

# Brute Force

Group 09

1. Lê Hữu Trung
2. Nguyễn Đỗ Mạnh Cường

01

Giới thiệu

02

Bài toán ví dụ

03

Ưu và nhược điểm

04

Ứng dụng

05

Tổng kết

06

Bài tập

# Table of Contents

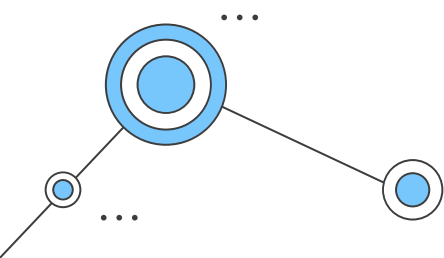
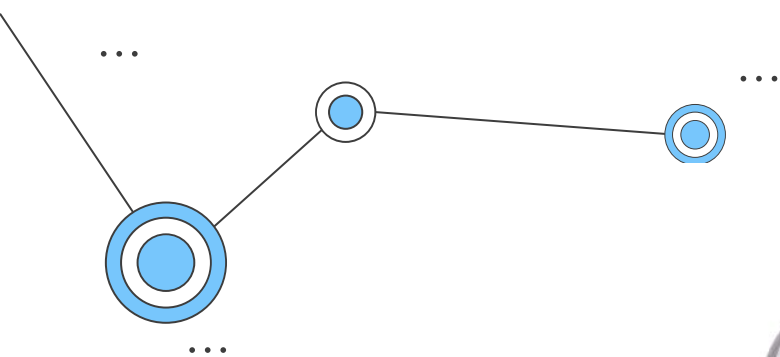




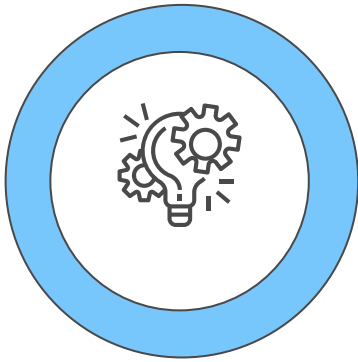
# 01

## Giới thiệu





**Đặt vấn đề:** Làm sao biết được chìa nào trong chùm khoá trên mở được ổ khoá?



## Khái niệm


Vét cạn là một cách tiếp cận đơn giản để giải quyết một vấn đề, thường trực tiếp dựa trên các phát biểu của vấn đề đó và định nghĩa của các khái niệm liên quan

Hay nói một cách đơn giản hơn, vét cạn là phương pháp tìm nghiệm của một bài toán bằng cách xem xét **tất cả** các trường hợp có thể xảy ra

