§3. THUẬT TOÁN QUAY LUI

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]⟩.

Trên phương diện quy nạp, có thể nói rằng thuật toán quay lui liệt kê các cấu hình n phần tử dạng x[1..n] bằng cách thử cho x[1] nhận lần lượt các giá trị có thể. Với mỗi giá trị thử gán cho x[1] bài toán trở thành liệt kê tiếp cấu hình n - 1 phần tử x[2..n].

Mô hình của thuật toán quay lui có thể mô tả như sau:

```
{Thủ tục này thử cho x[i] nhận lần lượt các giá trị mà nó có thể nhận}
procedure Attempt(i);
begin
  for ⟨mọi giá trị V có thể gán cho x[i]⟩ do
    begin
      ⟨Thử cho x[i] := V⟩;
      if ⟨x[i] là phần tử cuối cùng trong cấu hình⟩ then
        ⟨Thông báo cấu hình tìm được⟩
      else
        begin
          ⟨Ghi nhận việc cho x[i] nhận giá trị V (nếu cần)⟩;
          Attempt(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⟩;
        end;
    end;
end;
```

Thuật toán quay lui sẽ bắt đầu bằng lời gọi Attempt(1)

3.1. LIỆT KÊ CÁC DÃY NHỊ PHÂN ĐỘ DÀI N

Input/Output với khuôn dạng như trong P_1_02_1.PAS

Biểu diễn dãy nhị phân độ dài N dưới dạng x[1..n]. Ta sẽ liệt kê các dãy này bằng cách thử dùng các giá trị {0, 1} gán cho x[i]. Với mỗi giá trị thử gán cho x[i] lại thử các giá trị có thể gán cho x[i+1].Chương trình liệt kê bằng thuật toán quay lui có thể viết:

```
                    P_1_03_1.PAS * Thuật toán quay lui liệt kê các dãy nhị phân độ dài n
{$MODE DELPHI} (*This program uses 32-bit Integer [-2^31..2^31 - 1]*)
program Binary_String_Enumeration;
const
```

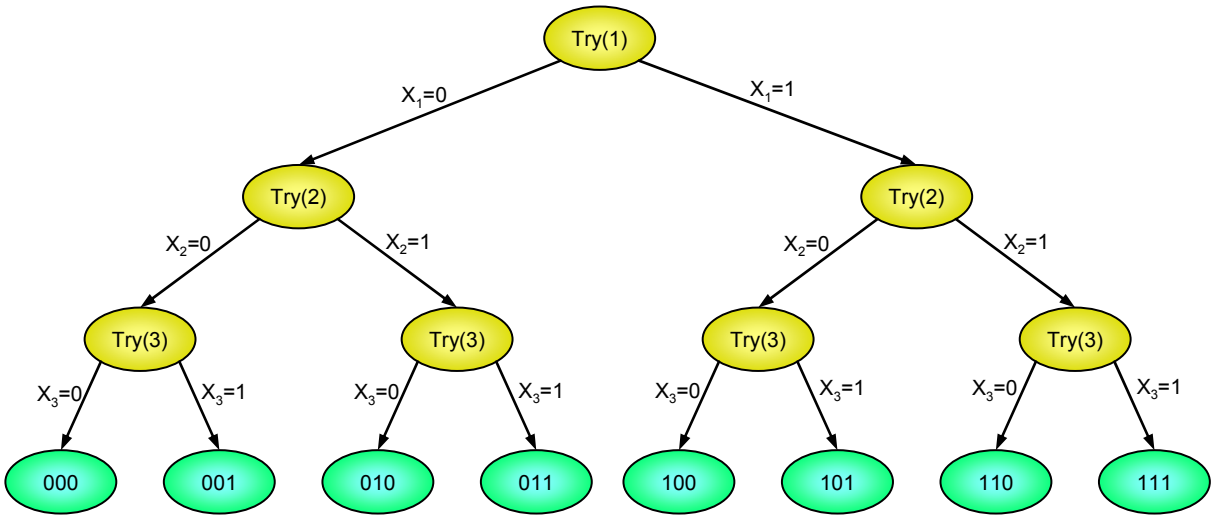
```
  InputFile  = 'BSTR.INP';
  OutputFile = 'BSTR.OUT';
  max = 100;
var
  x: array[1..max] of Integer;
  n: Integer;
  f: Text;

procedure PrintResult; {In cấu hình tìm được, do thủ tục tìm đệ quy Attempt gọi khi tìm ra một cấu hình}
var
  i: Integer;
begin
  for i := 1 to n do Write(f, x[i]);
  WriteLn(f);
end;

procedure Attempt(i: Integer); {Thử các cách chọn x[i]}
var
  j: Integer;
begin
  for j := 0 to 1 do {Xét các giá trị có thể gán cho x[i], với mỗi giá trị đó}
    begin
      x[i] := j; {Thử đặt x[i]}
      if i = n then PrintResult {Nếu i = n thì in kết quả}
      else Attempt(i + 1); {Nếu i chưa phải là phần tử cuối thì tìm tiếp x[i+1]}
    end;
end;

begin
  Assign(f, InputFile); Reset(f);
  ReadLn(f, n); {Nhập dữ liệu}
  Close(f);
  Assign(f, OutputFile); Rewrite(f);
  Attempt(1); {Thử các cách chọn giá trị x[1]}
  Close(f);
end.
```

Ví dụ: Khi n = 3, cây tìm kiếm quay lui như sau:



Hình 1: Cây tìm kiếm quay lui trong bài toán liệt kê dãy nhị phân

3.2. LIỆT KÊ CÁC TẬP CON K PHẦN TỬ

Input/Output có khuôn dạng như trong P_1_02_2.PAS

Để liệt kê các tập con k phần tử của tập S = {1, 2, …, n} ta có thể đưa về liệt kê các cấu hình x[1..n], ở đây các x[i] ∈ S và x[1] < x[2] < … < x[k]. Ta có nhận xét:

x[k] ≤ n

x[k-1] ≤ x[k] - 1 ≤ n - 1

…

x[i] ≤ n - k + i

…

x[1] ≤ n - k + 1.

Từ đó suy ra x[i-1] + 1 ≤ x[i] ≤ n - k + i (1 ≤ i ≤ k) ở đây ta giả thiết có thêm một số x[0] = 0 khi xét i = 1.

