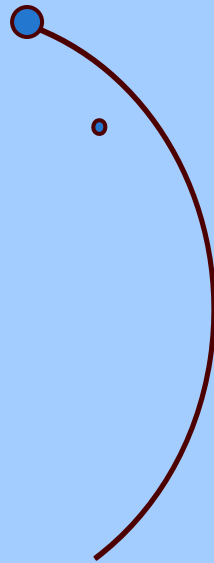
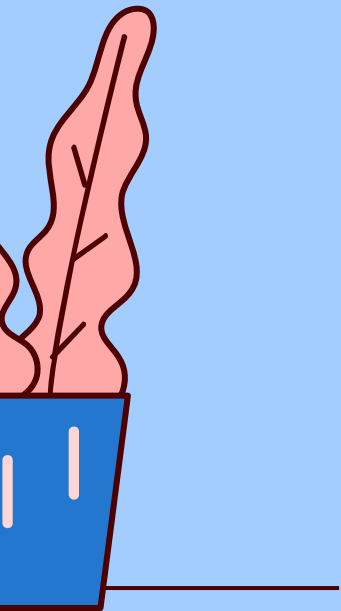


Phương pháp thiết kế thuật toán Backtracking

*Trường Đại học Công nghệ Thông tin
Lớp CS112.M11.KHTN - Ngày 11/10/2021
Nhóm 8:*

