Nguyễn Xuân Huy

**Back Tracking (Quay lui)**

**Các chuyên đề chọn lọc**

Hà Nội - 2023

[Phương pháp quay lui 3](#_Toc171252581)

[Các quân Hậu 4](#_Toc171252582)

[Queens Version 1. Tìm một nghiệm 4](#_Toc171252583)

[Queens Version 2. 9](#_Toc171252584)

[Queens Version 3. Lưu nghiệm 13](#_Toc171252585)

[Queens Version 4. Tìm mọi nghiệm 17](#_Toc171252586)

# Phương pháp quay lui

Diagram

Description automatically generated

Giả sử ta phải tìm trong một tập dữ liệu D cho trước một dãy dữ liệu: *v* = (*v*[1], *v*[2],..., *v*[*n*]) thoả mãn đồng thời hai tính chất P và Q. Trước hết ta chọn một trong hai tính chất đã cho để làm nền, giả sử ta chọn tính chất P. Tính chất P được gọi là *tính chất nền* hay *bất biến.*

Sau đó ta thực hiện các bước sau đây:

Bước 1. (Khởi trị) Xuất phát từ một dãy ban đầu *v* = (*v*[1],..., *v*[*i*]) nào đó của các phần tử trong D sao cho *v* thoả P.

Bước 2. Nếu *v* thoả Q ta dừng thuật toán và thông báo kết quả là dãy *v*, ngược lại ta thực hiện Bước 3.

Bước 3. Tìm tiếp một phần tử *v*[*i*+1] để bổ sung cho *v* sao cho *v* = (*v*[1],...,*v*[*i*],*v*[*i*+1]) vẫn thoả bất biến P.

Có thể xảy ra các trường hợp sau đây:

* 3.1 Tìm được phần tử *v*[*i*+1]: quay lại bước 2.
* 3.2 Không tìm được *v*[*i*+1], tức là với mọi *v*[*i*+1] có thể lấy trong D, dãy *v* = (*v*[1],..., *v*[*i*], *v*[*i*+1]) không thoả P. Điều này có nghĩa là đi theo đường *v* = (*v*[1],..., *v*[*i*]) sẽ không dẫn tới kết quả. Ta phải đổi hướng tại một vị trí nào đó. Để thoát khỏi ngõ cụt này, ta tìm cách thay *v*[*i*] bằng một giá trị khác trong D. Nói cách khác, ta loại *v*[*i*] khỏi dãy *v*, giảm *i* đi một đơn vị rồi quay lại Bước 3.

Cách làm như trên được gọi là *quay lui*: lùi lại một bước.

Dĩ nhiên ta phải đánh dấu *v*[*i*] là phần tử đã loại tại vị trí *i* để sau đó không lấy lại phần tử đó.

Khi nào thì có thể trả lời là không tồn tại dãy *v* thoả đồng thời hai tính chất P và Q? Nói cách khác, khi nào thì ta có thể trả lời là bài toán vô nghiệm?

Dễ thấy, bài toán vô nghiệm khi ta đã duyệt hết mọi khả năng. Ta nói là đã vét cạn mọi khả năng. Chú ý rằng có thể đến một lúc nào đó ta phải lùi liên tiếp nhiều lần. Từ đó suy ra rằng, thông thường bài toán vô nghiệm khi ta không còn có thể lùi được nữa.

Có nhiều sơ đồ giải các bài toán quay lui, dưới đây là một sơ đồ khá đơn giản, không đệ quy.

|  |
| --- |
| *Sơ đồ 1: Giải bài toán quay lui  (tìm 1 nghiệm)* |
| Khởi trị dãy v thoả P  while true do  if (v thoả Q) then  Ghi nhận nghiệm  exit  end if  if (hết cách đi) then  Thông báo vô nghiệm  exit  end if  if (Tìm được 1 nước đi) then Tiến  else Lùi  end if  end while |

Thông thường ta khởi trị cho *v* là dãy rỗng (không chứa phần tử nào) hoặc dãy có một phần tử. Ta chỉ yêu cầu dãy *v* được khởi trị sao cho *v* thoả bất biến *P*. Lưu ý là cả dãy *v* thoả *P* chứ không phải từng phần tử trong *v* thoả *P*.