Như vậy ta sẽ xét tất cả các cách chọn x[1] từ 1 (=x[0] + 1) đến n - k + 1, với mỗi giá trị đó, xét tiếp tất cả các cách chọn x[2] từ x[1] +1 đến n - k + 2, … cứ như vậy khi chọn được đến x[k] thì ta có một cấu hình cần liệt kê. Chương trình liệt kê bằng thuật toán quay lui như sau:

P_1_03_2.PAS * Thuật toán quay lui liệt kê các tập con k phần tử

```
{$MODE DELPHI} (*This program uses 32-bit Integer [-2^31..2^31 - 1]*)
program Combination_Enumeration;
const
  InputFile  = 'SUBSET.INP';
  OutputFile = 'SUBSET.OUT';
  max = 100;
var
  x: array[0..max] of Integer;
  n, k: Integer;
  f: Text;

procedure PrintResult;   (*In ra tập con {x[1], x[2], …, x[k]}*)
var
  i: Integer;
begin
  Write(f, '{');
  for i := 1 to k - 1 do Write(f, x[i], ', ');
  WriteLn(f, x[k], '}');
end;

procedure Attempt(i: Integer); {Thử các cách chọn giá trị cho x[i]}
var
  j: Integer;
begin
  for j := x[i - 1] + 1 to n - k + i do
    begin
      x[i] := j;
      if i = k then PrintResult
      else Attempt(i + 1);
    end;
end;

begin
  Assign(f, InputFile); Reset(F);
  ReadLn(f, n, k);
  Close(f);
  Assign(f, OutputFile); Rewrite(f);
```

```
  x[0] := 0;
  Attempt(1);
  Close(f);
end.
```

Nếu để ý chương trình trên và chương trình liệt kê dãy nhị phân độ dài n, ta thấy về cơ bản chúng chỉ khác nhau ở thủ tục Attemp(i) - chọn thử các giá trị cho x[i], ở chương trình liệt kê dãy nhị phân ta thử chọn các giá trị 0 hoặc 1 còn ở chương trình liệt kê các tập con k phần tử ta thử chọn x[i] là một trong các giá trị nguyên từ x[i-1] + 1 đến n - k + i. Qua đó ta có thể thấy tính phổ dụng của thuật toán quay lui: mô hình cài đặt có thể thích hợp cho nhiều bài toán, khác với phương pháp sinh tuần tự, với mỗi bài toán lại phải có một thuật toán sinh kế tiếp riêng làm cho việc cài đặt mỗi bài một khác, bên cạnh đó, không phải thuật toán sinh kế tiếp nào cũng dễ cài đặt.

3.3. LIỆT KÊ CÁC CHỈNH HỢP KHÔNG LẶP CHẬP K

Để liệt kê các chỉnh hợp không lặp chập k của tập S = {1, 2, …, n} ta có thể đưa về liệt kê các cấu hình x[1..k] ở đây các x[i] ∈ S và khác nhau đôi một.

Như vậy thủ tục Attempt(i) - xét tất cả các khả năng chọn x[i] - sẽ thử hết các giá trị từ 1 đến n, mà các giá trị này chưa bị các phần tử đứng trước chọn. Muốn xem các giá trị nào chưa được chọn ta sử dụng kỹ thuật dùng mảng đánh dấu:

- Khởi tạo một mảng c[1..n] mang kiểu logic boolean. Ở đây c[i] cho biết giá trị i có còn tự do hay đã bị chọn rồi. Ban đầu khởi tạo tất cả các phần tử mảng c là TRUE có nghĩa là các phần tử từ 1 đến n đều tự do.
- Tại bước chọn các giá trị có thể của x[i] ta chỉ xét những giá trị j có c[j] = TRUE có nghĩa là chỉ chọn những giá trị tự do.
- Trước khi gọi đệ quy tìm x[i+1]: ta đặt giá trị j vừa gán cho x[i] là **đã bị chọn** có nghĩa là đặt c[j] := FALSE để các thủ tục Attempt(i + 1), Attempt(i + 2)… gọi sau này không chọn phải giá trị j đó nữa
- Sau khi gọi đệ quy tìm x[i+1]: có nghĩa là sắp tới ta sẽ thử gán một **giá trị khác** cho x[i] thì ta sẽ đặt giá trị j vừa thử đó thành **tự do** (c[j] := TRUE), bởi khi xi đã nhận một giá trị khác rồi thì các phần tử đứng sau: x[i+1], x[i+2] … hoàn toàn có thể nhận lại giá trị j đó. Điều này hoàn toàn hợp lý trong phép xây dựng chỉnh hợp không lặp: x[1] có n cách chọn, x[2] có n - 1 cách chọn, …Lưu ý rằng khi thủ tục Attempt(i) có i = k thì ta không cần phải đánh dấu gì cả vì tiếp theo chỉ có in kết quả chứ không cần phải chọn thêm phần tử nào nữa.

Input: file văn bản ARRANGE.INP chứa hai số nguyên dương n, k (1 ≤ k ≤ n ≤ 100) cách nhau ít nhất một dấu cách

Output: file văn bản ARRANGE.OUT ghi các chỉnh hợp không lặp chập k của tập {1, …, n}

ARRANGE.INP	ARRANGE.OUT
3 2	1 2 1 3 2 1 2 3 3 1 3 2

P_1_03_3.PAS * Thuật toán quay lui liệt kê các chỉnh hợp không lặp chập k

```
{$MODE DELPHI} (*This program uses 32-bit Integer [-2^31..2^31 - 1]*)
program Arrangement_Enumeration;
const
  InputFile  = 'ARRANGES.INP';
  OutputFile = 'ARRANGES.OUT';
  max = 100;
var
  x: array[1..max] of Integer;
  c: array[1..max] of Boolean;
  n, k: Integer;
  f: Text;

procedure PrintResult; {Thủ tục in cấu hình tìm được}
var
  i: Integer;
begin
  for i := 1 to k do Write(f, x[i], ' ');
  WriteLn(f);
end;

procedure Attempt(i: Integer); {Thử các cách chọn x[i]}
var
  j: Integer;
begin
  for j := 1 to n do
    if c[j] then {Chỉ xét những giá trị j còn tự do}
      begin
        x[i] := j;
        if i = k then PrintResult {Nếu đã chọn được đến xk thì chỉ việc in kết quả}
        else
          begin
            c[j] := False; {Đánh dấu: j đã bị chọn}
            Attempt(i + 1); {Thủ tục này chỉ xét những giá trị còn tự do gán cho x[i+1]}
            c[j] := True; {Bỏ đánh dấu: j lại là tự do, bởi sắp tới sẽ thử một cách chọn khác của x[i]}
          end;
      end;
end;

begin
  Assign(f, InputFile); Reset(f);
  ReadLn(f, n, k);
  Assign(f, OutputFile); Rewrite(f);
  FillChar(c, SizeOf(c), True); {Tất cả các số đều chưa bị chọn}
  Attempt(1); {Thử các cách chọn giá trị của x[1]}
  Close(f);
end.
```

Nhận xét: khi k = n thì đây là chương trình liệt kê hoán vị

3.4. BÀI TOÁN PHÂN TÍCH SỐ

3.4.1. Bài toán

Cho một số nguyên dương n ≤ 30, hãy tìm tất cả các cách phân tích số n thành tổng của các số nguyên dương, các cách phân tích là hoán vị của nhau chỉ tính là 1 cách.

3.4.2. Cách làm:

Ta sẽ lưu nghiệm trong mảng x, ngoài ra có một mảng t. Mảng t xây dựng như sau: t[i] sẽ là tổng các phần tử trong mảng x từ x[1] đến x[i]: t[i] := x[1] + x[2] + … + x[i].