*Nguyễn Đức Minh Khang
Huỳnh Hoàng Vũ*





Gọi k là số quân hậu đã đặt trên bàn cờ

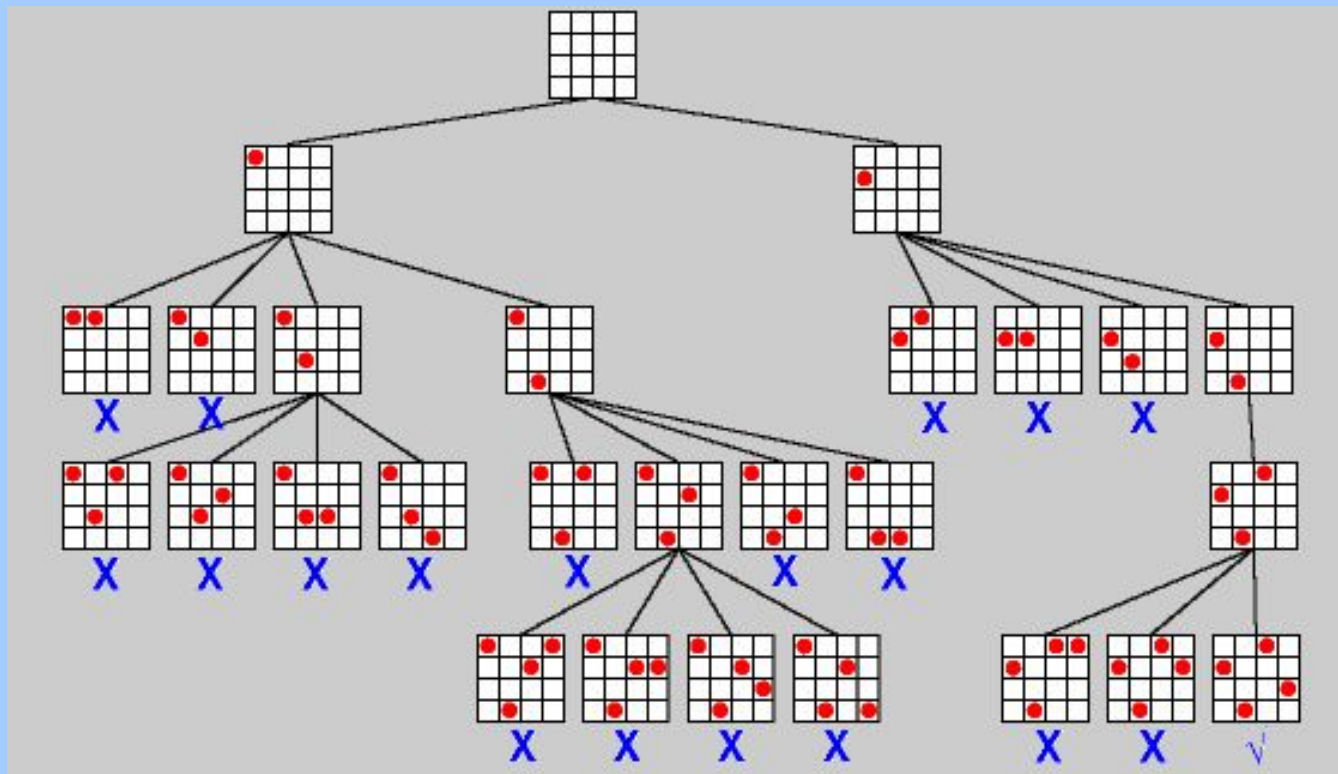
$k=0$

$k=1$

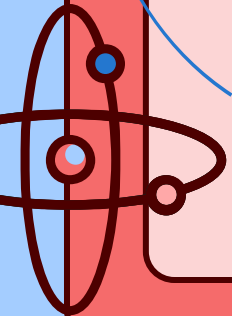
$k=2$

$k=3$

$k=4$



Nội dung

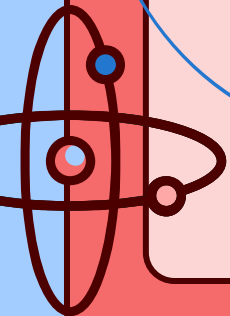


- Giới thiệu
- Bài toán minh họa
- Đặc điểm

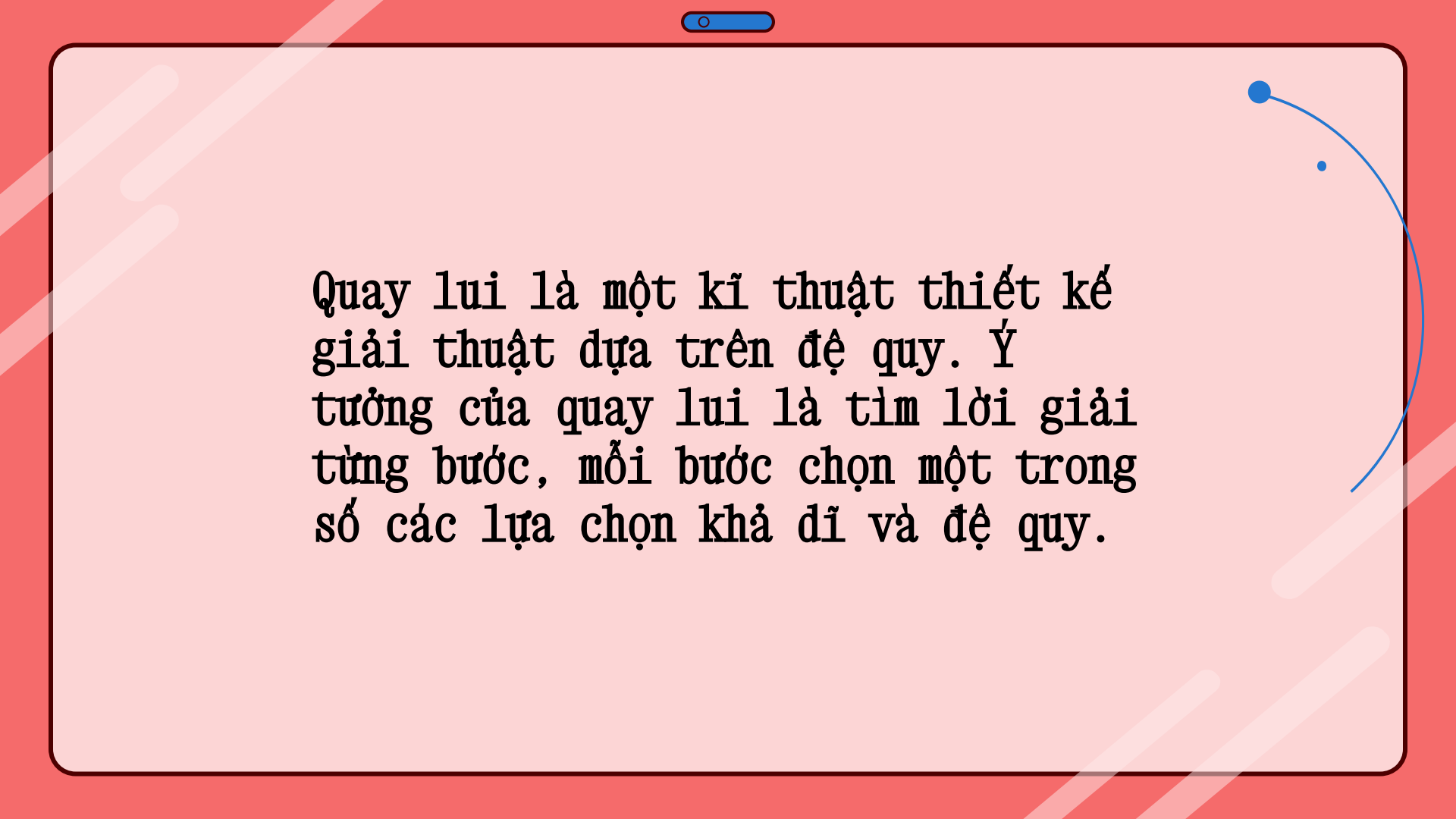
- Khác gì so với Brute Force?
- Ưu điểm, nhược điểm
- Ứng dụng

