Bài toán liệt kê

Nội dung

1. Thuật toán

2. Phương pháp sinh

Giới thiệu bài toán

Bài toán đếm	Bài toán liệt kê
- Đếm số lượng cấu hình thoả mãn ràng buộc	- Chỉ ra tường minh các cấu hình thoả mãn ràng buộc
- Tìm công thức cho số lượng cấu hình thoả mãn	- Thuật toán/Cách sinh ra tất cả các cấu hình thoả mãn

- Nguyên tắc trong bài toán liệt kê:
  - Không được lặp lại các cấu hình
  - Không bỏ sót các cấu hình
- Khó khăn:
  - Khả năng tính toán
  - Khả năng lưu trữ

Khái niệm thuật toán (1)

- Khái niệm thuật toán đã tồn tại từ thời cổ đại. Các thuật toán số học, chẳng hạn như thuật toán chia, được sử dụng bởi các nhà toán học Babylon cổ đại vào khoảng 2500 TCN và các nhà toán học Ai Cập vào khoảng 1550 TCN.
- Bản thân từ thuật toán (algorithm) từ bắt nguồn từ nhà toán học thế kỷ thứ 9 Muḥammad ibn Mūsā al-Khwārizmī, tên ông được Latinh hóa thành Algoritmi.

Khái niệm thuật toán (2)

- Thuật toán là tập hợp hữu hạn các bước có hướng dẫn rõ ràng cần thiết để thu được lời giải mong muốn.
- Một số đặc điểm của thuật toán
  - *Đầu vào*: Thuật toán nhận dữ liệu vào từ một nguồn nào đó
  - *Đầu ra*: Tương ứng với dữ liệu đầu vào, thuật toán trả ra dữ liệu đầu ra sau xử lý
  - Chính xác: Các tính toán trong các bước cần được phát biểu tường minh, rõ ràng
  - *Hữu hạn*: Thuật toán phải trả ra kết quả sau hữu hạn bước với mọi dữ liệu đầu vào
  - *Đơn trị:* Các kết quả tính toán trung gian chỉ phụ thuộc vào đầu vào và các tính toán nằm trong bước trước
  - *Tổng quát*: Thuật toán có thể áp dụng với mọi bài toán trong dạng đã cho

Ví dụ

- Ví dụ 1: Tìm số lớn nhất trong 3 số đã biết
  - Đầu vào: a, b, c
  - Thuật toán:
    - Bước 1: Đặt biến $x = a$
    - Bước 2: Nếu $b > x$ thì $x = b$
    - Bước 3: Nếu $c > x$ thì $x = c$
  - Đầu ra: x
- Ví dụ 2: Tìm số lớn nhất trong một dãy số cho trước
- Ví dụ 3: Tìm số nguyên tố nhỏ nhất lớn hơn số n cho trước

Độ phức tạp thuật toán

- ***Vấn đề***: Thuật toán đúng nhưng không thể tìm ra đáp án vì giới hạn năng lực (thời gian, bộ nhớ) tính toán.
- Phần tích thuật toán là đánh giá về thời gian tính toán và dung lượng bộ nhớ cần cho thuật toán.
- Trong khuôn khổ môn học này, chúng ta chỉ quan tâm tới thời gian tính toán của thuật toán.
- Thời gian tính toán là một hàm của dữ liệu đầu vào. Tuy nhiên khó thực hiện điều này, nên chúng ta đánh giá qua kích thước dữ liệu đầu vào.

Đánh giá thời gian tính toán

- Thời gian tính toán:
  - Thời gian tốt nhất: Thời gian tối thiểu cần thiết để thực hiện thuật toán với mọi bộ dữ liệu đầu vào có kích thước n.
  - Thời gian tồi nhất:
  - Thời gian trung bình:
- Cách tính thời gian tính toán
  - Đếm số câu lệnh cần thực hiện
  - Đếm số phép toán cần thực hiện
- Phân tích thuật toán tìm số lớn nhất trong n số!

