

Backtracking

Thuật toán quay lui

Bạn còn nhớ hay đã quên?

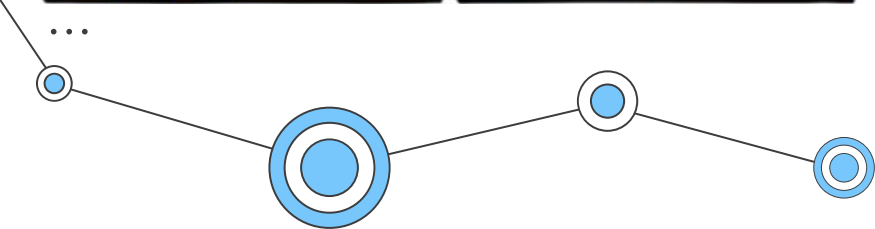
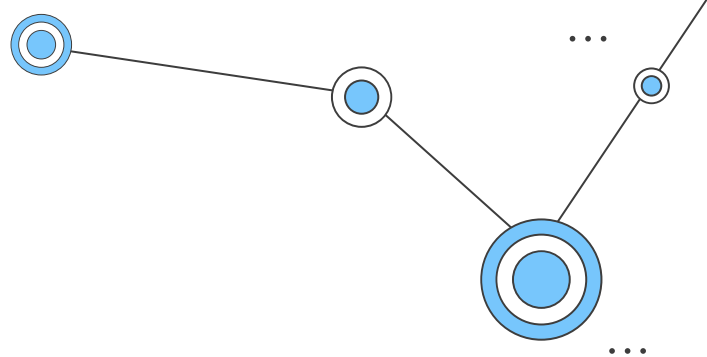
Đệ quy



Đệ quy

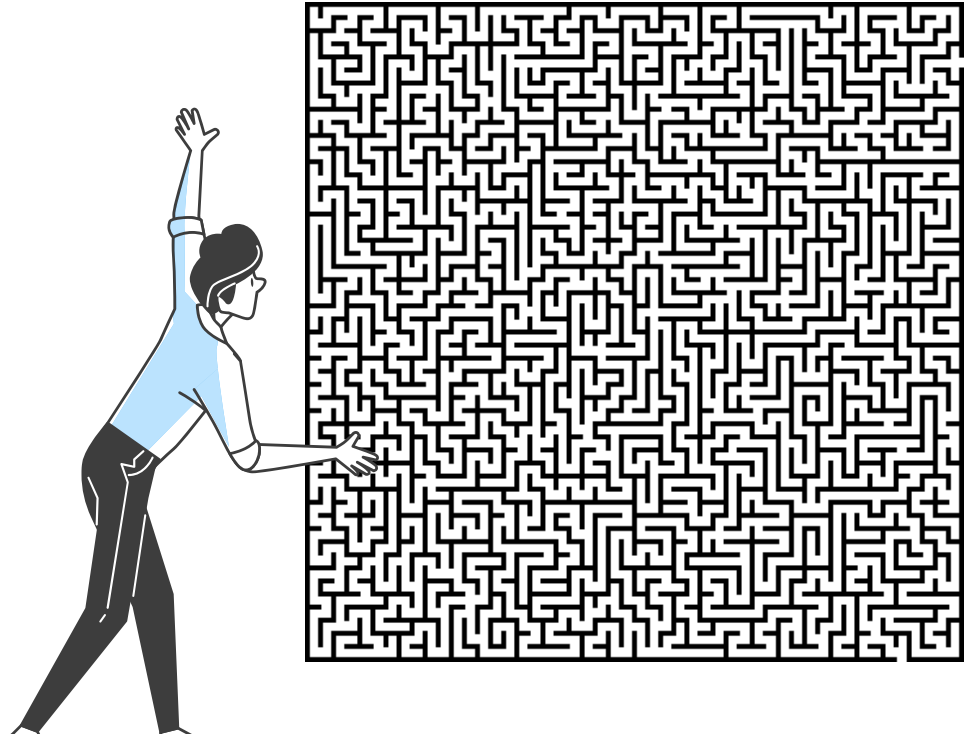
Đệ quy là một kỹ thuật lập trình cho phép một hàm gọi chính nó để giải quyết một vấn đề.

Kỹ thuật này thường được sử dụng để giải quyết các vấn đề liên quan đến tìm kiếm và liệt kê các giải pháp có thể.

