NGUYỄN XUÂN HUY

BACK TRACKING

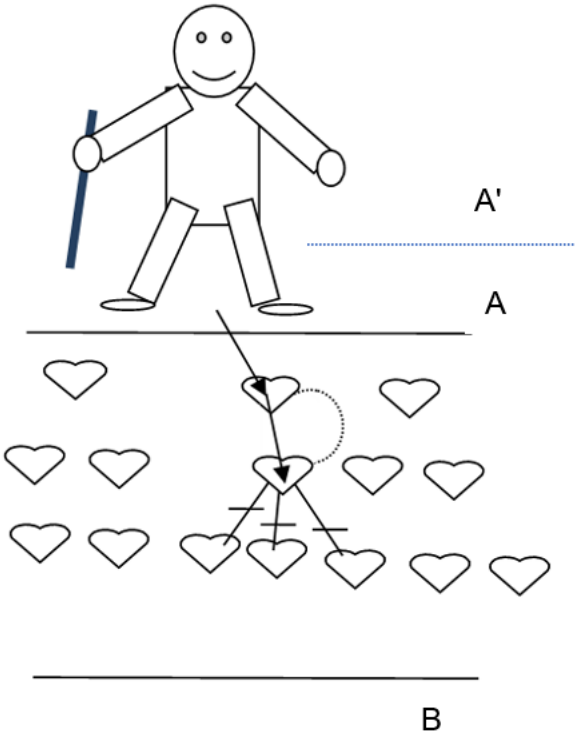
Thân mến tặng các sinh viên
Học viện Bưu chính Viễn Thông

Ngày 15 Tháng 10 Năm2021
Tác giả

Nguyễn Xuân Huy

CONTENTS

Introduction



Giả sử ta phải tìm trong một tập dữ liệu D cho trước một dãy dữ liệu:

$$v = (v[1], v[2],..., v[n])$$

thoả mãn đồng thời hai tính chất P và Q. Trước hết ta chọn một trong hai tính chất đã cho để làm nền, giả sử ta chọn tính chất P.

Sau đó ta thực hiện các bước sau đây:

Bước 1. (Khởi trị) Xuất phát từ một dãy ban đầu $v = (v[1],..., v[i])$ nào đó của các phần tử trong D sao cho v thoả P.

Bước 2. Nếu v thoả Q ta dừng thuật toán và thông báo kết quả là dãy v, ngược lại ta thực hiện Bước 3.

Bước 3. Tìm tiếp một phần tử $v[i + 1]$ để bổ sung cho v sao cho

$$v = (v[1],..., v[i], v[i + 1]) \text{ thoả P.}$$

Có thể xảy ra các trường hợp sau đây:

3.1. Tìm được phần tử $v[i + 1]$: quay lại bước 2.

3.2. Không tìm được $v[i + 1]$ như vậy, tức là với mọi $v[i + 1]$ có thể lấy trong D, dãy $v = (v[1],..., v[i], v[i + 1])$ không thoả P. Điều này có nghĩa là đi theo đường

$$v = (v[1],..., v[i])$$

sẽ không dẫn tới kết quả. Ta phải đổi hướng tại một vị trí nào đó. Để thoát khỏi ngõ cụt này, ta tìm cách thay $v[i]$ bằng một giá trị khác trong D. Nói cách khác, ta loại $v[i]$ khỏi dãy v, giảm i đi một đơn vị rồi quay lại Bước 3.

Cách làm như trên được gọi là quay lui: lùi lại một bước.

Dĩ nhiên ta phải đánh dấu *v*[*i*] là phần tử đã loại tại vị trí *i* để sau đó không đặt lại phần tử đó vào vị trí *i* trong dãy *v*.

Khi nào thì có thể trả lời là không tồn tại dãy *v* thoả đồng thời hai tính chất P và Q? Nói cách khác, khi nào thì ta có thể trả lời là bài toán vô nghiệm?

Dễ thấy, bài toán vô nghiệm khi ta đã duyệt hết mọi khả năng. Ta nói là đã vét cạn mọi khả năng. Chú ý rằng có thể đến một lúc nào đó ta phải lùi liên tiếp nhiều lần. Từ đó suy ra rằng, thông thường bài toán vô nghiệm khi ta không còn có thể lùi được nữa. Có nhiều sơ đồ giải các bài toán quay lui, dưới đây là hai sơ đồ khá đơn giản, không đệ quy.

```
Algorithm Back Tracking (one solution)
Input: n
Output: vector v = (v1, …, vn)
begin
  Init k = 1
  while true do
    if k < 1:
       print "no solution"
       return
    endif
    if k > n: print solution
      return
    endif
    if Find a step: go ahead
      else: go back
     endif
  endwhile
end Back Tracking
```

Thông thường ta khởi trị cho *v* là dãy rỗng (không chứa phần tử nào) hoặc dãy có một phần tử. Ta chỉ yêu cầu dãy *v* được khởi trị sao cho *v* thoả *P*. Lưu ý là cả dãy *v* thoả *P* chứ không phải từng phần tử trong *v* thoả *P*.

Có bài toán yêu cầu tìm toàn bộ (mọi nghiệm) các dãy *v* thoả đồng thời hai tính chất P và Q. Nếu biết cách tìm một nghiệm ta dễ dàng suy ra cách tìm mọi nghiệm như sau: mỗi khi tìm được một nghiệm, ta thông báo nghiệm đó trên màn hình hoặc ghi vào một tệp rồi thực hiện thao tác Lùi, tức là giả vờ như không công nhận nghiệm đó, do đó phải loại *v*[*i*] cuối cùng trong dãy v để tiếp tục tìm hướng khác. Phương pháp này có tên là phương pháp giả sai. Hai sơ đồ trên sẽ được sửa một chút như sau để tìm mọi nghiệm.

```
Algorithm Back Tracking (all solutions)
Input: n
Output: vector v = (v1, …, vn)
begin
  Init k = 1
  while true do
    if k < 1:
       summary
       return
    endif
    if k > n: print solution
```

.....

```
      go back
    endif
    if Find a step: go ahead
      else: go back
     endif
  endwhile
end Back Tracking
```

Queens

Hãy đặt n quân Hậu trên bàn cờ n × n sao cho không quân nào ăn được quân nào.

Một phương án đặt Hậu.