Thời gian tính toán ~ Tốc độ tăng thời gian tính toán

n	$t(n) = 60n^2 + 9n + 9$	$60n^2$
10	9099	6000
100	600909	600000
1000	60009009	60000000
10000	6000090009	6000000000

- $t(n) = \Theta(n^2)$ ta nói hàm $t(n)$ bậc 2.

Định nghĩa

Giả sử f và g là các hàm đối số nguyên dương.

- Ta viết $f(n) = O(g(n))$ và nói $f(n)$ có **bậc không quá** $g(n)$ nếu tồn tại hằng số dương C_1 và số nguyên dương N_1 sao cho $|f(n)| \leq C_1|g(n)|$ với mọi $n \geq N_1$.
- Ta viết $f(n) = \Omega(g(n))$ và nói $f(n)$ có **bậc ít nhất** là $g(n)$ nếu tồn tại hằng số dương C_2 và số nguyên dương N_2 sao cho $|f(n)| \geq C_2|g(n)|$ với mọi $n \geq N_2$.
- Ta viết $f(n) = \Theta(g(n))$ và nói $f(n)$ có bậc là $g(n)$ nếu $f(n) = O(g(n))$ và $f(n) = \Omega\big(g(n)\big)$.

Ví dụ (1)

- **Tìm O, Ω và Θ của $f(n) = 60n^2 + 9n + 9$**
- Ta có $f(n) = 60n^2 + 9n + 9 \leq 60n^2 + 9n^2 + n^2 = 70n^2$ do $n \geq 1$, ta chọn $C_1 = 70$, theo định nghĩa ta suy ra $60n^2 + 9n + 9 = O(n^2)$.
- Do $60n^2 + 9n + 9 \geq 60n^2$, ta chọn $C_2 = 60$, theo định nghĩa suy ra $60n^2 + 9n + 9 = \Omega(n^2)$.
- Do $60n^2 + 9n + 9 = O(n^2)$ và $60n^2 + 9n + 9 = \Omega(n^2)$, ta suy ra $60n^2 + 9n + 9 = \Theta(n^2)$

Ví dụ (2)

- Ta có $1^k + 2^k + \cdots + n^k \leq n^k + n^k + \cdots + n^k = n^{k+1}$ với $n \geq 1$, ta suy ra $1^k + 2^k + \cdots + n^k = O(n^{k+1})$
- Mặt khác, ta có $1^k + 2^k + \cdots + n^k \geq \left\lceil \frac{n}{2} \right\rceil^k + \cdots + (n-1)^k + n^k \geq \left\lceil \frac{n}{2} \right\rceil^k + \cdots + \left\lceil \frac{n}{2} \right\rceil^k = \left(\frac{n}{2}\right)\left(\frac{n}{2}\right)^k = n^{k+1}/2^{k+1}$, nên $1^k + 2^k + \cdots + n^k = \Omega\left(n^{k+1}\right)$.
- Do đó, $1^k + 2^k + \cdots + n^k = \Theta(n^{k+1})$

Ví dụ (3): Chứng minh $\lg(n!) = \Theta\big(\mathrm{n}\lg(n)\big)$

- Ta có $\lg(n!) = \ \lg(n) + \lg(n-1) + \cdots + \lg(2) + \ \lg(1)$. Do hàm lg là hàm tăng nên $\lg(n) + \ \lg(n-1) + \ \ldots + \lg(2) + \ \lg(1) \leq \lg(n) + \ \lg(n) + \cdots + \lg(n) = \ \mathrm{n}\ \lg(n)$. Vì vậy, $\lg(n!) = O(\mathrm{n}\ \lg(n))$.
- Mặt khác, ta cũng có $\lg(n) + \ \lg(n-1) + \cdots + \lg(2) + \lg(1) \geq \lg(n) + \lg(n-1) + \cdots + \lg\left(\frac{n}{2}\right) \geq \lg\left(\frac{n}{2}\right) + \cdots + \lg\left(\frac{n}{2}\right) = \frac{n}{2}\lg\left(\frac{n}{2}\right) \geq \mathrm{n}\ \lg(n)/4$. Suy ra, $\lg(n!) = \Omega(n\ \lg(n))$.
- Vậy, ta suy ra $\lg(n!) = \Theta\big(\mathrm{n}\lg(n)\big)$

