ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

BÁO CÁO SEMNINAR

Môn học: Phân tích và thiết kế thuật toán

Sinh viên thực hiện: *18520766 – Trần Thái Hòa*

18520811 – Trần Kim Hưng

18520915 – Hồ Sỹ Đăng Khoa

Giảng viên hướng dẫn: *T.S Nguyễn Thanh Sơn*

Thành phố Hồ Chí Minh, tháng 1 năm 2021

MỤC LỤC

I. TÓM TẮT

Dưới góc nhìn lý thuyết, việc nghiên cứu thuật toán được nhận định là nền tảng của khoa học máy tính. David Harel và quyển sách với tựa đề "Algorithmics:the Spirit of Computing" đã nói: "Thuật toán là một sự tồn tại to lớn hơn một nhánh của khoa học máy tính. Nó là cái cốt lõi của khoa học máy tính và công bằng với mọi thứ, nó cũng liên quan đến những lĩnh vực khác như khoa học, kinh tế và công nghệ" [1].

Dù một người không nghiên cứu về lập trình máy tính thì vẫn luôn có một lý do đủ thuyết phục để một người có thể tìm hiểu về thuật toán. Tức là, lập trình máy tính sẽ không tồn tại nếu như không có sự tồn tại của thuật toán [1]. Với sự bành trướng của ứng dụng máy tính vào mọi khía cạnh của cuộc sống, mọi chuyên ngành và đời sống cá nhân thì việc học và tìm hiểu về thuật toán sẽ trở thành điều tất yếu với sự vận động xã hội như thế.

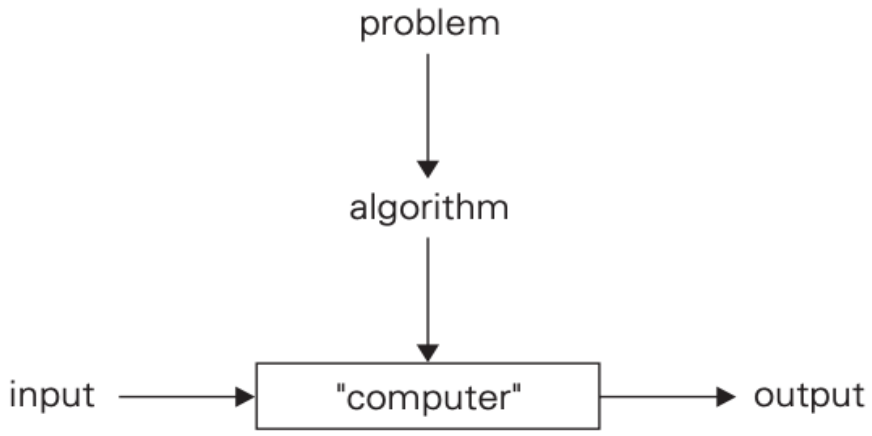
Trong đồ án này, chúng tôi sẽ giới thiệu về các thuật toán đã được học trong các buổi thảo luận và đưa ra các hướng giải quyết bài toán dưới góc nhìn của một thuật giải nhất định và cuối cùng là đưa ra các hướng tiếp cận đã được đưa vào thực tiễn.

II. GIỚI THIỆU

Khi phân tích và thiết kế thuật toán, ta phải hiểu khái niệm thuật toán là như thế nào và khi hiểu được khái niệm của thuật toán, việc phân tích và thiết kế thuật toán có thể được thực thi một cách có hệ thống.

1. Thuật toán là gì?

Thuật toán là một khái niệm không có sự thống nhất chặc chẽ về khái niệm nhưng có một thoả thuận chung được đặt ra về ý tưởng của thuật toán là "Thuật toán là một chuỗi các chỉ thị nhằm giải quyết một vấn đề hoặc tìm ra một giá trị đầu ra trong một khoảng thời gian hữu hạn" [1] (Hình. 1.1).



Hình. 1.1. Khái niệm về thuật toán đã được mô hình hoá[1]

Trong môn học phân tích và thiết kế thuật toán, chúng ta muốn tìm ra một lời giải cho một bài toán bằng một giải thuật thích hợp nhất. Hoặc, như các nhóm thảo luận trước thường nhắc là chọn ra một phương án tối ưu nhất.

2. Tối ưu là gì?

Khi giải quyết một vấn đề (hoặc một bài toán) là chọn ra một phương án tốt nhất cho bài toán. Sự tối ưu được thể hiện trong thường tập trung vào hai loại tối ưu cơ bản. Đó là tối ưu về thời gian và không gian. Hai phương án tối ưu này có một mối liên kết chặt chẽ với nhau và mỗi liên kết của chúng bù trừ cho nhau. Đôi lúc khi ta muốn tối ưu về thời gian thì ta cần hi sinh bằng