Hậu 1 đặt tại dòng 1

Hậu 2 đặt tại dòng 5

Hậu 3 đặt tại dòng 8

Hậu 4 đặt tại dòng 6

Hậu 5 đặt tại dòng 3

Hậu 6 đặt tại dòng 7

Hậu 7 đặt tại dòng 2

Hậu 8 đặt tại dòng 4

```
v = [1,5,8,6,3,7,2,4]
```

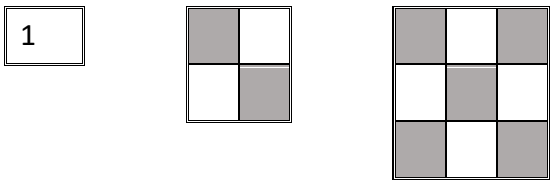
Tư duy trong sáng, minh bạch

Queens Version 1. Tìm một nghiệm

Ta biết quân Hậu trên bàn cờ Vua (cờ Quốc tế) kiểm soát hàng dọc, hàng ngang và các đường chéo cùng màu với ô Hậu đứng. Ta giải bài toán tổng quát với n quân Hậu trên bàn cờ n × n. Dễ hiểu là chỉ có thể đặt tối đa n quân hậu, vì mỗi quân Hậu sẽ kiểm duyệt một hàng dọc. Ta mã hóa mỗi quân Hậu theo hàng dọc 1, 2,…, n.

Với n = 1 ta có nghiệm tầm thường: đặt Hậu tại ô duy nhất trên bàn cờ.

Với n = 2 và n = 3 bài toán vô nghiệm.



Queens(1), Queens(2), Queens(3)

Ta dùng phương pháp quay lui (Back Tracking) để giải bài toán này.

Ý tưởng của phương pháp quay lui là như sau:

Mỗi bước đi ta chọn một khả năng dẫn đến đích.

- Nếu có một khả năng như vậy ta tiến thêm một bước theo khả năng đó.

……

- Nếu không, ta sẽ lùi lại một bước, trở về cấu hình của bước trước đó.
- Thuật toán sẽ kết thúc khi gặp một trong hai tình huống sau đây:
    - Đến đích: hiển thị nghiệm
    - Vô nghiệm: khi đã lùi về giới hạn

Sơ đồ trên có ưu điểm là đơn giản và không đệ quy.

Tổ chức dữ liệu

Ta sẽ dùng một mảng v để ghi nhận vị trí của mỗi Hậu trên bàn cờ.

Với ví dụ đã cho, khi hoàn tất, ta sẽ thu được một nghiệm

```
v = [1,5,8,6,3,7,2,4]
```

với ý nghĩa:

Đặt Hậu 1 tại dòng 1

Đặt Hậu 2 tại dòng 5

Đặt Hậu 3 tại dòng 8

Đặt Hậu 4 tại dòng 6

Đặt Hậu 5 tại dòng 3

Đặt Hậu 6 tại dòng 7

Đặt Hậu 7 tại dòng 2

Đặt Hậu 8 tại dòng 4

Một phương án đặt Hậu.

Hậu 1 đặt tại dòng 1

Hậu 2 đặt tại dòng 5

Hậu 3 đặt tại dòng 8

Hậu 4 đặt tại dòng 6

Hậu 5 đặt tại dòng 3

Hậu 6 đặt tại dòng 7

Hậu 7 đặt tại dòng 2

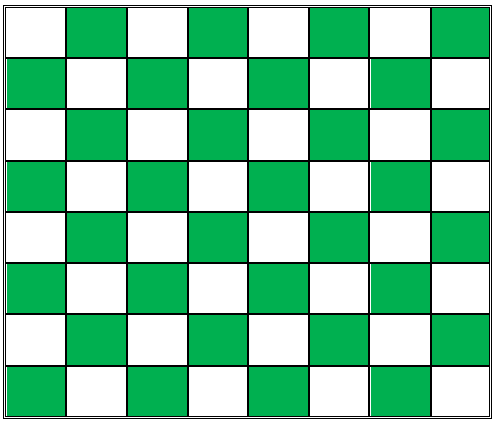
Hậu 8 đặt tại dòng 4

`v = [1,5,8,6,3,7,2,4]`

Khi khởi trị ta xếp các Hậu ở trước vạch xuất phát ngoài bàn cờ, tức là ta gán

```
v = [0,0,0,0,0,0,0,0]
```

v 0 0 0 0 0 0 0 0



v = [0,0,0,0,0,0,0,0]

Ta lần lượt di chuyển từng quân Hậu vào bàn cờ. Gọi k là quân Hậu ta cần di chuyển từ vị trí hiện đứng v[k] đến vị trí mới i để đặt Hậu k.

```
Algorithm Queens(n)Q
Input: n Hậu
Output: một phương án đặt Hậu
          v = (v1, v2, …, vn)
Begin
  Khởi trị:
  v[i] = 0, i = 1..n
  k = 1 # cầm Hậu 1
  while  True:
    if k < 1:
        Thông báo vô nghiệm
        return
    if k > n:
        print(v)
        return
    if Tìm được một nước đi: # (1)
        Tiến 1 bước
    else:
         Lùi 1 bước
```

Theo sơ đồ, ta phải xác định hai điều kiện quan trọng cho *tính dừng* của thuật toán.

Dừng khi không thành công

Dừng khi đã gặp một nghiệm.

Dễ thấy, khi ta đã đặt xong Hậu thứ n, tức là ta đã xếp xong đủ n Hậu, thì theo thói quen ta sẽ chuyển qua Hậu n+1. Vậy điều kiện dừng sau khi đã gặp một nghiệm sẽ là k > n.

Tương tự, sau nhiều lần quay lui ta đưa các Hậu ra khỏi bàn cờ về vạch xuất phát lúc đầu, theo thói quen, ta sẽ cầm Hậu trước đó, tức là Hậu k = 0. Vậy điều kiện dừng khi vô nghiệm sẽ là k < 1.

Ta viết hàm Find(k,n) xác định điều kiện *Tìm được một nước đi* cho Hậu k trên bàn cờ $n \times n$.

……

Dịch dần Hậu k từ dòng đang đứng v[k] xuống đến dòng cuối cùng n.

- Nếu tìm được một dòng i tại đó Hậu k không bị các Hậu 1, 2, …, k-1 đặt trước chiếu thì i là dòng tìm được.
- Ngược lại, với mọi i = v[k] + 1 ... n ta không tìm được dòng đặt Hậu k thì hàm cho ra giá trị 0.
- Hàm NicePlace(k,i) cho giá trị True nếu có thể đặt Hậu k tại dòng i, ngược lại, hàm cho ra giá trị False.
- Ta biết, Hai hậu k và j chiếu nhau khi và chỉ khi hai Hậu đứng trên cùng dòng: v[k] = v[j] hoặc tạo thành hai đỉnh đối diện của một hình vuông cạnh k-j: k - j = abs(v[j] - i). Để ý rằng do mỗi Hậu chiếm giữ một cột và Hậu j được đặt trước Hậu k nên j < k.

```
/*********************************************************
          Queens. Ver. 1: Tim mot nghiem

  *********************************************************/
  #include <iostream>
  #include <cmath>


  using namespace std;

  const int MN = 100;
 int v[MN]; // v[k] = dong dat Hau k

 void Answer(int n) {
   for (int i = 1; i <= n; ++i)
     cout << " " << v[i];
 }

 // Neu Hau k dung tren dong i
 // thi ko dung voi cac Hau 1..k-1
 bool NicePlace(int n, int k, int i) {
   for (int j = 1; j < k; ++j)
     if (v[j] == i || k-j == abs(v[j]-i))
       return false;
   return true;
 } // NicePlace

 // Di chuyen Hau k tu dong v[k]+1
 // den dong xuong n, tim dong dau tien
 // khong dung do voi cac Hau  dat truoc
 // 1..k-1
 int Find(int n, int k) {
   for (int i = v[k]+1; i <= n; ++i)
     if (NicePlace(n, k, i)) return i;
   return 0;
 } // Find
```

```
// Back Tracking
void Queens(int n) {
  cout << "\n\n " << n << " Queen(s): ";
  for (int i = 1; i <= n; ++i) v[i] = 0;
  int k = 1; // Cam Hau 1

  while (1) {
    if (k > n) {
      Answer(n);
      return;
    }
    if (k < 1) {
      cout << "\n No solution.";
      return;
    }
    v[k] = Find(n,k);
    if (v[k] > 0) ++k;
    else --k;
  } // while
} // Queens

main() {
  for (int n = 1; n <= 20; ++n)
    Queens(n);
  cout << "\n T H E   E N D ";
  return 0;
}
```

Queens Vesion 2. Tìm mọi nghiệm

Thay đổi ít nhất nhưng hưởng lợi nhiều nhất

```
Algorithm Queens(n)Q
Input: n Hậu
Output: Mọi phương án đặt Hậu
          v = (v1, v2, …, vn)
Begin
  Khởi trị:
  v[i] = 0, i = 1..n
  k = 1 # cầm Hậu 1
  while  True:
    if k < 1:
        Tổng kết
        return
    if k > n: // giả sai
        print(v)
        Lùi
    if Tìm được một nước đi: # (1)
        Tiến 1 bước
    else:
         Lùi 1 bước
```

```
/*******************************************************
          Queens. Ver 2: All solutions

  *******************************************************/
  #include <iostream>
  #include <cmath>


  using namespace std;

  const int MN = 100;
 int v[MN]; // v[k] = dong dat Hau k

 // Hien thi nghiem thu d
 void Answer(int n, int d = 1) {
   cout << "\n " << d << ". ";
   for (int i = 1; i <= n; ++i)
     cout << " " << v[i];
 }

 // Neu Hau k dung tren dong i
 // thi ko dung voi cac Hau 1..k-1
 bool NicePlace(int n, int k, int i) {
   for (int j = 1; j < k; ++j)
     if (v[j] == i || k-j == abs(v[j]-i))
       return false;
   return true;
 } // NicePlace

 // Di chuyen Hau k tu dong v[k]+1
 // den dong xuong n, tim dong dau tien
 // khong dung do voi cac Hau  dat truoc
 // 1..k-1
 int Find(int n, int k) {
   for (int i = v[k]+1; i <= n; ++i)
     if (NicePlace(n, k, i)) return i;
   return 0;
 } // Find

 // Back Tracking
 void Queens(int n) {
   cout << "\n\n " << n << " Queen(s): ";
   for (int i = 1; i <= n; ++i) v[i] = 0;
   int k = 1; // Cam Hau 1

   while (1) {
     if (k > n) {
       Answer(n);
       return;
     }
     if (k < 1) {
       cout << "\n No solution.";
       return;
```

```
    }
    v[k] = Find(n,k);
    if (v[k] > 0) ++k;
    else --k;
  } // while
} // Queens

// Back Tracking
void QueensAll(int n) {
  int d = 0; // dem so nghiem
  cout << "\n\n " << n << " Queen(s): ";
  for (int i = 1; i <= n; ++i) v[i] = 0;
  int k = 1; // Cam Hau 1

  while (1) {
    if (k > n) {
      ++d;
      Answer(n,d);
      k = n; // gia sai
    }
    if (k < 1) {
      if (d > 0)
        cout << "\n Total " << d << " solution.";
        else cout << "\n no " << " solution.";
      return;
    }
    v[k] = Find(n,k);
    if (v[k] > 0) ++k;
    else --k;
  } // while
} // Queens

main() {
  Queens(8);
  QueensAll(8);
  cout << "\n T H E   E N D ";
  return 0;
}
```

Cải tiến

Hà Nội không . . . được đâu

- Đặt kiểm soát hướng
- Tổ chức lưu nghiệm

Từ chuẩn

Một từ loại M là một dãy các chữ số, mỗi chữ số nằm trong khoảng từ 1 đến M. Số lượng các chữ số có mặt trong một từ được gọi là chiều dài của từ đó. Từ loại M được gọi là từ chuẩn nếu nó không chứa hai khúc (từ con) liền nhau mà giống nhau.

a) Với giá trị N cho trước, hiển thị trên màn hình một từ chuẩn loại 3 có chiều dài N.

b) Với mỗi giá trị N cho trước, tìm và ghi vào tệp văn bản tên TUCHUAN.OUT mọi từ chuẩn loại 3 có chiều dài N.

$$1 \le N \le 40000.$$

Thí dụ:

1213123 là từ chuẩn loại 3, chiều dài 7.

1213213 không phải là từ chuẩn vì nó chứa liên tiếp hai từ con giống nhau là 213.

Tương tự, 12332 không phải là từ chuẩn vì chứa liên tiếp hai từ con giống nhau là 3.

Bài giải

Ta dùng mảng *v*[1..*n*] để lưu từ cần tìm. Tại mỗi bước *i* ta xác định giá trị *v*[*i*] trong khoảng 1..*m* sao cho *v*[1..*i*] là từ chuẩn.

Điều kiện P: *v*[1..*i*] là từ chuẩn.

Điều kiện Q: Dừng thuật toán theo một trong hai tình huống sau đây:

- nếu i = *n* thì bài toán có nghiệm *v*[1..*n*].
- nếu *i* = 0 thì bài toán vô nghiệm.

Hàm `Tim` hoạt động như sau: duyệt các giá trị tại vị trí *v*[*i*] của từ *v*[1..*i*] kể từ *v*[*i*] + 1 đến m sao cho *v*[1..*i*] là từ chuẩn.