Có bài toán yêu cầu tìm toàn bộ (mọi nghiệm) các dãy *v* thoả đồng thời hai tính chất P và Q. Nếu biết cách tìm một nghiệm ta dễ dàng suy ra cách tìm mọi nghiệm như sau: mỗi khi tìm được một nghiệm, ta thông báo nghiệm đó trên màn hình hoặc ghi vào một file rồi thực hiện thao tác Lùi, tức là giả vờ như không công nhận nghiệm đó, do đó, phải loại *v*[*i*] cuối cùng trong dãy v để tiếp tục tìm hướng khác. Phương pháp này có tên là phương pháp *giả sai*. Sơ đồ trên sẽ được sửa một chút như sau để tìm mọi nghiệm.

|  |
| --- |
| *Sơ đồ 2: Giải bài toán quay lui  (tìm mọi nghiệm theo phương pháp giả sai)* |
| Khởi trị dãy v thoả P  while true do  if (v thoả Q) then  Ghi nhận nghiệm  Lùi (giả sai)  end if  if (hết cách đi) then  Thông báo số nghiệm  exit  end if  if (Tìm được 1 nước đi) then Tiến  else Lùi  end if  end while |

Độ phức tạp

Trong trường hợp vô nghiệm, các thuật toán quay lui phải khảo sát mọi khả năng, do đó độ phức tạp n!.

## Các quân Hậu

Quân Hậu trên bàn cờ Vua có thể diệt quân đối phương theo hàng, theo cột chứa nó hoặc theo đường chéo của hình vuông nhận nó làm đỉnh.

* Tìm một cách đặt N quân Hậu trên bàn cờ Vua kích thước N× N sao cho không quân nào diệt được quân nào.
* Tìm mọi cách đặt N quân Hậu theo điều kiện trên.

Hiển thị kết quả trên màn hình.

Ví dụ

A screenshot of a game

Description automatically generated with medium confidence

### Queens Version 1. Tìm một nghiệm

Ta biết quân Hậu trên bàn cờ Vua (cờ Quốc tế) kiểm soát hàng dọc, hàng ngang và các đường chéo cùng màu với ô Hậu đứng. Ta giải bài toán tổng quát với n quân Hậu trên bàn cờ n×n. Dễ hiểu là chỉ có thể đặt tối đa n quân hậu, vì mỗi quân Hậu sẽ kiểm duyệt một hàng dọc. Ta mã hóa mỗi quân Hậu theo hàng dọc 1, 2,…, n.

Với n = 1 ta có nghiệm tầm thường: đặt Hậu tại ô duy nhất trên bàn cờ.

Với n = 2 và n = 3 bài toán vô nghiệm.

Chart

Description automatically generated

Tổ chức dữ liệu

Ta sẽ dùng mảng v để ghi nhận vị trí của mỗi Hậu trên bàn cờ. Khi khởi trị với tám Hậu ta đặt các Hậu trước vạch xuất phát ngoài bàn cờ, tức là:

v = [0,0,0,0,0,0,0,0,0]

trong đó ta không sử dụng phần tử v[0].

Background pattern

Description automatically generated

Sau khi tìm được nghiệm đầu tiên ta có

Chart

Description automatically generated

v = [0,1,5,8,6,3,7,2,4]

với ý nghĩa:

Đặt Hậu 1 tại dòng 1

Đặt Hậu 2 tại dòng 5

Đặt Hậu 3 tại dòng 8

Đặt Hậu 4 tại dòng 6

Đặt Hậu 5 tại dòng 3

Đặt Hậu 6 tại dòng 7

Đặt Hậu 7 tại dòng 2

Đặt Hậu 8 tại dòng 4

Thuật toán

Ta dùng phương pháp quay lui (Back Tracking) để giải bài toán này.

Ý tưởng của phương pháp quay lui là như sau:

* Mỗi bước đi ta chọn một khả năng dẫn đến đích.
* Nếu có một khả năng như vậy ta tiến thêm một bước theo khả năng đó.
* Nếu không, ta sẽ lùi lại một bước, trở về cấu hình của bước trước đó.
* Thuật toán sẽ kết thúc khi gặp một trong hai tình huống sau đây:
* Đến đích: hiển thị nghiệm
* Vô nghiệm: khi đã lùi về giới hạn

|  |
| --- |
| Thuật toán Queens |
| Đặt n quân Hậu trên bàn cờ n×n |
| Khởi trị: đặt n Hậu tại vạch 0  Cầm Hậu k = 1  while True do  if k = 0 then  Thông báo vô nghiệm  return  end if  if k > n then  Thông báo nghiệm v  return  end if  if Tìm được một nước đi then  Tiến 1 bước: k = k+1  else  Lùi 1 bước: k = k-1  end if  end while |

Sơ đồ trên có ưu điểm là đơn giản và không đệ quy.