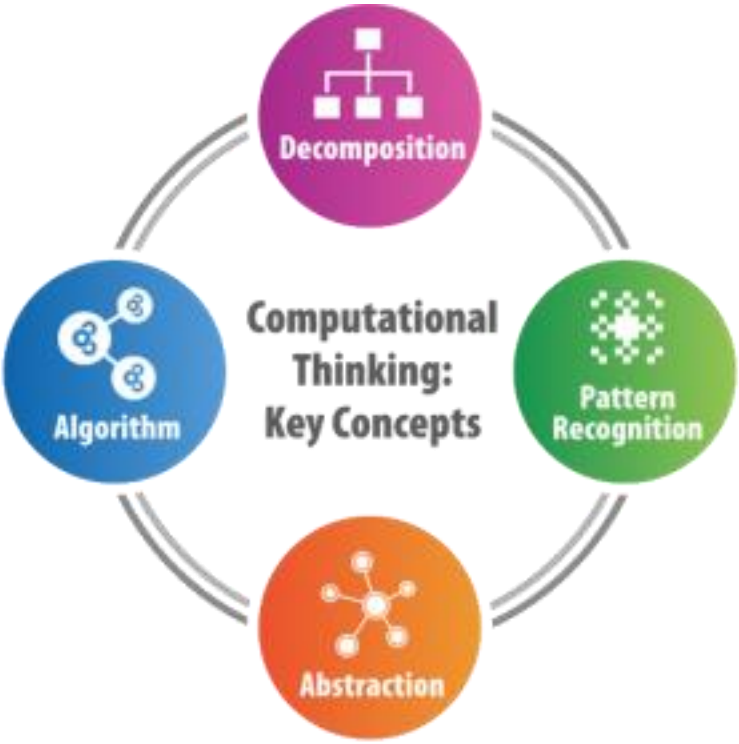
cách tăng độ phức tạp về không gian. Và trong thời đại nơi mà sự phát triển về mặt phần cứng được cải thiện hơn so với quá khứ, thì việc hi sinh không gian để đổi lấy thời gian không phải là một phương án tốt hơn so với việc hi sinh thời gian nhưng phương án đó trở thành một phương án dễ thực hiện và dễ thương lượng hơn .Nhưng sự thật là ta muốn tìm ra phương án có thể tối ưu cho cả hai phương án.

Với chúng tôi, vấn đề tối ưu hoá chỉ đơn giản là đi tìm cực đại hoặc cực tiểu của một bài toán và công thức để tính ra cực đại và cực tiểu là một công thức hoặc một độ đo được đề ra bởi người lập trình. Với tối ưu hoá về hiệu suất hoạt động của chương trình thì độ đo của chúng được định nghĩa lần lượt là thời gian thực thi và dung lượng sử dụng cho một lần thực thi. Trong những bài toán như bài toán căn lề văn bản thì việc tính toán độ thẩm mỹ của một văn bản sẽ được định nghĩa ra thành một công thức số học và nhiệm vụ của người lập trình là phải tìm được cực tiểu nhỏ nhất (nếu định nghĩa độ xấu của văn bản) hoặc cực đại (nếu định nghĩa là độ đẹp của văn bản).

3. Làm thế nào để giải quyết một bài toán?

Trong môn học phân tích và thiết kế thuật toán, chúng ta đã được tìm hiểu về tư duy lập trình. Với tư duy lập trình, ta có thể mô hình hoá hoặc hệ thống hoá quá trình giải quyết bài toán. Tư duy lập trình được hiểu là quá trình như sau:

- **Tóm tắt (Abstraction):** Được định nghĩa là tìm ra mục tiêu đầu ra mà bài toán đang hướng đến.
- **Nhận dạng (Pattern Recognition):** Từ bài toán đã được đơn giản hoá tìm những phương án thiết kế đã được sử dụng để giải quyết bài toán tương tự.
- **Phân tách (Decomposition):** Tổng quát hoá bài toán thành một công thức chung nhất hoặc phân tách bài toán ra thành các vấn đề dễ giải quyết hơn.
- **Thiết kế thuật toán (Algorithm):** Thiết kế thuật toán sau khi đã thực hiện việc phân tích bài toán.



Hình. 1.1. Vòng "luân hồi" của tư duy lập trình.

III. CÁC PHƯƠNG PHÁP THIẾT KẾ

Để phục vụ cho việc nhận dạng một bài toán cũng như giải quyết bài toán, chúng ta cần phải biết những phương pháp thiết kế.

Trong khi các kỹ thuật thiết kế giải thuật cung cập cho người lập trình nhiều hướng tiếp cận tổng quát nhất để giải quyết một bài toán. Thiết kế một giải thuật để giải quyết riêng cho một bài toán vẫn là một công việc khó khăn. Một số kỹ thuật đôi khi không có khả năng áp dụng vào một bài toán. Đôi khi để giải quyết một bài toán, người lập trình cần kết hợp nhiều phương pháp thiết kế thuật toán để giải quyết một bai toán [1].

Các tiểu mục sau sẽ nêu các ý tưởng của các giải thuật đã được giới thiệu bởi các nhóm thảo luận trước đó.