`Tim = true` nếu tồn tại một giá trị *v*[*i*] như vậy. Ngược lại, nếu với mọi *v*[*i*] = *v*[*i*] + 1..m từ *v*[1..*i*] đều không chuẩn thì `Tim = false`.

Để kiểm tra tính chuẩn của từ *v*[1..*i*], ta lưu ý rằng từ *v*[1..*i*-1] đã chuẩn (tính chất P), do đó chỉ cần khảo sát các cặp từ có chứa *v*[*i*], cụ thể là khảo sát các cặp từ có chiều dài k đứng cuối từ v. Đó là các cặp từ *v*[(i–*k*–*k*+1)..(*i*–*k*)] và *v*[*i*–*k*+1..*i*] với *k* = 1..(*i div* 2). Nếu với mọi *k* như vậy hai từ đều khác nhau thì **`Chuan=true`**. Ngược lại,

Hàm `Bang(i,k)` kiểm tra xem hai từ kề nhau chiều dài *k* tính từ *i* trở về trước có bằng nhau hay không.

Hai từ được xem là khác nhau nếu chúng khác nhau tại một vị trí nào đó.

Program

```
// NormalWord.CPP
// Back Tracking
#include <iostream>

#include <windows.h>

using namespace std;

string s;
int n; // len of s
```

.....

```
void Go() {
   cout << " ? ";
   fflush(stdin);
   if (cin.get() == '.') exit(0);
}

// s------------(k-d)----k
string Str(int k, int d) {
   string w = "";
   for (int i = 0; i < d; ++i)
      w = s[k-i] + w;
    return w;
}

bool Norm(int k) {
   if (k == 0) return true;
   int k2 = (k+1) / 2;
   int start = (k % 2 == 0) ? 1 : 0;
   for (int d = 1; d <= k2; ++d) {
       if (Str(k, d) == Str(k-d, d))
         return false;
   }
   return true;
}

bool Find(int k) {
   for (int i = 0; i < 3; ++i) {
      ++s[k];
      if (s[k] > '3') return false;
      if (Norm(k)) return true;
   }
   return false;
}


void NW(int len) {
   n = len;
   s = "";
   for (int i = 0; i < n; ++i) {
      s += '0';
   }
   //cout << "\n Init s: " << s;
   int k = 0;
   while (true) {
        if (k < 0) {
           cout << " No solution.";
           return;
        }
        if (k >= n) {
           cout << " " << s;
           return;
        }
        if (Find(k)) ++k;
         else {s[k] = '0'; --k; }
   }
}
```

```
Test() {
    for (int n = 1; n <= 22; ++n) {
     cout << "\n n = " << n << " : ";
     NW(n);
     Go();
   }
}

main() {
   Test(); // <= 2 s.
   cout << "\n T h e   E n d";
}
```

Result

```
 n = 1 :  1 ?

 n = 2 :  12 ?

 n = 3 :  121 ?

 n = 4 :  1213 ?

 n = 5 :  12131 ?

 n = 6 :  121312 ?

 n = 7 :  1213121 ?

 n = 8 :  12131231 ?

 n = 9 :  121312313 ?

 n = 10 :  1213123132 ?

 n = 11 :  12131231321 ?

 n = 12 :  121312313212 ?

 n = 13 :  1213123132123 ?

 n = 14 :  12131231321231 ?

 n = 15 :  121312313212312 ?

 n = 16 :  1213123132123121 ?

 n = 17 :  12131231321231213 ?

 n = 18 :  121312313212312131 ?

 n = 19 :  1213123132123121312 ?

 n = 20 :  12131231321231213123 ?

 n = 21 :  121312313212312131231 ?
```

```
 n = 22 :  1213123132123121312313 ?
 // n = 700
1213123132123121312313213121321231213123132123121321231321312132123121312313212
3213121321231213123132131213212312131232123121321231321312132123121312313212312
1321231321312132123121312321231213212313213121321231213231232123121312313212312
1312321231213212313213121321231213123132123121321231321312132123121312321231213
2123132131213212312132312321231213212313213121321231213123132123121321231321312
1321231213123212312132123132131213212313231213123132123121312313231213123212312
1312313212312131232123121321231321312132123121312313212312132123132131213212312
1312321231213212313213121321231213231232123121312313212312131232123121321231321
31213212312131231321231213212313213121321231213123212312132123132131
 T h e   E n d
```

Với N = 16, M = 3, có tổng cộng 798 nghiệm, tức là 798 từ chuẩn chiều dài 16 tạo từ các chữ số 1, 2 và 3. Dưới đây là 20 nghiệm đầu tiên tìm được theo thuật toán.

```
Nghiem thu 1:  1213123132123121
Nghiem thu 2:  1213123132123213
Nghiem thu 3:  1213123132131213
Nghiem thu 4:  1213123132131231
Nghiem thu 5:  1213123132131232
Nghiem thu 6:  1213123132312131
Nghiem thu 7:  1213123132312132
Nghiem thu 8:  1213123132312321
Nghiem thu 9:  1213123212312131
Nghiem thu 10: 1213123212312132
Nghiem thu 11: 1213123212313212
Nghiem thu 12: 1213123212313213
Nghiem thu 13: 1213123212313231
Nghiem thu 14: 1213123213121321
Nghiem thu 15: 1213123213121323
Nghiem thu 16: 1213123213231213
Nghiem thu 17: 1213123213231232
Nghiem thu 18: 1213123213231321
Nghiem thu 19: 1213212312131231
Nghiem thu 20: 1213212312131232
```

Tìm đường trong mê cung.

Mê cung là một đồ thị vô hướng bao gồm N đỉnh, được mã số từ 1 đến N, với các cạnh, mỗi cạnh nối hai đỉnh nào đó với nhau. Cho hai đỉnh S và T trong một mê cung. Hãy tìm một đường đi bao gồm các cạnh gối đầu nhau liên tiếp bắt đầu từ đỉnh S, kết thúc tại đỉnh T sao cho không qua đỉnh nào quá một lần.

