**Thuật toán quay lui – Đệ quy**

# **Thuật toán Quay lui (Backtracking)**

Quay lui, vét cạn, thử sai, duyệt … là một số tên gọi tuy không đồng nghĩa nhưng cùng chỉ một phương pháp trong tin học: *tìm nghiệm của một bài toán bằng cách xem xét tất cả các phương án có thể*. Đối với con người phương pháp này thường là không khả thi vì số phương án cần kiểm tra lớn. Tuy nhiên đối với máy tính, nhờ tốc độ xử lí nhanh, máy tính có thể giải rất nhiều bài toán bằng phương pháp quay, lui vét cạn.

Ưu điểm của phương pháp quay lui, vét cạn là *luôn đảm bảo tìm ra nghiệm đúng, chính xác*. Tuy nhiên, hạn chế của phương pháp này là *thời gian thực thi lâu, độ phức tạp lớn*. Do đó vét cạn thường chỉ phù hợp với các bài toán có kích thước nhỏ.

**Thuật toán quay lui** dùng để giải bài toán liệt kê các cấu hình. Mỗi cấu hình được xây dựng bằng cách xây dựng từng phần tử, mỗi phần tử được chọn bằng cách thử tất cả các khả năng. Giả sử cấu hình cần liệt kê có dạng x[1..n], khi đó thuật toán quay lui thực hiện qua các bước:

1) Xét tất cả các giá trị x[1] có thể nhận, thử cho x[1] nhận lần lượt các giá trị đó. Với mỗi giá trị thử gán cho x[1] ta sẽ:

2) Xét tất cả các giá trị x[2] có thể nhận, lại thử cho x[2] nhận lần lượt các giá trị đó. Với mỗi giá trị thử gán cho x[2] lại xét tiếp các khả năng chọn x[3] … cứ tiếp tục như vậy đến bước: …

n) Xét tất cả các giá trị x[n] có thể nhận, thử cho x[n] nhận lần lượt các giá trị đó, thông báo cấu hình tìm được <x[1], x[2], …, x[n]>.

Trong nhiều bài toán, việc tìm nghiệm có thể quy về việc tìm vector hữu hạn , độ dài vectơ có thể xác định trước hoặc không. Vectơ này cần phải thoả mãn một số điều kiện tùy thuộc vào yêu cầu của bài toán. Các thành phần được chọn ra từ tập hữu hạn .

Tuỳ từng trường hợp mà bài toán có thể yêu cầu: tìm một nghiệm, tìm tất cả nghiệm hoặc đếm số nghiệm.

Trên thực tế, thuật toán quay lui vét cạn thường được dùng bằng mô hình đệ quy như sau:

|  |
| --- |
| **void backtrack(i);**  **{**  **for <mọi giá trị V có thể gán cho x[i]>**  **{**  **<Thử cho x[i] := V>;**  **if <x[i] là phần tử cuối cùng trong cấu hình>**  **<Thông báo cấu hình tìm được**>  **else**  **{**  **<Ghi nhận việc cho x[i] nhận giá trị V (nếu cần)>;**  **backtrack (i + 1);** {Gọi đệ quy để chọn tiếp x[i+1]}  **<Nếu cần, bỏ ghi nhận việc thử x[i] := V để thử giá trị khác>;**  **}**  **}**  **Thuật toán quay lui sẽ bắt đầu bằng lời gọi backtrack(1)** |

Khi áp dụng lược đồ tổng quát của thuật toán quay lui cho các bài toán cụ thể, có ba vấn đề quan trọng cần làm:

* Tìm cách biểu diễn nghiệm của bài toán dưới dạng một dãy các đối tượng được chọn dần từng bước .
* Xác định tập các ứng cử viên được chọn làm thành phần thứ của nghiệm. Chọn cách thích hợp để biểu diễn .
* Tìm các điều kiện để một vectơ đã chọn là nghiệm của bài toán.

