Liệt kê các tổ hợp k phần tử của các số từ 1 đến n.

(đang phân vân???)

Ý tưởng:

Gọi 1 tổ hợp đó là { x1, x2, ... xk }  
Đầu tiên ta xét các giá trị mà x1 có thể nhận được  
Với mỗi giá trị đó lại tìm các giá trị x2 có thể nhận.  
Cứ như vậy cho đến khi tới xk thì nhận được 1 tổ hợp  
  
Áp dụng vào bài này :  
vì x1 <= x2 <= ...<=xk <= n  
Với 1 số i bất kì x(i) > x( i - 1 ) => x(i) >= x( i -1 ) + 1  
lại có x(k) <= n  
x( k - 1 ) <= n -1  
....  
x( i ) <= n - k + i

Liệt kê các hoán vị của n phần tử.

Biểu diễn hoán vị dưới dạng p1, p2,.., pn, trong đó pi nhận giá trị từ 1 đến n và pi ≠ pj với i ≠j. Các giá trị từ 1 đến n được đề cử cho pi, trong đó giá trị j được chấp nhậnnếu nó chưa được dùng trước đó. Do đó cần phải ghi nhớ xem giá trị j đã được dùng chưa. Ta thực hiện điều này bằng một mảng B, trong đó Bj = true nếu j chưa được dùng và ngược lại. Đầu tiên các giá trị trong B này phải được khởi tạo là true, sau khigán j cho xi thì ghi nhận Bj = false, sau khi gọi xong thủ tục Try(i+1) thì thiết lập lại Bj = true, để đánh dấu nó chưa được dùng để cho bước thử tiếp theo.

Void **Try**(int i)

{

for <mọi giá trị v có thể gán cho x[i]> do

{

<Thử cho x[i] bằng giá trị v>

if <x[i] là phần tử cuối cùng trong cấu hình hoặc i==n>

then

<Thông báo cấu hình tìm được>

Else

{

<Ghi nhận việc cho x[i] nhận giá trị v (nếu cần thiết)>

Try(i+1);// gọi đệ quy cho tiếp chi x[i+1].

<Nếu cần thì bỏ ghi nhận việc thử x[i]:= v để thử giá trị khác>

}

}

}

Thuật toán:

#include<conio.h>

#include<stdio.h>

#include<stdlib.h>

#define true 1

#define false 0

int \*B,\*A,n;

void init(int\*B,int n)

{

for (int i=1;i<=n;i++)

B[i]=true;

}

void process(int A[],int n,int count)

{

printf("\nhoan vi thu %5d:",count);

for(int i=1;i<=n;i++) printf("%d",A[i]);

}

void Try(int i,int n,int &count)

{

for(int j=1;j<=n;j++)

if(B[j])

{

A[i]=j;

B[j]=false;

if(i==n)

{

count++;

process(A,n,count);

}

else

Try(i+1,n,count);

B[j]=true;

}

}

main()

{

printf("nhap so phan tu ");

scanf("%d",&n);

A=(int\*)malloc(n\*sizeof(int));

B=(int\*)malloc(n\*sizeof(int));

init(B,n);

int count =0;

Try(1,n,count);

free(A);

free(B);

getch();

}

**Bài toán xếp quân cờ**

Phân tích bài toán:

