được lời giải tốt, mặc dù không phải là tốt nhất **kết quả chấp nhận được**.
* Thực hiện dễ dàng nhanh chóng so với thuật giải tối ưu **hiệu quả**.
* Giải pháp **tự nhiên, mang tính thông minh nhất định** tương tự con người, gần gũi với cách giải của con người.

### 5.3.3. Best – First Search (BFS).

**Tư tưởng:**

* Kết hợp 2 phương pháp tìm kiếm theo chiều sâu và chiều rộng, được hướng dẫn bởi hàm đánh giá.
* Tại mỗi bước, chọn đi theo trạng thái có **khả năng cao nhất** trong số các trạng thái đã được xét cho đến thời điểm đó.
  + Ưu tiên đi vào nhánh có khả năng nhất, đồng thời vẫn “quan sát: những nhánh khác.
  + Nếu càng đi sâu vào một hướng mà phát hiện càng đi thì càng tệ không đi tiếp mà chọn đi theo một hướng tốt nhất trong số những hướng chưa đi.

A diagram of a diagram of a number of circles and lines

Description automatically generated with medium confidence

### 5.3.4. Hàm heuristic.

Độ tốt của một trạng thái được tính dựa trên hai giá trị:

* : một ước lượng về chi phí từ trạng thái hiện hành cho đến đích.
* : “chiều dài quãng đường” đã đi từ trạng thái ban đầu cho đến trạng thái hiện tại (chi phí thực sự).
* Quy ước là và đều không âm và càng nhỏ nghĩa là càng tốt.

### 5.3.5. Search Algorithms.

A blue sign with white letters

Description automatically generatedA close-up of a sign

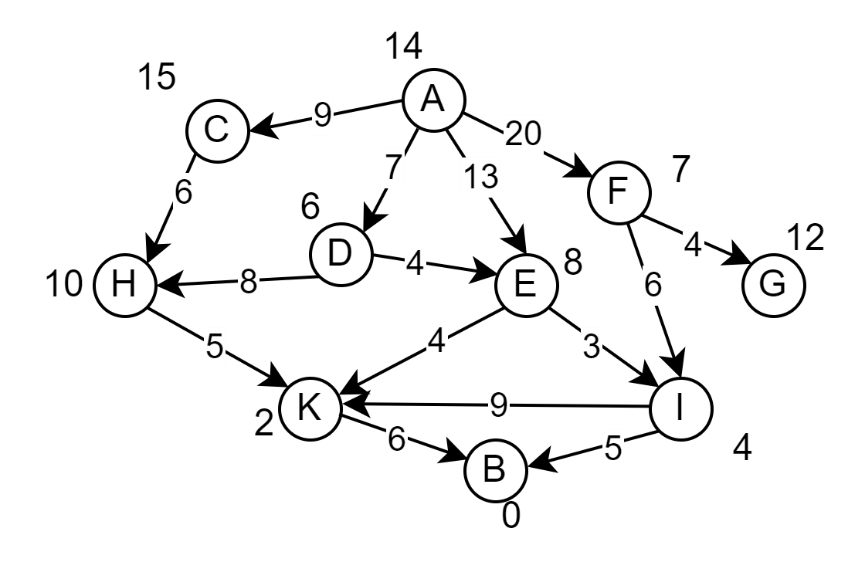
Description automatically generatedHàm đánh giá cho mỗi trạng thái/node.