Khi khởi trị ta xếp các Hậu ở trước vạch xuất phát ngoài bàn cờ:

v = [0,0,...,0]

Ta lần lượt di chuyển từng quân Hậu vào bàn cờ. Gọi k là quân Hậu cần di chuyển từ vị trí hiện đứng v[k] đến vị trí mới i để đặt Hậu k.

Theo sơ đồ, ta phải xác định hai điều kiện quan trọng cho tính dừng của thuật toán.

* + Dừng khi không thành công
  + Dừng khi đã gặp một nghiệm.

Dễ thấy, khi ta đã đặt xong Hậu thứ n, tức là ta đã xếp xong đủ n Hậu, thì theo thói quen ta sẽ chuyển qua Hậu k = n+1. Vậy điều kiện dừng sau khi đã gặp một nghiệm sẽ là k > n.

Tương tự, sau nhiều lần quay lui ta đưa các Hậu ra khỏi bàn cờ về vạch xuất phát lúc đầu, theo thói quen, ta sẽ cầm Hậu trước đó, tức là Hậu k = 0. Vậy điều kiện dừng khi vô nghiệm sẽ là k < 1.

Ta viết hàm Find(k,n) xác định điều kiện *Tìm được một nước đi cho Hậu k* trên bàn cờ n × n.

* Dịch dần Hậu k từ dòng đang đứng v[k] xuống đến dòng cuối cùng n.
* Nếu tìm được một dòng i tại đó Hậu k không đụng độ với các Hậu 1, 2, …, k-1 đã đặt trước thì i là dòng tìm được.
* Ngược lại, với mọi i = v[k]+1 ... n ta không tìm được dòng đặt Hậu k thì hàm cho ra giá trị 0.

Find(k, n):

for i = v[k]+1..n do

if GoodPlace(k, i) then

return i

end if

end for

return 0

end Find

Hàm GoodPlace(k,i) cho giá trị True nếu có thể đặt Hậu k tại dòng i, ngược lại, hàm cho ra giá trị False.

Ta biết, Hai hậu k (tại cột k) và j (tại cột j ≠ k) *đụng độ* khi và chỉ khi hai Hậu đứng trên cùng dòng: v[k] = v[j] hoặc tạo thành hai đỉnh đối diện của một hình vuông cạnh k - j = abs(v[j] - i). Để ý rằng, do mỗi Hậu chiếm giữ một cột và Hậu j được đặt trước Hậu k nên j < k.

GoodPlace(k, i):

for j=1..k do

if v[j] = i or k-j = abs(v[j]-i) then

return false

end if

end for

return true

end GoodPlace

Các chương trình được minh họa với các bàn cờ *n*×*n*, kích thước từ *n* = -1 đến *n* = 19:

Chương trình

# Queens, Ver. 1: Tìm một nghiệm

from time import time

MN = 20

def Go(msg = ' ? '):

if input(msg) == '.': exit(0)

def Queens(n):

global v

print(n, 'Queens:')

if n >= MN:

print(" Large chess board size: ", n)

return

if n < 1 or n == 2 or n == 3:

print(" No solution.")

return

if n == 1:

print([1])

return

# Back tracking

v = [0]\*(n+1) # Đặt n Hậu ngoài bàn cờ

k = 1 # cầm Hậu 1

while True:

if k < 1:

print(" No solution.")

return

if k > n:

print(" Solution:", v[1:])

return

v[k] = Find(k, n)

if v[k] > 0: k += 1 # tien

else: k -= 1 # lui

# Có thể đặt Hậu k tại dòng i ?

def GoodPlace(k, i):

# xét các Hậu đã đặt trước Hậu k

for j in range(1, k):

if v[j] == i or k-j == abs(v[j]-i):

return 0 # Hau k va j dung do

return 1

# Tìm dòng đặt Hậu k trên bàn cờ nxn:

# Đẩy Hậu k xuống các dòng dưới

# kiểm tra có thể đặt Hậu k tại dòng nào

def Find(k, n):

for i in range(v[k]+1, n+1):

if GoodPlace(k, i): return i

return 0

def Run():

for n in range(-1, MN+1):

Queens(n)

Go()

# APPLICATION

t = time()

Run()

print(time()-t)

print(' T h e E n d')

Output

-1 Queens:

No solution.