Nội dung

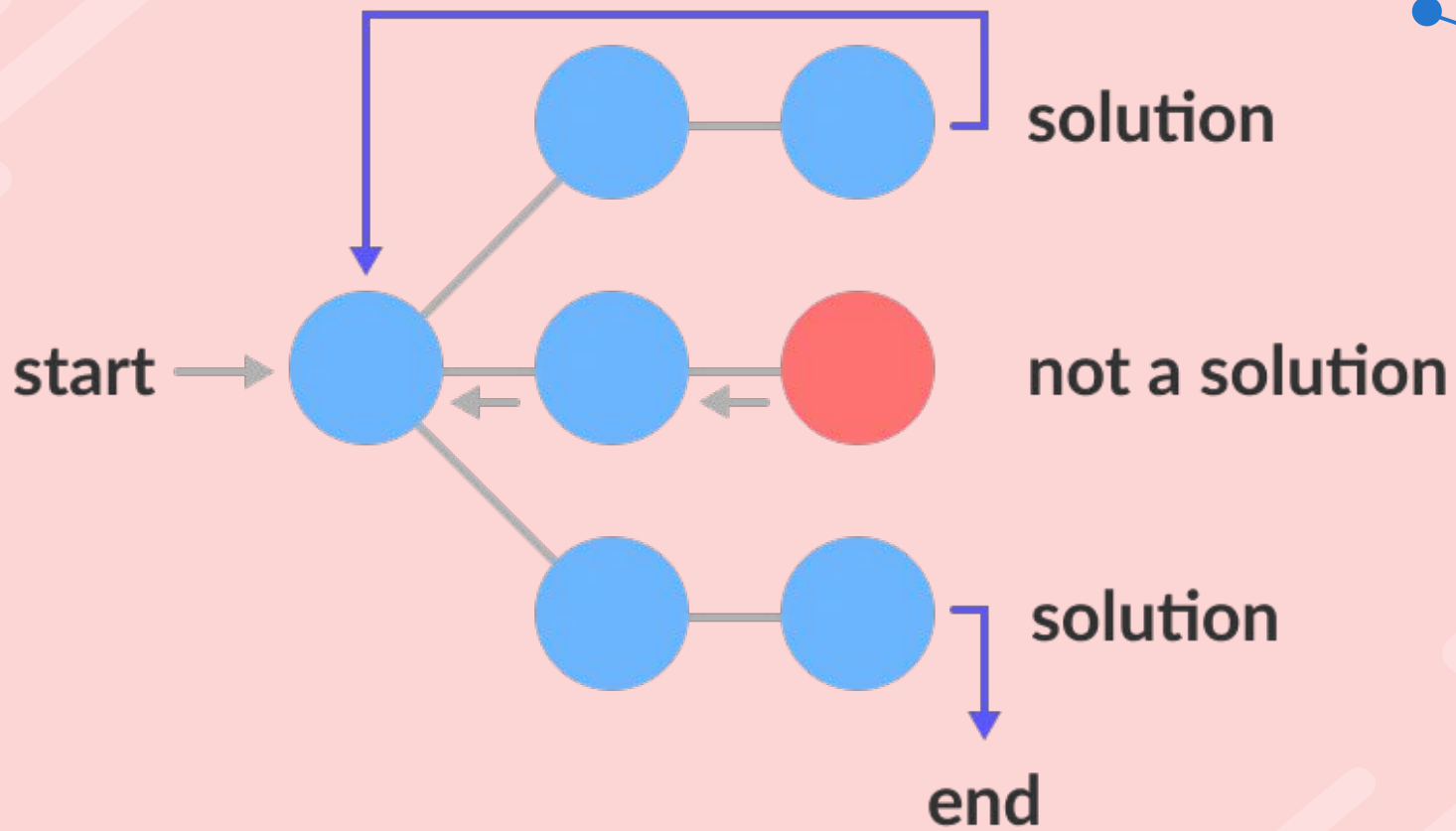
Backtracking là gì?