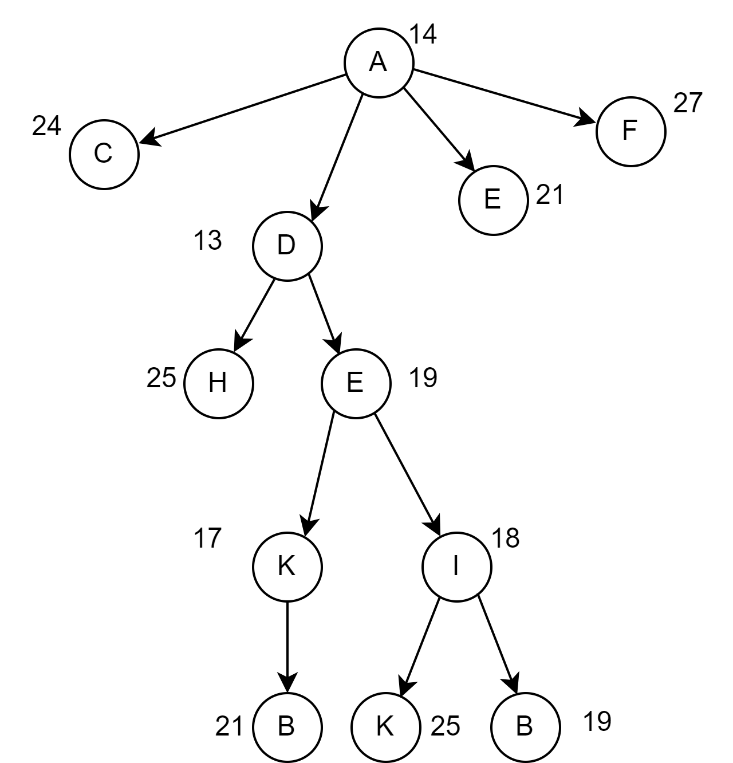
Chọn node tốt nhất trước – “Best – first search”

* Uniform Cost Search:
* Greedy Search: arbitrary
* A search: arbitrary
* A\* search: **admissible**

### 5.3.6. A\* Search.

Tìm đường đi ngắn nhất:





**Implementation**:

Mỗi đỉnh tương ứng với 1 số độ tốt

**Bước 1**:

**Bước 2:** While ()

2.1. Chọn thuộccó nhỏ nhất.

2.2. Nếu là trạng thái kết thúc thì thoát, thông báo kết quả.

2.3. Chuyển qua , và mở các sau .

2.4. Xét các đỉnh kề của

TH1:  **và**

; // là giá trị heuristic

Thêm vào

TH2:

if() // nếu đến được bằng path ngắn hơn thì cập nhật lại trong

TH3:

if() // nếu đến được q bằng path ngắn hơn

Bỏ khỏi

Thêm vào

Cập nhật các đỉnh chịu ảnh hưởng từ sự thay đổi của .

**Bước 3:** Không tìm được.

## 5.4. Conquer Algorithmics.

### 5.4.1. Divide and Conquer (Chia để trị).

**Ý tưởng:**

* Để giải một bài toán có kích thước **, chia** bài toán đã cho thành 1 số bài toán con có kích thước nhỏ hơn.
* Giải các bài toán con rồi **tổng hợp kết quả lại** để được lời giải của bài toán ban đầu.
  + Giải các bài toán con: lại chia kích thước nhỏ hơn nữa.
  + Quá trình dẫn đến những bài toán mà lời giải hiển nhiên, dễ dàng thực hiện **(bài toán cơ sở)**.

**The Divide and Conquer Pradigm**

1. DIVIDE into smaller subproblems.
2. CONQUER subproblems recursively.
3. COMBINE solutions of subproblems into one for the original problem.

**Mô hình:**

A black and white text

Description automatically generated

**Ví dụ:**

Ví dụ 1: Merge Sort.

**Chia:**

* Nếu mảng A rỗng hoặc chỉ có một phần tử thì trả về chính A (đã có thứ tự).
* Ngược lại, chia A thành 2 mảng con.

**Đệ quy:**

* Sắp xếp 2 mảng con.

**Tổng Hợp:**

* Sắp xếp xen kẽ hai mảng con đã có thứ tự.

A close-up of a card

Description automatically generated

Ví dụ 2: Bài toán Min/Max.

Tìm giá trị Max, Min trong đoạn của mảng A có phần tử.

Cách 1: Sort + a[l]/a[r]

Cách 2: min=a[0]

max=a[0]

for()

Cách 3: Chia để trị.

**Chia:**

* Nếu giải trực tiếp.
* Ngược lại, chia bài toán thành 2 bài toán con rời nhau.

**Tri:**

* Tìm kiếm Max1, Min1 trên bài toán con 1.
* Tìm kiếm Max2, Min2 trên bài toán con 2.

**Tổng hợp:** Tổng hợp kết quả.

A white background with black text

Description automatically generated

Ví dụ 3: Thuật toán Quick Sort.