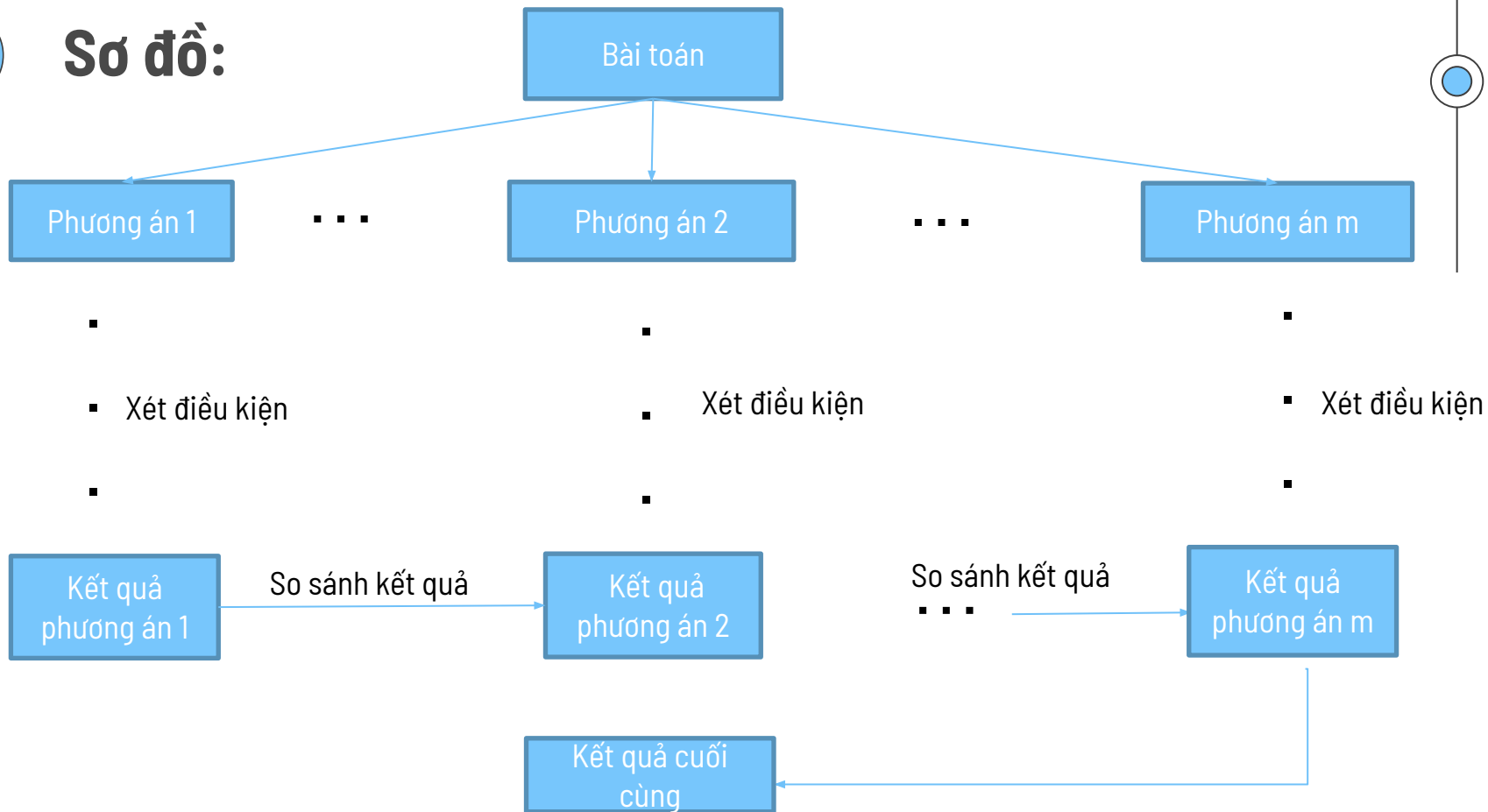
...



## Ý tưởng:

- Một bài toán có  $m$  phương án.
  - Thực hiện vòng lặp qua từng phương án.
  - Xét điều kiện từng phương án để đưa ra kết quả của phương án.
  - So sánh kết quả của phương án đó với các phương án trước (nếu là bài toán tối ưu) để đưa ra kết quả cuối cùng
- ...
- 

## Sơ đồ:



# Dạng tổng quát

```
c ← first(P)
```

```
while c ≠  $\Lambda$  do
```

```
  if valid(P, c) then
```

```
    output(P, c)
```

```
    c ← next(P, c)
```

```
  end while
```

Vòng lặp để duyệt qua các trường hợp

Xét điều kiện để đưa ra kết quả thích hợp



Bản chất của vét cạn là sử dụng vòng lặp duyệt qua **tất cả** các trường hợp của bài toán. Sau đó xét điều kiện của các trường hợp để tìm ra được kết quả cuối cùng.