0 Queens:

No solution.

1 Queens:

[1]

Solution: [1]

2 Queens:

No solution.

3 Queens:

No solution.

4 Queens:

Solution: [2, 4, 1, 3]

...

8 Queens:

Solution: [1, 5, 8, 6, 3, 7, 2, 4]

...

19 Queens:

Solution: [1, 3, 5, 2, 4, 9, 13, 15, 17, 19, 7, 16, 18, 11, 6, 8, 10, 12, 14]

20 Queens:

Large chess board size: 20

T h e E n d

### Queens Version 2.

Ta cải tiến hàm Queens(n) với các điều chỉnh sau đây:

* Bằng những quan sát đơn giản, ta thấy với kích thước bàn cờ n quá lớn thì thuật toán quay lui sẽ không khả dụng. Ta chọn giới hạn trên cho n là 30, tức là ta chỉ xét số Hậu tối đa là 29.
* Nếu n < 1: ta bỏ qua, xem như vô nghiệm;
* n = 1 có một nghiệm duy nhất [1];
* n = 2 và n = 3: vô nghiệm;
* n = 4 có nghiệm [2, 4, 1, 3];
* Từ n = 5 trở đi ta sẽ xuất phát từ nghiệm n = 4 và bắt đầu duyệt quân Hậu k = 5 trở đi.
* Để tăng tốc độ cho hàm Find ta bổ sung thêm ba mảng:
* Mảng r đánh dấu những dòng đã đặt Hậu: r[i] = 0 cho biết trên dòng i đã có Hậu, r[i] = 1 cho biết dòng này còn trống.
* Mảng x đánh dấu đường *chéo phải* gồm các ô chéo phải cùng màu với ô Hậu k đang đứng:

x[n+i-k] = 0 cho biết đường chéo phải n+i-k bị Hậu k trên dòng i kiểm soát.

x[n+i-k] = 1 cho biết đường chéo này chưa bị Hậu nào kiểm soát.

* Mảng y đánh dấu đường *chéo trái* gồm các ô chéo trái cùng màu với ô Hậu k đang đứng:

y[i+k] = 0 cho biết đường chéo trái i+k bị Hậu k trên dòng i kiểm soát.

y[i+k] = 1 cho biết trên đường chéo này còn trống.

Table

Description automatically generated with medium confidence

Tóm lại, nếu Hậu k đứng tại tại dòng i thì sẽ kiểm soát

dòng i: row[i] = 0

chéo phải n+i-k: x[n+i-k] = 0

chéo trái i+k: y[i+k] = 0

Ví dụ

Với n = 8 ta cần ba mảng định vị các dòng (row), đường chéo phải (x) và đường chéo trái (y) cho các Hậu từ 1..8 như sau:

A table with numbers and letters

Description automatically generated

Khi nhấc Hậu k khỏi dòng i ta cần giải phóng các mảng định vị:

x[n+i-k] = y[i+k] = r[i] = 1

Giá trị 1 cho biết các hướng này hiện đang trống.

Khi đặt Hậu k vào dòng j ta cần đánh dấu các mảng định vị:

x[n+j-k] = y[j+k] = r[j] = 0

Giá trị 0 cho biết các hướng này hiện đang bị chiếm.

Nhờ ba mảng này, việc tìm một vị trí (dòng) còn trống để đặt Hậu k trở nên đơn giản:

Find(k, n):

for i = v[k]+1..n do

if x[n+i-k] and y[i+k] and r[i] then:

return i

end if

return 0

end Find

trong đó giá trị k cho biết Hậu k hiện di chuyển trên cột k, v[k] là dòng Hậu k đang đứng, n là kích thước của bàn cờ.

Với n = 4 ta hiển thị ngay kết quả là nghiệm có sẵn: v = [2, 4, 1, 3].

Nếu n > 4 ta xuất phát từ nghiệm v(4) = [2, 4, 1, 4] và bắt đầu thuật toán quay lui với k = 5 trở đi.