Khi liệt kê các dãy x có tổng các phần tử đúng bằng n, để tránh sự trùng lặp ta đưa thêm ràng buộc x[i-1] ≤ x[i].

Vì số phần tử thực sự của mảng x là không cố định nên thủ tục PrintResult dùng để in ra 1 cách phân tích phải có thêm tham số cho biết sẽ in ra bao nhiêu phần tử.

Thủ tục đệ quy Attempt(i) sẽ thử các giá trị có thể nhận của x[i] (x[i] ≥ x[i - 1])

Khi nào thì in kết quả và khi nào thì gọi đệ quy tìm tiếp ?

Lưu ý rằng t[i - 1] là tổng của tất cả các phần tử từ x[1] đến x[i-1] do đó

- Khi t[i] = n tức là (x[i] = n - t[i - 1]) thì in kết quả
- Khi tìm tiếp, x[i+1] sẽ phải lớn hơn hoặc bằng x[i]. Mặt khác t[i+1] là tổng của các số từ x[1] tới x[i+1] không được vượt quá n. Vậy ta có t[i+1] ≤ n ⇔ t[i-1] + x[i] + x[i+1] ≤ n ⇔ x[i] + x[i+1] ≤ n - t[i-1] tức là x[i] ≤ (n - t[i-1])/2. Ví dụ đơn giản khi n = 10 thì chọn x[1] = 6, 7, 8, 9 là việc làm vô nghĩa vì như vậy cũng không ra nghiệm mà cũng không chọn tiếp x[2] được nữa.

Một cách dễ hiểu: ta gọi đệ quy tìm tiếp khi giá trị x[i] được chọn còn cho phép chọn thêm một phần tử khác lớn hơn hoặc bằng nó mà không làm tổng vượt quá n. Còn ta in kết quả chỉ khi x[i] mang giá trị đúng bằng số thiếu hụt của tổng i-1 phần tử đầu so với n.

Vậy thủ tục Attempt(i) thử các giá trị cho x[i] có thể viết như sau: (để tổng quát cho i = 1, ta đặt x[0] = 1 và t[0] = 0).

- Xét các giá trị của x[i] từ x[i - 1] đến (n - t[i-1]) div 2, cập nhật t[i] := t[i - 1] + x[i] và gọi đệ quy tìm tiếp.
- Cuối cùng xét giá trị x[i] = n - t[i-1] và in kết quả từ x[1] đến x[i].

Input: file văn bản ANALYSE.INP chứa số nguyên dương n ≤ 100

Output: file văn bản ANALYSE.OUT ghi các cách phân tích số n.

ANALYSE.INP	ANALYSE.OUT
6	6 = 1+1+1+1+1+1 6 = 1+1+1+1+2 6 = 1+1+1+3 6 = 1+1+2+2 6 = 1+1+4 6 = 1+2+3 6 = 1+5 6 = 2+2+2 6 = 2+4 6 = 3+3 6 = 6

P_1_03_4.PAS * Thuật toán quay lui liệt kê các cách phân tích số

```
{$MODE DELPHI} (*This program uses 32-bit Integer [-2^31..2^31 - 1]*)
program Analysis_Enumeration;
const
  InputFile  = 'ANALYSE.INP';
  OutputFile = 'ANALYSE.OUT';
  max = 100;
var
  n: Integer;
  x: array[0..max] of Integer;
  t: array[0..max] of Integer;
  f: Text;

procedure Init; {Khởi tạo}
begin
  Assign(f, InputFile); Reset(f);
  ReadLn(f, n);
  Close(f);
  x[0] := 1;
  t[0] := 0;
end;

procedure PrintResult(k: Integer);
var
  i: Integer;
begin
  Write(f, n, ' = ');
  for i := 1 to k - 1 do Write(f, x[i], '+');
  WriteLn(f, x[k]);
end;

procedure Attempt(i: Integer);
var
  j: Integer;
begin
  for j := x[i - 1] to (n - T[i - 1]) div 2 do {Trường hợp còn chọn tiếp x[i+1]}
    begin
      x[i] := j;
      t[i] := t[i - 1] + j;
      Attempt(i + 1);
    end;
  x[i] := n - T[i - 1]; {Nếu x[i] là phần tử cuối thì nó bắt buộc phải là n-T[i-1], in kết quả}
  PrintResult(i);
end;

begin
  Init;
  Assign(f, OutputFile); Rewrite(f);
  Attempt(1);
  Close(f);
```

```
end.
```

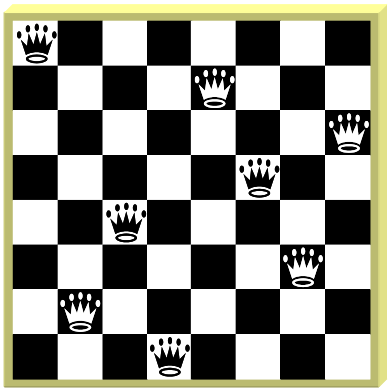
Bây giờ ta xét tiếp một ví dụ kinh điển của thuật toán quay lui:

3.5. BÀI TOÁN XẾP HẬU

3.5.1. Bài toán

Xét bàn cờ tổng quát kích thước nxn. Một quân hậu trên bàn cờ có thể ăn được các quân khác nằm tại các ô cùng hàng, cùng cột hoặc cùng đường chéo. Hãy tìm các xếp n quân hậu trên bàn cờ sao cho không quân nào ăn quân nào.

Ví dụ một cách xếp với n = 8:



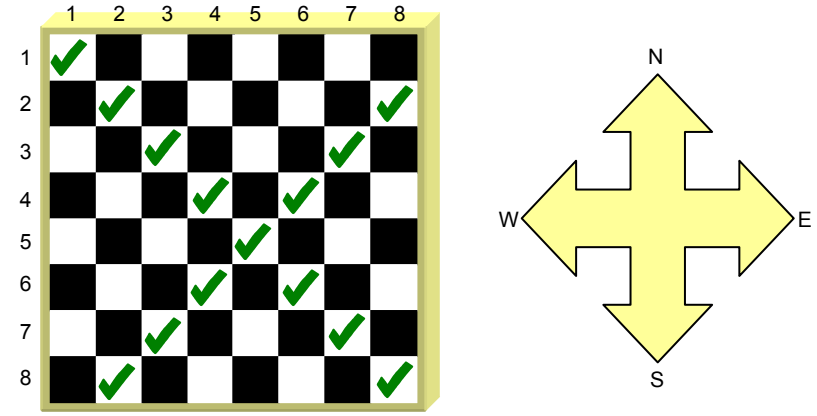
Hình 2: Xếp 8 quân hậu trên bàn cờ 8x8

3.5.2. Phân tích

Rõ ràng n quân hậu sẽ được đặt mỗi con một hàng vì hậu ăn được ngang, ta gọi quân hậu sẽ đặt ở hàng 1 là quân hậu 1, quân hậu ở hàng 2 là quân hậu 2… quân hậu ở hàng n là quân hậu n. Vậy một nghiệm của bài toán sẽ được biết khi ta tìm ra được **vị trí cột của những quân hậu**.