Backtracking được sử dụng để giải quyết các vấn đề trong đó một chuỗi các **đối tượng** được chọn từ một **tập hợp xác định** để chuỗi thỏa mãn một số **tiêu chí**.



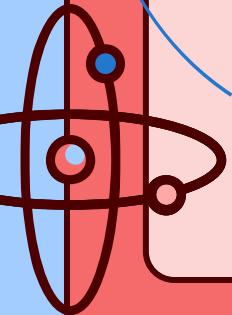
Quay lui là một kĩ thuật thiết kế giải thuật dựa trên đệ quy. Ý tưởng của quay lui là tìm lời giải từng bước, mỗi bước chọn một trong số các lựa chọn khả dĩ và đệ quy.



Giả thiết cấu hình của lời giải
có dạng mảng X gồm n phần tử

```
Backtrack(X[k]){  
    if (k==n-1) return true;  
    for(each X[k+1])  
        if Backtrack(X[k+1])  
            return true;  
    return false;  
}
```

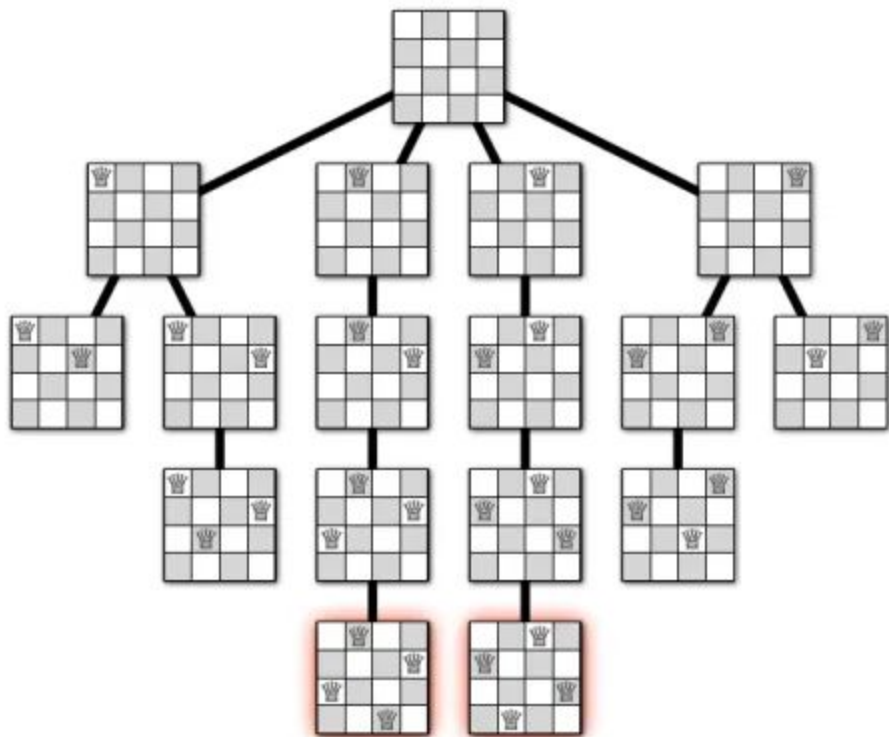
Khi nào dùng
Backtracking?



Các bài toán mà ta không quan tâm trạng thái của lời giải ở từng bước. Mà chỉ quan tâm đến 1 số câu hỏi như:

- Có tồn tại lời giải không?
- Có bao nhiêu lời giải?

Các bài toán mà không gian lời giải tăng theo cấp số so với đầu vào



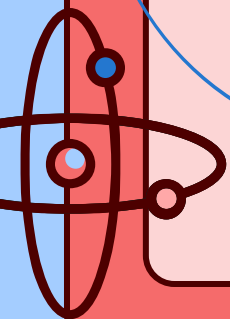
Trạng thái lời
giải được xác
định bởi biến
Xi trong miền
Di