Nội dung

1. Thuật toán

2. Phương pháp sinh

Phương pháp sinh

- Phương pháp sinh có thể áp dụng để giải bài toán liệt kê tổ hợp nếu như 2 điều kiện sau thoả mãn:
  - Có thể xác định được một thứ tự trên các cấu hình tổ hợp cần liệt kê. Từ đó có thể xác định được cấu hình đầu tiên và cấu hình cuối cùng trong thứ tự đã xác định.
  - Xây dựng được thuật toán từ cầu hình chưa phải là cấu hình cuối cùng đang xét, đưa ra cấu hình kế tiếp của nó.

Bài toán: Liệt kê tất cả các dãy nhị phân độ dài n

- Ví dụ với $n = 3$ (liệt kê hình bên)
- Nhận xét:
  - Dãy đầu tiên 0 0 0
  - Dãy cuối cùng 1 1 1
  - Dãy tiếp theo = dãy liền trước + 1
- Quy luật sinh dãy số kế tiếp:
  - Tìm i đầu tiên theo thứ tự $n, n-1, n-2, \dots, 1$ thoả mãn $b_i = 0$
  - Gán $b_i = 1$ và $b_j = 0$ với $j > i$
- Ví dụ về quy luật sinh dãy số kế tiếp: Dãy nhị phân độ dài 10, 1101011111. Ta có $i = 5$. Do đó, đặt $b_5 = 1$ và $b_j = 0, j = 6,7,8,9,10$ ta thu được nhị phân kế tiếp là 1101100000.

b	$p(b)$
0 0 0	0
0 0 1	1
0 1 0	2
0 1 1	3
1 0 0	4
1 0 1	5
1 1 0	6
1 1 1	7

Bài toán: Liệt kê tập con m phần tử của tập n phần tử

- Bài toán: cho tập $X = \{1,2, \dots, n\}$. Hãy liệt kê các tập con m phần tử của X. Mỗi tập con m phần tử của X có thể biểu diễn bởi bộ có thứ tự gồm m thành phần $a = (a_1, a_2, \dots, a_m\}$ thoả mãn $1 \leq a_1 < a_2 < \cdots < a_m \leq n$.
- Ví dụ: $X = \{1,2,3,4,5\}$ và $m = 3$. Danh sách các tập con 3 phần tử của X (hình bên)
- Xây dựng quy tắc xác định thứ tự: Ta nói tập con $a = (a_1, a_2, \dots, a_m)$ đứng trước tập con $a' = (a'_1, a'_2, \dots, a'_m)$ nếu tìm được số k $(1 \leq k \leq m)$ sao cho $a_1 = a'_1, a_2 = a'_2, \dots, a_{k-1} = a'_{k-1}, a_k < a_k{}'$
- Theo thứ tự, ta có thấy tập con đầu tiên là $\{1,2, \dots, m\}$ (ví dụ $\{1,2,3\}$) và tập con cuối cùng là $\{n-m+1, n-m+2, \dots, n\}$ (ví dụ {3,4,5}).

1	2	3
1	2	4
1	2	5
1	3	4
1	3	5
1	4	5
2	3	4
2	3	5
2	4	5
3	4	5

Bài toán: Liệt kê tập con m phần tử của tập n phần tử

- Giả sử $a = \{a_1, a_2, \dots, a_m\}$ là tập con đang có và chưa phải là cuối cùng. Chúng ta sẽ xác định thuật toán sinh tập con kế tiếp theo thứ tự như sau:
  - Tìm từ bên phải dãy $a_1, a_2, \dots, a_m$ phần tử $a_i \neq n - m + i$ (không phải là tập con cuối cùng)
  - Thay a_i bởi $a_i + 1$
  - Thay a_j bởi $a_i + j - i$ với $j = i + 1,\ i + 2, \dots, m$
- Ví dụ: $n = 6, m = 4$, giả sử tập con đang xem xét $\{1,2,5,6\}$, cần xây tập con kế tiếp. Ta có $i = 2$, thay $a_2 = 3$ và $a_3 = 4, a_4 = 5$. Suy ra tập con kế tiếp $\{1,3,4,5\}$.