02

**Bài toán ví dụ**



# Linear Search

**Đề bài:** Cho dãy các số 10, 14, 19, 26, 27, 31, 33, 35, 42, 44. Tìm vị trí phần tử 33 trong dãy.

Linear Search



# Linear Search

**Đề bài:** Cho dãy các số 10, 14, 19, 26, 27, 31, 33, 35, 42, 44. Tìm vị trí phần tử thứ 33 trong dãy.

**Thuật toán:**

```
def linearSearch(a,n):  
    for i in range (len(a)):  
        if(a[i]==n):  
            return i+1  
    return -1
```

```
 $c \leftarrow \text{first}(P)$   
while  $c \neq \Lambda$  do  
    if  $\text{valid}(P, c)$  then  
         $\text{output}(P, c)$   
         $c \leftarrow \text{next}(P, c)$   
end while
```

**Độ phức tạp thuật toán:**  $O(n)$

# Selection Sort


**Đề bài:** Cho dãy các số 15, 19, 27, 33, 46, 50, 5, 9, 37. Sắp xếp chúng theo thứ tự tăng dần.

```
def selectionSort(a):  
    # Duyệt qua các trường hợp  
    for i in range(len(a)-1):  
        min_idx=i  
        for j in range(i+1,len(a),1):  
            # Xét điều kiện  
            if a[min_idx]>a[j]:  
                min_idx=j  
        a[i], a[min_idx] = a[min_idx], a[i]  
    return a
```

```
c ← first(P)  
while c ≠ Λ do  
    if valid(P,c) then  
        output(P, c)  
        c ← next(P, c)  
end while
```

**Độ phức tạp thuật toán:**  $O(n^2)$


...

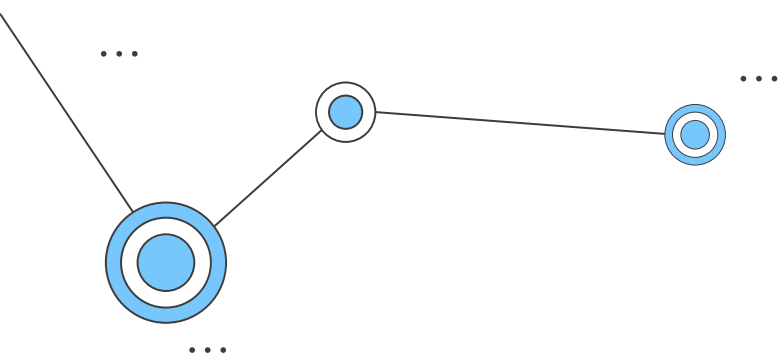


```
def selectionSort(a):  
    # Duyệt qua các trường hợp  
    for i in range(len(a)-1):  
        min_idx=i  
        for j in range(i+1,len(a),1):  
            # Xét điều kiện  
            if a[min_idx]>a[j]:  
                min_idx=j  
        a[i], a[min_idx] = a[min_idx], a[i]  
    return a
```

...

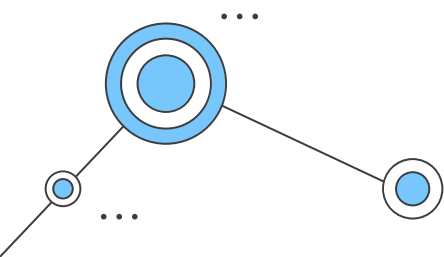
Bài toán tìm giá trị nhỏ nhất trong mảng. Một trong các ví dụ cơ bản nhất của vét cạn.





Chạy tay minh họa:

	15	19	27	33	46	50	<b>5</b>	9	37
5		19	27	33	46	50	15	<b>9</b>	37
5	9		27	33	46	50	<b>15</b>	19	37
5	9	15		33	46	50	27	<b>19</b>	37
5	9	15	19		46	50	<b>27</b>	33	37
5	9	15	19	27		50	46	<b>33</b>	37
5	9	15	19	27	33		46	50	<b>37</b>
5	9	15	19	27	33	37		50	<b>46</b>
5	9	15	19	27	33	37	46		<b>50</b>