Nếu ta định hướng Đông (Phải), Tây (Trái), Nam (Dưới), Bắc (Trên) thì ta nhận thấy rằng:

- Một đường chéo theo hướng Đông Bắc - Tây Nam (ĐB-TN) bất kỳ sẽ đi qua một số ô, các ô đó có tính chất: Hàng + Cột = C (Const). Với mỗi đường chéo ĐB-TN ta có 1 hằng số C và với một hằng số C: $2 \le C \le 2n$ xác định duy nhất 1 đường chéo ĐB-TN vì vậy ta có thể đánh chỉ số cho các đường chéo ĐB- TN từ 2 đến 2n
- Một đường chéo theo hướng Đông Nam - Tây Bắc (ĐN-TB) bất kỳ sẽ đi qua một số ô, các ô đó có tính chất: Hàng - Cột = C (Const). Với mỗi đường chéo ĐN-TB ta có 1 hằng số C và với một hằng số C: $1 - n \le C \le n - 1$ xác định duy nhất 1 đường chéo ĐN-TB vì vậy ta có thể đánh chỉ số cho các đường chéo ĐN- TB từ 1 - n đến n - 1.



Hình 3: Đường chéo ĐB-TN mang chỉ số 10 và đường chéo ĐN-TB mang chỉ số 0

Cài đặt:

Ta có 3 mảng logic để đánh dấu:

- Mảng a[1..n]. a[i] = TRUE nếu như cột i còn tự do, a[i] = FALSE nếu như cột i đã bị một quân hậu khống chế
- Mảng b[2..2n]. b[i] = TRUE nếu như đường chéo ĐB-TN thứ i còn tự do, b[i] = FALSE nếu như đường chéo đó đã bị một quân hậu khống chế.
- Mảng c[1-n..n-1]. c[i] = TRUE nếu như đường chéo ĐN-TB thứ i còn tự do, c[i] = FALSE nếu như đường chéo đó đã bị một quân hậu khống chế.

Ban đầu cả 3 mảng đánh dấu đều mang giá trị TRUE. (Các cột và đường chéo đều tự do)

Thuật toán quay lui:

- Xét tất cả các cột, thử đặt quân hậu 1 vào một cột, với mỗi cách đặt như vậy, xét tất cả các cách đặt quân hậu 2 không bị quân hậu 1 ăn, lại thử 1 cách đặt và xét tiếp các cách đặt quân hậu 3…Mỗi cách đặt được đến quân hậu n cho ta 1 nghiệm
- Khi chọn vị trí cột j cho quân hậu thứ i, thì ta phải chọn ô(i, j) không bị các quân hậu đặt trước đó ăn, tức là phải chọn cột j còn tự do, đường chéo ĐB-TN (i+j) còn tự do, đường chéo ĐN-TB(i-j) còn tự do. Điều này có thể kiểm tra (a[j] = b[i+j] = c[i-j] = TRUE)
- Khi thử đặt được quân hậu thứ i vào cột j, nếu đó là quân hậu cuối cùng (i = n) thì ta có một nghiệm. Nếu không:

  Trước khi gọi đệ quy tìm cách đặt quân hậu thứ i + 1, ta đánh dấu cột và 2 đường chéo bị quân hậu vừa đặt khống chế (a[j] = b[i+j] = c[i-j] := FALSE) để các lần gọi đệ quy tiếp sau chọn cách đặt các quân hậu kế tiếp sẽ không chọn vào những ô nằm trên cột j và những đường chéo này nữa.

  Sau khi gọi đệ quy tìm cách đặt quân hậu thứ i + 1, có nghĩa là sắp tới ta lại thử một cách đặt khác cho quân hậu thứ i, ta bỏ đánh dấu cột và 2 đường chéo bị quân hậu vừa thử đặt khống chế (a[j] = b[i+j] = c[i-j] := TRUE) tức là cột và 2 đường chéo đó lại thành tự do, bởi khi đã đặt quân hậu i sang vị trí khác rồi thì cột và 2 đường chéo đó hoàn toàn có thể gán cho một quân hậu khác

Hãy xem lại trong các chương trình liệt kê chỉnh hợp không lặp và hoán vị về kỹ thuật đánh dấu. Ở đây chỉ khác với liệt kê hoán vị là: liệt kê hoán vị chỉ cần một mảng đánh dấu xem giá trị có tự do không, còn bài toán xếp hậu thì cần phải đánh dấu cả 3 thành phần: Cột, đường chéo ĐB-TN, đường chéo ĐN- TB. Trường hợp đơn giản hơn: Yêu cầu liệt kê các cách đặt n quân xe lên bàn cờ nxn sao cho không quân nào ăn quân nào chính là bài toán liệt kê hoán vị

❖ **Input:** file văn bản QUEENS.INP chứa số nguyên dương n ≤ 100

❖ **Output:** file văn bản QUEENS.OUT, mỗi dòng ghi một cách đặt n quân hậu

QUEENS.INP	QUEENS.OUT
5	(1, 1); (2, 3); (3, 5); (4, 2); (5, 4); (1, 1); (2, 4); (3, 2); (4, 5); (5, 3); (1, 2); (2, 4); (3, 1); (4, 3); (5, 5); (1, 2); (2, 5); (3, 3); (4, 1); (5, 4); (1, 3); (2, 1); (3, 4); (4, 2); (5, 5); (1, 3); (2, 5); (3, 2); (4, 4); (5, 1); (1, 4); (2, 1); (3, 3); (4, 5); (5, 2); (1, 4); (2, 2); (3, 5); (4, 3); (5, 1); (1, 5); (2, 2); (3, 4); (4, 1); (5, 3); (1, 5); (2, 3); (3, 1); (4, 4); (5, 2);

P_1_03_5.PAS * Thuật toán quay lui giải bài toán xếp hậu

```
{$MODE DELPHI} (*This program uses 32-bit Integer [-2^31..2^31 - 1]*)
program n_Queens;
const
  InputFile  = 'QUEENS.INP';
  OutputFile = 'QUEENS.OUT';
  max = 100;
var
  n: Integer;
  x: array[1..max] of Integer;
  a: array[1..max] of Boolean;
  b: array[2..2 * max] of Boolean;
  c: array[1 - max..max - 1] of Boolean;
  f: Text;

procedure Init;
begin
  Assign(f, InputFile); Reset(f);
  ReadLn(f, n);
  Close(f);
  FillChar(a, SizeOf(a), True); {Mọi cột đều tự do}
  FillChar(b, SizeOf(b), True); {Mọi đường chéo Đông Bắc - Tây Nam đều tự do}
  FillChar(c, SizeOf(c), True); {Mọi đường chéo Đông Nam - Tây Bắc đều tự do}
end;

procedure PrintResult;
var
  i: Integer;
begin
  for i := 1 to n do Write(f, '(', i, ', ', x[i], '); ');
  WriteLn(f);
end;

procedure Attempt(i: Integer); {Thử các cách đặt quân hậu thứ i vào hàng i}
var
  j: Integer;
begin
  for j := 1 to n do
    if a[j] and b[i + j] and c[i - j] then {Chỉ xét những cột j mà ô (i, j) chưa bị khống chế}
```

```
      begin
        x[i] := j; {Thử đặt quân hậu i vào cột j}
        if i = n then PrintResult
        else
          begin
            a[j] := False; b[i + j] := False; c[i - j] := False; {Đánh dấu}
            Attempt(i + 1); {Tìm các cách đặt quân hậu thứ i + 1}
            a[j] := True; b[i + j] := True; c[i - j] := True; {Bỏ đánh dấu}
          end;
      end;
end;

begin
  Init;
  Assign(f, OutputFile); Rewrite(f);
  Attempt(1);
  Close(f);
end.
```