**Chia**:

* Chọn 1 phần tử bất kỳ trong mảng làm nút trục, xác định vị trí hợp lệ của nút này trong mảng (vị trí pivot).
* **Phân hoạch** các phần tử còn lại sao cho từ vị trí 0 đến pivot – 1 đều có giá trị nhỏ hơn hoặc bằng nút trục, từ vị trí pivot + 1 đến n – 1 lớn hơn nút trục.

**Đệ quy:**

* Sắp xếp 2 mảng con.

**Tổng hợp**:

* **Không tổng hợp kết quả** 1 cách tường minh (đã thực hiện trong quá trình phân hoạch).

A computer code with black text

Description automatically generated

Ví dụ 4: Bài toán tìm hạng trong không gian 2D.

* Cho tập S có n điểm trong 2D, hạng của điểm X là số lượng các điểm mà X trội hơn.

Cho điểm A() và B(). A được gọi là “trội hơn” B nếu và .

Thiết kế thuật toán để sắp hạng các điểm trong tập S?

A diagram of a circle with red circles and blue dots

Description automatically generated

Dựa vào hình ảnh (trực giác), ta có thể dễ dàng giải quyết:

Giải bằng chia để trị:

A screenshot of a computer

Description automatically generated

Ví dụ 5:

Một mảng được gọi là “weakly unimodal” nếu nó có thể được tách thành 2 mảng con bao gồm một mảng con không giảm được theo sau bởi một mảng con không tăng. Ví dụ: mảng a với các phần tử sau: 1 3 4 5 7 8 10 10 13 14 10 9 9 6 2 2 được xem là một mảng thoả tính chất trên. Bài toán đặt ra là: Cho một mảng số nguyên a gồn n phần tử thoả tính chất “weakly unimodal”, tìm giá trị lớn nhất trong mảng.

Yêu cầu:

* Hãy thiết kế một thuật toán theo chiến lược “**Chia để trị (Divide and Conquer)**” để giải bài toán trên. Thuật toán phải được trình bày dưới dạng mã giả, có chú thích và minh hoạ qua ví dụ cho người đọc dễ hiểu.

### 5.4.2. Decrease and Conquer (Giảm để trị).

Ý tưởng:

* Để giải một bài toán có kích thước , ta **giảm** kích thước của nó**, giải 1 bài toán con**, có kích thước nhỏ hơn.
* Giải bài toán con rồi tìm cách suy ra lời giải của bài toán ban đầu.

Mô hình:

* Kiểu top – down (dẫn đến 1 giải thuật đệ quy).
* Kiểu bottom – up (giải thuật lặp).

A black text on a white background

Description automatically generated

**3 trường hợp giảm kích thước:**

1. Decrease by a constant: hằng số -1, -2, -3, …
2. Decrease by a factor: hệ số
3. Variable size decrease.

Ví dụ 1: Tính luỹ thừa *.*

* Công thức đệ quy:
* Giảm kích thước bởi cùng 1 hằng số (không đổi) là 1.

Pow(x, n) {

if(n == 0 ) return 1;

return Pow(x, n – 1) \* x;

}

​Ví dụ 2: Binary Search.

* Giảm kích thước bởi cùng 1 hệ số là
* binarySearch(a, n) chuyển thành tìm kiếm trên mảng con có kích thước giảm đi 1 nửa

int binarySearch(int a[], int n, int l, int r, int x) {

if(l > r) return -1;

int mid = (l + r)/2;

if(x == a[mid]) return mid;

if(x < a[mid]) return binarySearch(a, n, l, mid – 1, x);

return binarySearch(a, n, mid + 1, r, x);

}

​Ví dụ 3: Tìm USCLN (a, b).

* **Brute – force** (dựa theo định nghĩa của USCLN): phân tích 2 số ra thừa số nguyên tố tìm thừa số chung lớn nhất.
* 2 cách giải theo kiểu Giảm để trị: dãy liên tiếp các phép toán
* Giảm kích thước của biến/tham số.

A white background with black text

Description automatically generated

A white background with black text

Description automatically generated

Ví dụ 4: Tìm hoán vị của 1 tập gồm n phần tử

* Cách 1: Quay lui – Bactracking.
* Cách 2: Giảm để trị - Decrease and Conquer.