Dữ liệu vào: Tệp văn bản tên **`MECUNG.INP`** với cấu trúc như sau:

- Dòng đầu tiên, được gọi là dòng 0, chứa ba số tự nhiên *N, S* và *T* ghi cách nhau bởi dấu cách, trong đó *N* là số lượng đỉnh của mê cung, *S* là đỉnh xuất phát, *T* là đỉnh kết thúc.
- Dòng thứ *i, i* = 1..(*N* - 1) cho biết có hay không cạnh nối đỉnh *i* với đỉnh *j, j* = (*i* + 1)..*N.*

Thí dụ:

MECUNG.INP
9 6 7
1 0 1 1 1 0 0 0
1 1 0 0 0 0 0
0 0 0 1 0 0
0 1 1 0 0
0 0 0 0
0 0 0
0 0
1

Với thí dụ đã cho kết quả có thể là:

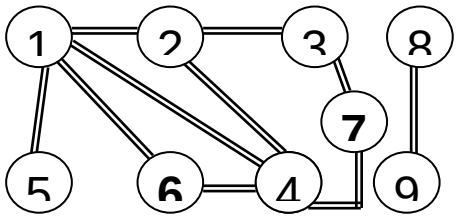
5

6 4 2 3 7

Từ đỉnh 6 có thể đến được đỉnh 7, qua 5 đỉnh theo đường bốn khúc:

6 → 4 → 2 → 3 → 7.

Với mê cung đã cho, nếu yêu cầu tìm đường đi từ đỉnh 6 đến đỉnh 9, tức là với dữ liệu vào như trên thì sẽ nhận được kết quả 0 với ý nghĩa là không có đường đi từ đỉnh 6 đến đỉnh 9, do mê cung đã cho không liên thông, đỉnh 6 và đỉnh 9 nằm trong hai vùng liên thông khác nhau.



Thuật toán

Xuất phát từ đỉnh *v*[1] = *s*, mỗi bước lặp *i* ta thực hiện các kiểm tra sau. Gọi *k* là số đỉnh đã đi qua và được tích luỹ trong mảng giải trình đường đi *v*, cụ thể là xuất phát từ đỉnh *v*[1] = *s*, sau một số lần duyệt ta quyết định chọn đường đi qua các đỉnh *v*[1], *v*[2], *v*[3],…, *v*[*k*]. Có thể gặp các tình huống sau:

a) (Đến đích?) nếu *v*[*k*] = t tức là đã đến được đỉnh *t*: thông báo kết quả, dừng thuật toán, ngược lại thực hiện *b*.

b) (Thất bại?) *k* = 0: nếu đã quay trở lại vị trí xuất phát *v*[*i*] = *s* mà từ đó không còn đường đi nào khác thì phải lùi một bước nữa, do đó *k* = 0. Trường hợp này chứng tỏ bài toán vô nghiệm, tức là, do đồ thị không liên thông nên không có đường đi từ đỉnh *s* đến đỉnh *t*. Ta thông báo vô nghiệm và dừng thuật toán.

c) (Đi tiếp?) nếu từ đỉnh *v*[*k*] tìm được một cạnh chưa đi qua và dẫn đến một đỉnh *i* nào đó thì tiến theo đường đó, nếu không: thực hiện bước *d*.

d) (Lùi một bước) Bỏ đỉnh *v*[*k*], lùi lại đỉnh *v*[*k*-1].

Thuật toán trên có tên là *sợi chỉ Arian* được phỏng theo một truyền thuyết cổ Hy Lạp sau đây. Anh hùng Te-dây phải tìm diệt con quái vật nhân ngưu (đầu người, mình trâu) Minotav ẩn náu trong một phòng của mê cung có nhiều ngõ ngách rắc rối đã từng làm lạc bước nhiều dũng sĩ và những người này đều trở thành nạn nhân của Minotav. Người yêu của chàng Te-dây là công chúa của xứ Mino đã đưa cho chàng một cuộn chỉ và dặn chàng như sau: Chàng hãy buộc một đầu chỉ vào cửa mê cung (phòng xuất phát s), sau đó, tại mỗi phòng trong mê cung, chàng hãy tìm xem có Minotav ẩn trong đó không. Nếu có, chàng hãy chiến đấu dũng cảm để hạ thủ nó rồi cuốn chỉ quay ra cửa hang, nơi em trông ngóng chàng. Nếu chưa thấy Minotav tại phòng đó, chàng hãy kiểm tra xem chỉ có bị rối hay không. Cuộn chỉ bắt đầu rối khi nào từ phòng chàng đứng có hai sợi chỉ đi ra hai cửa khác nhau. Nếu chỉ rối như vậy, chàng hãy cuộn chỉ để lùi lại một phòng và nhớ đánh dấu đường đã đi để khỏi lạc bước vào đó lần thứ hai.

Nếu không gặp chỉ rối thì chàng hãy yên tâm dò tìm một cửa chưa đi để qua phòng khác. Đi đến đâu chàng nhớ nhả chỉ theo đến đó. Nếu không có cửa để đi tiếp hoặc từ phòng chàng đang đứng, mọi cửa ra đều đã được chàng đi qua rồi, thì chàng hãy cuốn chỉ để lùi lại một phòng rồi tiếp tục tìm cửa khác.

Ta xuất phát từ sơ đồ tổng quát cho lớp bài toán quay lui.

Thủ tục **`Doc`** - đọc dữ liệu từ tệp **`MECUNG.INP`** vào mảng hai chiều a. Đây chính là ma trận kề của đồ thị biểu diễn mê cung. Mảng a sẽ đối xứng vì mê cung là đồ thị vô hướng. Đây cũng chính là lí do giải thích dữ liệu vào chỉ cho dưới dạng nửa trên của ma trận kề.

Thủ tục `Xem` – hiển thị dữ liệu trên màn hình để kiểm tra việc đọc có đúng không. Với những người mới lập trình cần luôn luôn viết thủ tục `Xem`. Khi nộp bài thì có thể bỏ lời gọi thủ tục này. Các hằng kiểu **`string bl = #32`** là mã ASCII của dấu cách, hằng **`nl = #13#10`** là một xâu chứa hai kí tự điều khiển có mã ASCII là xuống dòng **`#13`**, tức là ứng với phím **`RETURN`** và đưa con trỏ màn hình về đầu dòng **`#10`**. Khi đó lệnh **`writeln`** sẽ tương đương với lệnh **`write(nl)`**.

Hàm **`Tim`** - từ đỉnh *v*[*k*] tìm một bước đi đến đỉnh *i*. Điều kiện: *i* phải là đỉnh chưa thăm và đương nhiên có cạnh đi từ *v*[*k*] đến *i*, nghĩa là giá trị *a*[*v*[*k*], *i*] trong ma trận kề phải là 1. Ta dùng một mảng d đánh dấu đỉnh *i* đã thăm chưa. *d*[*i*] = 0 – đỉnh *i* chưa thăm, *d*[*i*] = 1 – đỉnh *i* đã thăm và đã từng được chọn để đưa vào mảng *v* là mảng giải trình đường đi. Nếu tìm kiếm thành công ta gán cho hàm `Tim` giá trị *i*, chính là đỉnh cần đến. Ngược lại, khi việc tìm kiếm thất bại, nghĩa là không tìm được đỉnh *i* để có thể đi từ đỉnh *v*[*k*] đến đó, ta gán cho hàm **`Tim`** giá trị 0.