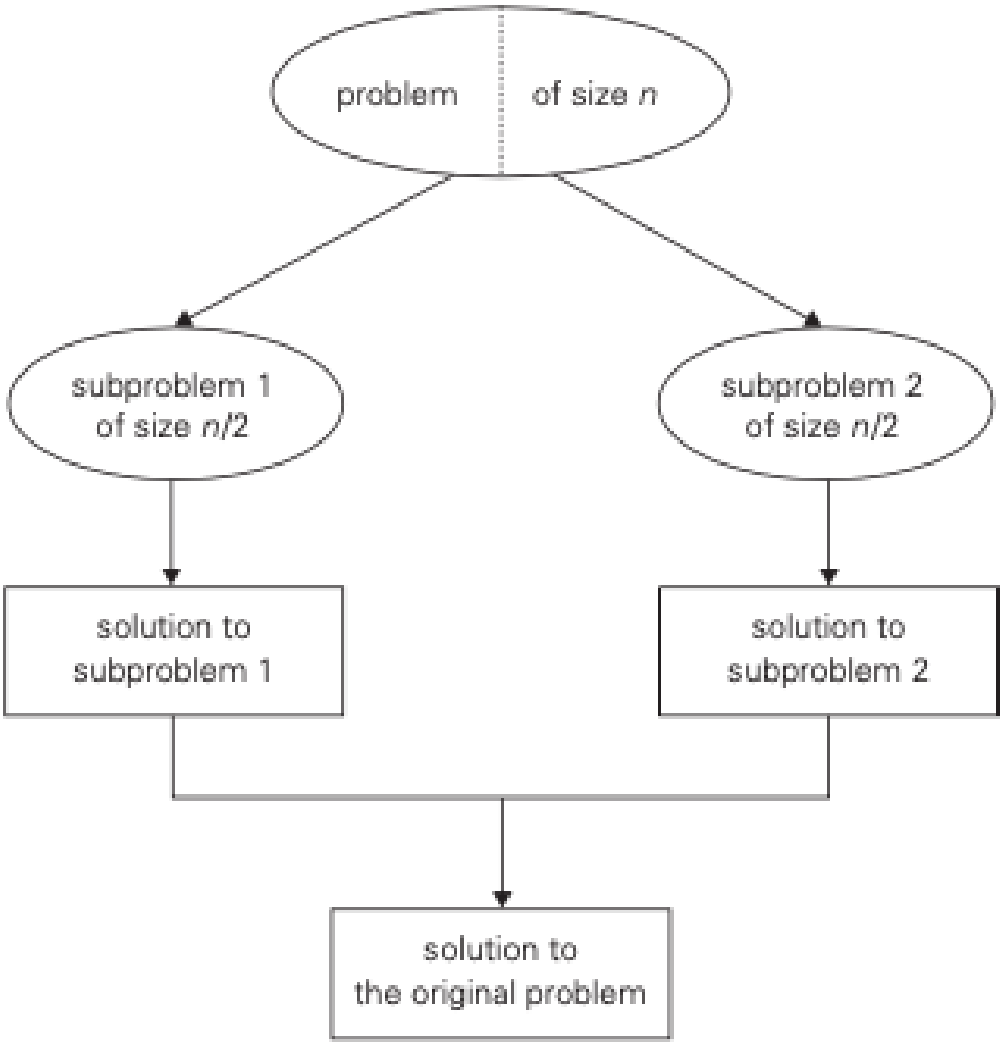
1. Chia để trị

Chia để trị là một phương pháp thiết kế thuật toán với ý tưởng là một bài toán lớn có thể được chia thành nhiều bài toán con (Hình. 2.1). Chiến thuật để thiết kê thuật toán được chia thành các bước như sau [1]:

1. Chia một bài toán lớn thành các bài toán con có cùng kích thước.
2. Các bài toán con được giải quyết hoặc theo hướng đệ quy hoặc giải quyết với nhiều giải thuật khác nhau.
3. Nếu cần thiết, phải kết hợp các bài toán con với nhau để giải quyết bài toán lớn gốc.

Giả sử như ta đang giải quyết bài toán về tính tổng của một dãy n số là $a_0 + \ldots + a_{n-1}$. Nếu n > 1, ta có thể chia các bài toán thành hai bài toán con nhỏ hơn có cùng kích thước: Tính tổng từ 1 đến [n/2] và tính tổng từ [n/2] đến n. Khi ra được kết quả của hai bài toán con và tổng lại thì chúng ta sẽ được kết quả bài toán gốc [1].

$$a_0 + \ldots + a_{n-1} = (a_0 + \ldots + a_{[n/2]-1}) + (a_{[n/2]} + \ldots + a_{n-1}). \text{ [1]}$$



Hình. 2.1. Minh hoạ trường hợp điển hình trong chia để trị [1]

2. Tham lam

Trong hướng tiếp cận theo thiết kế giải thuật tham lam, việc xây dựng một lời giải cho một bài toán bằng mỗi chuỗi các bước, mỗi khi mở rộng một phần lời giải sẽ được nhận vào, cho đến khi gặp lời giải cho bài toán [1]. Mỗi bước, thuật toán sẽ đưa các lựa chọn có những đặc tính như sau:

- **Khả thi (feasible):** thoã điều kiện yêu cầu của bài toán

- **Tối ưu cục bộ (locally optimal):** tức là lựa chọn là lựa chọn tối ưu nhất so với những lựa chọn đã có.
- **Không thể thu hồi (irrevocable):** Một khi đã chọn thì không được quay về các bước con trong thuật toán.

3. Vét cạn

Vét cạn là một cách tiếp cận đơn giản để giải quyết một vấn đề, thường là dựa trên cốt lõi vấn đề và định nghĩa của các khái niệm đã giải quyết. Chiến lược vét cạn được cho là chiến lược dễ áp dụng nhất và có thể áp dụng hầu hết các bài toán.