## **2. Một số ví dụ áp dụng**

2.1 Sinh dãy nhị phân có độ dài n

|  |  |
| --- | --- |
| input | output |
| 3 | 000  001  010  011  100  101  110  111 |

Thuật toán:

|  |
| --- |
| void Try(int k) {  // Duyệt qua 2 giá trị của bit: 0 và 1  for (int i = 0; i <= 1; i++) {  x[k] = i; // Gán giá trị cho vị trí k trong mảng x  // Nếu k đã bằng N, tức là đã sinh xâu nhị phân đủ độ dài  if (k == N) {  // In ra kết quả (xâu nhị phân)  for (int j = 1; j <= N; j++) {  cout << x[j];  }  cout << endl; // Xuống dòng sau mỗi xâu nhị phân  }  else {  // Nếu chưa đủ độ dài, tiếp tục gọi đệ quy sinh phần tử tiếp theo  Try(k + 1);  }  }  } |

2.2 **Tổ hợp**

**- Bài toán tính tổ hợp:** Giải quyết bài toán tính số cách chọn k phần tử từ n phần tử mà không quan tâm đến thứ tự. Tổ hợp có thể được sử dụng để tính số lượng các kết quả có thể có trong các tình huống chọn lựa.

-Ví dụ: Chọn 3 sinh viên từ một nhóm 5 sinh viên, tổ hợp sẽ cho phép bạn tính toán số cách chọn.

**- Bài toán sinh tổ hợp**

Một tổ hợp chập của là một tập con phần tử của tập phần tử. Chẳng hạn tập có các tổ hợp chập là:

.

Vì trong tập hợp các phần tử không phân biệt thứ tự nên tập cũng là tập , do đó, ta coi chúng chỉ là một tổ hợp.

Tổ hợp chập 3 của tập {1, 2, 3, 4} sẽ là {1, 2, 3}, {1, 2, 4}, {1, 3, 4}, {2, 3, 4}.

**🡪Tổ hợp không quan tâm đến thứ tự**

**Bài toán:** Hãy xác định tất cả các tổ hợp chập của tập phần tử. Để đơn giản ta chỉ xét bài toán tìm các tổ hợp của tập các số nguyên từ đến . Đối với một tập hữu hạn bất kì, bằng cách đánh số thứ tự của các phần tử, ta cũng đưa được về bài toán đối với tập các số nguyên từ đến . Nghiệm của bài toán tìm các tổ hợp chập của phần tử phải thoả mãn các điều kiện sau:

* có thành phần: ;
* lấy giá trị trong tập ;
* Ràng buộc: với mọi giá trị từ đến **(vì tập hợp không phân biệt thứ tự phần tử nên ta sắp xếp các phần tử theo thứ tự tăng dần).**

Ta có: , do đó tập (tập các ứng cử viên được chọn làm thành phần thứ ) là từ đến . Để điều này đúng cho cả trường hợp ta thêm vào .

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;  int x[101]; // Mảng lưu tổ hợp  int n, k;  void in()  {  for (int i = 1; i <= k; ++i)  cout << x[i] << " ";  cout << "\n";  }  void tohop(int i, int start)  {  for (int j = start; j <= n; ++j)  {  x[i] = j;  if (i == k)  {  in();  }  else  {  tohop(i + 1, j + 1);  }  }  }  main()  {  cin >> n >> k;  tohop(1, 1);  } |

Ví dụ về Input / Output của chương trình:

|  |  |
| --- | --- |
| TOHOP.INP | TOHOP.OUT |
| 4 2 | 1 2  1 3  1 4  2 3  2 4  3 4 |

Theo công thức, số lượng tổ hợp chập của là:

## **2.2. Chỉnh hợp lặp**

Chỉnh hợp lặp chập của là một dãy thành phần, mỗi thành phần là một phần tử của tập phần tử, có xét đến thứ tự và không yêu cầu các thành phần khác nhau.

Một ví dụ dễ thấy nhất của chỉnh hợp lặp là các dãy nhị phân. Một dãy nhị phân độ dài là một chỉnh hợp lặp chập của tập phần tử . Các dãy nhị phân độ dài :

.

Vì có xét thứ tự nên dãy và dãy là dãy khác nhau.

Như vậy, bài toán xác định tất cả các chỉnh hợp lặp chập của tập phần tử yêu cầu tìm các nghiệm như sau:

* có thành phần: ;
* lấy giá trị trong tập ;
* Không có ràng buộc nào giữa các thành phần.

Chú ý là cũng như bài toán tìm tổ hợp, ta chỉ xét đối với tập số nguyên từ đến . Nếu phải tìm chỉnh hợp không phải là tập các số nguyên từ đến thì ta có thể đánh số các phần tử của tập đó để đưa về tập các số nguyên từ đến .