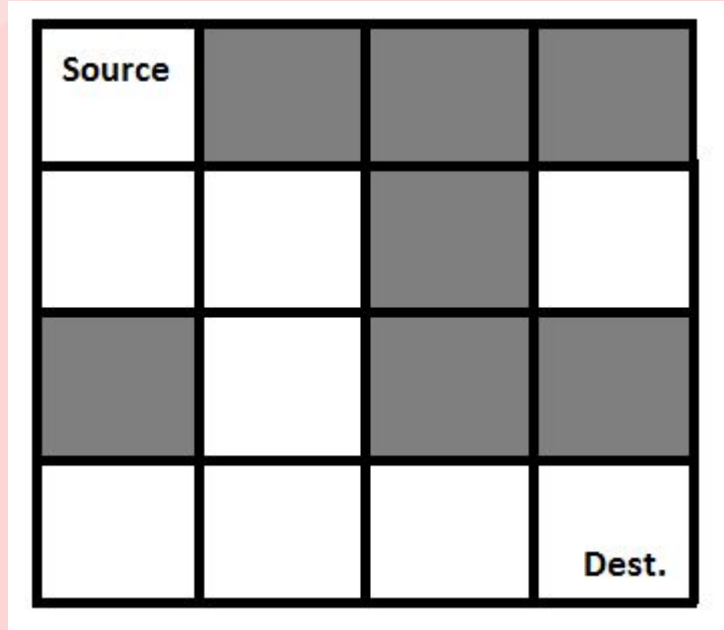
Mục tiêu là
một tập thỏa
các ràng buộc



- **Variables** WA, NT, Q, NSW, V, SA, T
- **Domains** $D_i = \{\text{red, green, blue}\}$
- **Constraints:** adjacent regions must have different colors
- e.g., $WA \neq NT$, or (WA, NT) in $\{(\text{red, green}), (\text{red, blue}), (\text{green, red}), (\text{green, blue}), (\text{blue, red}), (\text{blue, green})\}$

Một số bài toán minh họa





Source			
			Dest.

Rat in a Maze

Một mê cung được cho bởi một ma trận $n \times n$ chứa các số 0 và 1 trong đó ô nào chứa số 1 là có thể đi qua được còn chứa số 0 là vật cản không thể đi qua được. Một con chuột được thả vào góc trên bên trái mê cung để tìm lối thoát ở góc dưới bên phải.

Nếu có đường đi, in 1 đường đi bất kỳ dưới dạng ma trận $n \times n$ với các ô 1 là ô con chuột đi qua, 0 là ô con chuột không đi qua.

Nếu không có đường đi, hãy cho biết.



Input

1 0 0 0

1 1 0 1

0 1 0 0

1 1 1 1

Output

1 0 0 0

1 1 0 0

0 1 0 0

0 1 1 1

Ý tưởng

Xét từ ô 0, 0

Với mỗi ô đang xét, ta lại xét các ô liền kề với nó có đi đến lối thoát hay không bằng đệ quy

Các ô ta xét có thể trả về:

- True: nếu tồn tại ô kề nó hợp lệ hoặc nó là ô cuối cùng và đi được
- False: ngược lại

Các ô đã đi qua sẽ được đánh là False

Tạo một ma trận đầu ra với mọi ô bằng 0

Tạo một hàm đệ quy, lấy ma trận ban đầu, ma trận đầu ra và vị trí của con chuột (i, j).

Nếu vị trí nằm ngoài ma trận hoặc vị trí không hợp lệ thì quay trở lại.

Đánh dấu đầu ra vị trí $[i][j]$ là 1 và kiểm tra xem vị trí hiện tại có phải là đích hay không. Nếu đến đích, in ma trận đầu ra và trả về.

Gọi đệ quy cho vị trí $(i + 1, j)$ và $(i, j + 1)$.

Bỏ đánh dấu vị trí (i, j) , tức là đầu ra $[i][j] = 0$.

```

def solveMaze( maze ):
    sol = [ [ 0 for j in range(n) ] for i in range(n) ]
    if solveMazeUtil(maze, 0, 0, sol) == False:
        print("Solution doesn't exist");
        return False
    printSolution(sol)
    return True

def solveMazeUtil(maze, x, y, sol):
    if x == n - 1 and y == n - 1 and maze[x][y] == 1: # ô cuối cùng
        sol[x][y] = 1
        return True
    if isSafe(maze, x, y) == True:
        if sol[x][y] == 1: # ô đã đi
            return False
        sol[x][y] = 1
        if solveMazeUtil(maze, x + 1, y, sol) == True: # ô dưới
            return True
        if solveMazeUtil(maze, x, y + 1, sol) == True: # ô phải
            return True
        if solveMazeUtil(maze, x - 1, y, sol) == True: # ô trên
            return True
        if solveMazeUtil(maze, x, y - 1, sol) == True: # ô trái
            return True
        sol[x][y] = 0 # ô chưa đi
        return False