### 5.4.3. Transform and Conquer (Biến đổi để trị).

…

## 5.5. Dynamic Programming (Quy hoạch động).

Ví dụ dẫn vào bài:

Dãy số Fibonacci:

long Fibo(int n) {

    if(n == 1 || n == 2) return 1;

    return Fibo(n - 1) + Fibo(n - 2);

}

* Đây là thuật toán sử dụng đệ quy với chiến lược chia để trị.
* Độ phức tạp:
  + Tại sao ? Là do có các bài toán con lặp lại.

A diagram of a diagram

Description automatically generated

Quy hoạch động ra đời nhằm khắc phục các nhược điểm của chia để trị, nên ta có thể xem QHĐ là chia để trị phiên bản nâng cấp !

**Phương pháp quy hoạch động:**

* **Chia để trị**: một số bài toán con nào đó có thể giải nhiều lần Chi phí cáo, có thể .
* **Quy hoạch động:** Như divide and conquer (kết hợp lời giải của các bài toán con) nâng cấp, khắc phục nhược điểm của D & C.
* Thường được vận dụng để giải bài toán tối ưu, bài toán có công thức truy hồi.

### 5.5.1. Dấu hiệu một bài toán có thể sử dụng QHĐ:

* Bài toán có Công thức truy hồi
* Bài toán có cấu trúc con tối ưu

**Optimal substructure:** An optimal solution to a problem (instance) contains optimal solutions to the subproblems.

* Bài toán có các bài toán con trùng lắp

**Overlapping substructure:** A recursive solution contains a “small” number of distinct subproblems repeated many times.

### 5.5.2. Ý tưởng QHĐ:

Khắc phục việc giải dư thừa 1 số bài toán con: **lưu trữ** kết quả + **tra cứu**.

* **Tạo ra 1 bảng (hay vector)** để lưu trữ kết quả các các bài toán con.
* Sử dụng kết quả đã lưu trong bảng mà không cần giải lại bài toán con.
* **Giải các bài toán con theo hướng bottom – up (từ nhỏ đến lớn).**

**3 – steps of developing DP algorithm.**

Các bước thực hiện:

* **Bước 1:** Phân tích đặc trưng của cấu trúc lời giải tối ưu + Xác định hàm (phương trình) quy hoạch động.
* **Bước 2:** Tạo bảng: lắp đầy bảng theo 1 quy luật nào đó.
  + Gán giá trị cho 1 số ô nào đó.
  + Xác định giá trị của 1 số ô khác nhờ vào giá trị các ô trước đó.
* **Bước 3:** Tra bảng và truy xuất/ xây dựng lời giải của bài toán.

### 5.5.3. Ưu, nhược điểm.

**Ưu điểm:**

* Thực hiện nhanh không giải lại 1 số bài toán con.
* Thường được vận dụng để giải các bài toán tối ưu, bài toán có công thức truy hồi.

**Nhược điểm:**

* Không hiệu quả khi:
  + Không tìm được công thức truy hồi.
  + Số lượng bài toán con và lưu trữ kết quả rất lớn.
  + Sự kết hợp lời giải của các bài toán con chưa chắc cho lời giải của bài toán đầu.

### 5.5.4. Ví dụ.

**Dãy Fibonacci:**

* Thuật toán quy hoạch động: lưu kết quả bài toán con đã tính để dùng lại.
* Khai báo: int \*F = new int[n];
* Phương trình QHĐ:

Nó là 1 cái phương trình/công thức mà phải gắn liền với cái bảng, nó sẽ giúp ta xác định được phần tử thứ nào đó.

Ví dụ:

long Fibo(int n) {

    int \*F = new int[n];

    F[1] = 1;

    F[2] = 1;

    for(int i = 3; i <= n; i++) {

        F[i] = F[i - 1] + F[i - 2];

    }

    return F[n];

}

Độ phức tạp (Fibonacci):

**Ví dụ 2: Tính tổ hợp chập k của n**.

long C(int n, int k) {

    if(k == 0 || k == n) return 1;

    if(k < 0 || k > n) return 0;

    return C(n - 1, k - 1) + C(n - 1, k);

}

Thuật toán chia để trị: Độ phức tạp: