Quay lui (8)

Nguyễn Thanh Bình
Khoa Công nghệ Thông tin
Trường đại học Bách khoa
Đại học Đà Nẵng

Quay lui (backtracking)

- *Tìm kiếm vét cạn* trong một không gian trạng thái của bài toán
  - Các giải pháp của bài toán được biểu diễn bởi một không gian trạng thái (cụ thể là một cây)
  - Tìm kiếm giải pháp = tìm kiếm vét cạn trên không gian trạng thái

- Thường được sử dụng để giải quyết các bài toán yêu cầu tìm kiếm các phần tử của một tập hợp thoả mãn một số *ràng buộc*

- Nhiều bài toán được giải quyết bởi thuật toán quay lui có dạng:

  « ***Tìm tập con S của A_1 x A_2 x … x A_n (A_k là một tập hợp) sao cho mỗi phần tử s = (s_1,s_2,…,s_n) $\in$ S thoả mãn ràng buộc nào đó*** »

- Ví dụ
  - Tìm tất cả các hoán vị của {1,2, …, n}

    A_k = {1,2,…,n} với $\forall k$

    $s_i \neq s_k$ với $\forall i \neq k$

Quay lui

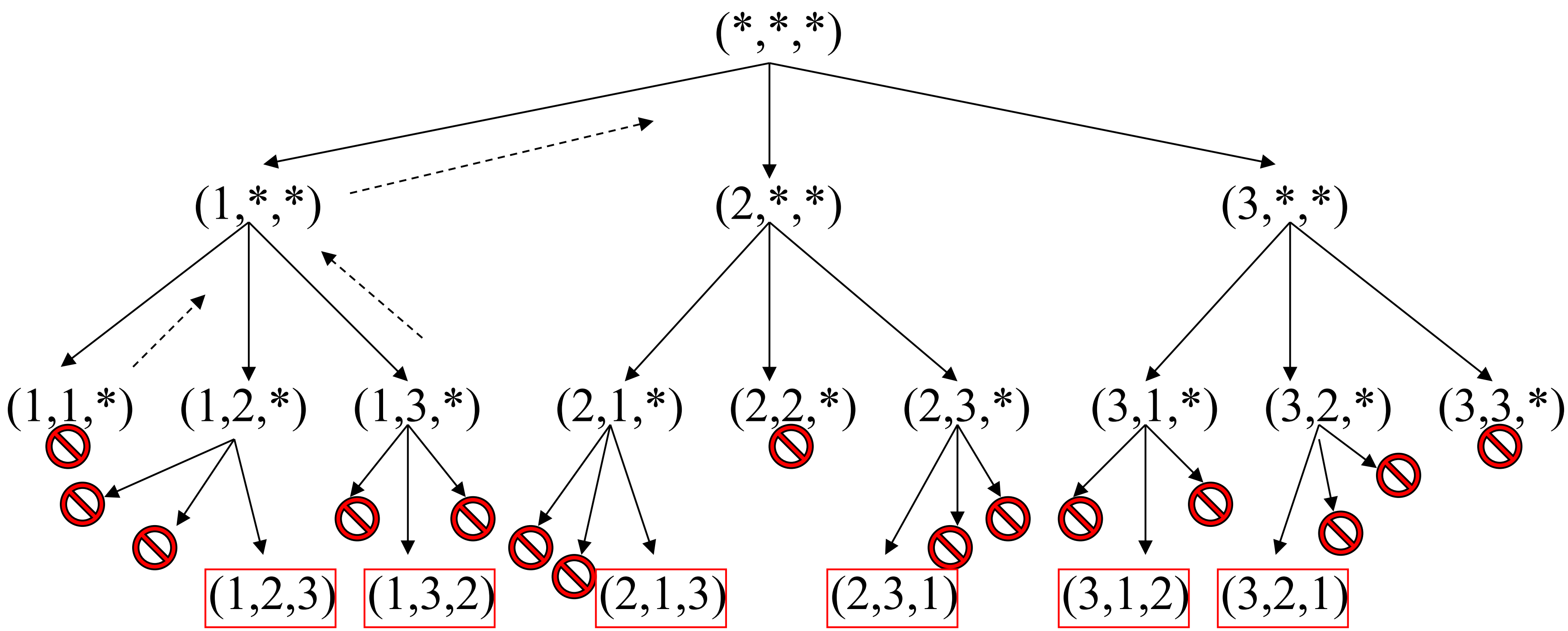
- Ý tưởng
  - Giải pháp được xây dựng từng thành phần ở mỗi bước
  - *Tìm kiếm vét cạn* tất cả các giải pháp có thể trên cây không gian trạng thái
    - Mất nhiều thời gian thực thi
  - *Tỉa bớt* các thành phần không đưa đến giải pháp
    - Chỉ những giải pháp từng phần có triển vọng được sử dụng
      - Giải pháp từng phần có triển vọng nếu nó có thể dẫn đến giải pháp cuối cùng, nếu không thì gọi là giải pháp không có triển vọng
    - Những giải pháp từng phần không có triển vọng sẽ bị loại bỏ
  - Nếu tất cả các giá trị của một thành phần không dẫn đến một giải pháp từng phần có triển vọng thì quay lui thành phần trước và thử giá trị khác

Quay lui

- Giải pháp quay lui xây dựng không gian trạng thái dưới dạng cây
  - Nút gốc tương ứng với trạng thái đầu (trước khi việc tìm kiếm giải pháp bắt đầu)
  - Mỗi nút trong tương ứng với một giải pháp từng phần có triển vọng
  - Các nút lá tương ứng với hoặc giải pháp từng phần không có triển vọng hoặc giải pháp cuối cùng

Quay lui

- Ví dụ
  - Cây không gian trạng thái



Quay lui

- Tìm kiếm giải pháp
  - Tìm kiếm theo chiều sâu trước trên cây không gian trạng thái
  - Nếu tìm kiếm theo chiều sâu gặp một nút lá
    - thì kiểm tra giải pháp hiện tại có thoả mãn các ràng buộc hay không
    - có thể đưa thêm các điều kiện để kiểm tra một giải pháp có tối ưu không

Quay lui

- Các bước thiết kế thuật toán quay lui

1. Chọn cách *biểu diễn giải pháp*
2. Xây dựng các *tập A1, A2, …, An* và xếp thứ tự để các phần tử của chúng được xử lý
3. Xây dựng các *điều kiện* từ ràng buộc của bài toán để xác định một giải pháp từng phần có là triển vọng không, gọi là *điều kiện tiếp tục*
4. Chọn *tiêu chí để xác định một giải pháp từng phần có là giải pháp cuối cùng* không

Quay lui

- Ví dụ: hoán vị của {1, 2, …, n}
  - *biểu diễn giải pháp*
    - mỗi hoán vị là một véc-tơ $s=(s_1, s_2, \ldots, s_n)$, $s_i \neq s_j$ với $\forall i \neq j$
  - *tập $A_1, A_2, \ldots, A_n$ và thứ tự các phần tử*
    - $A_k = \{1, 2, \ldots, n\}$, mọi k
    - các phần tử sẽ được xử lý tăng dần
  - *điều kiện tiếp tục*
    - mỗi giải pháp từng phần $s=(s_1, s_2, \ldots, s_k)$, $s_i \neq s_k$ với $\forall i \neq k$
  - *tiêu chí để xác định một giải pháp từng phần có là giải pháp cuối cùng*
    - $k = n$

Quay lui

- Thuật toán quay lui tổng quát: đệ quy

```
quaylui-dequy (k)
begin
   if (s=(s1, s2, …, sk-1) là giải pháp) then xuly(s)
   else
      for j from 1 to |Ak| do // thử tất cả các giá trị có thể
         sk = akj                  // Ak = {ak1, ak2, …}
         if ((s1, s2, …, sk) là có triển vọng) then
            quaylui-dequy (k+1) // xây dựng thành phần tiếp theo
         endif
      endfor
   endif
end
```

Quay lui

- Thuật toán quay lui tổng quát: lặp

```
quaylui-lap(A₁, A₂, …, Aₙ)
begin
   k=1;  i_k=0
   while (k > 0) do
      i_k=i_k+1
      v=false
      while (v=false and i_k≤|A_k|) do
         s_k=a^k_ik
         if ((s_1,…,s_k) là có triển vọng) then v=true
         else i_k=i_k+1 endif     // thử giá trị tiếp theo
      endwhile
      if (v=true) then
          if (s=(s1,…,sk) là giải pháp cuối cùng) then xuly(s)
          else k=k+1              // xây dựng thành phần tiếp theo
                 i_k=0
          endif
      else k=k-1 endif            // quay lui lại thành phần trước đó
    endwhile
end
```

Một số ứng dụng

- Hoán vị
- Chuỗi nhị phân
- Tập con
- Xếp n con hậu
- Tìm đường đi
- Xếp ba lô 0-1
- Mê cung

Hoán vị

- Bài toán
  - Tìm tất cả các hoán vị của {1,2, …, n}

- Các bước thiết kế
  - *biểu diễn giải pháp*
    - mỗi hoán vị là một véc-tơ $s=(s_1, s_2, \ldots, s_n)$, $s_i \neq s_j$ với $\forall i \neq j$
  - *tập $A_1, A_2, \ldots, A_n$ và thứ tự các phần tử*
    - $A_k = \{1, 2, \ldots, n\}$, mọi k
    - các phần tử sẽ được xử lý tăng dần
  - *điều kiện tiếp tục*
    - mỗi giải pháp từng phần $s=(s_1, s_2, \ldots, s_k)$, $s_i \neq s_k$ với $\forall i \neq k$
  - *tiêu chí để xác định một giải pháp từng phần có là giải pháp cuối cùng*
    - $k = n$

Hoán vị

- Thuật toán

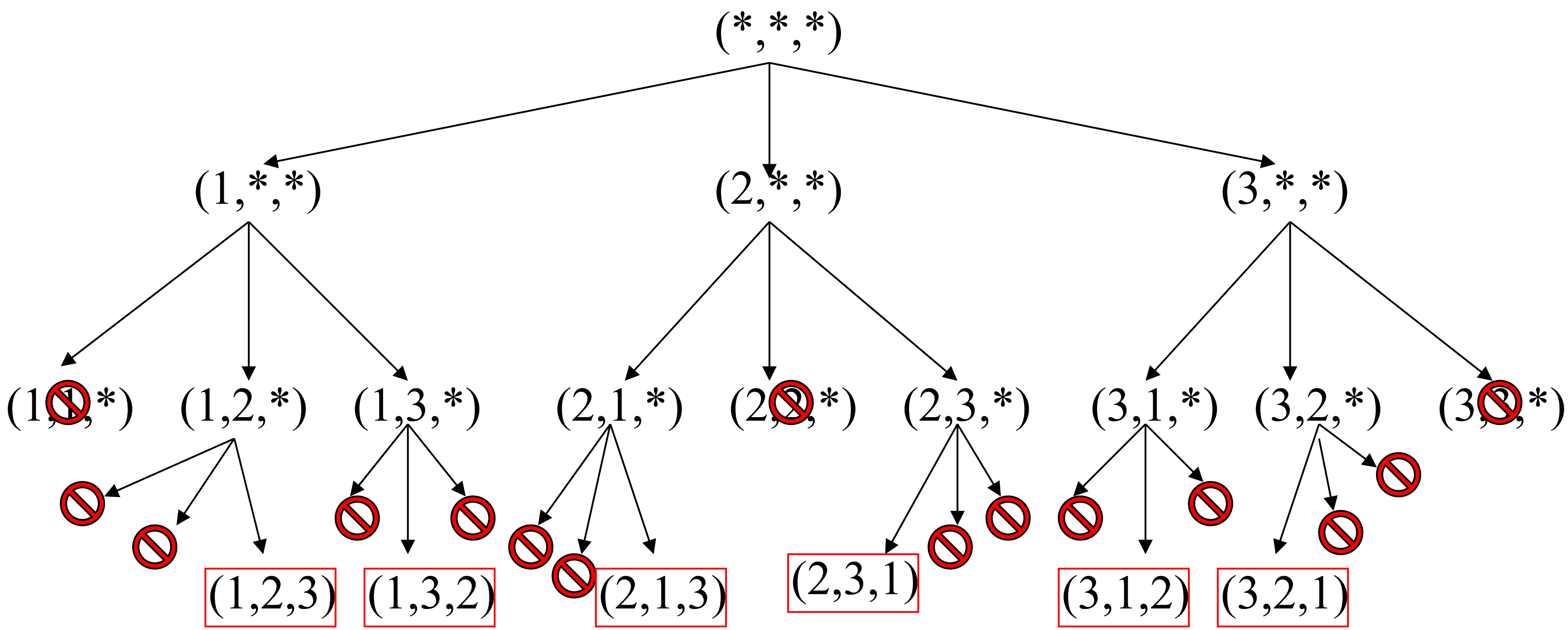
```
hoanvi (k)
begin
  if (k=n+1) then print(s[1..n])
  else
     for i from 1 to n do
        s[k]=i
        if (trienvong(s[1..k])) then
           hoanvi (k+1)
        endif
     endfor
  endif
end
```

```
trienvong(s[1..k])
begin
   for i from 1 to k-1 do
      if (s[k]=s[i]) then
         return false
      endif
   endfor
   return true
end
```

sử dụng: hoanvi(1)

Hoán vị

- Minh hoạ: n=3



Chuỗi nhị phân

- Bài toán
  - Tìm tất cả các chuỗi nhị phân có độ dài n
- Các bước thiết kế
  - *biểu diễn giải pháp*
    - mỗi chuỗi nhị phân là một véc-tơ $s=(s_1, s_2, \ldots, s_n)$
  - *tập $A_1, A_2, \ldots, A_n$ và thứ tự các phần tử*
    - $A_k = \{0, 1\}$, mọi k
    - các phần tử sẽ được xử lý tăng dần
  - *điều kiện tiếp tục*
    - mọi giải pháp từng phần đều có triển vọng
  - *tiêu chí để xác định một giải pháp từng phần có là giải pháp cuối cùng*
    - $k = n$

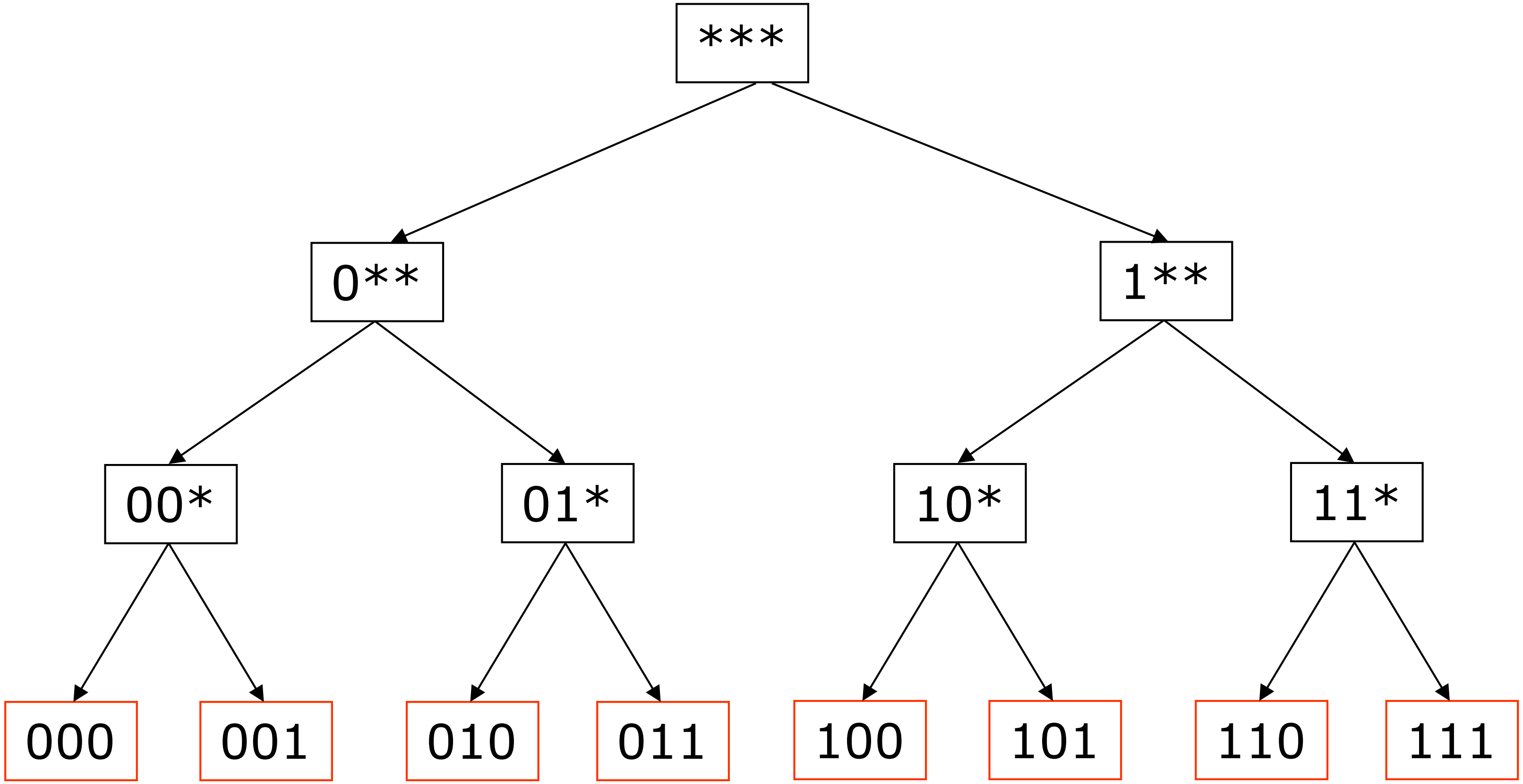
Chuỗi nhị phân

- Thuật toán

```
chuoinhiphan (k)
begin
  if (k=n+1) then print(s[1..n])
  else
     for i from 0 to 1 do
        s[k]=i
        chuoinhiphan (k+1)
     endfor
  endif
end
```

Chuỗi nhị phân

- Minh hoạ: n = 3



Chuỗi nhị phân

- Phân tích thuật toán
  - C(n) thời gian thực thi *chuoinhiphan(n)*
  - Vậy

$$C(n) = \begin{cases} c & if\ n = 0 \\ 2C(n-1) + d & else \end{cases}$$

  - Bằng phương pháp thay thế, ta có

$$C(n) = 2^n(c+d) - d$$

  - Vậy $C(n) = O(2^n)$
  - Thuật toán tối ưu, vì có 2^n chuỗi nhị phân có độ dài bằng n
  - *Không cần bước tỉa bớt các giải pháp thành phần*

Tập con

- Bài toán
  - Cho tập hợp $A=\{a_1,a_2,\ldots,a_n\}$, hãy sinh ra tất cả các tập con của A

- Các bước thiết kế
  - *biểu diễn giải pháp*
    - mỗi tập hợp con được biểu diễn bởi một véc-tơ $s=(s_1, s_2, \ldots, s_n)$, nếu $s_i = 1$ thì a_i thuộc s, $s_i = 0$ thì ngược lại
  - *tập $A_1, A_2, \ldots, A_n$ và thứ tự các phần tử*
    - $A_k = \{0, 1\}$, mọi k
    - các phần tử sẽ được xử lý tăng dần
  - *điều kiện tiếp tục*
    - giải pháp từng phần $s=(s_1, s_2, \ldots, s_k)$ có ít hơn n phần tử, hay $k \leq n$
  - *tiêu chí để xác định một giải pháp từng phần có là giải pháp cuối cùng*
    - $k = n$

Tập con

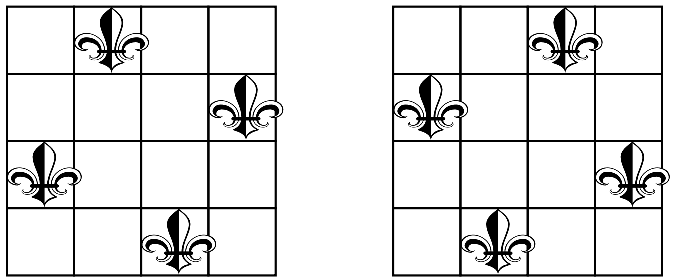
- Thuật toán

```
tapcon (k)
begin
  if (k-1=n) then
      print (s[1..k-1])
  else
     s[k]=0; tapcon (k+1)
     s[k]=1; tapcon (k+1)
  endif
end
```

  - Tại sao không có hàm *trienvong* kiểm tra điều kiện tiếp tục ?
  - Chỉnh sửa thuật toán để tạo ra tất cả các tập hợp con của A có đúng m phần tử (m<n)

Xếp n con hậu

- Bài toán
  - Tìm tất cả các khả năng xếp n con hậu trên một bàn cờ có nxn ô sao cho các con hậu không tấn công nhau, nghĩa là
    - mỗi *hàng* chỉ chứa một con hậu
    - mỗi *cột* chỉ chứa một con hậu
    - mỗi *đường chéo* chỉ chứa một con hậu
  - Ví dụ
    - n=3: không tồn tại giải pháp
    - n=4: có hai giải pháp
    - n=8: có 92 giải pháp



Xếp n con hậu

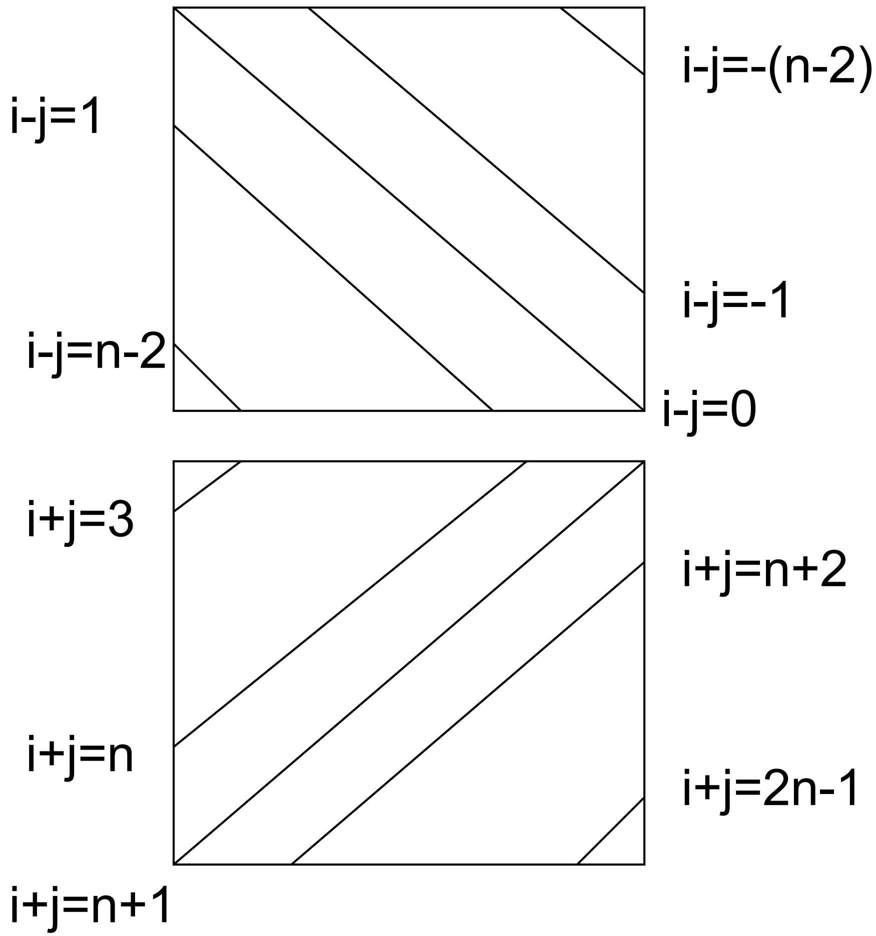
- Các bước thiết kế
  - *biểu diễn giải pháp*
    - giả sử con hậu k được đặt trên hàng k, như thế đối với mỗi con hậu chỉ cần mô tả cột chứa nó. Khi đó giải pháp được biểu diễn bởi véc-tơ s=(s_1, s_2, …, s_n) với s_k = *cột mà con hậu k được đặt trên đó*
  - *tập A_1, A_2, …, A_n và thứ tự các phần tử*
    - A_k = {1,…,n}, mọi k
    - các phần tử sẽ được xử lý tăng dần
  - *điều kiện tiếp tục*
    - giải pháp từng phần s=(s_1, s_2, …, s_k) phải thoả mãn ràng buộc bài toán (mỗi hàng/cột/đường chéo chỉ chứa đúng một con hậu)
  - *tiêu chí để xác định một giải pháp từng phần có là giải pháp cuối cùng*
    - k = n

Xếp n con hậu

- *Điều kiện tiếp tục (1)*
  - giải pháp từng phần $s=(s_1, s_2, \ldots, s_k)$ phải thoả mãn ràng buộc bài toán (mỗi hàng/cột/đường chéo chỉ chứa đúng một con hậu)
  - mỗi hàng chứa đúng một con hậu: điều này luôn đúng do cách biểu diễn giải pháp
  - mỗi cột chứa đúng một con hậu: $s_i \neq s_k$ với mọi $i \neq k$. Chỉ cần kiểm tra $s_k \neq s_i$ với mọi $i \leq k-1$
  - Mỗi đường chéo chỉ chứa một con hậu: $|j-i| \neq |s_j-s_i|$ với mọi $i \neq j$. Chỉ cần kiểm tra $|k-i| \neq |s_k-s_i|$ với mọi $i \leq k-1$
    - Tại sao ?

Xếp n con hậu

- *Điều kiện tiếp tục (2)*
  - Hai con hậu i và j ở trên cùng một đường chéo nếu:
    $i\text{-}s_i = j\text{-}s_j$ hay $j\text{-}i = s_j\text{-}s_i$
    hoặc
    $i+s_i = j+s_j$ hay $j\text{-}i = s_i\text{-}s_j$
    nghĩa là: $|j-i| = |s_j-s_i|$



Xếp n con hậu

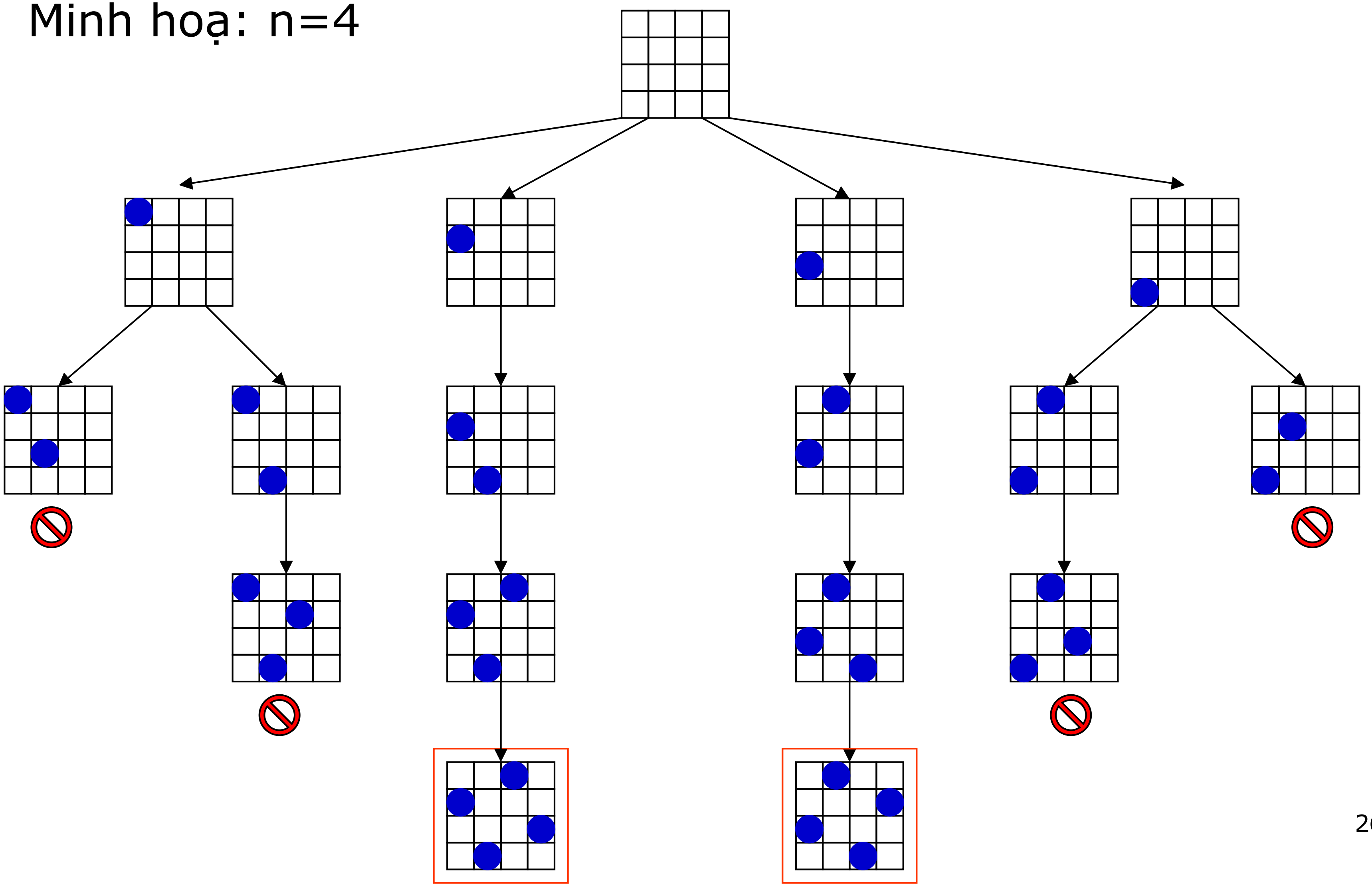
- Thuật toán

```
xephau (k)
begin
   if (k=n+1) then
      print(s[1..n])
   else
      for i from 1 to n do
         s[k]=i
         if (trienvong(s[1..k])) then
            xephau(k+1)
         endif
      endfor
   endif
end
```

```
trienvong(s[1..k])
begin
   for i from 1 to k-1 do
      if (s[k]=s[i] or |i-k|=|s[i]-s[k]|)
         then return false
      endif
   endfor
   return true
end
```