Tên gọi thuật toán quay lui, đứng trên phương diện cài đặt có thể nên gọi là kỹ thuật vét cạn bằng quay lui thì chính xác hơn, tuy nhiên đứng trên phương diện bài toán, nếu như ta coi công việc giải bài toán bằng cách xét tất cả các khả năng cũng là 1 cách giải thì tên gọi Thuật toán quay lui cũng không có gì trái logic. Xét hoạt động của chương trình trên cây tìm kiếm quay lui ta thấy tại bước thử chọn x[i] nó sẽ gọi đệ quy để tìm tiếp x[i+1] có nghĩa là quá trình sẽ duyệt tiến sâu xuống phía dưới đến tận nút lá, sau khi đã duyệt hết các nhánh, tiến trình lùi lại thử áp đặt một giá trị khác cho x[i], đó chính là nguồn gốc của tên gọi "thuật toán quay lui"

Bài tập:

Bài 1

Một số chương trình trên xử lý không tốt trong trường hợp tầm thường (n = 0 hoặc k = 0), hãy khắc phục các lỗi đó

Bài 2

Viết chương trình liệt kê các chỉnh hợp lặp chập k của n phần tử

Bài 3

Cho hai số nguyên dương l, n. Hãy liệt kê các xâu nhị phân độ dài n có tính chất, bất kỳ hai xâu con nào độ dài l liền nhau đều khác nhau.

Bài 4

Với n = 5, k = 3, vẽ cây tìm kiếm quay lui của chương trình liệt kê tổ hợp chập k của tập {1, 2, …, n}

Bài 5

Liệt kê tất cả các tập con của tập S gồm n số nguyên {S[1], S[2], …, S[n]} nhập vào từ bàn phím

Bài 6

Tương tự như bài 5 nhưng chỉ liệt kê các tập con có max - min ≤ T (T cho trước).

Bài 7

Một dãy x[1..n] gọi là một hoán vị hoàn toàn của tập {1, 2, …, n} nếu nó là một hoán vị và thoả mãn x[i] ≠ i với ∀i: 1 ≤ i ≤ n. Hãy viết chương trình liệt kê tất cả các hoán vị hoàn toàn của tập trên (n vào từ bàn phím).

Bài 8

Sửa lại thủ tục in kết quả (PrintResult) trong bài xếp hậu để có thể vẽ hình bàn cờ và các cách đặt hậu ra màn hình.

Bài 9

Mã đi tuần: Cho bàn cờ tổng quát kích thước nxn và một quân Mã, hãy chỉ ra một hành trình của quân Mã xuất phát từ ô đang đứng đi qua tất cả các ô còn lại của bàn cờ, mỗi ô đúng 1 lần.

Bài 10

Chuyển tất cả các bài tập trong bài trước đang viết bằng sinh tuần tự sang quay lui.

Bài 11

Xét sơ đồ giao thông gồm n nút giao thông đánh số từ 1 tới n và m đoạn đường nối chúng, mỗi đoạn đường nối 2 nút giao thông. Hãy nhập dữ liệu về mạng lưới giao thông đó, nhập số hiệu hai nút giao thông s và d. Hãy in ra tất cả các cách đi từ s tới d mà mỗi cách đi không được qua nút giao thông nào quá một lần.

§4. KỸ THUẬT NHÁNH CẬN

4.1. BÀI TOÁN TỐI ƯU

Một trong những bài toán đặt ra trong thực tế là việc tìm ra **một** nghiệm thoả mãn một số điều kiện nào đó, và nghiệm đó là **tốt nhất** theo một chỉ tiêu cụ thể, nghiên cứu lời giải các lớp bài toán tối ưu thuộc về lĩnh vực quy hoạch toán học. Tuy nhiên cũng cần phải nói rằng trong nhiều trường hợp chúng ta chưa thể xây dựng một thuật toán nào thực sự hữu hiệu để giải bài toán, mà cho tới nay việc tìm nghiệm của chúng vẫn phải dựa trên mô hình **liệt kê** toàn bộ các cấu hình có thể và đánh giá, tìm ra cấu hình tốt nhất. Việc liệt kê cấu hình có thể cài đặt bằng các phương pháp liệt kê: Sinh tuần tự và tìm kiếm quay lui. Dưới đây ta sẽ tìm hiểu phương pháp liệt kê bằng thuật toán quay lui để tìm nghiệm của bài toán tối ưu.

4.2. SỰ BÙNG NỔ TỔ HỢP

Mô hình thuật toán quay lui là tìm kiếm trên 1 cây phân cấp. Nếu giả thiết rằng ứng với mỗi nút tương ứng với một giá trị được chọn cho x[i] sẽ ứng với chỉ 2 nút tương ứng với 2 giá trị mà x[i+1] có thể nhận thì cây n cấp sẽ có tới 2^n nút lá, con số này lớn hơn rất nhiều lần so với dữ liệu đầu vào n. Chính vì vậy mà nếu như ta có thao tác thừa trong việc chọn x[i] thì sẽ phải trả giá rất lớn về chi phí thực thi thuật toán bởi quá trình tìm kiếm lòng vòng vô nghĩa trong các bước chọn kế tiếp x[i+1], x[i+2], … Khi đó, một vấn đề đặt ra là trong quá trình liệt kê lời giải ta cần tận dụng những thông tin đã tìm được để loại bỏ sớm những phương án chắc chắn không phải tối ưu. Kỹ thuật đó gọi là kỹ thuật đánh giá nhánh cận trong tiến trình quay lui.

4.3. MÔ HÌNH KỸ THUẬT NHÁNH CẬN

Dựa trên mô hình thuật toán quay lui, ta xây dựng mô hình sau:

```
procedure Init;
begin
  ⟨Khởi tạo một cấu hình bất kỳ BESTCONFIG⟩;
end;

{Thủ tục này thử chọn cho x[i] tất cả các giá trị nó có thể nhận}
procedure Attempt(i: Integer);
begin
  for ⟨Mọi giá trị V có thể gán cho x[i]⟩ do
    begin
      ⟨Thử cho x[i] := V⟩;
      if ⟨Việc thử trên vẫn còn hi vọng tìm ra cấu hình tốt hơn BESTCONFIG⟩ then
        if ⟨x[i] là phần tử cuối cùng trong cấu hình⟩ then
          ⟨Cập nhật BESTCONFIG⟩
        else
          begin
            ⟨Ghi nhận việc thử x[i] = V nếu cần⟩;
            Attempt(i + 1); {Gọi đệ quy, chọn tiếp x[i+1]}
            ⟨Bỏ ghi nhận việc thử cho x[i] = V (nếu cần)⟩;
          end;
    end;
end;
```

```
begin
  Init;
  Attempt(1);
  ⟨Thông báo cấu hình tối ưu BESTCONFIG⟩;
end.
```

Kỹ thuật nhánh cận thêm vào cho thuật toán quay lui khả năng đánh giá theo từng bước, nếu tại bước thứ i, giá trị thử gán cho x[i] không có hi vọng tìm thấy cấu hình tốt hơn cấu hình BESTCONFIG thì thử giá trị khác ngay mà không cần phải gọi đệ quy tìm tiếp hay ghi nhận kết quả làm gì. Nghiệm của bài toán sẽ được làm tốt dần, bởi khi tìm ra một cấu hình mới (tốt hơn BESTCONFIG - tất nhiên), ta không in kết quả ngay mà sẽ cập nhật BESTCONFIG bằng cấu hình mới vừa tìm được

4.4. BÀI TOÁN NGƯỜI DU LỊCH

4.4.1. Bài toán

Cho n thành phố đánh số từ 1 đến n và m tuyến đường giao thông hai chiều giữa chúng, mạng lưới giao thông này được cho bởi bảng C cấp nxn, ở đây C[i, j] = C[j, i] = Chi phí đi đoạn đường trực tiếp từ thành phố i đến thành phố j. Giả thiết rằng C[i, i] = 0 với $\forall$i, C[i, j] = $+\infty$ nếu không có đường trực tiếp từ thành phố i đến thành phố j.

Một người du lịch xuất phát từ thành phố 1, muốn đi thăm tất cả các thành phố còn lại mỗi thành phố đúng 1 lần và cuối cùng quay lại thành phố 1. Hãy chỉ ra cho người đó hành trình với chi phí ít nhất. Bài toán đó gọi là bài toán người du lịch hay bài toán hành trình của một thương gia (Traveling Salesman)

4.4.2. Cách giải

Hành trình cần tìm có dạng x[1..n + 1] trong đó x[1] = x[n + 1] = 1 ở đây giữa x[i] và x[i+1]: hai thành phố liên tiếp trong hành trình phải có đường đi trực tiếp (C[i, j] $\neq +\infty$) và ngoại trừ thành phố 1, không thành phố nào được lặp lại hai lần. Có nghĩa là dãy x[1..n] lập thành 1 hoán vị của (1, 2, …, n).

Duyệt quay lui: x[2] có thể chọn một trong các thành phố mà x[1] có đường đi tới (trực tiếp), với mỗi cách thử chọn x[2] như vậy thì x[3] có thể chọn một trong các thành phố mà x[2] có đường đi tới (ngoài x[1]). Tổng quát: x[i] có thể chọn 1 trong các thành phố **chưa đi qua** mà **từ x[i-1] có đường đi trực tiếp tới ($1 \le i \le n$).**

Nhánh cận: Khởi tạo cấu hình BestConfig có chi phí = $+\infty$. Với mỗi bước thử chọn x[i] xem chi phí đường đi cho tới lúc đó có < Chi phí của cấu hình BestConfig?, nếu không nhỏ hơn thì thử giá trị khác ngay bởi có đi tiếp cũng chỉ tốn thêm. Khi thử được một giá trị x[n] ta kiểm tra xem x[n] có đường đi trực tiếp về 1 không ? Nếu có đánh giá chi phí đi từ thành phố 1 đến thành phố x[n] cộng với chi phí từ x[n] đi trực tiếp về 1, nếu nhỏ hơn chi phí của đường đi BestConfig thì cập nhật lại BestConfig bằng cách đi mới.

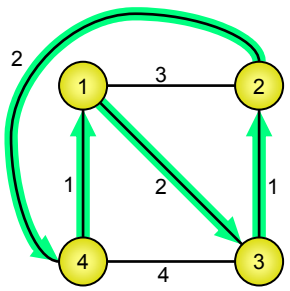
Sau thủ tục tìm kiếm quay lui mà chi phí của BestConfig vẫn bằng $+\infty$ thì có nghĩa là nó không tìm thấy một hành trình nào thoả mãn điều kiện đề bài để cập nhật BestConfig, bài toán

không có lời giải, còn nếu chi phí của BestConfig < +∞ thì in ra cấu hình BestConfig - đó là hành trình ít tốn kém nhất tìm được

Input: file văn bản TOURISM.INP

- Dòng 1: Chứa số thành phố n (1 ≤ n ≤ 100) và số tuyến đường m trong mạng lưới giao thông
- m dòng tiếp theo, mỗi dòng ghi số hiệu hai thành phố có đường đi trực tiếp và chi phí đi trên quãng đường đó (chi phí này là số nguyên dương ≤ 10000)

Output: file văn bản TOURISM.OUT, ghi hành trình tìm được.