# Bubble Sort

**Đề bài:** Cho dãy các số 15, 19, 27, 33, 46, 50, 5, 9, 37. Sắp xếp chúng theo thứ tự tăng dần.

**Thuật toán:**

```
def bubbleSort(arr):  
    n = len(arr)  
    # Duyệt qua các trường hợp  
    for i in range(n-1):  
        for j in range(0, n-i-1):  
            # Xét điều kiện  
            if arr[j] > arr[j+1]:  
                arr[j], arr[j+1] = arr[j+1], arr[j]  
    return arr
```

```
 $c \leftarrow \text{first}(P)$   
while  $c \neq \Lambda$  do  
    if  $\text{valid}(P, c)$  then  
         $\text{output}(P, c)$   
         $c \leftarrow \text{next}(P, c)$   
end while
```

**Độ phức tạp thuật toán:**  $O(n^2)$

# Nhận xét

## So sánh với chia để trị (Quick sort / Merge sort):

### Ưu điểm:

- Dễ cài đặt (chỉ với vài dòng for)

### Nhược điểm:

- Độ phức tạp thuật toán lớn ( $O(n^2)$ ) trong khi với Quick sort / Merge sort độ phức tạp là  $O(n\log(n))$



Vét cạn là phương pháp đầu tiên và đơn giản nhất  
ta nên nghĩ tới khi giải quyết một bài toán

...

\* [Code tham khảo](#)



# String matching

**Bài toán:** Cho một văn bản  $T[1,2,\dots,n]$  có chiều dài  $n$  và một chuỗi mẫu  $P[1,2,\dots,m]$ . Tìm vị trí đầu tiên của chuỗi  $P$  trong văn bản  $T$ , nếu không trả về  $-1$

## Ý tưởng:

- Bắt đầu từ vị trí đầu tiên của văn bản  $T$ , dịch dần chuỗi  $P$  qua bên phải từng ký tự một.
- Nếu ta dịch đến ký tự thứ  $i$ , so sánh chuỗi  $T[i,i+1,\dots,i+m-1]$  với chuỗi  $P[1,2,\dots,m]$  nếu bằng thì trả về  $i$ .
- Nếu không thì dịch đến thứ tự  $i+1$ .

Minh hoạ:

T	H	I	S		I	S		A		S	I	M	P	L	E		E	X	A	M	P	L	E
---	---	---	---	--	---	---	--	---	--	---	---	---	---	---	---	--	---	---	---	---	---	---	---

S	I	M	P	L	E																		
	S	I	M	P	L	E																	
		S	I	M	P	L	E																
			S	I	M	P	L	E															
				S	I	M	P	L	E														
					S	I	M	P	L	E													
						S	I	M	P	L	E												
							S	I	M	P	L	E											
								S	I	M	P	L	E										
									S	I	M	P	L	E									
										S	I	M	P	L	E								

## Thuật toán:

```
def string_match(string, sub_str):  
    # Duyệt qua các trường hợp  
    for i in range(len(string)-len(sub_str)+1):  
        count = 0  
        for j in range(len(sub_str)):  
            # Xét điều kiện  
            if string[i+j] == sub_str[j]:  
                count += 1  
            else:  
                break  
        if count == len(sub_str):  
            return i  
    return -1
```

```
 $c \leftarrow \text{first}(P)$   
while  $c \neq \Lambda$  do  
    if  $\text{valid}(P, c)$  then  
         $\text{output}(P, c)$   
         $c \leftarrow \text{next}(P, c)$   
end while
```

Độ phức tạp thuật toán:  $O(n^2)$

# Knapsack 0/1 Problem

## Ý tưởng:

- Một tập hợp có  $n$  phần tử có  $2^n$  tập con
- Tính tổng weight, value trong từng tập hợp con
- Tập con có tổng value lớn nhất và thoả điều kiện (weight  $\leq W$  với  $W$  là weight tối đa có thể) chính là kết quả tối ưu mà ta cần tìm

## Thuật toán:

```
class subset():
    def __init__(self, lst):
        self.list = lst
    def weight(self):
        sum = 0
        for i in self.list:
            sum += w[i]
        return sum
    def value(self):
        sum = 0
        for i in self.list:
            sum += v[i]
        return sum
def knapsack_01(w,v,W):
    max_value = 0
    n = len(w)
    total_cases = 1 << n
    for i in range(total_cases):
        lst = []
        for j in range(n):
            if i & (1 << j):
                lst.append(j)
        a = subset(lst)
        val_sub = a.value()
        if (a.weight() <= W and val_sub > max_value):
            max_value = val_sub
    return max_value
```