Xếp n con hậu

- Minh hoạ: n=4



Xếp n con hậu

- Nhận xét
  - Với n=4
    - Nếu sử dụng kỹ thuật *tỉa bớt* chỉ có 2 giải pháp cuối cùng được xét
    - Nếu không sử dụng kỹ thuật tỉa bớt, nghĩa là tìm tất cả các khả năng đặt n con hậu trên bàn cờ, sau đó kiểm tra xem mỗi khả năng có hợp lệ không.

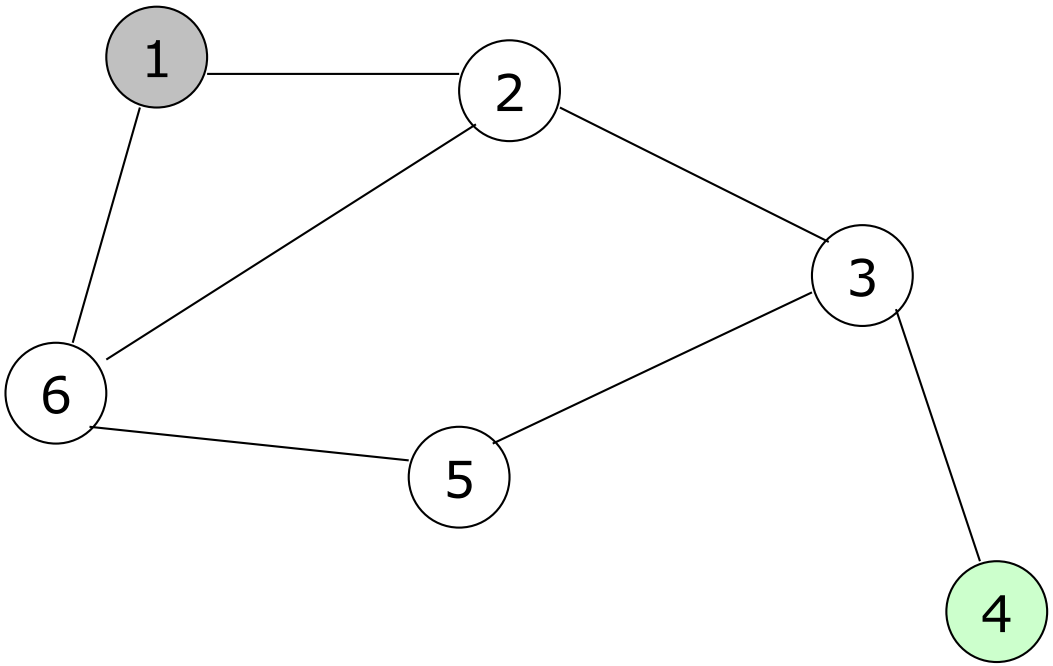
      Có $C_{n}^{n^2}$ khả năng

      Với n = 4, có 1820 khả năng

    - Như vậy, kỹ thuật tỉa bớt của thuật toán quay lui tiết kiệm rất lớn thời gian thực thi

Tìm đường đi

- Bài toán
  - Cho tập hợp n thành phố, có một mạng lưới giao thông nối các thành phố này. Tìm tất cả các đường đi nối hai thành phố cho trước sao cho không đi qua một thành phố nào hai lần



Đi từ 1 đến 4

$1 \to 2 \to 3 \to 4$

$1 \to 2 \to 6 \to 5 \to 3 \to 4$

$1 \to 6 \to 2 \to 3 \to 4$

$1 \to 6 \to 5 \to 3 \to 4$

Tìm đường đi

- Phân tích bài toán
  - Giả sử mạng lưới giao thông nối các thành phố được mô tả bởi ma trận C[1..n,1..n]

$$C[i,j] = \begin{cases} 0 & \text{nếu không tồn tại đường đi trực tiếp giữa i và j} \\ 1 & \text{nếu không tồn tại đường đi trực tiếp giữa i và j} \end{cases}$$