TOURISM.INP	TOURISM.OUT
4 6 1 2 3 1 3 2 1 4 1 2 3 1 2 4 2 3 4 4	1->3->2->4->1 Cost: 6

P_1_04_1.PAS * Kỹ thuật nhánh cận dùng cho bài toán người du lịch

```
{$MODE DELPHI} (*This program uses 32-bit Integer [-2^31..2^31 - 1]*)
program Travelling_Salesman;
const
  InputFile  = 'TOURISM.INP';
  OutputFile = 'TOURISM.OUT';
  max = 100;
  maxE = 10000;
  maxC = max * maxE;{+∞}
var
  C: array[1..max, 1..max] of Integer; {Ma trận chi phí}
  X, BestWay: array[1..max + 1] of Integer; {X để thử các khả năng, BestWay để ghi nhận nghiệm}
  T: array[1..max + 1] of Integer; {T[i] để lưu chi phí đi từ X[1] đến X[i]}
  Free: array[1..max] of Boolean; {Free để đánh dấu, Free[i]= True nếu chưa đi qua tp i}
  m, n: Integer;
  MinSpending: Integer; {Chi phí hành trình tối ưu}

procedure Enter;
var
  i, j, k: Integer;
  f: Text;
begin
  Assign(f, InputFile); Reset(f);
  ReadLn(f, n, m);
  for i := 1 to n do {Khởi tạo bảng chi phí ban đầu}
    for j := 1 to n do
      if i = j then C[i, j] := 0 else C[i, j] := maxC;
  for k := 1 to m do
    begin
      ReadLn(f, i, j, C[i, j]);
      C[j, i] := C[i, j]; {Chi phí như nhau trên 2 chiều}
    end;
  Close(f);
end;

procedure Init; {Khởi tạo}
begin
  FillChar(Free, n, True);
```

```
  Free[1] := False; {Các thành phố là chưa đi qua ngoại trừ thành phố 1}
  X[1] := 1; {Xuất phát từ thành phố 1}
  T[1] := 0; {Chi phí tại thành phố xuất phát là 0}
  MinSpending := maxC;
end;

procedure Attempt(i: Integer); {Thử các cách chọn xi}
var
  j: Integer;
begin
  for j := 2 to n do {Thử các thành phố từ 2 đến n}
    if Free[j] then {Nếu gặp thành phố chưa đi qua}
      begin
        X[i] := j; {Thử đi}
        T[i] := T[i - 1] + C[x[i - 1], j]; {Chi phí := Chi phí bước trước + chi phí đường đi trực tiếp}
        if T[i] < MinSpending then {Hiển nhiên nếu có điều này thì C[x[i - 1], j] < +∞ rồi}
          if i < n then {Nếu chưa đến được x[n]}
            begin
              Free[j] := False; {Đánh dấu thành phố vừa thử}
              Attempt(i + 1); {Tìm các khả năng chọn x[i+1]}
              Free[j] := True; {Bỏ đánh dấu}
            end
          else
            if T[n] + C[x[n], 1] < MinSpending then {Từ x[n] quay lại 1 vẫn tốn chi phí ít hơn trước}
              begin {Cập nhật BestConfig}
                BestWay := X;
                MinSpending := T[n] + C[x[n], 1];
              end;
      end;
end;

procedure PrintResult;
var
  i: Integer;
  f: Text;
begin
  Assign(f, OutputFile); Rewrite(f);
  if MinSpending = maxC then WriteLn(f, 'NO SOLUTION')
  else
    for i := 1 to n do Write(f, BestWay[i], '->');
  WriteLn(f, 1);
  WriteLn(f, 'Cost: ', MinSpending);
  Close(f);
end;

begin
  Enter;
  Init;
  Attempt(2);
  PrintResult;
end.
```

Trên đây là một giải pháp nhánh cận còn rất thô sơ giải bài toán người du lịch, trên thực tế người ta còn có nhiều cách đánh giá nhánh cận chặt hơn nữa. Hãy tham khảo các tài liệu khác để tìm hiểu về những phương pháp đó.

4.5. DÃY ABC

Cho trước một số nguyên dương N ($N \leq 100$), hãy tìm một xâu chỉ gồm các ký tự A, B, C thoả mãn 3 điều kiện:

❖ Có độ dài N

❖ Hai đoạn con bất kỳ liền nhau đều khác nhau (đoạn con là một dãy ký tự liên tiếp của xâu)

❖ Có ít ký tự C nhất.

Cách giải:

Không trình bày, đề nghị tự xem chương trình để hiểu, chỉ chú thích kỹ thuật nhánh cận như sau:

Nếu dãy X[1..n] thoả mãn 2 đoạn con bất kỳ liền nhau đều khác nhau, thì trong 4 ký tự liên tiếp bất kỳ bao giờ cũng phải có 1 ký tự "C". Như vậy với một dãy con gồm k ký tự liên tiếp của dãy X thì số ký tự C trong dãy con đó bắt buộc phải ≥ k div 4.

Tại bước thử chọn X[i], nếu ta đã có T[i] ký tự "C" trong đoạn đã chọn từ X[1] đến X[i], thì cho dù các bước đệ quy tiếp sau làm tốt như thế nào chăng nữa, số ký tự "C" sẽ phải chọn thêm bao giờ cũng ≥ (n - i) div 4. Tức là nếu theo phương án chọn X[i] như thế này thì số ký tự "C" trong dãy kết quả (khi chọn đến X[n]) cho dù có làm tốt đến đâu cũng ≥ **T[i] + (n - i) div 4**. Ta dùng con số này để đánh giá nhánh cận, nếu nó nhiều hơn số ký tự "C" trong BestConfig thì chắc chắn có làm tiếp cũng chỉ được một cấu hình tồi tệ hơn, ta bỏ qua ngay cách chọn này và thử phương án khác.

Input: file văn bản ABC.INP chứa số nguyên dương n ≤ 100

Output: file văn bản ABC.OUT ghi xâu tìm được

ABC.INP	ABC.OUT
10	ABACABCBAB "C" Letter Count : 2

P_1_04_2.PAS * Dãy ABC

```
{$MODE DELPHI} (*This program uses 32-bit Integer [-2^31..2^31 - 1]*)
program ABC_STRING;
const
  InputFile  = 'ABC.INP';
  OutputFile = 'ABC.OUT';
  max = 100;
var
  N, MinC: Integer;
  X, Best: array[1..max] of 'A'..'C';
  T: array[0..max] of Integer; {T[i] cho biết số ký tự "C" trong đoạn từ X[1] đến X[i]}
  f: Text;

{Hàm Same(i, l) cho biết xâu gồm l ký tự kết thúc tại X[i] có trùng với xâu l ký tự liền trước nó không ?}
function Same(i, l: Integer): Boolean;
var
  j, k: Integer;
begin
  j := i - l; {j là vị trí cuối đoạn liền trước đoạn đó}
  for k := 0 to l - 1 do
    if X[i - k] <> X[j - k] then
      begin
        Same := False; Exit;
      end;
  Same := True;
end;

{Hàm Check(i) cho biết X[i] có làm hỏng tính không lặp của dãy X[1..i] hay không}
```

```
function Check(i: Integer): Boolean;
var
  l: Integer;
begin
  for l := 1 to i div 2 do {Thử các độ dài l}
    if Same(i, l) then {Nếu có xâu độ dài l kết thúc bởi X[i] bị trùng với xâu liền trước}
      begin
        Check := False; Exit;
      end;
  Check := True;
end;

{Giữ lại kết quả vừa tìm được vào BestConfig (MinC và mảng Best)}
procedure KeepResult;
begin
  MinC := T[N];
  Best := X;
end;

{Thuật toán quay lui có nhánh cận}
procedure Attempt(i: Integer); {Thử các giá trị có thể của X[i]}
var
  j: 'A'..'C';
begin
  for j := 'A' to 'C' do {Xét tất cả các giá trị}
    begin
      X[i] := j;
      if Check(i) then {Nếu thêm giá trị đó vào không làm hỏng tính không lặp}
        begin
          if j = 'C' then T[i] := T[i - 1] + 1 {Tính T[i] qua T[i - 1]}
          else T[i] := T[i - 1];
          if T[i] + (N - i) div 4 < MinC then {Đánh giá nhánh cận}
            if i = N then KeepResult
            else Attempt(i + 1);
        end;
    end;
end;

procedure PrintResult;
var
  i: Integer;
begin
  for i := 1 to N do Write(f, Best[i]);
  WriteLn(f);
  WriteLn(f, '"C" Letter Count : ', MinC);
end;

begin
  Assign(f, InputFile); Reset(f);
  ReadLn(f, N);
  Close(f);
  Assign(f, OutputFile); Rewrite(f);
  T[0] := 0;
  MinC := N; {Khởi tạo cấu hình BestConfig ban đầu rất tồi}
  Attempt(1);
  PrintResult;
  Close(f);
end.
```