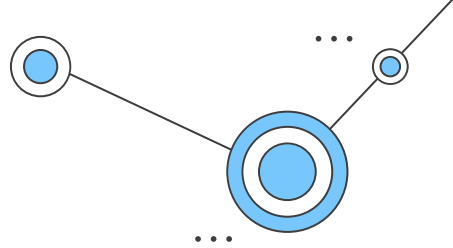


Thử thách quá là khó

*Có thể hay không việc giải trò chơi
mê cung và thành công ngay trong
lần đầu tiên?*

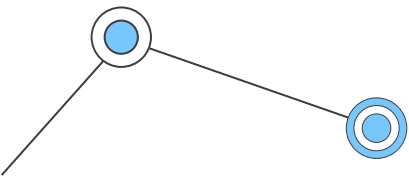
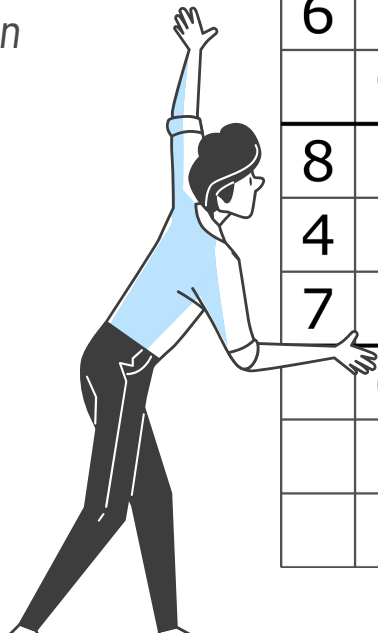


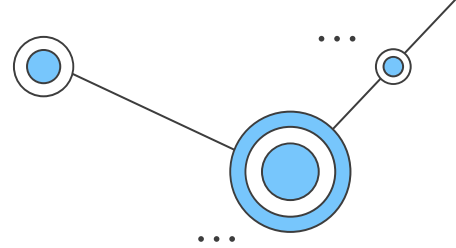
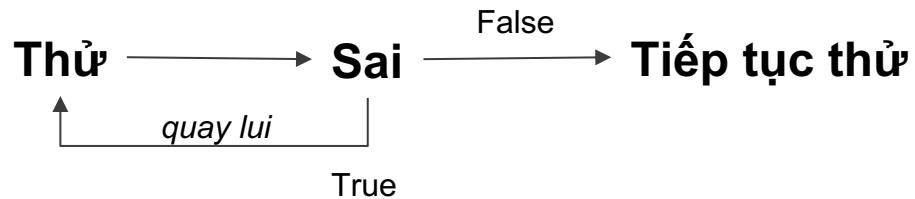
Thử thách quá là khó



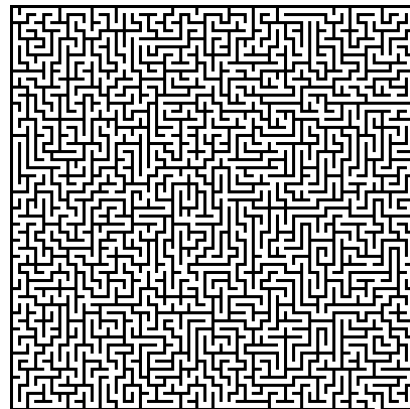
Không được quyền đi lại, bạn có thể giải trò chơi Sudoku trong một lần chơi duy nhất?

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

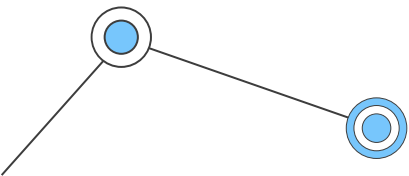




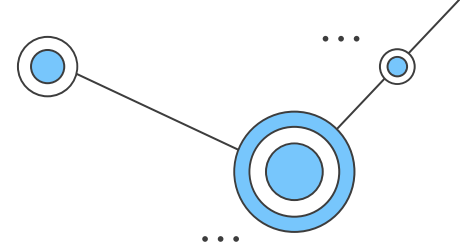
5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9



Back Tracking - Quay lui



Contents



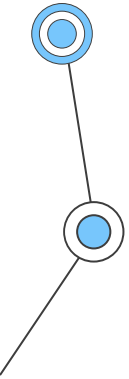
01 Giới thiệu
Backtraking là gì?

02 Ý tưởng
Backtracking hoạt động như thế nào?

03 Nhận diện
Khi nào thì sử dụng backtracking

04 Thuật toán
Thuật toán backtracking ra sao?

05 Nhận xét
Ưu - nhược điểm của backtracking





01

Giới thiệu

Backtracking là gì?