  - Tìm tất cả các đường đi $s=(s_1,s_2,\ldots,s_m)$, s_k trong {1,…,n} chỉ ra thành phố đi đến ở bước k sao cho

    - s_1 là thành phố xuất phát
    - s_m là thành phố đích
    - $C[s_{i-1},s_i] = 1$ (tồn tại đường đi nối trực tiếp giữa hai thành phố s_{i-1} và s_i)
    - $s_i \neq s_j$ với mọi $i \neq j$ (một thành phố không được đi qua hai lần)

Tìm đường đi

- Các bước thiết kế
  - *biểu diễn giải pháp*
    - giải pháp được biểu diễn bởi véc-tơ $s=(s_1, s_2, \ldots, s_m)$ với s_k *thành phố đi đến ở bước k*
  - *tập $A_1, A_2, \ldots, A_n$ và thứ tự các phần tử*
    - $A_k = \{1,\ldots,n\}$, mọi k
    - các phần tử sẽ được xử lý tăng dần
  - *điều kiện tiếp tục*
    - giải pháp từng phần $s=(s_1, s_2, \ldots, s_k)$ phải thoả mãn:

      $s_i \neq s_k$ với mọi i trong $\{1,\ldots,k-1\}$

      $C[s_{k-1},s_k]=1$
  - *tiêu chí để xác định một giải pháp từng phần có là giải pháp cuối cùng*
    - s_k = thành phố đích

Tìm đường đi

- Thuật toán

```
duongdi (k)
begin
  if (s[k-1]=thành phố đích) then
      print(s[1..k-1])
  else
     for j from 1 to n do
       s[k]=j
       if (trienvong(s[1..k])) then
          duongdi (k+1)
       endif
     endfor
  endif
end
```

```
trienvong(s[1..k])
begin
   if (C[s[k-1],s[k]] = 0) then
       return false
   endif
   for i from 1 to k-1 do
       if (s[i]=s[k]) then
           return false
       endif
    endfor
    return true
end
```

s_1 = thành phố xuất phát
duongdi(2)

Xếp ba lô 0-1

- Bài toán
  - Có n đồ vật có trọng lượng $w_1,\ldots,w_n$ và giá trị tương ứng $v_1, \ldots, v_n$. Xếp các đồ vật vào ba lô có sức chứa W sao cho tổng giá trị lớn nhất
- Thuật toán quay lui liệt kê tất cả các giải pháp có thể hoặc giải pháp đầu tiên tìm thấy
- Bài toán xếp ba lô 0-1 cần một giải pháp tối ưu
  - Khi tìm được một giải pháp, thì so sánh với giải pháp trước đó để xác định giải pháp tốt hơn
  - Cuối cùng, tìm được **giải pháp tốt nhất**

Xếp ba lô 0-1

- Các bước thiết kế
  - *biểu diễn giải pháp*
    - giải pháp được biểu diễn bởi véc-tơ $s=(s_1, s_2, \ldots, s_n)$ với nếu $s_i = 1$ thì đồ vật i được chọn, $s_i = 0$ ngược lại thì không
  - *tập $A_1, A_2, \ldots, A_n$ và thứ tự các phần tử*
    - $A_k = \{1,0\}$, mọi k
    - các phần tử sẽ được xử lý tăng dần
  - *điều kiện tiếp tục*
  - *tiêu chí để xác định một giải pháp từng phần có là giải pháp cuối cùng*
    - k = n

      $$\sum_{i=1}^{k} s_i w_i \le W$$
    - So sánh giải pháp với giải pháp trước đó, lưu lại giải pháp có tổng giá trị lớn nhất

Xếp ba lô 0-1

- Thuật toán

giatri-totnhat được khởi gán bằng 0

```
xepbalo (k)
begin
   if (k-1 = n) then
      if ( ∑_{i=1}^{n} s_i w_i ≤ W ) then
         giatri = ∑_{i=1}^{n} s_i v_i
         if (giatri > giatri-totnhat) then
            giatri-totnhat=giatri
            giaphap-totnhat={s_1,s_2,…,s_n}
         endif
      endif
   else
     s[k] = 0; xepbalo (k+1)
     s[k] = 1; xepbalo (k+1)
   endif
end
```

Chọn giải pháp tốt hơn

Xếp ba lô 0-1

- Cải tiến thuật toán (1)
  - Tỉa bớt các lời gọi đệ quy không bao giờ cho giải pháp
  - Chỉnh lại *điều kiện tiếp tục*
  - *điều kiện tiếp tục*
    - giải pháp từng phần $s=(s_1, s_2, \ldots, s_k)$ phải thoả mãn:

      $k \le n$

      $$\sum_{i=1}^{k} s_i w_i \le W$$

Xếp ba lô 0-1

- Cải tiến thuật toán (2)

```
xepbalo (k)
begin
   if (k-1 = n) then
      giatri = ∑_{i=1}^{n} s_i v_i
      if (giatri > giatri-totnhat) then
         giatri-totnhat=giatri
         giaphap-totnhat={s1,s2,…,sn}
      endif
   else
      s[k] = 0; xepbalo (k+1)
      if (∑_{i=1}^{k} s_i w_i ≤ W) then
         s[k] = 1; xepbalo (k+1)
      endif
   endif
end
```

$$giatri = \sum_{i=1}^{n} s_i v_i \qquad giaphap\text{-}totnhat = \{s_1, s_2, \ldots, s_n\} \qquad \sum_{i=1}^{k} s_i w_i \le W$$