Ta lưu ý là mỗi đỉnh chỉ đi đến không quá một lần. Đương nhiên khi lùi thì ta buộc phải quay lại đỉnh đã đến, do đó, chính xác hơn ta phải gọi *d*[*i*]=1 là giá trị đánh dấu khi tiến đến đỉnh *i*.

Nếu tìm được đỉnh chưa thăm thoả các điều kiện nói trên ta tiến thêm một bước theo cạnh (*v*[*k*], *i*). Ta cũng đánh dấu đỉnh *i* là đã thăm bằng lệnh gán *d*[*i*]: = 1. Đó là nội dung của thủ tục `NhaChi` (nhả chỉ).

Nếu từ đỉnh *v*[*k*] ta không tìm được đỉnh nào để đi tiếp thì ta phải thực hiện thủ tục `CuonChi` (cuộn chỉ) như dưới đây. Thủ tục này chỉ đơn giản là lùi một bước từ đỉnh Te-dây hiện đang đứng trở về đỉnh trước đó, nếu có, tức là $k \geq 1$, ta đánh dấu cạnh (*v*[*k* - 1], v[k]) là đã đi hai lần. Ta nhận xét rằng, nếu không tính lần trở lại một đỉnh khi phải lùi một bước thì mỗi đỉnh trong mê cung chỉ cần thăm tối đa là một lần, do đó thay vì đánh dấu cạnh ([*v*[*k* - 1], *v*[*k*]) ta chỉ cần đánh dấu đỉnh *v*[*k*] là đủ.

Với thí dụ đã cho trong đề bài, bạn hãy chạy thử chương trình `MECUNG.PAS` với hai dữ liệu kiểm thử, một dữ liệu kiểm thử có nghiệm và một dữ liệu kiểm thử vô nghiệm.

Chú ý

Đường đi tìm được không phải là đường ngắn nhất. Thuật toán Dijkstra sẽ cho đường đi ngắn nhất.

Sudoku

Sudoku là trò chơi Nhật Bản. Đó là một bảng 9×9 ô trong đó có một số ô đã chứa sẵn một số số trong khoảng 1 đến 9. Người chơi cần điền nốt các ô còn lại hiện chứa số 0. Kết quả cuối cùng phải là bảng số đáp ứng được các yêu cầu sau đây:

Mọi dòng, mọi cột và mọi khối con 3×3 phải chứa đầy đủ các số từ 1 đến 9.

4	0	0	0	0	0	**6**	0	0
0	**9**	0	0	**8**	0	0	**5**	0
5	0	0	0	0	**9**	**2**	0	0
6	0	0	0	0	0	**1**	0	0
0	**2**	0	0	7	0	0	**3**	0
0	0	**4**	0	0	0	0	0	**9**
0	0	**1**	**3**	0	0	0	0	**8**
0	**3**	0	0	**2**	0	0	**4**	0
0	0	**9**	0	0	0	0	0	**6**

Đề bài

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

Đáp án

Input text file sudoku.inp

```
4 0 0 0 0 0 6 0 0
0 9 0 0 8 0 0 5 0
5 0 0 0 0 9 2 0 0
6 0 0 0 0 0 1 0 0
0 2 0 0 7 0 0 3 0
0 0 4 0 0 0 0 0 9
0 0 1 3 0 0 0 0 8
0 3 0 0 2 0 0 4 0
0 0 9 0 0 0 0 0 6
```

Thuật toán: Quay lui.

Phương án 1. Sau khi đọc dữ liệu vào mảng 9×9 a ta lần lượt duyệt các ô trống (lúc đầu chứa số 0). Với mỗi ô trống a[si][sj] ta gọi hàm Find để chỉnh lại giá trị cần điền cho ô đó. Nếu giá trị hiện hành của ô trống này là v thì ta duyệt từ v+1 đến 9 để tìm giá trị đầu tiên c thỏa điều kiện:

Dòng si không chứa c;

Cột sj không chứa c;

Khối chứa ô (si,sj) không chứa c.

Nếu tìm được giá trị c như trên, ta điền c vào ô a[si][sj] và chuyển qua xử lí ô trống tiếp theo.

Nếu không tìm được giá trị c ta lùi lại ô trống trước đó.

……

Thuật toán kết thúc thành công khi mọi ô trống đều được điền giá trị hợp lí.

Thuật toán kết thúc vô nghiệm nếu ta quay lui về điểm xuất phát sau khi đã duyệt hết mọi khả năng.

Để xác định được ô trống ta cần lưu lại giá trị input vào mảng 2 chiều s. Mảng hai chiều thứ hai là a sẽ chứa kết quả. Hai biến si và sj dùng để ghi nhận chỉ số dòng và cột của ô hiện hành.

```
int BT1(){
  int i,j;
  DiemXuatPhat();
  cout << "\n Diem xuat phat: si = " << si << "   sj = " << sj;
  while(1){
     if (si > 9) return 1;
     if (si < 1) return 0;
     a[si][sj] = Find();
     if (a[si][sj] > 0) NextCell(); // Tien
     else PredCell(); // Lui
  }

  return 1;
}
```

Thủ tục DiemXuatPhat() tìm ô trống đầu tiên (si,sj) để làm ô xuất phát cho quá trình duyệt.

Hàm Find() trước hết đánh dấu các số đã ghi trên dòng si, cột sj và khối 3×3 chứa ô đang xét (si,sj). Mảng c dùng để đánh dấu được ghi nhận như sau: c[i] = 0 cho biết số i chưa xuất hiện, ngược lại, c[i] > 0 cho biết số i đã xuất hiện trên dòng si hoặc cột sj hoặc trong khối chứa ô (si,sj).

```
int Find(){
  memset(c,0,sizeof(c));
  int i, j, di, dj;
 // Danh dau cac so tren dong si, cot sj
  for (i = 1; i <= 9; ++i){
    c[a[si][i]] = c[a[i][sj]] = 1;
  }
  // Dau khoi: (di,dj)
  di = (si < 4) ? 1 : ((si < 7) ? 4 : 7);
  dj = (sj < 4) ? 1 : ((sj < 7) ? 4 : 7);
  for (i = 0; i < 3; ++i)
    for (j = 0; j < 3; ++j)
        c[a[di+i][dj+j]] = 1;
  // Tim so de cap nhat o (si,sj)
  for (i = a[si][sj]+1; i <= 9; ++i)
    if (c[i] == 0) return i;
  return 0;
}
```