Nếu ta thay bài toán là tìm xâu ít ký tự 'B' nhất mà vẫn viết chương trình tương tự như trên thì chương trình sẽ chạy chậm hơn chút ít. Lý do: thủ tục Attempt ở trên sẽ thử lần lượt các giá trị 'A', 'B', rồi mới đến 'C'. Có nghĩa ngay trong cách tìm, nó đã tiết kiệm sử dụng ký tự 'C' nhất

nên trong phần lớn các bộ dữ liệu nó nhanh chóng tìm ra lời giải hơn so với bài toán tương ứng tìm xâu ít ký tự 'B' nhất. Chính vì vậy mà nếu như đề bài yêu cầu ít ký tự 'B' nhất ta cứ lập chương trình làm yêu cầu ít ký tự 'C' nhất, chỉ có điều khi in kết quả, ta đổi vai trò 'B', 'C' cho nhau. Đây là một ví dụ cho thấy sức mạnh của thuật toán quay lui khi kết hợp với kỹ thuật nhánh cận, nếu viết quay lui thuần tuý hoặc đánh giá nhánh cận không tốt thì với N = 100, tôi cũng không đủ kiên nhẫn để đợi chương trình cho kết quả (chỉ biết rằng > 3 giờ). Trong khi đó khi N = 100, với chương trình trên chỉ chạy hết hơn 1 giây cho kết quả là xâu 27 ký tự 'C'.

Nói chung, ít khi ta gặp bài toán mà chỉ cần sử dụng một thuật toán, một mô hình kỹ thuật cài đặt là có thể giải được. Thông thường các bài toán thực tế đòi hỏi phải có sự tổng hợp, pha trộn nhiều thuật toán, nhiều kỹ thuật mới có được một lời giải tốt. Không được lạm dụng một kỹ thuật nào và cũng không xem thường một phương pháp nào khi bắt tay vào giải một bài toán tin học. Thuật toán quay lui cũng không phải là ngoại lệ, ta phải biết phối hợp một cách uyển chuyển với các thuật toán khác thì khi đó nó mới thực sự là một công cụ mạnh.

Bài tập:

Bài 1

Một dãy dấu ngoặc hợp lệ là một dãy các ký tự "(" và ")" được định nghĩa như sau:

i. Dãy rỗng là một dãy dấu ngoặc hợp lệ độ sâu 0

ii. Nếu A là dãy dấu ngoặc hợp lệ độ sâu k thì (A) là dãy dấu ngoặc hợp lệ độ sâu k + 1

iii. Nếu A và B là hay dãy dấu ngoặc hợp lệ với độ sâu lần lượt là p và q thì AB là dãy dấu ngoặc hợp lệ độ sâu là max(p, q)

Độ dài của một dãy ngoặc là tổng số ký tự "(" và ")"

Ví dụ: Có 5 dãy dấu ngoặc hợp lệ độ dài 8 và độ sâu 3:

1. (((()()))
2. ((()())())
3. ((()))()
4. (()(()))
5. ()((()))

Bài toán đặt ra là khi cho biết trước hai số nguyên dương n và k. Hãy liệt kê hết các dãy ngoặc hợp lệ có độ dài là n và độ sâu là k (làm được với n càng lớn càng tốt).

Bài 2

Cho một bãi mìn kích thước mxn ô vuông, trên một ô có thể có chứa một quả mìn hoặc không, để biểu diễn bản đồ mìn đó, người ta có hai cách:

Cách 1: dùng bản đồ đánh dấu: sử dụng một lưới ô vuông kích thước mxn, trên đó tại ô (i, j) ghi số 1 nếu ô đó có mìn, ghi số 0 nếu ô đó không có mìn

Cách 2: dùng bản đồ mật độ: sử dụng một lưới ô vuông kích thước mxn, trên đó tại ô (i, j) ghi một số trong khoảng từ 0 đến 8 cho biết tổng số mìn trong các ô lân cận với ô (i, j) (ô lân cận với ô (i, j) là ô có chung với ô (i, j) ít nhất 1 đỉnh).

Giả thiết rằng hai bản đồ được ghi chính xác theo tình trạng mìn trên hiện trường.

Về nguyên tắc, lúc cài bãi mìn phải vẽ cả bản đồ đánh dấu và bản đồ mật độ, tuy nhiên sau một thời gian dài, khi người ta muốn gỡ mìn ra khỏi bãi thì vấn đề hết sức khó khăn bởi bản đồ đánh dấu đã bị thất lạc !!. Công việc của các lập trình viên là: Từ bản đồ mật độ, hãy tái tạo lại bản đồ đánh dấu của bãi mìn.

Dữ liệu: Vào từ file văn bản MINE.INP, các số trên 1 dòng cách nhau ít nhất 1 dấu cách

- Dòng 1: Ghi 2 số nguyên dương m, n ($2 \le m, n \le 30$)
- m dòng tiếp theo, dòng thứ i ghi n số trên hàng i của bản đồ mật độ theo đúng thứ tự từ trái qua phải.

Kết quả: Ghi ra file văn bản MINE.OUT, các số trên 1 dòng ghi cách nhau ít nhất 1 dấu cách

- Dòng 1: Ghi tổng số lượng mìn trong bãi
- m dòng tiếp theo, dòng thứ i ghi n số trên hàng i của bản đồ đánh dấu theo đúng thứ tự từ trái qua phải.

Ví dụ:

MINE.INP	MINE.OUT
10 15	80
0 3 2 3 3 3 5 3 4 4 5 4 4 4 3	1 0 1 1 1 1 0 1 1 1 1 1 1 1 1
1 4 3 5 5 4 5 4 7 7 7 5 6 6 5	0 0 1 0 0 1 1 1 0 1 1 1 0 1 1
1 4 3 5 4 3 5 4 4 4 4 3 4 5 5	0 0 1 0 0 1 0 0 1 1 1 0 0 1 1
1 4 2 4 4 5 4 2 4 4 3 2 3 5 4	1 0 1 1 1 0 0 1 0 0 0 0 0 1 1
1 3 2 5 4 4 2 2 3 2 3 3 2 5 2	1 0 0 0 1 1 1 0 0 1 0 0 1 0 1
2 3 2 3 3 5 3 2 4 4 3 4 2 4 1	0 0 0 0 1 0 0 0 0 1 1 0 1 0 0
2 3 2 4 3 3 2 3 4 6 6 5 3 3 1	0 1 1 0 0 1 0 0 1 1 0 0 1 0 0
2 6 4 5 2 4 1 3 3 5 5 5 6 4 3	1 0 1 0 1 0 1 0 1 1 1 1 0 1 0
4 6 5 7 3 5 3 5 5 6 5 4 4 4 3	0 1 1 0 1 0 0 0 0 0 1 1 1 1 1
2 4 4 4 2 3 1 2 2 2 3 3 3 4 2	1 1 1 1 1 0 1 1 1 1 0 0 0 0 1