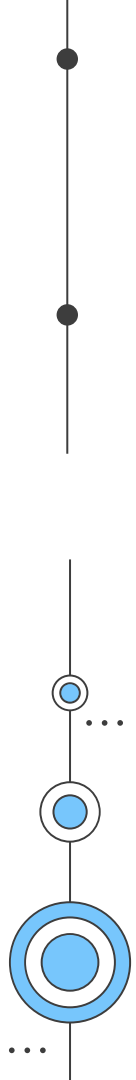


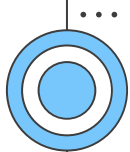


02

Ý tưởng

Backtracking hoạt động như thế nào?

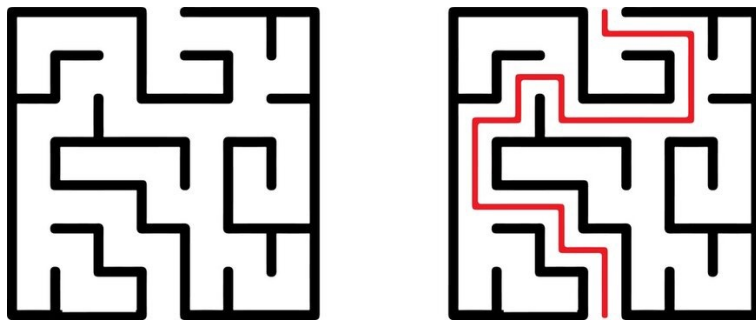


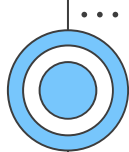


2. Ý tưởng

Xét lại ví dụ mê cung:

Thuật toán backtracking được sử dụng để giải quyết vấn đề này bằng cách **thử từng bước đi** và **quay lại** nếu không tìm thấy lối ra. Thuật toán sẽ tiếp tục thử các bước khác cho đến khi tìm thấy lối ra hoặc không còn bước nào để thử.





2. Ý tưởng

Backtracking hoạt động như thế nào?

❑ Dùng để giải bài toán liệt kê các cấu hình. Mỗi cấu hình được xây dựng bằng từng phần tử. Mỗi phần tử lại được chọn bằng cách thử tất cả các khả năng

❑ Các bước trong việc liệt kê cấu hình dạng $X[1...n]$:

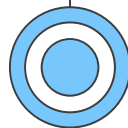
1. Xét tất cả các giá trị $X[1]$ có thể nhận, thử $X[1]$ nhận các giá trị đó. Với mỗi giá trị của $X[1]$ ta sẽ:

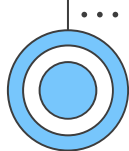
2. Xét tất cả giá trị $X[2]$ có thể nhận, lại thử $X[2]$ cho các giá trị đó. Với mỗi giá trị $X[2]$ lại xét khả năng giá trị của $X[3]$...tiếp tục như vậy cho tới bước:

...

n. Xét tất cả giá trị $X[n]$ có thể nhận, thử cho $X[n]$ nhận lần lượt giá trị đó.

Kết luận: Thông báo cấu hình tìm được.





2. Ý tưởng

Tóm tắt ý tưởng

Đặc trưng: là các bước hướng tới lời giải cuối cùng của bài toán hoàn toàn được làm thử

Tại mỗi bước:

- Nếu có một lựa chọn được chấp nhận thì ghi nhận lại lựa chọn này và tiến hành các bước thử tiếp theo
- Ngược lại, không có sử lựa chọn nào thích hợp thì làm lại bước trước, xóa bỏ sự ghi nhận và quay về chu trình thử các lựa chọn còn lại



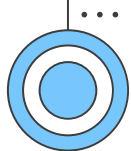


03

Nhận diện

Khi nào thì sử dụng backtracking?





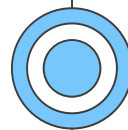
3. Nhận diện

Quay lui:

- ☐ quay lui được sử dụng khi có nhiều giải pháp và bạn muốn tất cả các giải pháp đó
- ☐ được sử dụng để giải bài toán liệt kê các cấu hình. Mỗi cấu hình được xây dựng bằng từng phần tử. Mỗi phần tử lại được chọn bằng cách thử tất cả các khả năng