Độ phức tạp thuật toán:  $O(n \cdot 2^n)$

```
c ← first(P)
while c ≠ Λ do
    if valid(P,c) then
        output(P, c)
        c ← next(P, c)
end while
```

# Nhận xét

**So sánh với tham lam:**

**Ưu điểm:**

- Knapsack 0/1 là nhược điểm của tham lam, nên nếu sử dụng vét cạn thì ta đưa ra được kết quả chính xác

**Nhược điểm:**

- Độ phức tạp thuật toán lớn  $O(n \cdot 2^n)$  trong khi với tham lam độ phức tạp là  $O(n \log(n))$



Kết quả của phương pháp vét cạn có độ chính xác cao

...

\* [Code tham khảo](#)

# Exponentiation Problem

Cho 2 số nguyên không âm  $a, b$  với  $a \neq 0, 10^5 \leq b \leq 10^{18}$ .

Tính  $a^b \bmod (10^9 + 7)$ .

[Code minh họa](#)

# Exponentiation Problem

(Divide and Conquer Strategy)

**Solution:**

$$\begin{aligned} a^b &= a^{b/2} * a^{b/2} \\ &= a^{b/4} * a^{b/4} * a^{b/4} * a^{b/4} \\ &= \dots \\ &= \underbrace{a * a * \dots * a}_{b\text{-times}} \end{aligned}$$

Từ đó,

$$a^b = \begin{cases} a^{b/2} * a^{b/2} & (\text{nếu } b \text{ chẵn}) \\ a^{b/2} * a^{b/2} * a & (\text{nếu } b \text{ lẻ}) \end{cases}$$

[Code minh họa](#)