Một ví dụ điển hình của vét cạn là bài toán tính lũy thừa. Bằng định nghĩa, ta có, a lũy thừa n là $a^n = \underbrace{a * a * ... * a}_{n\_times}$. Từ đó, việc giải quyết vấn đề trên đơn giản bằng cách nhân 1 cho a và nhân n lần [1].

4. Quy hoạch động

Quy trình động là một kỹ thuật để giải quyết các vấn đề với các bài toán con chồng chéo nhau. Thông thường, các vấn đề con này phát sinh từ sự lặp lại từ các bài toán con nhỏ hơn của nó. Thay vì giải các bài toán con lặp đi lặp lại, quy hoạch động đề xuất giải các bài toán con nhỏ hơn chỉ một lần, từ đó có thể thu được giải pháp cho bài toán ban đầu. [1]

Cốt lõi của quy hoạch động là hi sinh sự phức tạp về không gian đổi lại độ phức tạp về thời gian sẽ được cải thiện. Quy hoạch động không phải lúc nào cũng cần có sự can thiệp của đệ quy nhưng ta vẫn cần có một kiến trúc có khả năng tận dụng câu trả lời của các bài toán con đã được giải quyết trước đó.

Việc thiết kế một giải thuật quy hoạch động là một công đoạn không hề khó và ta có thể nhìn nhận vấn đề theo nhiều góc độ khác nhau. Phương pháp thứ nhất để thiết kế giải thuật với hướng tiếp cận quy hoạch động là ta có thể giải một bài toán với cấu trúc con tối ưu bằng một quy trình ba bước:

1. Chia bài toán thành các bài toán con nhỏ hơn.
2. Giải các bài toán này một cách tối ưu bằng cách sử dụng đệ quy quy trình ba bước này.
3. Sử dụng các kết quả tối ưu đó để xây dựng một lời giải tối ưu cho bài toán ban đầu.

Một cách nhìn khác về hướng tiếp cận quy hoạch động với Erik Demaine là việc phân tích một bài toán có thể chia ra thành 5 bước đơn giản đó là [3]:

1. Định nghĩa bài toán con.
2. Đoán đường đi đến trạng thái kế tiếp của bài toán con.
3. Tìm mối quan hệ giữa các bài toán con.
4. Đệ quy + ghi nhớ hoặc xây dựng một bảng ghi quy hoạch động để xác định các trật tự liên kết của các bài toán con (subproblem topological order).
5. Giải quyết bài toán gốc bằng một bài toán con hoặc kết hợp mỗi chuỗi các lời giải của các bài toán con

Examples:	Fibonacci	Shortest Paths
subprobs:	F_k for $1 \le k \le n$	$\delta_k(s,v)$ for $v \in V$, $0 \le k < \lvert V\rvert$ $= \min s \to v$ path using $\le k$ edges
# subprobs:	n	V^2
guess:	nothing	edge into v (if any)
# choices:	1	$\text{indegree}(v) + 1$
recurrence:	$F_k = F_{k-1} + F_{k-2}$	$\delta_k(s,v) = \min\{\delta_{k-1}(s,u) + w(u,v) \mid (u,v) \in E\}$
time/subpr:	$\Theta(1)$	$\Theta(1 + \text{indegree}(v))$
topo. order:	for $k = 1, \ldots, n$	for $k = 0, 1, \ldots \lvert V\rvert - 1$ for $v \in V$
total time:	$\Theta(n)$	$\Theta(VE)$ + $\Theta(V^2)$ unless efficient about indeg. 0
orig. prob.:	F_n	$\delta_{\lvert V\rvert-1}(s,v)$ for $v \in V$
extra time:	$\Theta(1)$	$\Theta(V)$

Hình. 2.2. Hệ thống hoá việc thiết kế giải thuật vớ hướng tiếp cận quy hoạch động theo quy tắc 5 bước của Erik Demaine

Các bài toán con được giải bằng cách chia chúng thành các bài toán nhỏ hơn, và cứ tiếp tục như thế, cho đến khi ta đến được trường hợp đơn giản dễ tìm lời giải. Và để tránh việc có các bài toán con trùng nhau, ta lưu trữ lời giải của các bài toán con đã giải. Do vậy, nếu sau này ta cần

giải lại chính bài toán đó, ta có thể lấy và sử dụng kết quả đã được tính toán. Nếu ta chắc chắn rằng một lời giải nào đó không còn cần thiết nữa, ta có thể xóa nó đi để tiết kiệm không gian bộ nhớ [1].

Kỹ thuật này có thể được minh họa cụ thể nhất với bài toán tìm dãy số Fibonacci

$$F\ (n) = F(n-1) + F(n-2) \text{ với } n > 1 \text{ và } F(0) = 0, F(1) = 1 \text{ [1]}$$

Quy hoạch động không phải lúc nào cũng là một thuật giải tốt để giải quyết một bài toán nhưng với quy hoạch động ta có thể tìm ra một hướng giải quyết chung cho các bài toán.