Sau đây là chương trình hoàn chỉnh, chương trình sử dụng mô hình đệ quy để sinh tất cả các chỉnh hợp lặp chập của .

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;  int x[21], n, k, dem;  void Nhap() {  cin >> n >> k;  }  void GhiNghiem() {  cout << ++dem << ". ";  for (int i = 1; i <= k; ++i)  cout << x[i] << " ";  cout << "\n";  }  void ChinhHopLap(int i) {  for (int j = 1; j <= n; ++j) {  x[i] = j;  if (i == k) GhiNghiem();  else ChinhHopLap(i + 1);  }  }  main() {  Nhap();  ChinhHopLap(1);  return 0;  } |

Ví dụ về Input / Output của chương trình:

|  |  |
| --- | --- |
| CHINHHOPLAP.INP | CHINHHOPLAP.OUT |
| 2 3 | 1. 1 1 1  2. 1 1 2  3. 1 2 1  4. 1 2 2  5. 2 1 1  6. 2 1 2  7. 2 2 1  8. 2 2 2 |

Theo công thức, số lượng chỉnh hợp lặp chập của là:

.

**Bài toán: sinh dãy nhị phân có độ dài n.**

|  |  |
| --- | --- |
| CHINHHOPLAP.INP | CHINHHOPLAP.OUT |
| 2 3 | 1. 0 0 0  2. 0 0 1  3. 0 1 0  4. 0 1 1  5. 1 0 0  6. 1 0 1  7. 1 1 0  8. 1 1 1 |

## **2.3. Chỉnh hợp không lặp**

Khác với chỉnh hợp lặp là các thành phần được phép lặp lại (tức là có thể giống nhau), chỉnh hợp không lặp chập của tập () phần tử cũng là một dãy thành phần lấy từ tập phần tử có xét thứ tự nhưng các thành phần không được phép giống nhau.

**Ví dụ:** Có người, một cách chọn ra người để xếp thành một hàng là một chỉnh hợp không lặp chập của .

Một trường hợp đặc biệt của chỉnh hợp không lặp là hoán vị. Hoán vị của một tập phần tử là một chỉnh hợp không lặp chập của . Nói một cách trực quan thì hoán vị của tập phần tử là phép thay đổi vị trí của các phần tử (do đó mới gọi là hoán vị).

Nghiệm của bài toán tìm các chỉnh hợp không lặp chập của tập số nguyên từ đến là các vectơ thoả mãn các điều kiện:

* có thành phần: ;
* lấy giá trị trong tập ;
* Ràng buộc: các giá trị đôi một khác nhau, tức là với mọi .

Sau đây là chương trình hoàn chỉnh, chương trình sử dụng mô hình đệ quy để sinh tất cả các chỉnh hợp không lặp chập của phần tử.

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;  int x[21], n, k, dem;  bool d[21];  void Nhap() {  cin >> n >> k;  }  void in() {  for (int i = 1; i <= k; ++i)  cout << x[i] << " ";  cout << "\n";  }  void chinhhopklap(int i) {  for (int j = 1; j <= n; ++j) {  if (!d[j]) {  x[i] = j;  d[j] = true;  if (i == k) in();  else chinhhopklap(i + 1);  d[j] = false;  }  }  }  main() {  Nhap();  memset(d, 0, sizeof(d));  chinhhopklap(1);  } |

Ví dụ về Input / Output của chương trình:

|  |  |
| --- | --- |
| CHINHHOPKL.INP | CHINHHOPKL.OUT |
| 3 3 | 1 2 3  1 3 2  2 1 3  2 3 1  3 1 2  3 2 1 |

Theo công thức, số lượng chỉnh hợp không lặp chập của là:

**II. Bài tập luyện tập**

Bài 1:  Quân hậu.

Cho trước một bàn cờ vua kích thước . Hãy in ra mọi phương án đặt quân hậu sao cho vào bàn cờ sao cho không có 2 quân hậu bất kỳ nào khống chế nhau.

A black and white checkered flag

Description automatically generated with low confidence

Ví dụ một cách đặt quân hậu trên bàn cờ

INPUT: Đọc từ file **QUEEN.INP** dòng thứ nhất ghi số nguyên dương

OUTPUT: Ghi ra file **QUEE.OUT** gồm nhiều dòng:

- Mỗi dòng chứa số thể hiện một phương án đặt quân hậu trên bàn cờ trong đó tức là quân hậu được đặt tại cột dòng .