Bài toán: Liệt kê các hoán vị của tập n phần tử

- Bài toán: cho tập $X = \{1,2, \dots, n\}$. Hãy liệt kê các hoán vị từ n phần tử của X. Mỗi hoán vị từ n phần tử của X có thể biểu diễn bởi bộ có thứ tự gồm n phần tử $a = (a_1, a_2, \dots, a_n\}$ thoả mãn $a_i \in X, i = 1,2,3, \dots, n, a_p \neq a_q, p \neq q$
- $X = \{1,2,3\}$, các hoán vị 3 phần tử của X trong hình bên.
- Xây dựng quy tắc xác định thứ tự: Ta nói hoán vị $a = (a_1, a_2, \dots, a_{\mathrm{n}})$ đứng trước tập con $a' = (a'_1, a'_2, \dots, a'_{\mathrm{n}}\}$ nếu tìm được số k $(1 \leq k \leq \mathrm{n})$ sao cho $a_1 = a'_1, a_2 = a'_2, \dots, a_{k-1} = a'_{k-1}, a_k < a_k{}'$
- Theo thứ tự, hoán vị đầu tiên $(1,2, \dots, n\ )$ và hoán vị cuối cùng là $(n, n-1, n-2, \dots, 1)$

1	2	3
1	3	2
2	1	3
2	3	1
3	1	2
3	2	1

Bài toán: Liệt kê các hoán vị của tập n phần tử

- Giả sử đang xét hoán vị $a = (a_1, a_2, \dots, a_n)$, hoán vị này chưa phải hoán vị cuối cùng, nhiệm vụ là tìm hoán vị kế tiếp của a. Thuật toán tìm hoán vị kế tiếp như sau:
  - Tìm từ phải qua trái hoán vị đang có chỉ số j đầu tiên thoả mãn $a_j < a_{j+1}$
  - Tìm a_k là số nhỏ nhất và lớn hơn a_j trong các số ở bên phải a_j
  - Đổi chỗ a_j với a_k
  - Lật ngược đoạn từ a_{j+1} đến a_n