5. Hình học

Hình học đối với giác quan của con người thì khá quen thuộc và dễ nhận biết, còn hình học đối với máy tính thì lại không giống như vậy. Các giải thuật hình học thường được dùng để giải quyết các vấn đề về hình học như điểm, các đường thẳng, đa giác,… Có những bài toán ta phải giải quyết với chi phí thuật toán rất lớn đôi khi không thể chấp nhận được, nhưng nhờ vào những tính chất đặc biệt của hình học mà ta lại có thể giải quyết nó một cách hiệu quả và dễ dàng.[1]

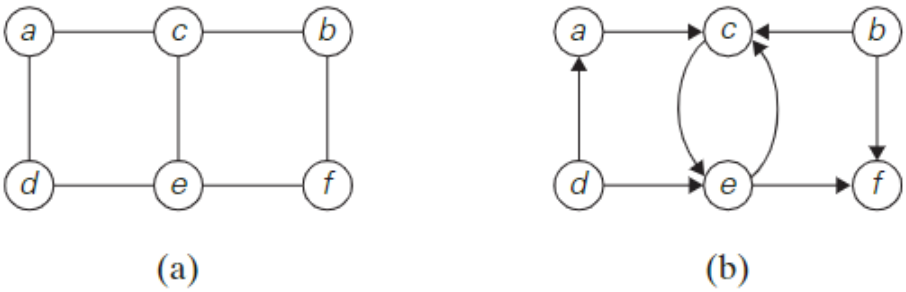
6. Đồ thị

Một trong những lĩnh vực lâu đời nhất và thú vị nhất trong thuật toán là đồ thị. Về mặt hình thức, đồ thị có thể được coi là tập hợp các điểm được gọi là đỉnh, một số điểm được nối với nhau bằng các đoạn thẳng gọi là cạnh [1].

Một đồ thị $G = \langle V, E \rangle$ bởi 2 thành phần là một tập V không rỗng được gọi là các đỉnh và một tập E được gọi là các cạnh. Thông thường, ta gán nhãn các đỉnh của đồ thị bằng các chữ cái và cặp chữ cái kết hợp nên thành cạnh. [1]

Đồ thị được mô tả trong Hình 6.1a có sáu đỉnh và bảy cạnh vô hướng

$$V = \{a,\ b,\ c,\ d,\ e,\ f\},\ E = \{(a,\ c),\ (a,\ d),\ (b,\ c),\ (b,\ f\ ),\ (c,\ e),\ (d,\ e),\ (e,\ f)\}$$



Hình. 6.1a,b. Đồ thị vô hướng và Đồ thị có hướng[1]

Đồ thị được mô tả trong Hình 1.6b có sáu đỉnh và tám cạnh có hướng:

$V = \{a, b, c, d, e, f\}$, $E = \{(a, c), (b, c), (b, f), (c, e), (d, a), (d, e), (e, c), (e, f)\}$

Đồ thị là một chủ đề thú vị để nghiên cứu, vì lý do lý thuyết và thực tiễn. Đồ thị có thể được sử dụng để mô hình hóa nhiều loại ứng dụng, bao gồm giao thông vận tải, truyền thông, mạng xã hội và kinh tế,… [1]

IV. CÁC BÀI TOÁN

I. Bài toán tìm đỉnh chóp

1. *Đặt vấn đề*

Cho một mảng các số nguyên. Tìm một phần tử đỉnh trong đó. Một phần tử mảng là đỉnh nếu nó KHÔNG nhỏ hơn các phần tử lân cận của nó.

2. *Input*

Một mảng các số nguyên $array = \{a_0, a_1, ..., a_n\}$

3. *Output*

Một mảng các số nguyên $array = \{a_0, a_1, ..., a_n\}$

VÍ DỤ

Input	**Output**
5, 10, 20, 15	20
10, 20, 15, 2, 23, 90, 67	20 or 90

4. *Thuật toán*

a. Ý tưởng

Các trường hợp sau cần chú ý khi giải bài toán

- Nếu mảng đầu vào được sắp xếp theo **thứ tự tăng dần**, phần tử **cuối cùng luôn là phần tử đỉnh**. Ví dụ: 50 là phần tử đỉnh trong {10, 20, 30, 40, 50}.
- Nếu mảng đầu vào được sắp xếp theo **thứ tự giảm dần**, phần tử **đầu tiên luôn là phần tử đỉnh**. 100 là phần tử đỉnh trong {100, 80, 60, 50, 20}.

- Nếu tất cả các phần tử của mảng đầu vào giống nhau thì mọi phần tử đều là phần tử đỉnh.

Ở đây ta thấy rằng, sẽ luôn tồn tại ít nhất một phần tử đỉnh từ mảng cho trước.

b. Hướng tiếp cận

Vét cạn (Brute Force)

- Nếu trong mảng, phần tử đầu tiên lớn hơn phần tử thứ hai hoặc phần tử cuối cùng lớn hơn phần tử cuối cùng thứ hai, in phần tử tương ứng và kết thúc chương trình.
- Nếu không duyệt mảng từ phần tử thứ hai đến phần tử kế cuối.
- Trong mảng $array[i]$, phần tử đỉnh sẽ lớn hơn cả hai phần tử láng giềng của nó, tức là $array[i] > array[i-1]$ và $array[i] < array[i+1]$, in phần tử đó và kết thúc.

Độ phức tạp về thời gian: $O(n)$ vì sử dụng 1 vòng lặp *for*.

Độ phức tạp không gian: $O(1)$ vì không sử dụng thêm vùng nhớ.