Trước hết ta cần định vị nghiệm v(4).

v[1] = 2; v[2] = 4; v[3] = 1; v[4] = 3;

for (int k = 1; k <= 4; ++k)

x[n+j-k] = y[j+k] = r[j] = 0

Ta viết thêm hàm Verify(n) kiểm tra các nghiệm xem việc đặt Hậu có đúng luật không. Hàm này có độ phức tạp với kích thước bàn cờ n khoảng vài chục thì không đáng kể.

def Verify(n):

print(' Verify:', end = ' ')

for k in range(1, n+1):

for j in range(1, k):

if v[j] == v[k]:

print(" Error at ", j, k)

return

if k-j == abs(v[j]-v[k]):

print(" Error at ", j, k)

return

print(" CORRECT.")

Ta đặt thêm biến step để đếm xem thuật toán quay lui cần bao nhiêu bước lặp. Ví dụ, với 19 Hậu ta cần 5068 bước tiến và lùi.

Chương trình

"""

Queens, Ver. 2B: Tim mot nghiem

Dùng các biến định vị r, x, y

Hau k tren dong i se kiem soat

r[i]

cheo phai x[n+i-k]

cheo trai y[i+k]

Dat Hau k tai dong i: r[i] = x[n+i-k] = y[i+k] = false;

Nhac Hau k khoi dong i: r[i] = x[n+i-k] = y[i+k] = true;

"""

MN = 20

def Go():

if input(' ? ') == '.':

exit(0)

def NhacHau(k):

if v[k] > 0:

r[v[k]] = x[n+v[k]-k] = y[v[k]+k] = 1

def DatHau(k):

r[v[k]] = x[n+v[k]-k] = y[v[k]+k] = 0

# Tim dong dat Hau k tren ban co nxn:

# day Hau k xuong cac dong duoi

# kiem tra co the dat Hau k tai dong do?

def Find(k, n):

for i in range(v[k]+1, n+1):

if r[i] and x[n+i-k] and y[i+k]:

return i

return 0

def Verify(n):

print(' Verify:', end = ' ')

for k in range(1, n+1):

for j in range(1, k):

if v[j] == v[k]:

print(" Error at ", j, k)

return

if k-j == abs(v[j]-v[k]):

print(" Error at ", j, k)

return

print(" CORRECT.")

def Queens(soHau):

global n, v, r, x, y

n = soHau

print(n, 'Queens:')

if n >= MN:

print(" Large chess board size: ", n)

return

if n == 1:

print([1])

if n < 4: # n = 2 | 3

print(" No solution.")

return

# n >= 4

v = [0,2,4,1,3] + [0]\*(n-4)

if n == 4:

print(v[1:])

Verify(4)

return

r = bytearray([1]\*(n+1)) # kiem soat dong

x = bytearray([1]\*(2\*n+1)) # cheo phai

y = bytearray([1]\*(2\*n+1)) # cheo trai

print(' Xuat phat tu Queens 4:')

for k in range(1,5):

DatHau(k)

# Back tracking

k = 5 # cam Hau 5

step = 0

while True:

step += 1

if k < 1:

print(" No solution.")

return

if k > n:

print(" Solution:", v[1:])

print(" step = ", step)

Verify(n)

return

NhacHau(k)

v[k] = Find(k, n)

if v[k] > 0:

# Chap nhan Hau k

DatHau(k)

k += 1 # tien

else:

k -= 1 # lui

def Run():

for n in range(-1, MN+1):

Queens(n)

Go()

print(" T h e E n d")

Run()

Output

. . .

4 Queens:

[2, 4, 1, 3]

Verify: CORRECT.

5 Queens:

Xuat phat tu Queens 4:

Solution: [2, 4, 1, 3, 5]

step = 2

Verify: CORRECT.

. .

8 Queens:

Xuat phat tu Queens 4:

Solution: [2, 4, 6, 8, 3, 1, 7, 5]

step = 45

Verify: CORRECT.

. . .

18 Queens:

Xuat phat tu Queens 4:

Solution: [2, 4, 1, 3, 8, 12, 14, 16, 18, 6, 15, 17, 10, 5, 7, 9, 11, 13]

step = 55133

Verify: CORRECT.

19 Queens:

Xuat phat tu Queens 4:

Solution: [2, 4, 1, 3, 5, 9, 13, 15, 17, 19, 7, 16, 18, 11, 6, 8, 10, 12, 14]

step = 5068

Verify: CORRECT.