# Exponentiation Problem

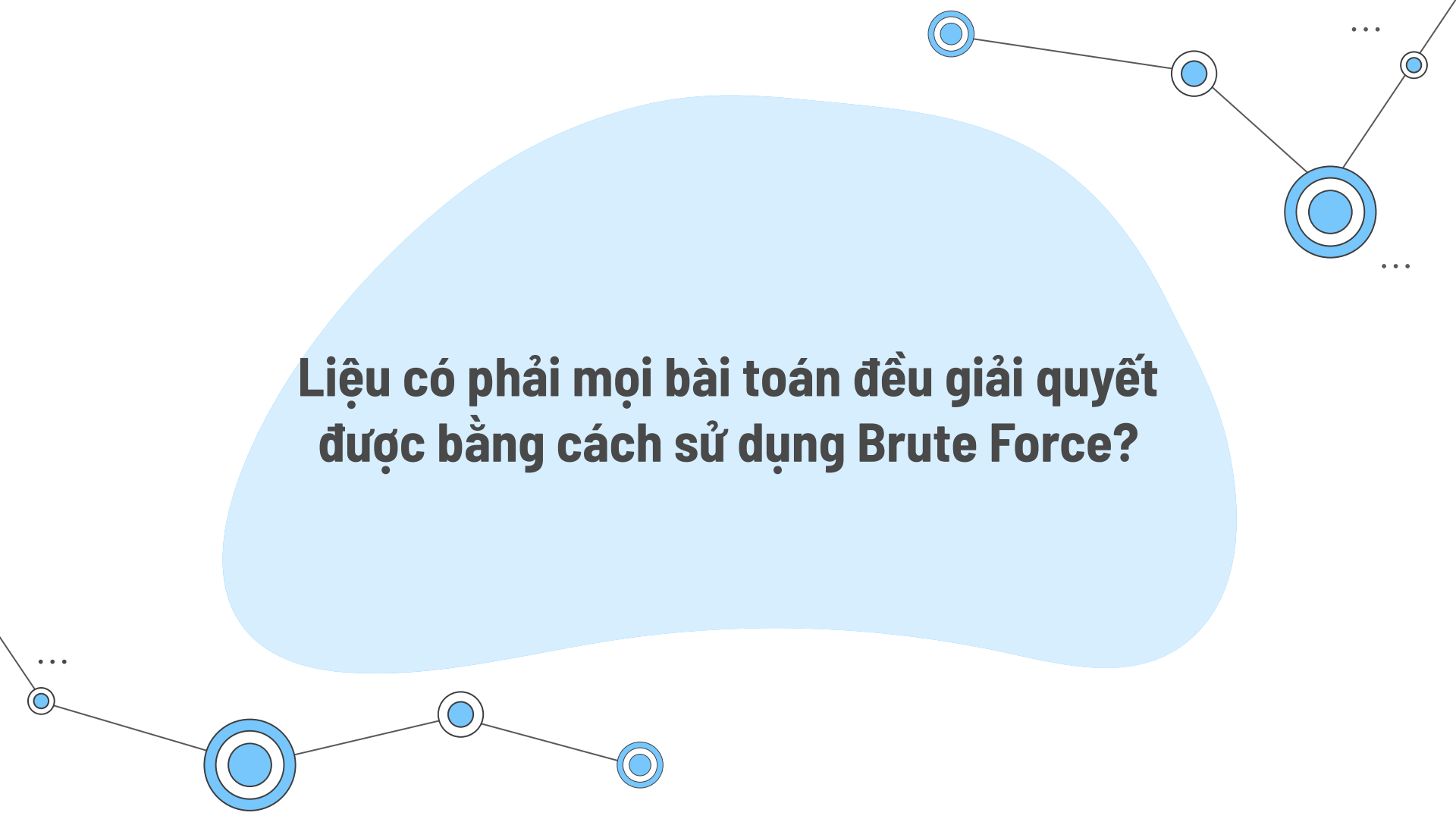
(Brute Force strategy)

**Solution:**

$$a^b = \underbrace{a * a * \dots * a}_{b\text{-times}}$$

...

[Code minh họa](#)



**Liệu có phải mọi bài toán đều giải quyết  
được bằng cách sử dụng Brute Force?**

# Nonlinear Equations

Giải phương trình sau bằng cách sử dụng Brute Force:  
 **$X^2 + 2X + 2 = 0 (X \in \mathbb{R})$**

...

# Nonlinear Equations

$$X^2 + 2X + 2 = 0 \quad (X \in \mathbb{R})$$

Nhận xét: **R** là một tập số vô hạn, do đó không thể quét hết không gian nghiệm để tìm ra nghiệm cho phương trình này. Vì vậy, không thể giải phương trình này bằng cách sử dụng Brute Force với một tập số vô hạn như vậy. Tuy nhiên, khi giải bài toán này bằng máy tính, với một sai số nhất định đủ nhỏ, ta vẫn có thể dùng Brute Force để tìm xấp xỉ nghiệm của phương trình.



03

**Ưu và nhược điểm**



# Ưu điểm



Dễ tiếp cận



Dễ cài đặt



Độ chính xác cao

## Nhược điểm



Độ phức tạp thuật toán tỉ lệ  
với độ lớn không gian  
nghiệm (thường lớn hơn  
những hướng tiếp cận khác)



Tốn thời gian khi  
giải quyết những  
bài toán có miền  
dữ liệu lớn



04

Ứng Dụng





# TESTCASE GENERATOR

TESTCASE 0

TESTCASE 1

TESTCASE 2



Độ chính xác cao

# BRUTE FORCE ATTACK



Password Space:

- Lowercase + Uppercase:  $26 + 26$
- Number: 10
- Symbols: 34

Number of cases = Number in password space<sup>Number of</sup>  
: characters



05

**Tổng Kết**





**Khi nào thì nên sử dụng Brute Force?**



# Khi nào thì nên sử dụng Brute Force?



Những bài toán đơn giản,  
xác định được không gian  
nghiệm



Bài toán yêu cầu tính đơn giản  
và độ chính xác cao hơn so  
với thời gian chạy

## Bài Tập Vận Dụng: Closest-Pair Problem

**Đề bài:** Cho một mảng các điểm trong mặt phẳng  $p_1(x_1, y_1), p_2(x_2, y_2), \dots, p_n(x_n, y_n)$ . Tìm và in ra 2 điểm có khoảng cách gần nhất và khoảng cách đó.