Chia để trị

Đối với bài toán tìm tất cả đỉnh chóp thì việc áp dụng tìm nhị phân là không thể thực thi vì ta cần phải tìm tất cả các đỉnh và tìm nhị phân sẽ chia mảng thành hai phần và chỉ tập trung vào một bên của mảng. Tuy nhiên, nếu yêu cầu của bài toán được chỉnh sửa lại thành chỉ tìm một đỉnh chóp thì bài toán có thể được tiếp cận theo phương pháp tương tự với tìm nhị phân.

5. *Cài đặt*

```
int findPeak(int arr[], int n)
{
    // first or last element is peak element
    if (n == 1)
      return arr[0];
    if (arr[0] >= arr[1])
        return 0;
    if (arr[n - 1] >= arr[n - 2])
        return n - 1;

    // check for every other element
    for (int i = 1; i < n - 1; i++) {

        // check if the neighbors are smaller
        if (arr[i] >= arr[i - 1] && arr[i] >= arr[i + 1])
            return i;
    }
}
```

Hình. 4.1. Cài đặt tìm đỉnh chóp với hướng tiếp cận vét cạn

II. Bài toán canh lề văn bản

1. *Đặt vấn đề*

Cho một danh sách A chứa những từ thuộc ngôn ngữ tự nhiên và một số nguyên b cho biết giới hạn về số lượng ký tự có thể chứa trong một dòng của văn bản. Bài toán đặt ra là bố trí các từ một cách thẩm mỹ vào các hàng để tạo nên một văn bản hoàn chỉnh.

2. *Input*

Một danh sách A

- Một từ trong danh sách A không được tồn tại kí tự " "
- Độ dài mỗi từ phải được đảm bảo là lớn hơn 0
- Đầu vào chương trình là A phải chứa ít nhất một kí tự

và số nguyên b (maxWitdth)

Điều kiện:

- $1 \leq A.length \leq 300$
- $1 \leq A[i].length \leq 300$
- $1 \leq b \leq 300$

3. *Output*

Văn bản đã canh lề có thẩm mỹ

VÍ DỤ

```
Input: words = ["This", "is", "an", "example", "of", "text", "justification."], maxWidth = 16
Output:
[
   "This    is    an",
   "example  of text",
   "justification.  "
]
```

4. *Thuật toán*

a. Ý tưởng

Trong quá khứ, Microsoft đã tiếp cận vấn đề bằng **phương pháp tham lam** nhưng phương pháp này được nhận định là không hiệu quả. Với hướng tiếp cận tham lam, việc căn chỉnh văn phải được định nghĩa là phải chứa tất cả các từ vào một dòng cho đến khi đạt đến giới hạn b. Tuy nhiên, với hướng tiếp cận tham lam như thế thì sẽ dễ dàng sảy ra trường hợp như sau:

Giả sử như ta có một dãy kí tự A = ["blah", "blah", "blah", "blah", "somethinglongs"] với b = 14. Khi căn chỉnh với ý tưởng của phương pháp tham lam thì ta có:

```
blah blah blah
blah
somethinglongs
```

Tuy nhiên, cách căng chỉnh văn bản mà ta muốn nên như thế này:

```
blah      blah
blah      blah
somethinglongs
```

b. Hướng tiếp cận

Quy hoạch động(Dynamic Programming)

Quy hoạch động là một phương pháp thiết kế giải thuật với mục tiêu chính là hi sinh sự độ phức tạp về không gian nhưng đổi lại có thể cải thiện độ phức tạp về thời gian(tuy nhiên, số mức độ hi sinh của không gian không đáng kể so với thời gian, nên quy hoạch động là một phương pháp cho ra một hướng giải quyết một bài toán theo hướng chung(general) đáng để sử dụng nếu được cài đặt một cách khôn ngoan). Chúng ta có thể lưu trử giá trị mỗi lần tính toán và gọi lại thay vì phải tái tính toán cho bước tính toán đó lần nữa.[1]

Trong LaTex, một hướng tiếp cận được đặt ra là tìm ra một công thức để biểu diễn độ thẩm mỹ hoặc ngược lại là độ mất thẩm mỹ của một dòng được tạo ra bằng cách lấy ra các từ trong danh sách A.

5. *Cài đặt*

Vì chương trình cài đặt phức tạp, vui lòng tham khảo thêm ở github của nhóm.

III. Bài toán mã đi tuần

1. *Đặt vấn đề*

Quân mã là một quân cờ đặt biệt trên bàn cờ với nước đi hình chữ L nó có thể xoay chuyển thế trận một cách đầy bất ngờ.

Trong một lần đánh cờ với chính bản thân, James Moriaty tự hỏi mất bao nhiêu bước để một quân Mã có thể đi hết bàn cờ như một phép ẩn dụ về bản thân cần bao nhiêu mưu đồ để nắm trọn Luân Đôn trong bàn tay.Các hành động được phát sinh lần lượt theo thứ tự: $\{-1,2\}$, $\{1,2\}$, $\{2,1\}$, $\{2,-1\}$, $\{1,-2\}$, $\{-1,-2\}$, $\{-2,\ -1\}$, $\{-2,\ 1\}$.Hãy giúp James tạo ra một chương trình nhập liệu giúp tính toán số lượng bước đi cần thiết để thôn tính bàn cờ.