Hai hàm NextCell tìm ô trống sát sau, hàm PredCell tìm ô trống sát trước ô trống đã duyệt.

```
// Tim o trong sau o [si,sj] trong ma tran s
void NextCell(){
```

```
   int i, j;
   // Duyet dong si
   for (j = sj+1; j <= 9; ++j)
     if (s[si][j] == 0) {
       sj = j; return;
   }
   // Duyet cac dong tu si+1 .. 9
   for (i = si+1; i <= 9; ++i)
     for (j = 1; j <= 9; ++j)
       if (s[i][j] == 0){
          si = i; sj = j; return;
       }
   si = sj = 10;
}

// Tim o trong truoc o [si,sj] trong ma tran s
void PredCell(){
   int i, j;
   // Duyet dong si
   for (j = sj-1; j > 0; --j)
     if (s[si][j] == 0) {
       sj = j; return;
   }
   // Duyet cac dong tu si+1 .. 9
   for (i = si-1; i > 0; --i)
     for (j = 9; j > 0; --j)
       if (s[i][j] == 0){
          si = i; sj = j; return;
       }
   si = sj = 0;
}
```

Phương án 2. Ta sử dụng thêm một số mảng hai chiều để ghi nhận các ô trống sát sau và sát trước mỗi ô trống. Trước hết ta duyệt xuôi các ô trống, từ ô trống đầu tiên đến ô trống cuối cùng để thiết lập trị cho các mảng iNext và jNext. iNext[i][j] chứa số hiệu dòng, jNext[i][j] chứa số hiệu cột của ô trống sát trước ô trống (i,j). Sau đó ta duyệt ngược các ô trống để thiết lập trị cho các mảng iPred và jPred. iPrredt[i][j] chứa số hiệu dòng, jPred[i][j] chứa số hiệu cột của ô trống sát sau ô trống (i,j).

```
// To chuc cac tro truoc va sau cho moi a[i][j]
 void Init(){
   int q, ii, jj, i, j;
   // Duyet xuoi
    q = ii = jj = 0;
   for (i = 1; i <= 9; ++i)
     for (j = 1; j <= 9; ++j)
       switch(q){
         case 0: // duyet doan 0
                if (a[i][j] == 0) {
                    iPred[i][j] = ii; jPred[i][j] = jj;
                    ii = i; jj = j;
                 } else q = 1;
              break;
         case 1: // Duyet doan > 0
              if (a[i][j] == 0) {
                 iPred[i][j] = ii; jPred[i][j] = jj;
```

```
                ii = i; jj = j; q = 0;
              }
              break;
        } // switch
        // Duyet nguoc
        q = 0; ii = jj = 10;
        for (i = 9; i > 0; --i)
          for (j = 9; j > 0; --j)
        switch(q){
          case 0: // duyet doan 0
                  if (a[i][j] == 0) {
                      iNext[i][j] = ii; jNext[i][j] = jj;
                      ii = i; jj = j;
                   } else q = 1;
              break;
          case 1: // Duyet doan > 0
              if (a[i][j] == 0) {
                 iNext[i][j] = ii; jNext[i][j] = jj;
                 ii = i; jj = j; q = 0;
              }
              break;
        } // switch
    }
```

Sau khi thiết lập được các ô trống sát sau và sát trước cho mỗi ô trống ta tổ chức phương án BT2 như sau:

```
int BT2(){
  int i,j;
  DiemXuatPhat();
  cout << "\n Diem xuat phat: si = " << si << "   sj = " << sj;
  while(1){
     if (si > 9) return 1;
     if (si < 1) return 0;
     a[si][sj] = Find();
     i = si; j = sj;
     if (a[si][sj] > 0) { // Tien
       si = iNext[i][j]; sj = jNext[i][j];
     }
     else { // Lui
       si = iPred[i][j]; sj = jPred[i][j];
     }
  }
}
```

Cuối cùng ta viết hàm Test để kiểm tra kết quả có thỏa các điều kiện sudoku hay không.