20 Queens:

Large chess board size: 20

T h e E n d

### Queens Version 3. Lưu nghiệm

Mấy năm gần đây, trong các đề thi quốc tế thường đề xuất các test theo quy trình *test sau dựa vào các test trước*. Đây là ý tưởng thường được vận dụng trong các bộ tìm kiếm thông minh trên mạng. Giả sử bạn đã tìm được các bài Toán Tin trong các kỳ thi Châu Á, nay bạn muốn tham khảo các bài Toán Tin Quốc tế, thì một bộ tìm kiếm thông minh sẽ không phải duyệt tìm trên Internet từ đầu mà chỉ tìm kiếm tiếp những trang ngoài Châu Á để bổ sung kết quả cho bạn.

Trở lại với bài toán Queens. Giả sử bạn gọi liên tiếp hai hàm Queens(10), sau đó là Queens(12).

Nếu tại bước trước bạn đã lưu lại nghiệm của Queens(10) thì khi cần tính Queens(12) bạn chỉ cần phát triển tiếp Queens(10) để duyệt với Hậu k = 11, rồi k = 12. Trong nhiều trường hợp việc lưu lại các nghiệm có thể giảm đáng kể thời gian thực hiện.

Version 3 sẽ tập trung giải quyết vấn đề này.

Chiến lược đại thể sẽ như sau:

Sau mỗi test Queens(m) ta sẽ *lưu lại nghiệm* v(m), trong đó v(m) là vector nghiệm của Queens(m).

Khi gặp test mới, Queens(n), nếu v(n) đã có sẵn trong kho nghiệm thì ta chỉ việc hiển thị v(n). Ngược lại, ta tìm trong kho một nghiệm v(m) gần nhất với n, m < n rồi phát triển v(m) để thu được v(n), sau đó ta lại nạp v(n) vào kho.

Cấu trúc dữ liệu

Ta biết, mỗi nghiệm Queens(n) là một danh sách (vector, mảng) v[0..n], ví dụ, Queens(4) cho ta nghiệm v = [2, 4, 1, 3]. Ta tổ chức kho lưu nghiệm là một từ điển dưới dạng một danh sách các danh sách vv[0..MN] trong đó mỗi phần tử vv[n] có độ dài n+1, phần tử đầu tiên là vv[n][0] cho biết tình trạng nghiệm của Queens(n), n phần tử tiếp theo vv[n][1..n], là một nghiệm cụ thể đã tìm được của Queens(n), 0 ≤ n < MN (giới hạn của n, ví dụ MN = 30), cụ thể là:

* vv[n][0] = -1 cho biết Queens(n) vô nghiệm.
* vv[n][0] = 0 cho biết Queens(n) chưa xác định.
* vv[n][0] = 1 cho biết Queens(n) có nghiệm cụ thể trong kho.

Ví dụ

vv[4] = [1, **2, 4, 1, 3**], n = 4 có nghiệm [2,4,1,3]

vv[5] = [0,...], n = 5 chưa xác định

vv[0] = [-1,...], n = 0 vô nghiem

vv[1] = [1,1], n = 1 có nghiệm [1]

vv[2] = [-1,...], n = 2 vô nghiem

vv[3] = [-1,...], n = 3 vô nghiem

vv[8] = [0,...], n = 8 chưa xác định

Với mỗi số Hậu n ta gọi Queens(n) theo các bước sau đây:

* Kiểm tra:
* nếu n > giới hạn thì thông báo kích thước bàn cờ vượt giới hạn
* nếu n < 1 hoặc n = 2 | 3 thì thông báo vô nghiệm
* Tìm nghiệm n trong kho:
* nếu n có trong kho thì hiển thị nghiệm n
* nếu nghiệm n không có sẵn trong kho thì xác định nghiệm m gần nhất với n (m < n) rồi gọi thuật toán quay lui tìm nghiệm n từ nghiệm m và nạp nghiệm n vào kho.

Chương trình

# Queens, Ver. 3: Lưu nghệm

MN = 30 # Tối đa 29 Hậu

def Go(msg = ' ? '):

if input(msg) == '.': exit(0)

def NhacHau(k):

if v[k] > 0: r[v[k]] = x[n+v[k]-k] = y[v[k]+k] = 1

def DatHau(k):

r[v[k]] = x[n+v[k]-k] = y[v[k]+k] = 0

# Khởi trị base MN nghiệm

def Init():

global vv

vv = [[-1],[1,1],[-1],[-1],[1,2,4,1,3]]

for i in range(5,MN): vv.append([0]\*(i+1))