2. *Input*

Hàng đầu tiên chứa 02 số nguyên dương m, n là kích thước bàn cờ $(0 \le m, n \le 1\times10^6)$

Hàng tiếp theo chứa một chuỗi ghi tọa độ ô đầu tiên mã đang đứng $(0 \le x < m)$ và $(0 \le y < n)$.

3. *Output*

Số lượng các bước mà quân Mã cần phải di chuyển.

VÍ DỤ

Input	**Output**
12 12 2 3	143
1 3 1 1	0

4. *Thuật toán*

a. Ý tưởng

Định lý Schwenk:

Cho bàn cờ $m\times n$ bất kỳ với $m \le n$, không có hành trình đóng nào của quân mã nếu một trong ba điều kiện dưới đây xảy ra:

- *Điều kiện 1:* m, n đều là số lẻ.
- *Điều kiện 2:* $m = 1, 2, 4$; $n \ne 1$.
- *Điều kiện 3:* $m = 3$; $n = 4, 6, 8$.

b. Hướng tiếp cận

Xác định có tồn tại hành trình đóng trên bàn cờ hay là không.

5. *Cài đặt*

```python
m, n = [int(i) for i in input().split()]
curr_x, curr_y = [int(i) for i in input().split()]

def knight(m, n, curr_x, curr_y):
    if m%2 != 0 and n%2 != 0:
        print(0)
        return
    if m == 1 or m == 2 or m == 4:
        print(0)
        return
    if m == 3:
        if n == 4 or n == 6 or n == 8:
            print(0)
            return
    if curr_x > m or curr_x < 0:
        print(0)
        return
    if curr_y > n or curr_y < 0:
        print(0)
        return

    print(m*n - 1)
    return

knight(m, n, curr_x, curr_y)
```

Hình.4.2. Cài đặt thuật giải cho bài toán căn chỉnh văn bản

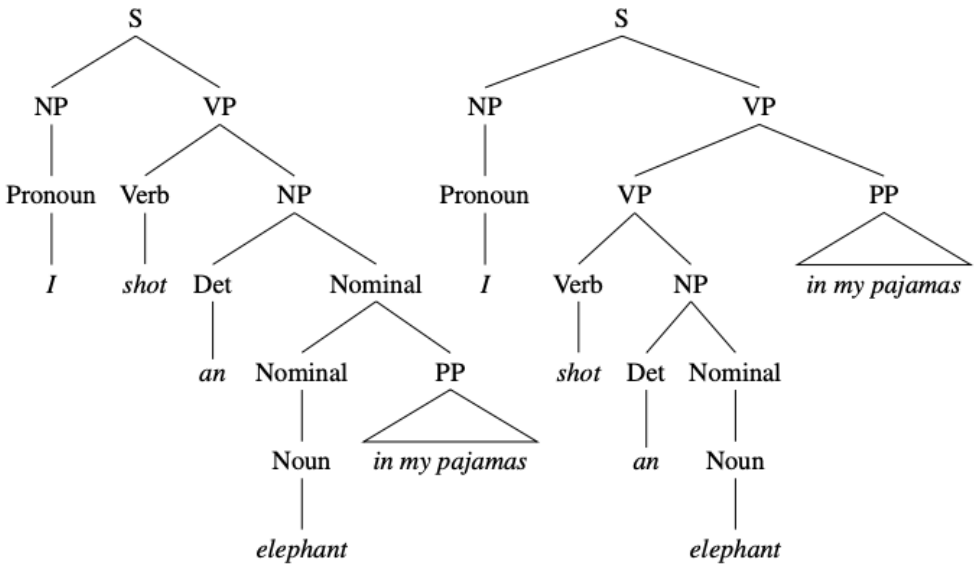
Test Case	**Giải thích**
0	Trường hợp hiển nhiên nhằm xác định tính đúng của chương trình
1	Xác định thoả điều kiện 1 của định lý schwerk
2	Xác định thoả điều kiện 2 của định lý schwerk
3	Xác định thoả điều kiện 3 của định lý schwerk

Test Case	**Giải thích**
4	Kiểm tra thời gian thực thi
5	Kiểm tra thời gian thực thi

V. ỨNG DỤNG

1. Phân tích cú pháp ngữ đoạn

Sự nhập nhằng về cấu trúc sảy ra khi văn phạm có thể gán cho ngữ đoạn nhiều hơn một danh tính trong một câu. Câu nói nổi tiếng của Groucho Marx thủ vai thuyền trưởng Spaulding trong Animal Cracker (Hình. 1.5) được là một trường biểu diễn cho sự nhập nhằng trong cấu trúc. Với từ in my pajamas đều có thể là một phần của cụm danh từ (NP) [2].



Hình. 5.1.Câu nói của thuyền trưởng Groucho trong bộ Animal Cracker khi phân tích thành cấu trúc ngữ đoạn [2]

Quy hoạch động (Dynamic programming) cung cấp một hướng giải quyết hiệu quả cho vấn đề nhập nhằng. Trong hướng tiếp cận quy hoạch động, bằng cách lưu lại các kết quả của các bài toán con, ta có thể quay lại và giải quyết tiếp những bài toán con. Trong bộ phân tích cú pháp, các bài toán con được thể hiện dưới dạng những cây văn phạm con được sinh ra trong quá trình phân tích [2].