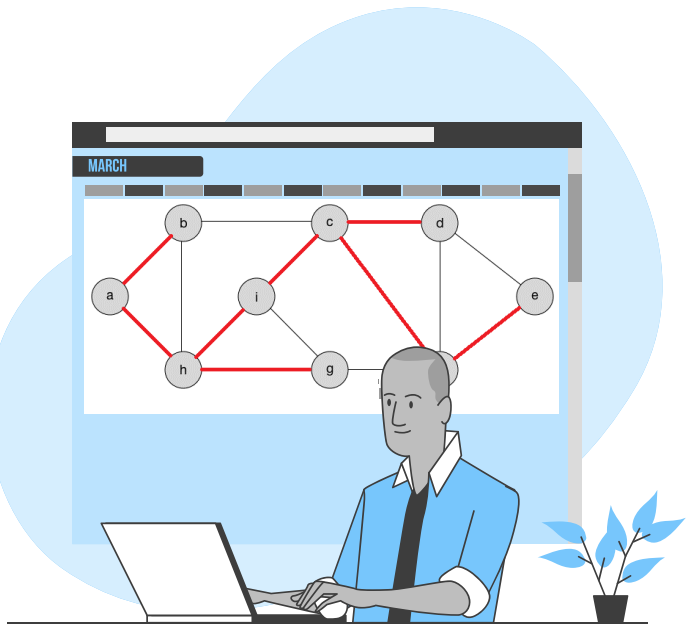
...



...

Bài toán

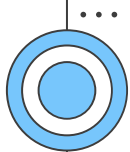
1. Bài toán viết dãy nhị phân có chiều dài n cho trước.
2. Xếp 2 nam 1 nữ vào băng ghế 3 chỗ (nữ không ngồi giữa)
3. Bài toán hoán vị
4. Bài toán 8 quân hậu
5. Bài toán xếp túi
6. Bài toán tìm đường đi trong mê cung
7. Bài toán tìm kiếm đường đi của quân mã trên bàn cờ
8. Bài toán tìm đường đi Hamilton



04

Thuật toán

Thuật toán backtracking ra sao?



4. Thuật toán

Backtracking hoạt động như thế nào?

Backtracking(pos):

for val in {tập các phương án X[pos]} có thể nhận:

if <chấp nhận phương án i>:

[thực hiện các lệnh khác]

if <đủ cấu hình>:

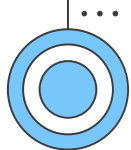
[Xuất kết quả]

else:

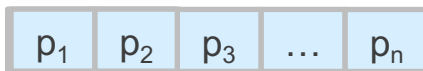
Backtracking(k+1)

// thoát khỏi bước thứ I, quay về bước thứ i-1 tiếp tục xét các phần tử

[Bỏ chọn i cho X[k]]



Xét từng vị trí:



Backtracking(pos):

for val in {tập các phương án $X[pos]$ } có thể nhận:

if <chấp nhận phương án i >:

[thực hiện các lệnh khác]

if <đủ cấu hình>:

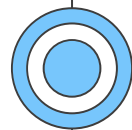
[Xuất kết quả]

else:

Backtracking($k+1$)

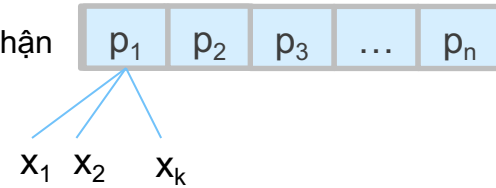
// thoát khỏi bước thứ i , quay về bước thứ $i-1$ tiếp tục xét các phần tử

[Bỏ chọn i cho $X[k]$]



Thử tất các giá trị có thể nhận

Xét từng vị trí:



Backtracking(pos):

for val in {tập các phương án $X[pos]$ } có thể nhận:

if <chấp nhận phương án i >:

[thực hiện các lệnh khác]

if <đủ cấu hình>:

[Xuất kết quả]

else:

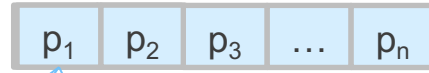
Backtracking($k+1$)

// thoát khỏi bước thứ I , quay về bước thứ $i-1$ tiếp tục xét các phần tử

[Bỏ chọn i cho $X[k]$]

Thử tất các giá trị có thể nhận

Xét từng vị trí:



x_1

x_2

x_k

Kiểm tra điều kiện

Backtracking(pos):

for val in {tập các phương án X[pos]} có thể nhận:

if <chấp nhận phương án i>:

[thực hiện các lệnh khác]

if <đủ cấu hình>:

[Xuất kết quả]

else:

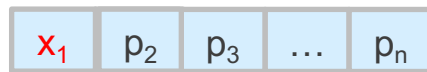
Backtracking(k+1)