VD:

|  |  |
| --- | --- |
| QUEEN.INP | QUEEN.OUT |
| 4 | 2 4 1 3  3 1 4 2 |

### **Bài 2. Cái túi.**

Một tên trộm lọt được vào một gia đình. Hắn mang theo một cái túi có thể tích M để đựng đồ ăn trộm. Trong nhà có N vật dụng (N ≤ 20), vật thứ i có trọng lượng là Wi ≤ 100 và giá trị là Vi ≤ 100. Vậy tên trộm phải chọn những vật nào để lấy đi mà tổng giá trị của các vật đó là lớn nhất.

**Input:** file văn bản BAG.INP

* Dòng 1: Chứa hai số N, M cách nhau một dấu cách
* N dòng tiếp theo mỗi dòng i ghi hai số Wi và Vi cách nhau một dấu cách

**Output:** file văn bản BAG.OUT

* Dòng 1: Ghi tổng giá trị lớn nhất mà tên trộm có thể lấy được
* Dòng 2: Ghi chỉ số những vật dụng bị lấy đi.

Ví dụ:

|  |  |
| --- | --- |
| **BAG.INP** | **BAG.OUT** |
| 5 11  3 3  4 4  5 4  9 10  4 4 | 11  1 2 5 |

* **Hướng dẫn giải:**

Xây dựng x[1..N] là phương án chọn đồ vật:

* + Với x[i] = 1 tức là chọn đồ vật thứ i
  + Với x[i] = 0 tức là không chọn đồ vật thứ i.

Như vậy ý tưởng là sinh ra mọi dãy nhị phân độ dài N lưu vào mảng x[1..N], với mỗi dãy nhị phân như vậy ta thử xem các đồ vật tương ứng ở vị trí của dãy nhị phân có thể cho vừa vào túi không, nếu có thì hãy tính tổng giá trị các đồ vật và lựa chọn tổng giá trị tối ưu.

Với test ví dụ của đề bài: ta có thể sinh được 1 dãy nhị phân độ dài 5 là 0 1 1 0 1. Tức là có thể sinh chọn các đồ vật thứ 2, 3 và 5.

Có thể dùng phương pháp quay lui để giải quyết bài toán dễ dàng.

### **Bài 3. Xếp valy**

Một valy có thể chứa tối đa W đơn vị trọng lượng. Có N đồ vật, số lượng mỗi loại không hạn chế. Loại đồ vật thứ i có trọng lượng là Ai và có giá trị là Ci. Hỏi nên chọn những đồ vật nào và số lượng bao nhiêu để xếp vào valy sao cho:

* Tổng trọng lượng của các đồ vật không vượt quá giới hạn W của valy
* Tổng giá trị của đồ vật là lớn nhất.
* Input: Đọc từ file **VALY.INP**
  + Dòng 1 chứa N và M (1 ≤ N ≤ 40, 1 ≤ M ≤ 100)
  + N dòng tiếp theo, mỗi dòng chứa hai số A[i] và C[i].
* Output: Ghi ra file VALY.OUT
  + Dòng đầu là giá trị lớn nhất

Ví dụ:

|  |  |
| --- | --- |
| **VALY.INP** | **VALY.OUT** |
| 4 100  50 50  19 20  80 90  21 25 | 110 |

* **Hướng dẫn giải**:

Với mỗi loại đồ vật i, gọi Ti là giá trị riêng của nó: Ti = Ci / Ai. Sau đó sắp các loại đồ vật theo thứ tự giảm dần của Ti.

Giả sử đã chọn được một số đồ vật trong i đồ vật đầu tiên. Điều kiện để chọn k vật của đồ vật thứ i + 1 là:

S + T[i+1]\*(M – W) > Smax

Trong đó:

- S là tổng giá trị của các vật đồ vật đang có trong valy

- W là tổng trọng lượng của các đồ vật đang có trong valy

- Smax là tổng giá trị valy của một các xếp đã tìm được nhưng chưa tối ưu. Có thể khởi tạo Smax ban đầu bằng 0.