Hướng tiếp cận quy hoạch động được sinh ra từ bản chất phi ngữ cảnh vốn có của quy tắc văn phạm. Khi một ngữ đoạn được phát hiện ra trong một phân đoạn của dữ liệu đầu vào. Sẽ được lưu trữ và luôn sẵn sàng cho bất kì giai đoạn mở rộng cần đến nó [2]. Từ các bước tính toán trên, bộ phân tích cú pháp có thể tìm ra các cây phân tích có thể có.

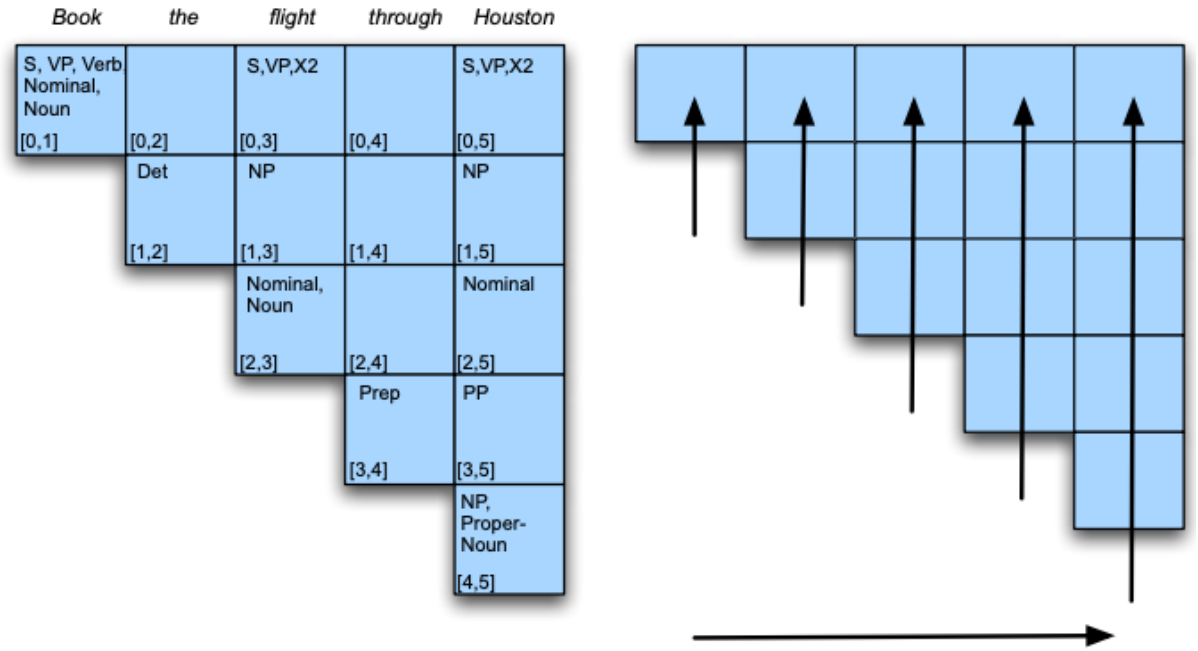
Thuật toán CKY sẽ được biểu diễn dưới dạng mã giả như sau:

```
function CKY-PARSE(words, grammar) returns table

for j ← from 1 to LENGTH(words) do
    for all {A | A → words[j] ∈ grammar}
            table[j − 1, j] ← table[j − 1, j] ∪ A
    for i ← from j − 2 down to 0 do
        for k ← i + 1 to j − 1 do
            for all {A | A → BC ∈ grammar and B ∈ table[i,k] and C ∈ table[k, j]}
                    table[i,j] ← table[i,j] ∪ A
```

Hình. 5.2. Giải thuật CKY [2]

Với chuỗi văn bản có chiều dài n thuật giải CKY sẽ xây dựng một ma trận tam giác vuôn cận trên với kích thước (n + 1) x (n + 1). Mỗi ô thứ [i j] trong ma trận thể hiện một ngữ đoạn [2] (Hình. 2.2).



Hình. 5.3. Kết quả một văn phạm được phân tích [2]

2. Kali - Linux

Trong Kali-linux, việc dò mật khẩu wifi hiện tại đang được tiếp cận theo hướng brute force. Bằng cách dò tất cả những tổ hợp có thể có từ một chuỗi ký tự cho trước, thuật toán brute force có thể dò ra được mật khẩu wifi của một router với điều kiện các ký tự trong mật khẩu wifi thuộc chuỗi ký tự được cho trước.

VI. KẾT LUẬN

Cấu trúc dữ liệu và các chiến lược thiết kế thuật toán là các lĩnh vực nghiên cứu gắn liền với nhau và là một trong những lĩnh vực nghiên cứu lâu đời của khoa học máy tính. Qua những phân hiểu biết và phân tích các chiến lược xây dựng thuật toán, chúng tôi hy vọng sẽ cung cấp cho các lập trình viên có nền tảng lý thuyết vững chắc và có nhiều lựa chọn hơn trong việc đưa ra các giải pháp cho các bài toán thực tế.

THAM KHẢO

[1] Anany Levitin, *"The design and Analasis of Algorithm – 3rd Edition", 2012.*

[2] Dan Jurafsky, *"Speech and Language Processing - 3rd Edition", 2020.*

[3] Erik Demaine, *"Dynamic Programming lecture MITopencourseware 6.006".*