// thoát khỏi bước thứ I, quay về bước thứ i-1 tiếp tục xét các phần tử

[Bỏ chọn i cho X[k]]

Thử tất các giá trị có thể nhận

Xét từng vị trí:



x_1 x_2 x_k

Kiểm tra điều kiện

↓
Đúng

Sai

Backtracking(pos):

for val in {tập các phương án $X[pos]$ } có thể nhận:

if <chấp nhận phương án i>:

[thực hiện các lệnh khác]

if <đủ cấu hình>:

[Xuất kết quả]

else:

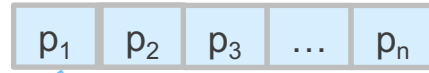
Backtracking(k+1)

// thoát khỏi bước thứ I, quay về bước thứ i-1 tiếp tục xét các phần tử

[Bỏ chọn i cho $X[k]$]

Thử tất các giá trị có thể nhận

Xét từng vị trí:



x_1

x_2

x_k

Kiểm tra điều kiện

Đúng

Sai

Backtracking(pos):

for val in {tập các phương án $X[pos]$ } có thể nhận:

if <chấp nhận phương án i >:

[thực hiện các lệnh khác]

if <đủ cấu hình>:

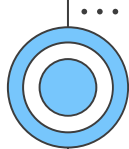
[Xuất kết quả]

else:

Backtracking($k+1$)

// thoát khỏi bước thứ i , quay về bước thứ $i-1$ tiếp tục xét các phần tử

[Bỏ chọn i cho $X[k]$]



...

Backtracking(pos):

for val in {tập các phương án $X[pos]$ } có thể nhận:

if <chấp nhận phương án i>:

[thực hiện các lệnh khác]

if <đủ cấu hình>:

[Xuất kết quả]

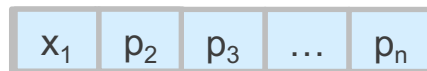
else:

Backtracking(k+1)

// thoát khỏi bước thứ I, quay về bước thứ i-1 tiếp tục xét các phần tử

[Bỏ chọn i cho $X[k]$]

Xét từng vị trí:



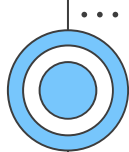
x_1 x_2 x_k



...



...



Xét từng vị trí:

x_1	x_2	x_3	...	x_k
-------	-------	-------	-----	-------

Backtracking(pos):

for val in {tập các phương án $X[pos]$ } có thể nhận:

if <chấp nhận phương án i >:

[thực hiện các lệnh khác]

if <đủ cấu hình>:

[Xuất kết quả]

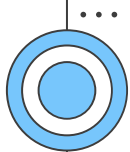
else:

Backtracking($k+1$)

// thoát khỏi bước thứ i , quay về bước thứ $i-1$ tiếp tục xét các phần tử

[Bỏ chọn i cho $X[k]$]





4. Thuật toán

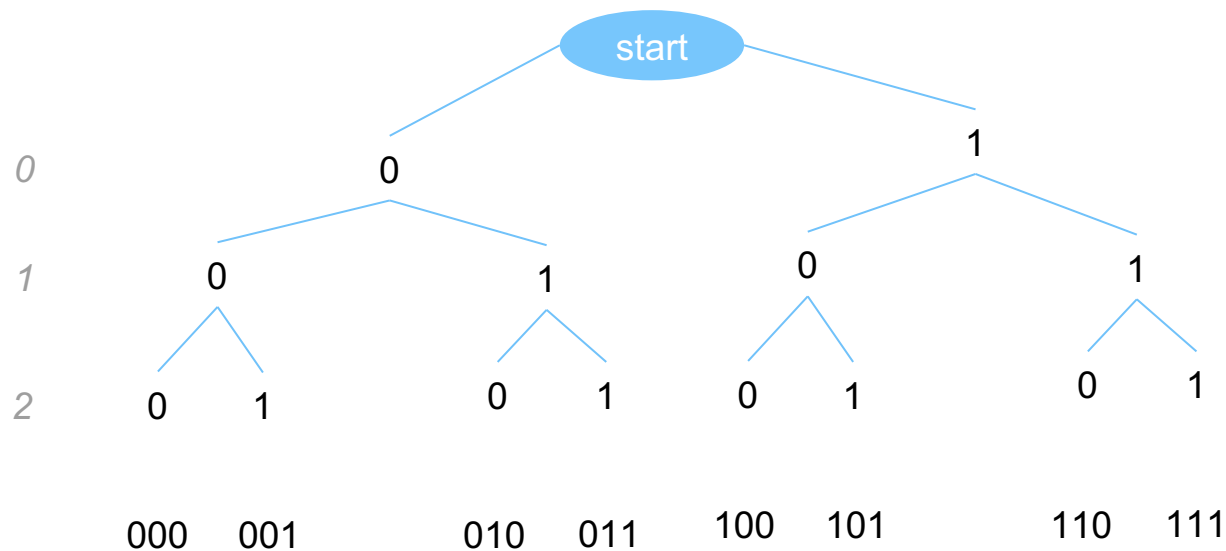
Ví dụ 1: tìm tất cả dãy nhị phân có độ dài n

```
1 def Try(pos):
2     for val in range(0, 2):
3         arr[pos] = val
4         if pos == n-1:
5             print(arr)
6         else:
7             Try(pos+1)
8
9 n = 3
10 arr = [None] * n
11 Try(0)
```