```

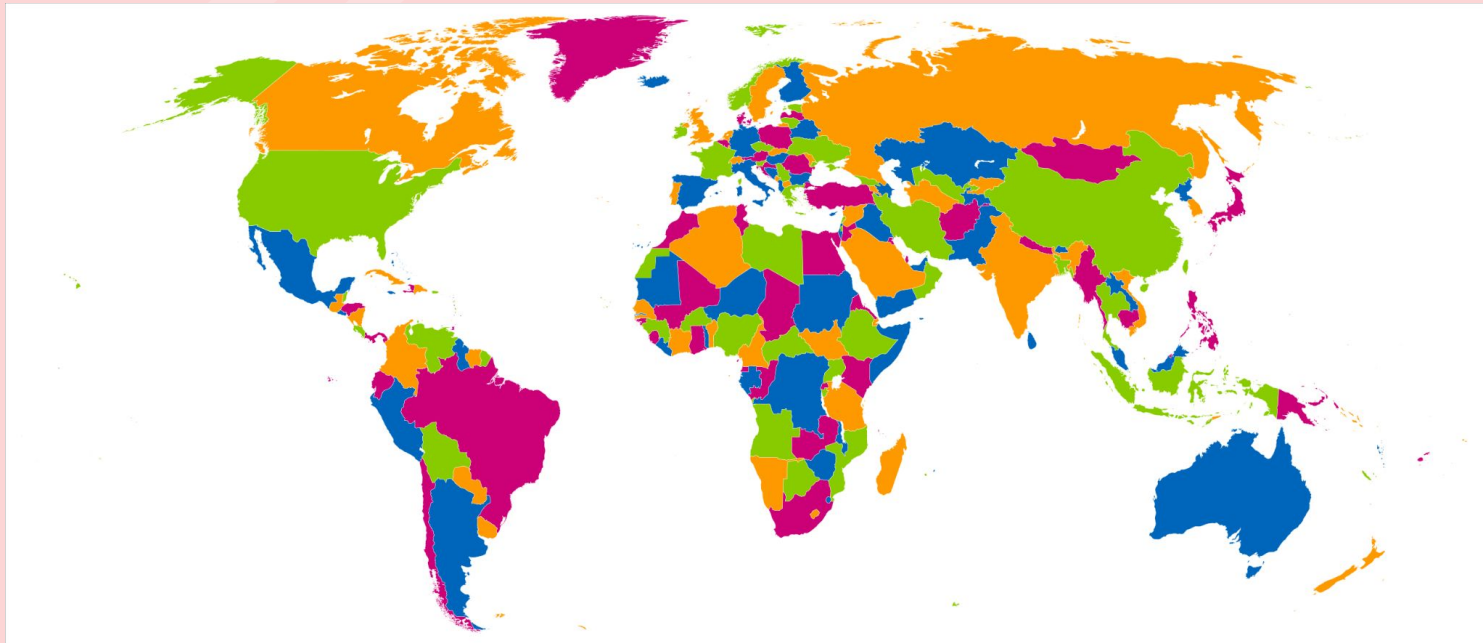
```

def isSafe( maze, x, y ):
    # if dưới check ô có nằm trong ma trận và = 1 không
    if x >= 0 and x < n and y >= 0 and y < n and maze[x][y] == 1:
        return True
    return False

# in ma trận kết quả
def printSolution( sol ):
    for i in sol:
        for j in i:
            print(str(j) + " ", end = "")
        print("")

if __name__ == "__main__":
    n = 4
    maze = [ [1, 0, 0, 0],
              [1, 1, 1, 1],
              [0, 1, 0, 1],
              [1, 1, 1, 1] ]
    solveMaze(maze)

```



Tô màu bản đồ

Cho đồ thị không liên thông có v đỉnh, ma trận kề của đồ thị này và số nguyên m . m là số màu tối đa có thể dùng để tô các đỉnh của đồ thị sao cho không có 2 đỉnh kề nào có màu trùng nhau.

Nếu tồn tại cách làm thỏa đề, xuất ra 1 cách bất kỳ

Nếu không, hãy cho biết

Input

4
0 1 1 1
1 0 1 0
1 1 0 1
1 0 1 0
3

Output

1
2
3
2

Tạo một hàm đệ quy lấy đồ thị, index, số đỉnh và mảng đầu ra đã tô màu.