Đệ quy giải các bài toán liệt kê

ĐỆ QUY QUAY LUI

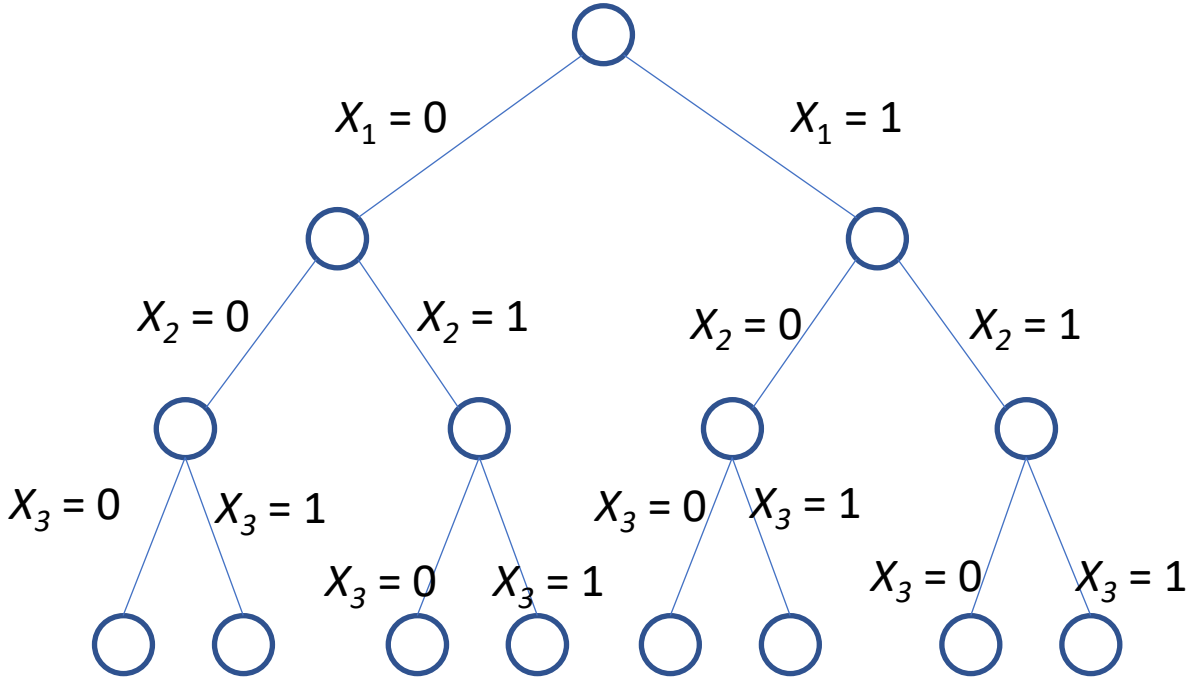
- Áp dụng để giải các bài toán liệt kê, bài toán tối ưu tổ hợp
- A = {(x_1, x_2, . . ., x_n) | $x_i \in A_i$, $\forall\ i$ = 1,. . ., n}
- Liệt kê tất cả các bộ $x \in A$ thoả mãn một thuộc tính P nào đó
- Thủ tục TRY(k):
  - Thử các giá trị v có thể gán cho x_k mà không vi phạm thuộc tính P
  - Với mỗi giá trị hợp lệ v:
    - Gán v cho x_k
    - Nếu $k < n$: gọi đệ quy TRY(k+1) để thử tiếp giá trị cho x_{k+1}
    - Nếu $k = n$: ghi nhận cấu hình

ĐỆ QUY QUAY LUI

```
TRY(k)
  Begin
    Foreach v thuộc A_k
     if check(v,k) /* kiểm tra xem v có hợp lệ không */
       Begin
         x_k = v;
         if(k = n) ghi_nhan_cau_hinh;
         else TRY(k+1);
       End
  End
Main()
Begin
  TRY(1);
End
```

ĐỆ QUY QUAY LUI: liệt kê xâu nhị phân

- Mô hình hoá cấu hình:
  - Mảng **x[n]** trong đó **x[i]** $\in$ {0,1} là bít thứ **i** của xâu nhị phân (**i= 1, . . ., n**)



ĐỆ QUY QUAY LUI: liệt kê xâu nhị phân

- Mô hình hoá cấu hình:
  - Mảng **x[n]** trong đó **x[i]** ∈{0,1} là bít thứ **i** của xâu nhị phân
    (**i= 1, . . ., n**)

```
void solution(){
   for(int i = 1; i<= n; i++)
                    printf("%d ",x[i]);
    printf("\n");
}
int check(int v, int k){
    return 1;
}
void Try(int k){
    for(int v = 0; v <= 1; v++){
        if(check(v,k)){
            x[k] = v;
            if(k == n) solution();
            else Try(k+1);
        }
    }
}
int main() {
  Try(1);
}
```

ĐỆ QUY QUAY LUI: liệt kê xâu nhị phân

- Liệt kê các xâu nhị phân sao cho không có 2 bit 1 nào đứng cạnh nhau
- Mô hình hoá cấu hình:
  - Mảng **x[n]** trong đó **x[i]** $\in$ {0,1} là bít thứ **i** của xâu nhị phân

    (**i= 1, . . ., n**)
  - Thuộc tính *P*: không có 2 bít 1 nào đứng cạnh nhau

```
int TRY(int k) {
  for(int v = 0; v <= 1; v++){
    if(x[k-1] + v < 2){
      x[k] = v;
      if(k == n)
        printSolution();
      else TRY(k+1);
    }
  }
}

int main() {
  x[0] = 0;
  TRY(1);
}
```