[0, 0, 0]
[0, 0, 1]
[0, 1, 0]
[0, 1, 1]
[1, 0, 0]
[1, 0, 1]
[1, 1, 0]
[1, 1, 1]

4. Thuật toán

Ví dụ 1: tìm tất cả dãy nhị phân có độ dài n – minh họa thuật toán



```
1 def Try(pos):  
2     for val in range(0, 2):  
3         arr[pos] = val  
4         if pos == n-1:  
5             print(arr)  
6         else:  
7             Try(pos+1)  
8  
9 n = 3  
10 arr = [None] * n  
11 Try(0)
```

4. Thuật toán

Ví dụ 2: Xếp 2 nam 1 nữ vào băng ghế 3 chỗ (nữ không ngồi giữa)

```
1 def Try(pos):
2     for i in range(len(students)):
3         if check[i] == False:
4             # if pos == 1 and students[i] == 'Nu':
5             #     break
6             ans[pos] = students[i]
7             check[i] = True
8             if pos == n-1:
9                 print(ans)
10            else:
11                Try(pos+1)
12            check[i] = False
13
14 n = 3
15 students = ['Nam 1', 'Nam 2', 'Nu']
16 ans = [None] * n
17 check = [False] * n
18 Try(0)
```

Nam 1 Nam 2 Nữ

Nam 2 Nữ Nam 1

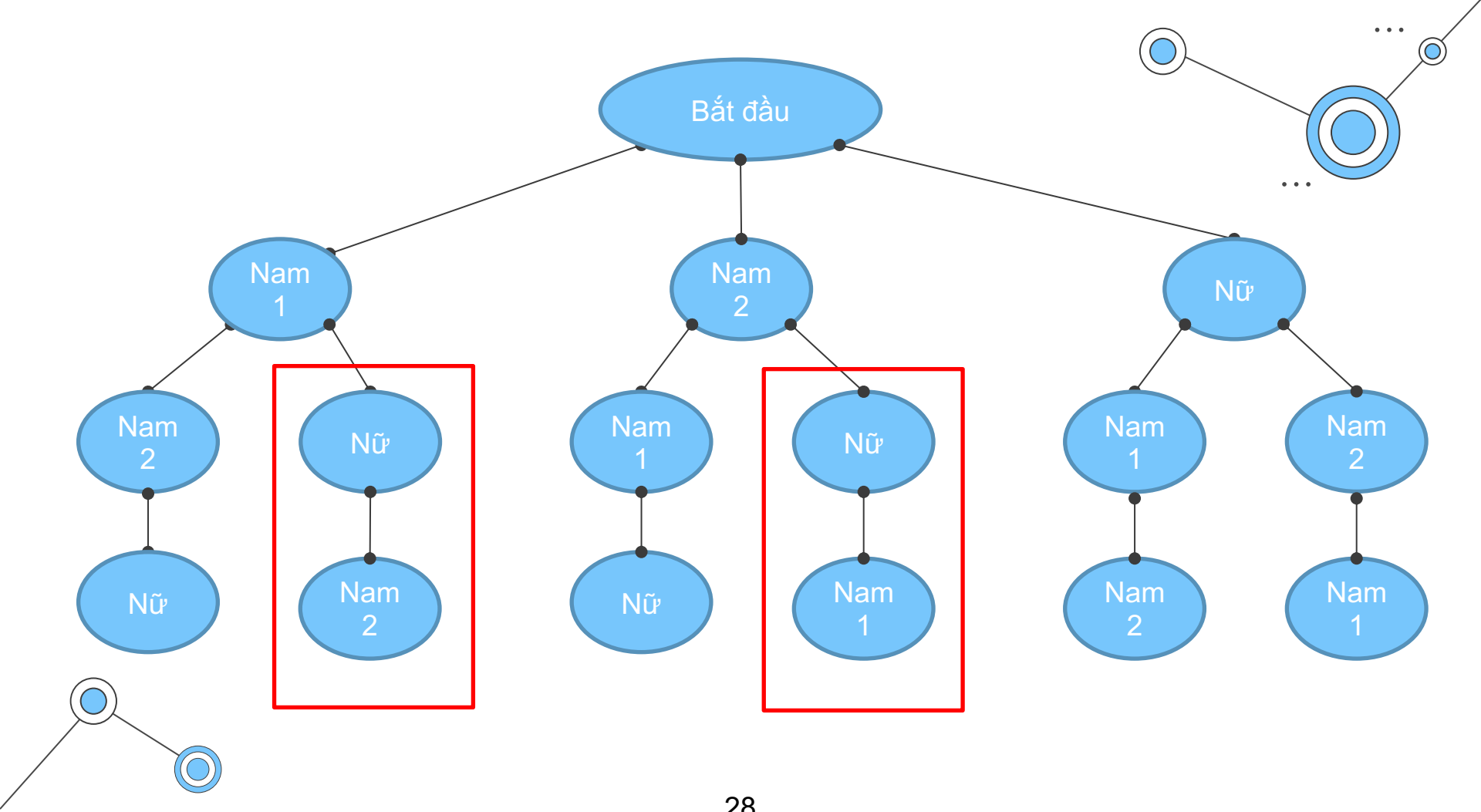
Nam 1 Nữ Nam 2

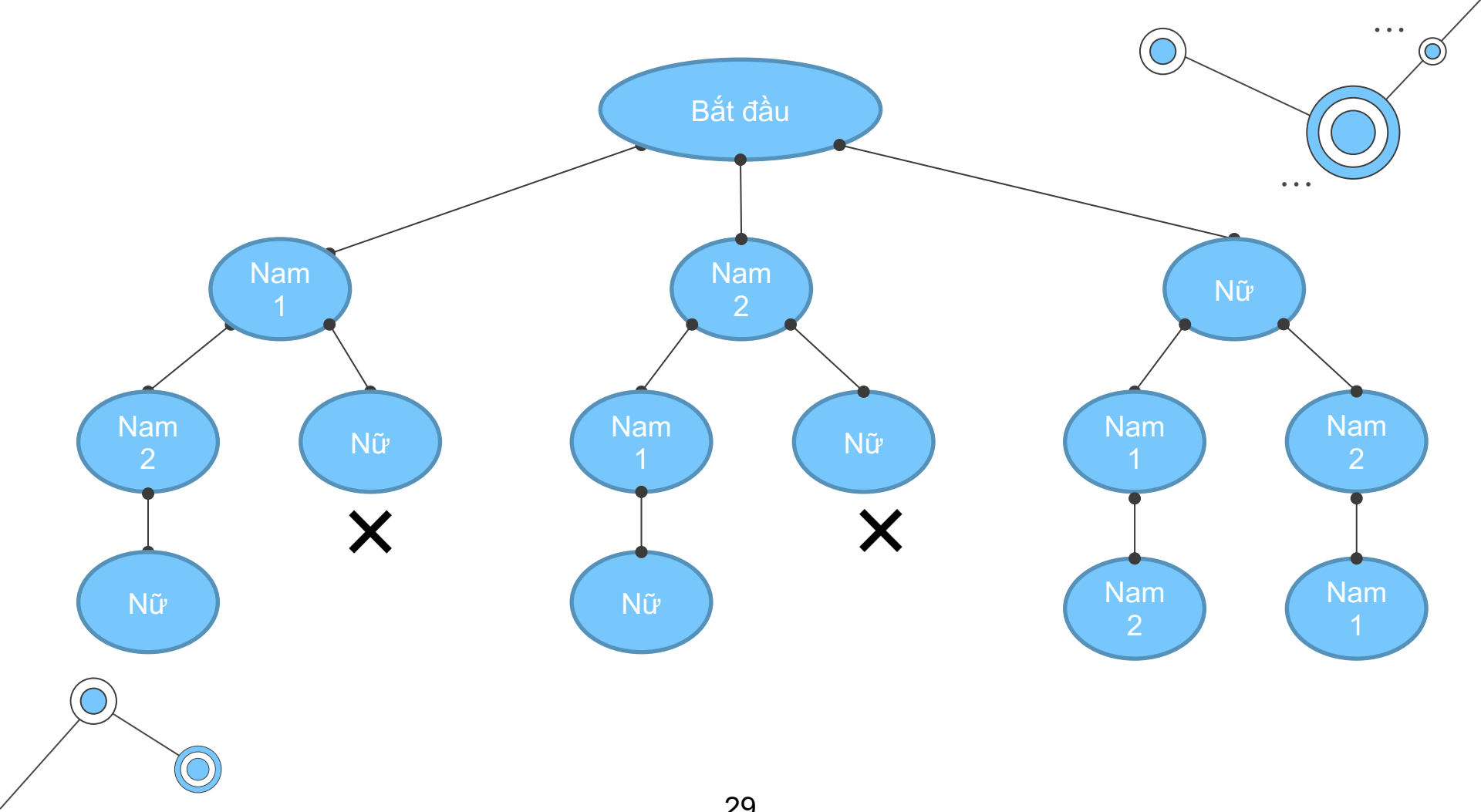
Nữ Nam 1 Nam 2

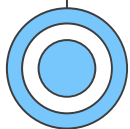