Nếu index bằng số đỉnh. In cấu hình màu.

Gán màu cho một đỉnh (1 đến m).

Đối với mỗi màu được chỉ định, kiểm tra xem cấu hình hợp lệ hay không, gọi hàm đệ quy với chỉ số và số đỉnh tiếp theo

Nếu bất kỳ hàm đệ quy nào trả về true, phá vỡ vòng lặp và trả về true.

Ngược lại thì trả về false.

```

class Graph():
    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)]\
                       for row in range(vertices)]
    # check các đỉnh kề có bị trùng màu không
    def isSafe(self, v, colour, c):
        for i in range(self.V):
            if self.graph[v][i] == 1 and colour[i] == c:
                return False
        return True

    def graphColourUtil(self, m, colour, v):
        if v == self.V:
            return True
        for c in range(1, m + 1):
            if self.isSafe(v, colour, c) == True:
                colour[v] = c
                if self.graphColourUtil(m, colour, v + 1) == True:
                    return True
                colour[v] = 0

```

```

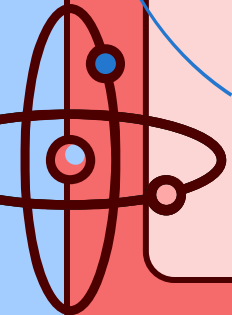
    def graphColouring(self, m):
        colour = [0] * self.V
        if self.graphColourUtil(m, colour, 0) == None:
            return False
        print ("Solution exist and Following\
              are the assigned colours:")
        for c in colour:
            print (c),
        return True

g = Graph(4)
g.graph = [[0, 1, 1, 1],
           [1, 0, 1, 0],
           [1, 1, 0, 1],
           [1, 0, 1, 0]]
m = 3
g.graphColouring(m)

```



Khác gì so với
Brute Force?



BACKTRACKING

- Có thể loại bỏ trường hợp kq đúng
- Lựa trường hợp kq tiềm năng chạy trước
- Tới bước gây sai thì quay lại
-

BRUTE FORCE

- Chạy tất cả trường hợp
-

Điểm khác nhau

Khái niệm

- Ràng buộc rõ ràng là giới hạn không gian cho từng biến. Ví dụ $x, y < 5$; $z > 3$.
- Ràng buộc ngầm định cần từ hai biến trở lên mới xác định được không gian bị giới hạn. Ví dụ: $x > y$; $x + y + z = 8$.
- Từ “biến” ở đây được dùng để chỉ một thành phần tạo nên kết quả (có thể sai hoặc đúng).

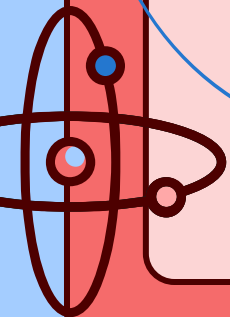
BACKTRACKING

- Kiểm tra ràng buộc ngầm định khi gán giá trị cho biến.
- Loại bỏ được nhiều trường hợp sai trước cả khi hình thành một ứng viên.

BRUTE FORCE

- Đi qua từng kết quả ứng viên trong không gian ràng buộc rõ ràng.
- Xác định đúng sai sau khi hình thành một ứng viên cụ thể.

Ưu điểm Nhược điểm





PROS

- Loại bỏ trường hợp chắc chắn không ra kq đúng, tiết kiệm bộ nhớ và thời gian từ trường hợp không cần thiết.
- Đơn giản
- Thích hợp khi chưa tìm được cách phù hợp nhất



CONS

- Độ phức tạp cao, có thể cấp số mũ.
- Gặp các “bế tắc” cùng nguyên nhân.
- Đệ quy gây tốn bộ nhớ.

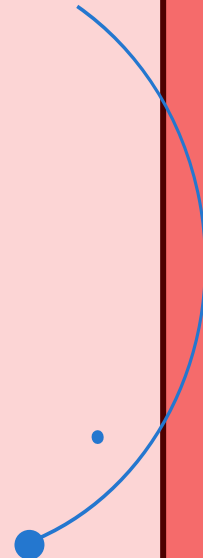
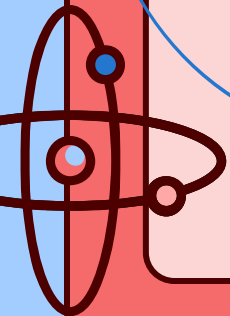
Ưu điểm

- Đảm bảo tìm được giải pháp, nếu giải pháp tồn tại.
- Dễ hiểu, dễ triển khai.
- Thể hiện từng bước giải quyết vấn đề, dễ debug.