ĐỆ QUY QUAY LUI: liệt kê tổ hợp

- Liệt kê các tổ hợp chập *k* của 1, 2, …, *n*
- Mô hình hoá cấu hình:
  - Mảng *x*[*k*] trong đó *x*[*i*] ∈{1, . . ., *n*} là phần tử thứ *i* của cấu hình tổ hợp (*i* = 1, . . ., *k*)
  - Thuộc tính *P*: *x*[*i*] < *x*[*i*+1], với mọi *i* = 1, 2, …, *k*-1

```
int TRY(int i) {
  for(int v = x[i-1]+1; v <= n-k+i;
          v++){
     x[i] = v;
     if(i == k)
        printSolution();
      else TRY(i+1);
     }
}

int main() {
  x[0] = 0;
  TRY(1);
}
```

ĐỆ QUY QUAY LUI: liệt kê hoán vị

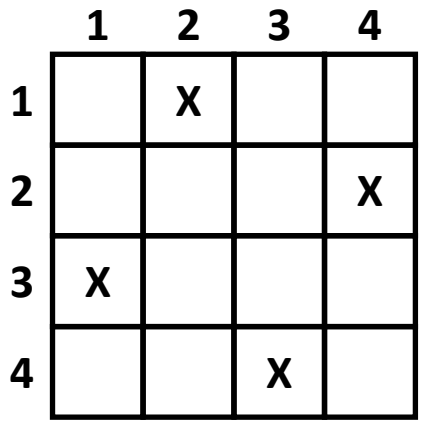
- Liệt kê các hoán vị của 1, 2, …, *n*
- Mô hình hoá cấu hình:
  - Mảng *x*[1,. . . , *n*] trong đó *x*[*i*] $\in${1, . . ., *n*} là phần tử thứ *i* của cấu hình hoán vị (*i* = 1, . . ., *n*)
  - Thuộc tính *P*:
    - $x[i] \neq x[j]$, với mọi $1 \leq i < j \leq n$
  - Mảng đánh dấu *m*[*v*] = true (false) nếu giá trị *v* đã xuất hiện (chưa xuất hiện) trong cấu hình bộ phận, với mọi *v* = 1, …, *n*

```
void TRY(int i) {
  for(int v = 1; v <= n; v++){
    if(!m[v]) {
      x[i] = v;
      m[v] = true; // đánh dấu
      if(i == n)
        printSolution();
      else TRY(i+1);
      m[v] = false;// khôi phục
    }
  }
}

void main() {
  for(int v = 1; v <= n; v++)
     m[v] = false;
  TRY(1);
}
```

ĐỆ QUY QUAY LUI: bài toán xếp hậu

- Xếp n quân hậu trên một bàn cờ quốc tế sao cho không có 2 quân hậu nào ăn được nhau
- Mô hình hoá
  - $x[1, \ldots, n]$ trong đó $x[i]$ là hàng của quân hậu xếp trên cột i, với mọi $i = 1, \ldots, n$
  - Thuộc tính P
    - $x[i] \neq x[j]$, với mọi $1 \leq i < j \leq n$
    - $x[i] + i \neq x[j] + j$, với mọi $1 \leq i < j \leq n$
    - $x[i] - i \neq x[j] - j$, với mọi $1 \leq i < j \leq n$



Lời giải $x = (3, 1, 4, 2)$

ĐỆ QUY QUAY LUI: bài toán xếp hậu

```
int check(int v, int k) {
  // kiểm tra xem v có thể gán được
  // cho x[k] không
  for(int i = 1; i <= k-1; i++) {
    if(x[i] == v) return 0;
    if(x[i] + i == v + k) return 0;
    if(x[i] – i == v – k) return 0;
  }
  return 1;
}
```

```
void TRY(int k) {
  for(int v = 1; v <= n; v++) {
    if(check(v,k)) {
      x[k] = v;
      if(k == n) printSolution();
      else TRY(k+1);
    }
  }
}
void main() {
    TRY(1);
}
```

ĐỆ QUY QUAY LUI: bài toán Sudoku

- Điền các chữ số từ 1 đến 9 vào các ô trong bảng vuông 9x9 sao cho trên mỗi hàng, mỗi cột và mỗi bảng vuông con 3x3 đều có mặt đầy đủ 1 chữ số từ 1 đến 9