# Qeens of 0: vô nghiệm

# Qeens of 1: [1]

# Qeens of 2, 3: vô nghiệm

# Qeens of 4: [2,4,1,3]

# Qeens of n >= 5: chưa xác định

def Find(k, n):

for i in range(v[k]+1, n+1):

if r[i] and x[n+i-k] and y[i+k]: return i

return 0

def Verify(n):

print(' Verify:', end = ' ')

for k in range(1, n+1):

for j in range(1, k):

if v[j] == v[k]:

print(" Error at ", j, k)

return

if k-j == abs(v[j]-v[k]):

print(" Error at ", j, k)

return

print(" CORRECT.")

def Queens(soHau):

global vv, n, v, r, x, y

n = soHau

print(n, 'Queens:')

if n < 1:

print(" No solution.")

return

if n >= MN:

print(" Large chess board size: ", n)

return

v = [0]\*(n+1) # nghiem

r = bytearray([1]\*(n+1)) # kiểm soát dòng

x = bytearray([1]\*(2\*n+1)) # chéo phải

y = bytearray([1]\*(2\*n+1)) # chéo trái

if vv[n][0] == -1:

print(" No solution.")

return

if vv[n][0] == 1: # nghiem co san trong base

print(" Solution in base:", vv[n][1:])

return

# Tìm trong kho nghiệm d gàn nhất với n

for d in range(n,3,-1):

if vv[d][0] == 1: break

# copy nghiệm d, thêm 0 cho đủ len n

v = vv[d].copy() + [0]\*(n-d)

print(' Starting from solution', d, ':', v[1:])

for k in range(1, d): DatHau(k) # các Hậu nghiệm d kiểm soát

k = d + 1 # cầm Hậu k

step = 0

while True:

step += 1

if k < 1:

print(" No solution.")

return

if k > n: # tim duoc nghiem n

print(" Solution:", v[1:])

print(" step = ", step)

Verify(n)

# Luu nghiem

vv[n] = v.copy()

return

NhacHau(k)

v[k] = Find(k, n)

if v[k] > 0:

# Chap nhan Hau k

DatHau(k)

k += 1 # tien

else: k -= 1 # lui

def Run():

a = [-3,2,4,3,0,30,5,8,5,10,13,17,10,8,20,12,20,25,5,4,17]

Init()

for n in a:

Queens(n)

Go()

# APPLICATION

Run()

print(" T h e E n d")

Output

-3 Queens:

No solution.

2 Queens:

No solution.

4 Queens:

Solution in base: [2, 4, 1, 3]

...

30 Queens:

Large chess board size: 30

5 Queens:

Starting from solution 4 : [2, 4, 1, 3, 0]

Solution: [2, 4, 1, 3, 5]

step = 2

Verify: CORRECT.

8 Queens:

Starting from solution 5 : [2, 4, 1, 3, 5, 0, 0, 0]

Solution: [2, 4, 6, 8, 3, 1, 7, 5]

step = 46

Verify: CORRECT.

5 Queens:

Solution in base: [2, 4, 1, 3, 5]

...

17 Queens:

Starting from solution 13 : [2, 4, 6, 8, 10, 12, 1, 3, 5, 7, 9, 11, 13, 0, 0, 0, 0]

Solution: [2, 4, 6, 8, 10, 12, 14, 16, 1, 3, 5, 7, 9, 11, 13, 15, 17]

step = 1171

Verify: CORRECT.

10 Queens:

Solution in base: [2, 4, 6, 8, 10, 1, 3, 5, 7, 9]

8 Queens:

Solution in base: [2, 4, 6, 8, 3, 1, 7, 5]

20 Queens:

Starting from solution 17 : [2, 4, 6, 8, 10, 12, 14, 16, 1, 3, 5, 7, 9, 11, 13, 15, 17, 0, 0, 0]

Solution: [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 3, 1, 7, 5, 11, 9, 15, 13, 19, 17]

step = 1838

Verify: CORRECT.

...

T h e E n d

### Queens Version 4. Tìm mọi nghiệm

Để triển khai thuật toán tìm mọi cách xếp *n* quân Hậu trên bàn cờ *n*×*n* chúng ta sẽ dựa trên một kinh nghiệm hữu hiệu của công nghệ phần mềm là “*Thay đổi ít nhất nhưng hưởng lợi nhiều nhất*”.

Như ta biết, có thể có đến 92 cách xếp Hậu cho bài Queens(8).