Tỉa bớt
Có thể thay bằng hàm triển vọng

Xếp ba lô 0-1

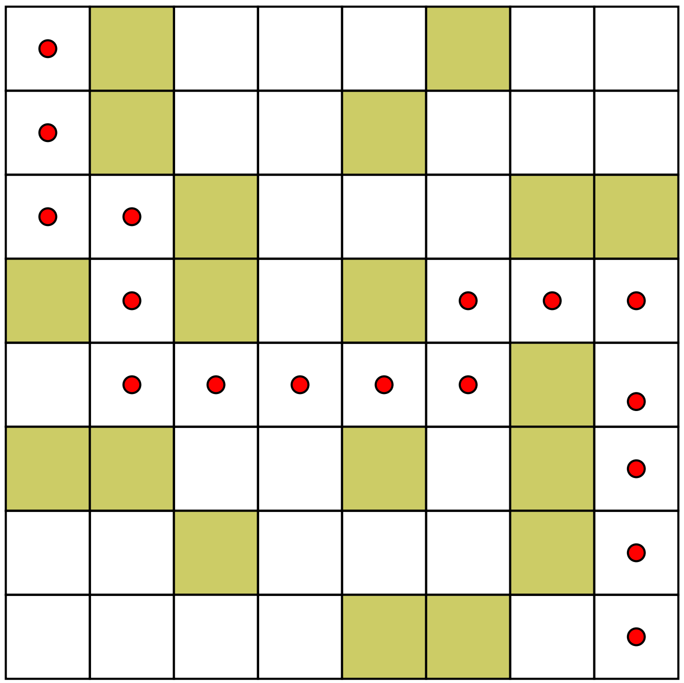
- Thuật toán đơn giản hơn

```
xepbalo (k, W)
begin
   if (k-1 = n) then
      giatri = ∑_{i=1}^{n} s_i v_i
      if (giatri > giatri-totnhat) then
         giatri-totnhat=giatri
         giaphap-totnhat={s_1,s_2,…,s_n}
      endif
   else
      s[k] = 0; xepbalo (k+1, W)
      if (W ≥ w_k) then
         s[k] = 1; xepbalo (k+1, W-w_k)
      endif
   endif
end
```

W được khởi gán là sức chứa tối đa của ba lô

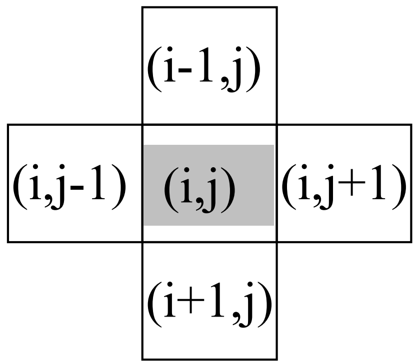
Mê cung

- Bài toán
  - Giả sử một mê cung được định nghĩa là một lưới có nxn ô. Tìm đường đi trong mê cung xuất phát từ ô (1,1) đến ô (n,n)



Chỉ có thể đi qua những ô rỗng

Từ một ô (i,j) có thể đến được một trong các ô:
(i-1,j), (i+1,j), (i,j-1), (i, j+1)



Mê cung

- Phân tích bài toán
  - Dùng ma trận M[1..n,1..n] để lưu trữ mê cung sao cho

$$M[i,j] = \begin{cases} 0 & \text{nếu ô (i,j) rỗng} \\ 1 & \text{nếu ô (i,j) đặc} \end{cases}$$

  - Tìm một đường đi $s=(s_1,s_2,\ldots,s_m)$, s_k trong $\{1,\ldots,n\}x\{1,\ldots,n\}$ chỉ ra chỉ số tương ứng ô đi đến ở bước k sao cho
    - s_1 là ô xuất phát (1,1)
    - s_m là ô đích (n,n)
    - $s_k \neq s_q$ với mọi $k \neq q$ (mỗi ô chỉ đi qua một lần)
    - $M(s_k) = 0$ (ô được đi đến phải rỗng)
    - s_{k-1} và s_k là các ô kề nhau

Mê cung

- Các bước thiết kế
  - *biểu diễn giải pháp*
    - giải pháp được biểu diễn bởi véc-tơ s=(s_1, s_2, …, s_m) với s_k *ô đi đến ở bước k*
  - *tập A_1, A_2, …, A_n và thứ tự các phần tử*
    - A_k = {1,…,n}x{1,…,n}, mọi k
    - các phần tử sẽ được xử lý tăng dần
  - *điều kiện tiếp tục*
    - giải pháp từng phần s=(s_1, s_2, …, s_k) phải thoả mãn:

      $s_k \neq s_q$ với mọi q trong {1,…,k-1}

      M(s_k) = 0

      s_{k-1} và s_k là các ô kề nhau
  - *tiêu chí để xác định một giải pháp từng phần có là giải pháp cuối cùng*
    - s_k là ô đích (n,n)

Mê cung

- Thuật toán (1)

```
mecung (k)
begin
   if (s[k-1]=(n,n)) then print (s[1..k-1])
   else
      s[k].i=s[k-1].i-1; s[k].j=s[k-1].j          //  đi lên
      if (trienvong(s[1..k])) then mecung(k+1) endif
      s[k].i=s[k-1].i+1; s[k].j=s[k-1].j          // đi xuống
      if (trienvong(s[1..k])) then mecung(k+1) endif
      s[k].i=s[k-1].i; s[k].j=s[k-1].j-1          // qua trái
      if (trienvong(s[1..k])) then mecung(k+1) endif
      s[k].i=s[k-1].i; s[k].j=s[k-1].j+1          // qua phải
      if (trienvong(s[1..k])) then mecung(k+1) endif
      endif
end
```

Mê cung

- Thuật toán (2)

```
trienvong (s[1..k])
begin
   if (s[k].i<1 or s[k].i>n or s[k].j<1 or s[k].j>n) then
       return false  // ô ngoài mê cung
   endif
   if (M[s[k].i,s[k].j]=1) then return false endif
   for q from 1 to k-1 do
      if (s[k].i=s[q].i and s[k].j=s[q].j) then return false endif
   endfor
   return true
end
```

Sử dụng: s[1] = (1,1)
mecung(2)