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

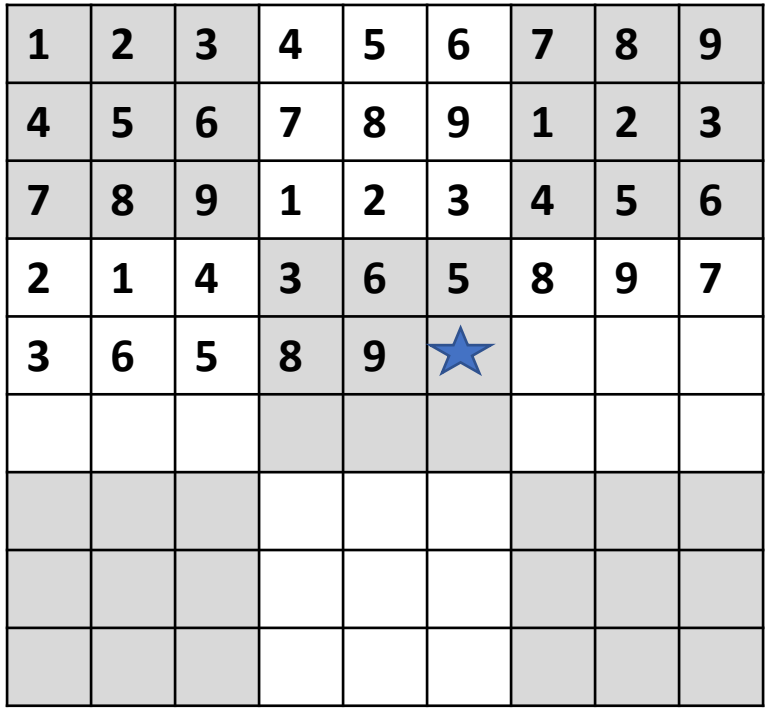
ĐỆ QUY QUAY LUI: bài toán Sudoku

- Mô hình hoá
  - Mảng 2 chiều $x[0..8, 0..8]$
  - Thuộc tính P
    - $x[i, j_2] \neq x[i, j_1]$, với mọi $i = 0,\ldots,8$, và $0 \leq j_1 < j_2 \leq 8$
    - $x[i_1, j] \neq x[i_2, j]$, với mọi $j = 0,\ldots,8$, và $0 \leq i_1 < i_2 \leq 8$
    - $x[3I+i_1, 3J+j_1] \neq x[3I+i_2, 3J+j_2]$, với mọi $I, J = 0,\ldots, 2$, và $i_1, j_1, i_2, j_2 \in \{0,1, 2\}$ sao cho $i_1 \neq i_2$ hoặc $j_1 \neq j_2$

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

ĐỆ QUY QUAY LUI: bài toán Sudoku

- Thứ tự duyệt: từ ô (0,0), theo thứ tự từ trái qua phải và từ trên xuống dưới



ĐỆ QUY QUAY LUI: bài toán Sudoku

```
bool check(int v, int r, int c){
  for(int i = 0; i <= r-1; i++)
    if(x[i][c] == v) return false;
  for(int j = 0; j <= c-1; j++)
    if(x[r][j] == v) return false;
  int I = r/3;
  int J = c/3;
  int i = r - 3*I;
  int j = c - 3*J;
  for(int i1 = 0; i1 <= i-1; i1++)
    for(int j1 = 0; j1 <= 2; j1++)
      if(x[3*I+i1][3*J+j1] == v)
        return false;
  for(int j1 = 0; j1 <= j-1; j1++)
    if(x[3*I+i][3*J+j1] == v)
       return false;
  return true;
}
```

```
void TRY(int r, int c){
  for(int v = 1; v <= 9; v++){
    if(check(v,r,c)){
      x[r][c] = v;
      if(r == 8 && c == 8){
        printSolution();
      }else{
        if(c == 8) TRY(r+1,0);
        else TRY(r,c+1);
      }
    }
  }
}

void main(){
  TRY(0,0);
}
```