# Bài Tập Vận Dụng: Closest-Pair Problem

**Input:** Một mảng các điểm trong mặt phẳng  $p_1(x_1, y_1), p_2(x_2, y_2), \dots, p_n(x_n, y_n)$ .

**Output:** 2 điểm có khoảng cách gần nhất và khoảng cách đó.

# Bài Tập Vận Dụng: Closest-Pair Problem

**Solution:**

$d \leftarrow \infty$

**for**  $i \leftarrow 1$  **to**  $n - 1$  **do**

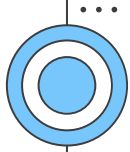
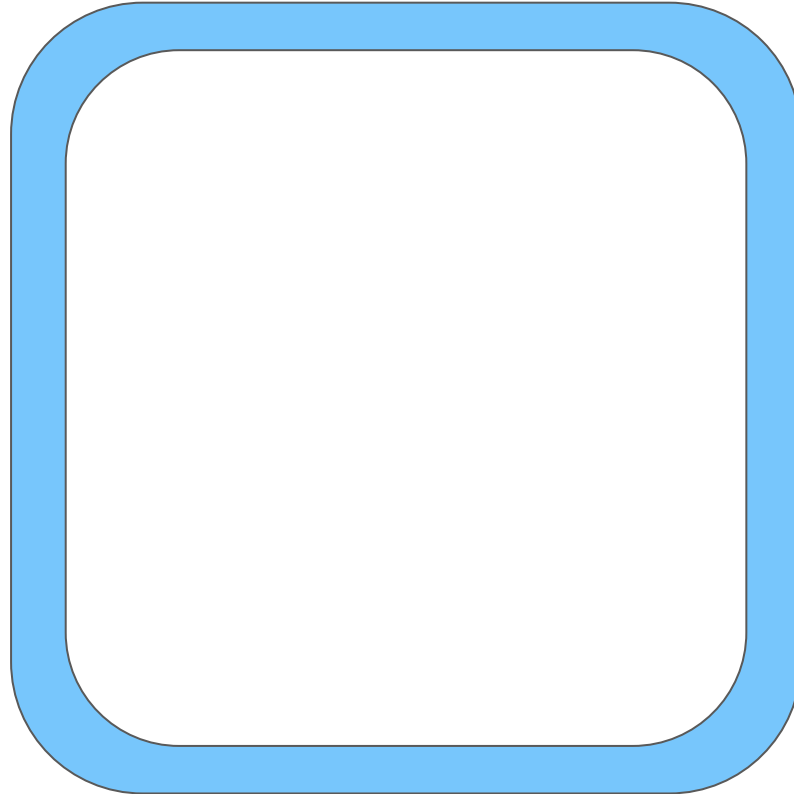
**for**  $j \leftarrow i + 1$  **to**  $n$  **do**

$d \leftarrow \min(d, \text{sqrt}((x_i - x_j)^2 + (y_i - y_j)^2))$

**return**  $d$



# MINIGAME KAHOOT





06

**Bài Tập**



# Longest Palindrome Substring



<https://www.hackerrank.com/brute-force-homework-l21-khtn-n09>

# Tài Liệu Tham Khảo

[1] Anany Levitin, Introduction to the Design and Analysis of Algorithms, 3rd Edition, 2014



# Thanks!

Do you have any questions?

**CREDITS:** This presentation template was created by [Slidesgo](#), including icons by [Flaticon](#), infographics & images by [Freepik](#) and illustrations by [Stories](#)