### **Bài 4. Đường đi của con rắn**

Một con rắn sống trên một lưới ô vuông 8 x 8 có chứa một vài vật cản (là các ô mang màu đen trong hình vẽ dưới đây). Con rắn di chuyển theo đường thẳng (khi còn có thể di chuyển được). Khi nó gặp vật cản hay gặp biên của lưới hình vuông, nó sẽ quay sang bên phải hay bên trái của nó. Nếu nó đối diện với ô đã đi qua, nó sẽ dừng hẳn.

A

S

B

C

D

E

F

G

H

1

2

3

4

5

6

7

8

*Câu A: Đ­ường đi khi rắn luôn quay bên trái (13 ô)*

A

S

B

C

D

E

F

G

H

1

2

3

4

5

6

7

8

*Câu B: Đường rắn đi với hành trình xa nhất.*

Con rắn luôn bắt đầu tại ô A1 và luôn bắt đầu việc di chuyển bằng cách đi hướng xuống dưới. Vị trí của các vật cản (tức là các ô đen) được cho biết trước.

**Yêu cầu:**

Tìm số ô mà con rắn đi qua cho đến khi không đi được nữa và thoả mãn:

a) Con rắn luôn đi về phía bên trái của nó

b) Con rắn đi sao cho số ô đi qua là lớn nhất.

**Ví dụ:**

Hình vẽ trên cho hình ảnh một lưoiứ ô vuông với ba ô đen (tại các vị trí A6, E2 và F5). Đường đi thoả mãn câu a) là: đi từ A1 đến A5, quay sang trái đến E5. Từ E5 quay trái đến E3, quay trái đi thẳng và dừng tại ô B3. Con rắn đã đi qua 13 ô.

Đường đi thoả mãn câu b) được cho bởi lưới ô vuông bên phải. Đường đi nói trên đi qua 32 ô.

**Dữ liệu vào:** Ghi trên file SNAIL.IN gồm n+1 dòng:

* Dòng đầu tiên chi số nguyên dương n (1≤n≤32) chi số lượng các ô đen
* Trên n dòng tiếp theo, mỗi dòng ghi toạ độ một ô đen dưới dạng Xn, trong đó X (là các chữ cái Latinh hoa từ A đến Z) chỉ cột và n (1≤n≤8) chỉ dòng.

Chú ý rằng không có ô đen tại các ô A1 và A2

**Dữ liệu ra:** Ghi ra file SNAIL.OUT gồm 2 dòng

* Dòng đầu ghi số lượng ô đi qua thoả mãn câu a)
* Dòng thứ hai ghi số lượng ô đi qua thoả mãn câu b)

*Ví dụ:*

|  |  |
| --- | --- |
| **SNAIL.IN** | **SNAIL.OUT** |
| 3  A6  E2  F5 | 13  32 |

### Bài 5. Tìm đường đi trong mê cung

Mê cung là một ma trận A có M dòng và N cột, chỉ chứa 0 hoặc 1, trong đó 0 nghĩa là có đường đi và 1 nghĩa là không có đường đi. Một người đứng tại vị trí (u,v) có thể đi đến ô kề cạnh với ô đang đứng nếu có đường đi, hãy chỉ ra cho người đó đường đi ngắn nhất để thoát ra khỏi mê cung. Người đó ra khỏi mê cung khi đứng ở một trong 4 cạnh của mê cung.

Dữ liệu vào: Bởi file MAZE.INP gồm:

* Dòng đầu tiên chứa hai số M và N
* M dòng tiếp theo mỗi dòng ghi N số 0 hoặc 1
* Dòng cuối cùng ghi hai số U và V

Dữ liệu ra: Ghi ra file MAZE.OUT gồm:

Nếu không có đường đi ra khỏi mê cung thì ghi -1.

Nếu có đường đi thì:

* Dòng đầu tiên ghi ra số “d” là số bước đi ngắn nhất để thoát ra khỏi mê cung
* d dòng tiếp theo mỗi dòng ghi hai số (i,j) là tọa độ các ô trong hành trình thoát ra khỏi mê cung.

Ví dụ

|  |  |
| --- | --- |
| MAZE.INP | MAZE.OUT |
| 5 6  1 0 1 0 1 0  0 0 1 0 0 1  1 0 0 0 0 1  1 0 0 1 0 1  1 1 1 0 1 0  3 2 | 2  2,2  1,2 |