Ta chỉ cần sửa chút ít sơ đồ tìm một nghiệm để thu được mọi nghiệm. Ý tưởng là như sau:

Phương pháp giả sai

Sau khi tìm được một nghiệm và hiển thị nghiệm đó, nghĩa là sau khi đã đặt xong Hậu thứ n, ta coi như không xếp được Hậu n này, do đó ta phải quay lui về Hậu n-1 để tìm tiếp nghiệm khác.

|  |
| --- |
| Khởi trị  while True do  if mọi Hậu đều đã quay lại vạch xuất phát then  return  end if  if Đến đích then  Thông báo một nghiệm  Lùi  end if  if Tìm được một nước đi then  Tiến 1 bước  else  Lùi 1 bước  end if  end while |
| *Sơ đồ thuật toán quay lui tìm mọi nghiệm* |

Phương án đơn giản dưới đây minh họa phương pháp giả sai.

Hàm Queens(sh, sn) lúc này sẽ có hai tham biến:

* sh là số Hậu cần đặt trên bàn cờ.
* sn là số nghiệm cần tìm: sn = 1 cho biết chỉ cần tìm một nghiệm; ngược lại, khi sn > 1, thì chương trình sẽ tìm tất các các nghiệm.

Chương trình

# Queens, Ver. 4: Tim mọi nghiệm

MN = 20 # Tối đa 19 Hậu

def Go(msg = ' ? '):

if input(msg) == '.': exit(0)

def NhacHau(k):

if v[k] > 0: r[v[k]] = x[n+v[k]-k] = y[v[k]+k] = 1

def DatHau(k):

r[v[k]] = x[n+v[k]-k] = y[v[k]+k] = 0

def Find(k, n):

for i in range(v[k]+1, n+1):

if r[i] and x[n+i-k] and y[i+k]: return i

return 0

def Verify(n):

print(' Verify:', end = ' ')

for k in range(1, n+1):

for j in range(1, k):

if v[j] == v[k]:

print(" Error at ", j, k)

return

if k-j == abs(v[j]-v[k]):

print(" Error at ", j, k)

return

print(" CORRECT.")

def Queens(soHau, songhiem = 1): # ngầm định: tim 1 nghiệm

global n, v, r, x, y

n = soHau

print(n, 'Queens:', end = ' ')

if n >= MN:

print(" Large chess board size: ", n)

return

if n == 1:

print(" Solution [1]")

print(' Total', 1, "solution(s)")

return

if n < 4:

print(" no solutions")

return

if songhiem == 1: print(' one solution')

else: print(' all solutions')

v = [0]\*(n+1)

r = bytearray([1]\*(n+1)) # kiem soat dong

x = bytearray([1]\*(2\*n+1)) # cheo phai

y = bytearray([1]\*(2\*n+1)) # cheo trai

k = 1

d = 0 # đếm nghiệm

while True:

if k < 1:

if d == 0:

print(" No solution.")

return

else:

print(' Total', d, "solution(s)")

return

if k > n:

d += 1

print(" Solution No",d, ':', v[1:])

Verify(n)

if songhiem == 1: return

else: k = n # gia sai

NhacHau(k)

v[k] = Find(k, n)

if v[k] > 0:

# Chap nhan Hau k

DatHau(k)

k += 1 # tien

else: k -= 1 # lui

def Run():

for n in range(-1, 10):

Queens(n)

Queens(n,2)

Go()

Queens(MN)

Queens(MN, 2)

# APPLICATION

Run()

print(" T h e E n d")

Output

-1 Queens: no solutions

-1 Queens: no solutions

?

...

1 Queens: Solution [1]

Total 1 solution(s)

1 Queens: Solution [1]

Total 1 solution(s)

?

...

4 Queens: one solution

Solution No 1 : [2, 4, 1, 3]

Verify: CORRECT.

4 Queens: all solutions

Solution No 1 : [2, 4, 1, 3]

Verify: CORRECT.

Solution No 2 : [3, 1, 4, 2]

Verify: CORRECT.

Total 2 solution(s)

?

...

?

8 Queens: one solution

Solution No 1 : [1, 5, 8, 6, 3, 7, 2, 4]

Verify: CORRECT.

8 Queens: all solutions

Solution No 1 : [1, 5, 8, 6, 3, 7, 2, 4]

Verify: CORRECT.

...

Solution No 92 : [8, 4, 1, 3, 6, 2, 7, 5]

Verify: CORRECT.

Total 92 solution(s)

?

...

20 Queens: Large chess board size: 20

20 Queens: Large chess board size: 20

T h e E n d