Nam 2 Nam 1 Nữ

Nữ Nam 2 Nam 1

Có tất cả 6 khả năng







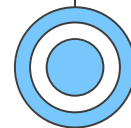
4. Thuật toán

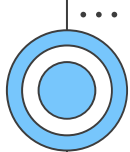
Ví dụ 2: Xếp 2 nam 1 nữ vào băng ghế 3 chỗ (nữ không ngồi giữa)

```
1 def Try(pos):
2     for i in range(len(students)):
3         if check[i] == False:
4             if pos == 1 and students[i] == 'Nu':
5                 break
6             ans[pos] = students[i]
7             check[i] = True
8             if pos == n-1:
9                 print(ans)
10            else:
11                Try(pos+1)
12            check[i] = False
13
14 n = 3
15 students = ['Nam 1', 'Nam 2', 'Nu']
16 ans = [None] * n
17 check = [False] * n
18 Try(0)
```

Chặn điều kiện nữ ngồi giữa

['Nam 1', 'Nam 2', 'Nu']
['Nam 2', 'Nam 1', 'Nu']
['Nu', 'Nam 1', 'Nam 2']
['Nu', 'Nam 2', 'Nam 1']



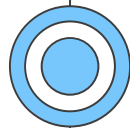


4. Thuật toán

Ví dụ 3: tìm dãy hoán vị từ 1 đến n

```
1 def Try(pos):
2     for val in range(1,n+1):
3         if exist[val] == False:
4             hv[pos] = val
5             exist[val] = True
6             if pos == n:
7                 print(hv[1:])
8             else:
9                 Try(pos+1)
10            exist[val] = False
11
12 n = 3
13 hv = [None] * (n+1)
14 exist = [False] * (n+1)
15 Try(1)
```

[1, 2, 3]
[1, 3, 2]
[2, 1, 3]
[2, 3, 1]
[3, 1, 2]
[3, 2, 1]






05

Nhận xét

Ưu nhược điểm backtracking?





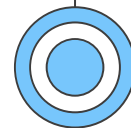
5. Nhận xét

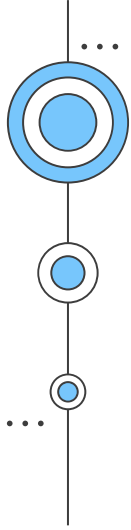
Ưu nhược điểm backtracking?

Ưu điểm: Việc quay lui là thử tất cả các tổ hợp để tìm được một lời giải. Thế mạnh của phương pháp này là nhiều cài đặt tránh được việc phải thử nhiều trường hợp chưa hoàn chỉnh, nhờ đó giảm thời gian chạy

Nhược điểm: Trong trường hợp xấu nhất độ phức tạp của quay lui vẫn là cấp số mũ. Vì nó mắc phải các nhược điểm sau:

- Rơi vào tình trạng "thrashing": quá trình tìm kiếm cứ gặp phải bế tắc với cùng một nguyên nhân.
- Thực hiện các công việc dư thừa: Mỗi lần chúng ta quay lui, chúng ta cần phải đánh giá lại lời giải trong khi đôi lúc điều đó không cần thiết.
- Không sớm phát hiện được các khả năng bị bế tắc trong tương lai. Quay lui chuẩn, không có cơ chế nhìn về tương lai để nhận biết đc nhánh tìm kiếm sẽ đi vào bế tắc.





Thanks for listening

