**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI**

**KHOA CÔNG NGHỆ THÔNG TIN**

**BÀI BÁO CÁO: TOÁN RỜI RẠC**

**TEAM: RECURSION (CNTT2 – K15)**

* **THÀNH VIÊN:**

**1. NGUYỄN HẢI DƯƠNG**

**2. PHẠM VĂN ĐẠT**

**3. NGUYỄN MINH ĐOÀN**

**4. HOÀNG MINH HIẾU**

**5. NGUYỄN HUY NGỌ (Leader)**

**LỜI MỞ ĐẦU**

Chúng tôi muốn bày tỏ lòng biết ơn đối với cô giáo Vũ Thị Tuyết Mai đã chỉ dạy tận tình cho chúng tôi trong thời gian giảng dạy bộ môn Toán rời rạc. Sự hiểu biết và lòng nhiệt tình của cô không những đã cung cấp cho chúng tôi những kiến thức quý báu mà còn là tấm gương sáng cho chúng tôi noi theo.

Do thời gian hạn hẹp, bài báo cáo này chỉ nêu được những nội dung trọng tâm của 10 thuật toán cơ bản chúng tôi đã được học trong bộ môn Toán rời rạc. Rất mong nhận được những lời nhận xét và góp ý của cô giáo và các bạn để chúng tôi hoàn thiện bài báo cáo này.

Trong bài báo cáo này, chúng tôi sử dụng ngôn ngữ lập trình C++, Pascal làm ngôn ngữ chính để mô tả các thuật toán khi lập trình.

**--0O0--MỤC LỤC--0O0--**

**CÁC THUẬT TOÁN CƠ BẢN**

**PHẦN I: SINH HOÁN VỊ, SINH TỔ HỢP**

**PHẦN II: KIỂM TRA TÍNH LIÊN THÔNG CỦA MỘT ĐỒ THỊ ĐƠN**

**PHẦN III: KIỂM TRA CHU TRÌNH EULER**

**PHẦN IV: KIỂM TRA CHU TRINH HAMILTON**

**PHẦN V: TÌM ĐƯỜNG ĐI NGẮN NHẤT VỚI THUẬT TOÁN DIJKSTRA**

**PHẦN VI: THUẬT TOÁN TÔ MÀU ĐỒ THỊ**

**PHẦN VII: THUẬT TOÁN TÌM KIẾM NHỊ PHÂN**

**PHẦN VIII: BA PHƯƠNG PHÁP DUYỆT CÂY**

**PHẦN IX: THUẬT TOÁN TÌM CÂY KHUNG NHỎ NHẤT**

**PHẦN X: KỸ THUẬT QUAY LUI GIẢI BÀI TOÁN 8 HẬU**

**Công việc thực hiện bài báo cáo của các thành viên:**

1. Nguyễn Hải Dương: Code bài IX: Tìm cây khung nhỏ nhất
2. Nguyễn Minh Đoàn: Code bài VI: Tô màu đồ thị

& Code bài: VII: Thuật toán tìm kiếm nhị phân

1. Phạm Văn Đạt: Trình bày vào word phần sinh tổ hợp, Kiểm tra chu trình Hamilton, thuật toán Dijkstra, Bài toán 8 hậu
2. Hoàng Minh Hiếu: Trình bày vào word phần sinh hoán vị, chu trình Euler, 3 phương pháp duyệt cây
3. Nguyễn Huy Ngọ: nêu ý tưởng trình bày và bản thảo các thuật toán sinh hoán vị, sinh tổ hợp, đồ thị liên thông, kiểm tra chu trình Euler, kiểm tra chu trình Hamilton, 3 phương pháp duyệt cây, bài toán 8 hậu, thuật toán dijkstra và code các bài trên; thuyết trình bài toán 8 hậu và sinh hoán vị.

**PHẦN I: SINH HOÁN VỊ, SINH TỔ HỢP**

1. **Phương pháp sinh là gì?**

Phương pháp sinh có thể áp dụng để giải bài toán liệt kê tổ hợp đặt ra nếu như hai điều kiện sau thỏa mãn:

* Có thể xác định được một thứ tự trên tập các cấu hình tổ hợp cần liệt kê. Từ đó có thể biết được cấu hình đầu tiên và cấu hình cuối cùng trong thứ tự đó.
* Xây dựng được thuật toán từ một cấu hình chưa phải cấu hình cuối, sinh ra được cấu hình kế tiếp nó.