\* Ta thấy khi đặt 1 quân hậu vào một ô ở hàng i cột j trên bàn cờ, nó sẽ chiếm dụng kiểm soát trên:  
- Hàng ngang i.  
- Cột dọc j.  
- 1 đường chéo song song với đường chéo chính.  
- 1 đường chéo song song với đường chéo phụ  
\* Nhận xét:   
- Với đường chéo song song với đường chéo chính đi qua ô ở hàng i và cột j luôn có cùng hiệu (i-j).  
- Với đường chéo song song với đường chéo phụ đi qua ô ở hàng i và cột j luôn có cùng tổng (i + j)  
- Với một bàn cờ cỡ NxN ta có:  
 Chỉ số đường chéo chính chạy từ -(n-1)...(n-1)  
 Chỉ số đường chéo phụ chạy từ 2...2n  
\* Do vậy khi đặt quân hậu vào một ô ở hàng i và cột j chúng ta có thể dùng các mảng boolean để đánh dấu các hàng, cột mà nó chiếm dụng như:  
- h[i] = true;// Hàng i đã bị chiếm.  
- c[j] = true;// Cột j đã bị chiếm.  
- cc[i-j] = true;// Chéo chính i - j đã bị chiếm  
- cp[i+j] = true;// Chéo phụ i + j đã bị chiếm.  
\* Do vậy để đặt 1 quân hậu vào một ô ở hàng i, cột j bất kỳ chúng ta chỉ cần kiểm tra xem hàng i, cột j và hai đường chéo đã bị chiếm dụng chưa.  
\* Lần lượt đặt thử con hậu vào các vị trí:  
- Đặt con hậu đầu tiên ở vị trí cột 1, hàng 1   
- Nếu được đặt con hậu tiếp theo vào các ô ở cột tiếp sao cho không bị ăn cứ như vậy theo cho đến đến khi đủ số con hậu cần.   
- Nếu không được quay lại bước trước và dịch chuyển quân hậu trước đó sang ô tiếp theo.

Thuật toán:

**Procedure Try** (i)   
For j=1 to n do   
If not h(j) And not c(i) And not cc(i-j) And not cp(i+j) then   
{   
solution(i)=j;   
If i  
c(i)=true;  
h(j)= true;  
cc(i-j)=true;   
cp(i+j)=true;

try\_col(i+1)

c(i)=false;   
h(j)=false;   
cc(i-j)=false;   
cp(i+j)=false;  
ELSE print\_solution();

}

Chương trình:

**import** java.util.\*;  
**publicclass** NQueen {  
**staticboolean***c* [], *h* [],*cc*[], *cp*[];  
**staticint***n*, *x*[],*soCach* = 0;  
**publicstaticvoid** Try(**int** i){  
**for**(**int** j = 1; j<=*n*; ++j){  
**if**(!*h*[j] && !*c*[i] &&!*cc*[j-i+*n*]&&!*cp*[i+ j]){  
*x*[i] = j;  
**if**(i<*n*){  
*h*[j] = **true**;  
*c*[i] = **true**;  
*cc*[j-i+*n*] = **true**;  
*cp*[i + j] = **true**;  
*Try*(i+1);  
*h*[j] = **false**;  
*c*[i] = **false**;  
*cc*[j-i+*n*] = **false**;  
*cp*[i + j] = **false**;  
}

**else**{  
++soCach;  
System.*out*.print("Cach " + *soCach* + ": ");  
**for**(**int** l = 1; l<*x*.length; ++l)  
System.*out*.print(*x*[l] + " ");  
System.*out*.println();  
}  
}  
}  
}  
**publicstaticvoid** main(String[] args) {  
Scanner keyIn = **new** Scanner(System.*in*);  
System.*out*.print("Nhap n = ");  
*n* = keyIn.nextInt();  
*c*= **newboolean**[*n*+1];  
*h* = **newboolean** [*n*+1];  
*cc* = **newboolean**[2\**n* + 1];//-n..n; 0 ..2n  
*cp* =**newboolean** [2\**n*+1]; //2 .. 2n  
*x* = **newint** [*n*+1];  
*Try*(1);  
}  
}

Phân tích bài toán:

Bài toán cách đi của con mã:

* Giải bằng phương pháp đệ quy: dùng thuật toán quay lui; xuất phát từ 1 ô, gọi số nước đi là t=1,ta cho quân mã thử đi tiếp 1 ô (có 8 nước đi có thể), nếu ô đi tiếp này chưa đi qua thì chọn làm bước đi tiếp theo. Tại mỗi nước đi kiểm tra xem tổng số nước đi bằng n\*n chưa, nếu bằng thì mã đã đi qua tất cả các ô ⇒ dừng (do chỉ cần tìm một giải pháp).Trường hợp ngược lại, gọi đệ quy để chọn nước đi tiếp theo. Ngoài ra, nếu tại một bước tìm đường đi, nếu không tìm được đường đi tiếp thì thuật toán sẽ quay lui lại nước đi trước và tìm đường đi khác…
* Mảng board[MAX][MAX]: lưu bàn cờ, trong đó board[i][j] là ô (i, j); giá trị của board[i][j] là 0 khi quân mã chưa đi qua, và >0 khi quân mã đã đi qua, giá trị board[i][j] lúc này chính là thứ tự nước đi trên hành trình. Vấn đề bổ sung liên quan đến đường biên, do mỗi lần di chuyển quân mã ta phải kiểm tra xem nước đi có ra ngoài biên haykhông. Ta có thể mở rộng mảng board để không cần phải kiểm tra bằng cách mở rộng hai ô về bốn hướng trên dưới trái phải. Khi đó chỉ cần gán giá trị cho các ô ngoài biên là -1.
* Mảng dr[8], dc[8]: lưu các độ dời của bước đi kế tiếp, có tám nước đi có thể cho vị trí quân mã hiện tại. Do đó để đi nước thứ i ta chỉ cần cộng thêm dr[i] cho dòng và dc[i] cho cột !
* Thuật giải đệ quy: Tại mỗi bước lần lượt cho quân mã thử tất cả các nước đi kế tiếp (tám nước đi kế tiếp). Với mỗi bước đi, kiểm tra xem nếu nước đi hợp lệ (chưa đi qua và ở trong bàn cờ) thì thử đi nước này. Nếu quân mã đã đi qua hết bàn cờ thì xuất kết quả. Ngược lại thì gọi đệ quy tiếp cho vị trí mới thử trên. Lưu ý là mỗi khí vị trí đã đi qua được đánh dấu chính bằng thứ tự nước đi trên bàn cờ. Sau khi khong tử vị trí này thì phải bỏ đánh dấu để chọn giải pháp khác ( trường hợp quay lui).

Thuật toán:

Try( int step, int i, j)

{

+ Với mỗi nước đi kế tiếp (ii, jj) từ (i, j)

+ Nếu (ii,jj) hợp lệ

**chọn (ii, jj) làm nước đi kế tiếp**

+ nếu đi hết bàn cờ

=>xuất 1 kết quả+ ngược lại Gọi đệ quy Try(**step +1,**ii, jj)

**Không chọn (ii, jj) là nước đi kế tiếp**

}

Chương trình:

#include <stdio.h>

#include <stdlib.h>

void Show(int board[12][12]);

void Init(int board[12][12])

{

for(int i=0;i<12;i++)

for(int j=0;j<12;j++)

if(i>=2 && i<=9 && j>=2 && j<=9)

board[i][j]=0;

else

board[i][j]=-1;

}

void Try(int step,int i,int j,int board[12][12],int \*I,int \*J)

{

for(int l=0;l<7;l++)//duyet cac truong hop theo kim dong ho

{

if(board[i+I[l]][j+J[l]]==0)

//o chua qua , dat duoc

{

if(step+1==63)//hoan tat

{

for(i=2;i<10;i++)

for(j=2;j<10;j++)

if(board[i][j]==0)

board[i][j]=63;

Show(board);

exit(0);

}

else

{

i=i+I[l];

j=j+J[l];

board[i][j]=step+1;

Try(step+1,i,j,board,I,J);

board[i][j]=0;

}

}

}

}

void Show(int board[12][12])

{

for(int i=0;i<12;i++)

{

for(int j=0;j<12;j++)

printf("%4d",board[i][j]);

printf("\n\n");

}

}

main()

{

int board[12][12];

int I[8]={-2,-1,1,2,2,1,-1,-2};

int J[8]={1,2,2,1,-1,-2,-2,-1};

Init(board);

board[6][3]=1;

Show(board);

Try(1,6,3,board,I,J);

}