Nhược điểm

- Tiêu tốn nhiều thời gian khi hệ số rẽ nhánh lớn.
- Dùng đệ quy dẫn đến hiệu suất kém, tiêu tốn bộ nhớ.
- Thường xuyên mắc cùng một lỗi vì không phát hiện yếu tố gây lỗi.
- Có nhiều bước đi dư thừa.
- Phát hiện xung đột muộn, chỉ phát hiện khi xung đột xảy ra.

Ứng dụng



Ứng dụng

- Giải quyết vấn đề thỏa mãn các ràng buộc - Constraint Satisfaction Problems (CSPs) như tìm kiếm, giải đố, lập lịch...
- Tối ưu hóa, Backtracking Search Optimization Algorithm (BSA).
- Một hướng tiếp cận trong Học tập củng cố - Reinforcement Learning.

Ứng dụng

- Artificial Intelligence
- Network Communication
- Robotics
- Electrical Engineering
- Materials Engineering

Ví dụ bài toán cụ thể

- Phân tích độ tin cậy dữ liệu trong mạng truyền thông

<https://www.sciencedirect.com/science/article/abs/pii/S016750600870726X>

- Các bài toán thuộc Constraint Satisfaction Problem trong AI

<http://www.cs.toronto.edu/~hojjat/384w09/Lectures/Lecture-04-Backtracking-Search.pdf>

- Dùng để tạo các con bot làm đối thủ trong game cờ vua, hoặc biết giải các game giải đố như sudoku, xếp hình
- Lập bản đồ cho các robot thám hiểm

<https://journals.sagepub.com/doi/full/10.5772/60043>

Ví dụ bài toán cụ thể

- Kiểm soát tần số tải đối với hệ thống năng lượng liên kết đa vùng

https://www.researchgate.net/publication/291312194_Application_of_backtracking_search_algorithm_in_load_frequency_control_of_multi-area_interconnected_power_system

- Hoạch định chuyển động và công việc trong hệ thống robot

<https://www.sciencedirect.com/science/article/pii/S000437021500051X>

- Tìm đường đi tối ưu trong AI

https://www.researchgate.net/publication/311312464_Adaptive_Backtracking_Search_Strategy_to_Find_Optimal_Path_for_Artificial_Intelligence_Purposes

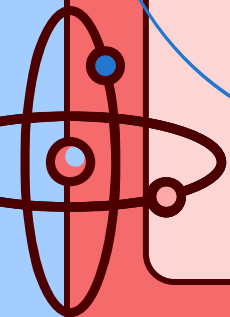
Ví dụ bài toán cụ thể


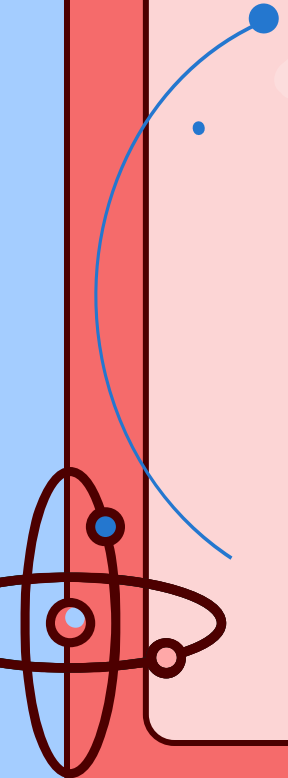
A simulation-based modified backtracking search algorithm for multi-objective stochastic flexible job shop scheduling problem with worker flexibility

Backtracking search algorithm with specular reflection learning for global optimization

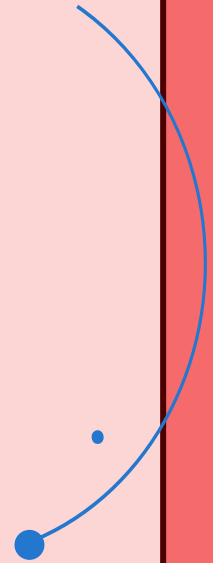


Human-machine interaction: A case study on fake news detection using a backtracking based on a cognitive system

Giới thiệu BTVN





Play Quiz



THANKS!

Do you have
any questions?

CREDITS: This presentation template was
created by **Slidesgo**, including icons by
Flaticon, infographics & images by **Freepik**