Chương trình C++

```
/*-------------------------------------------------
 SUDOKU.CPP
 Phuong an 1. Quay lui binh thuong.
 Phuong an 2. TO CHUC CON TRO TRUOC VA SAU CHO MOI O.
 -------------------------------------------------*/
#include <iostream>
```

```
#include <fstream>
using namespace std;

const int MN = 10;
int a[MN][MN]; // sudoku table
int s[MN][MN];
int iNext[MN][MN], jNext[MN][MN], iPred[MN][MN],jPred[MN][MN];
int c[MN];
int si,sj; // CHI SO O DANG DUYET
// Hien thi ma tran 2 chieu a[d..c][d..c]
void Print(int a[][MN], int d = 1, int c =  9){
  int i, j;
  for (i = d; i <= c; ++i){
     cout << endl;
    for (j = d; j <= c; ++j)
      cout << " " << a[i][j];
    }
}
// Hien thi mang 1 chieu a[d..c]
void Print(int a[], int d = 1, int c =  9){
  int i;
  cout << endl;
  for (i = d; i <= c; ++i)
     cout << " " << a[i];
}

void Read(){
  int i, j;
  ifstream f("sudoku.inp");
  for (i = 1; i <= 9; ++i)
    for (j = 1; j <= 9; ++j)
      f >> a[i][j];
  f.close();
}

// To chuc cac tro truoc va sau cho moi a[i][j]
 void Init(){
   int q, ii, jj, i, j;
   // Duyet xuoi
    q = ii = jj = 0;
   for (i = 1; i <= 9; ++i)
     for (j = 1; j <= 9; ++j)
       switch(q){
         case 0: // duyet doan 0
                if (a[i][j] == 0) {
                    iPred[i][j] = ii; jPred[i][j] = jj;
                    ii = i; jj = j;
                 } else q = 1;
              break;
         case 1: // Duyet doan > 0
              if (a[i][j] == 0) {
                 iPred[i][j] = ii; jPred[i][j] = jj;
                 ii = i; jj = j; q = 0;
              }
              break;
       } // switch
       // Duyet nguoc
       q = 0; ii = jj = 10;
       for (i = 9; i > 0; --i)
```

```
            for (j = 9; j > 0; --j)
          switch(q){
            case 0: // duyet doan 0
                   if (a[i][j] == 0) {
                       iNext[i][j] = ii; jNext[i][j] = jj;
                       ii = i; jj = j;
                    } else q = 1;
                 break;
            case 1: // Duyet doan > 0
                 if (a[i][j] == 0) {
                    iNext[i][j] = ii; jNext[i][j] = jj;
                    ii = i; jj = j; q = 0;
                 }
                 break;
          } // switch
       }

      int Find(){
        memset(c,0,sizeof(c));
        int i, j, di, dj;
        for (i = 1; i <= 9; ++i){
          c[a[si][i]] = c[a[i][sj]] = 1;
        }
        // Dau khoi: (di,dj)
        di = (si < 4) ? 1 : ((si < 7) ? 4 : 7);
        dj = (sj < 4) ? 1 : ((sj < 7) ? 4 : 7);
        for (i = 0; i < 3; ++i)
          for (j = 0; j < 3; ++j)
              c[a[di+i][dj+j]] = 1;
        for (i = a[si][sj]+1; i <= 9; ++i)
          if (c[i] == 0) return i;
        return 0;
      }
      // Tim o trong (si,sj) dau tien
      void DiemXuatPhat(){
        int i, j;
        si = sj = 0;
        for (i = 1; i <= 9; ++i){
          if (si > 0) return;
          for (j = 1; j <= 9; ++j)
            if (a[i][j] == 0) {
              si = i; sj = j; return;
            }
          }
      }

      int BT2(){
        int i,j;
        DiemXuatPhat();
        cout << "\n Diem xuat phat: si = " << si << "   sj = " << sj;
        while(1){
           if (si > 9) return 1;
           if (si < 1) return 0;
           a[si][sj] = Find();
           i = si; j = sj;
           if (a[si][sj] > 0) {
             si = iNext[i][j]; sj = jNext[i][j];
           }
           else {
```

```
            si = iPred[i][j]; sj = jPred[i][j];
          }
      }
    }

    int TestRow(int i){
        int j;
        cout << "\n Test row " << i << ": ";
        memset(c, 0, sizeof(c));
        for (j = 1; j <= 9; ++j)
          if (c[a[i][j]] > 0) {
            cout << " Failed!"; return 0;
          } else c[a[i][j]] = 1;
        cout << " Passed. "; return 1;
    }

    int TestColum(int i){
        int j;
        cout << "\n Test colum " << i << ": ";
        memset(c, 0, sizeof(c));
        for (j = 1; j <= 9; ++j)
          if (c[a[j][i]] > 0) {
            cout << " Failed!"; return 0;
          } else c[a[j][i]] = 1;
        cout << " Passed. "; return 1;
    }

    int TestBloc(int di, int dj){
        int i, j;
        cout << "\n Test bloc [" << di << "," << dj << "] : ";
        memset(c, 0, sizeof(c));
        for (i = 0; i < 3; ++i)
          for (j = 0; j < 3; ++j)
          if (c[a[di+i][dj+j]] > 0) {
            cout << " Failed!"; return 0;
          } else c[a[di+i][dj+j]] = 1;
        cout << " Passed. ";
    }

    int Test() {
      int i, j;
      for (i = 1; i <= 9; ++i)  TestRow(i);
      for (i = 1; i <= 9; ++i)  TestColum(i);
      for (i = 1; i < 8; i += 3)
        for (j = 1; j < 8; j += 3)
          TestBloc(i,j);
    }

    void Run2(){
      int k;
      cout << "\n SUDOKU Phuong an 2: "
           << " Su dung con tro truoc va sau.\n";
      Read();
      Print(a);
      Init();
      if (BT2()) {
         cout << "\n Result: \n";
         Print(a);
         Test();
```

```
  } else cout << "\n No solution.";
}

// Tim o trong sau o [si,sj] trong ma tran s
void NextCell(){
   int i, j;
   // Duyet dong si
   for (j = sj+1; j <= 9; ++j)
     if (s[si][j] == 0) {
       sj = j; return;
   }
   // Duyet cacs dong tu si+1 .. 9
   for (i = si+1; i <= 9; ++i)
     for (j = 1; j <= 9; ++j)
       if (s[i][j] == 0){
          si = i; sj = j; return;
       }
   si = sj = 10;
}

// Tim o trong truoc o [si,sj] trong ma tran s
void PredCell(){
   int i, j;
   // Duyet dong si
   for (j = sj-1; j > 0; --j)
     if (s[si][j] == 0) {
       sj = j; return;
   }
   // Duyet cac dong tu si+1 .. 9
   for (i = si-1; i > 0; --i)
     for (j = 9; j > 0; --j)
       if (s[i][j] == 0){
          si = i; sj = j; return;
       }
   si = sj = 0;
}

int BT1(){
  DiemXuatPhat();
  cout << "\n Diem xuat phat: si = " << si << "   sj = " << sj;
  while(1){
     if (si > 9) return 1;
     if (si < 1) return 0;
     a[si][sj] = Find();
     if (a[si][sj] > 0) NextCell();
     else PredCell();
  }

  return 1;
}


// Khong dung con tro
void Run1(){
  int k;
  cout << "\n SUDOKU Phuong an 1: "
       << " Khong su dung con tro truoc va sau.\n";
  Read();
  Print(a);
```

```
        memcpy(s,a,sizeof(s));
        if (BT1()) {
           cout << "\n Result: \n";
           Print(a);
           Test();
        } else cout << "\n No solution.";
      }

      main(){
       Run1();
       cout << endl; system("pause");
       Run2();
        cout << endl; system("pause");
        return 0;
      }

/*
DU LIEU TEST

4 0 0 0 0 0 6 0 0
0 9 0 0 8 0 0 5 0
5 0 0 0 0 9 2 0 0
6 0 0 0 0 0 1 0 0
0 2 0 0 7 0 0 3 0
0 0 4 0 0 0 0 0 9
0 0 1 3 0 0 0 0 8
0 3 0 0 2 0 0 4 0
0 0 9 0 0 0 0 0 6
Dap an

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

--------------------------
0 3 8 0 7 0 0 0 0
0 0 0 0 0 0 0 7 0
0 0 0 8 0 5 0 1 0
0 5 9 0 6 0 3 0 0
0 0 0 0 9 0 8 0 0
4 0 0 1 2 0 9 0 0
5 0 0 0 0 0 0 0 0
0 9 0 0 1 0 0 0 0
2 7 0 9 0 0 0 0 0
Dap an
6 3 8 2 7 1 4 9 5
1 4 5 6 3 9 2 7 8
9 2 7 8 4 5 6 1 3
7 5 9 4 6 8 3 2 1
3 1 2 5 9 7 8 6 4
4 8 6 1 2 3 9 5 7
5 6 1 3 8 2 7 4 9
8 9 4 7 1 6 5 3 2

.....................................................
```

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

-------------------
0 0 0 0 0 0 0 7 0
0 0 0 0 7 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 5 0
0 0 0 0 0 7 0 0 0
0 0 7 8 0 0 0 0 4
8 0 0 0 0 0 0 0 0

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

----------------------

*/
```