Phương pháp sinh có thể được mô tả như sau:

<Xây dựng cấu hình đầu tiên>;

do

<Đưa ra cấu hình đang có>;

<Từ cấu hình đang có sinh ra cấu hình kế tiếp nếu còn>;

while <hết cấu hình>;

1. **Thứ tự từ điển là gì?**

Trên các kiểu dữ liệu đơn giản chuẩn, người ta thường nói tới khái niệm thứ tự. Ví dụ trên kiểu số thì có quan hệ: 1<2; 2<3; 3<4; …, trên kiểu kí tự char thì cũng có quan hệ ‘A’ < ‘B’; ‘C’ < ‘c’ …

Trong các bộ từ điển, các từ được liệt kê theo thứ tự được gọi là thứ tự từ điển. Cho hai từ dưới dạng xâu của các ký tự:

x = x1x2…xm

y = y1y2…yn

Từ x được gọi là đứng trước từ y theo thứ tự từ điện nếu tồn tại chỉ số i, 1<=i<=min{m,n} sao cho:

Với mọi j<=i: xj = yj

xj đứng trước yj

1. **Sinh hoán vị**

Bài toán đặt ra: Nhập vào 1 danh sách n tên người. Liệt kê tất cả các cách sắp xếp n người vào 1 hàng

Bài toán cơ sở: Viết 1 chương trình liệt kê các hoán vị của {1, 2, …, n}

Phương pháp giải:

Ví dụ với n = 4, ta phải liệt kê đủ 24 hoán vị:

1.1234 2.1243 3.1324 4.1342 5.1423 6.1432

7.2134 8.2143 9.2314 10.2341 11.2413 12.2431

13.3124 14.3142 15.3214 16.3241 17.3412 18.3421

19.4123 20.4132 21.4213 22.4231 23.4312 24.4321

Như vậy hoán vị đầu tiên sẽ là (1,2, …, n). Hoán vị cuối cùng là (n, n-1, …., 1)

Hoán vị sẽ sinh ra phải lớn hơn hoán vị hiện tại, hơn thế nữa phải là hoán vị vừa đủ lớn hơn hoán vị hiện tại theo nghĩa không thể có một hoán vị nào khác chen giữa chúng khi sắp thứ tự.

Giả sử hoán vị hiện tại là x = (3, 2, 6, 5, 4, 1), xét 4 phần tử cuối cùng ta thấy chúng được sắp sếp giảm dần, điều đó có nghĩa là cho dù ta có hoán vị 4 phần tử này thế nào, ta cũng được một hoán vị bé hơn hoán vị hiện tại! Như vậy ta phải xét đến x2=2, thay nó bằng một giá trị khác. Ta sẽ thay bằng giá trị nào?, không thể là 1 bởi nếu vậy sẽ được hoán vị nhỏ hơn, không thể là 3 vì có x1 = 3 rồi (phần tử sau không được chọn vào những giá trị mà phần tử trước đã chọn). Còn lại các giá trị 4, 5, 6. Vì cần một hoán vị **vừa đủ lớn hơn hiện tại** nên ta chọn x2 = 4. Còn các giá trị (x3, x4, x5, x6) sẽ lấy trong tập {2, 6, 5, 1}. Cũng vì tính vừa đủ lớn nên ta sẽ tìm biểu diễn nhỏ nhất của 4 số này gán cho x3, x4, x5, x6 tức là (1, 2, 5, 6). Vậy hoán vị mới sẽ là (3, 4, 1, 2, 5, 6).

(3, 2, 6, 5, 4, 1) => (3, 4, 1, 2, 5, 6).

Ta có nhận xét gì qua ví dụ này: Đoạn cuối của hoán vị được xếp giảm dần, số x5 = 4 là số nhỏ nhất trong đoạn cuối giảm giần thỏa mãn điều kiện lớn hơn x2 = 2. Nếu đổi chỗ x5 cho x2 thì ta sẽ được x2 = 4 và đoạn cuối **vẫn** **được sắp xếp giảm dần**. Khi đó muốn biểu diễn nhỏ nhất cho các giá trị trong đoạn cuối thì ta chỉ cần đảo ngược đoạn cuối.

Trong trường hợp hoán vị hiện tại là (2, 1, 3, 4) thì hoán vị kế tiếp sẽ là (2, 1, 4, 3). Ta cũng có thể coi hoán vị (2, 1, 3, 4) có đoạn cuối giảm dần, đoạn cuối này chỉ gồm 1 phần tử (4)

**Vậy kỹ thuật sinh hoán vị kế tiếp từ hoán vị hiện tại có thể xây dựng như sau:**

* *Xác định đoạn cuối giảm dần dài nhất, tìm chỉ số i của phần tử xi đứng liền trước đoạn cuối đó. Điều này đồng nghĩa với việc tìm từ vị trí sát cuối dãy lên đầu, gặp chỉ số i đầu tiên thỏa mãn xi < xi+1. Nếu toàn dãy đã là giảm dần, thì đó là cấu hình cuối.*

i := n-1;

while (i > 0) and (xi > xi+1) do i := i-1;

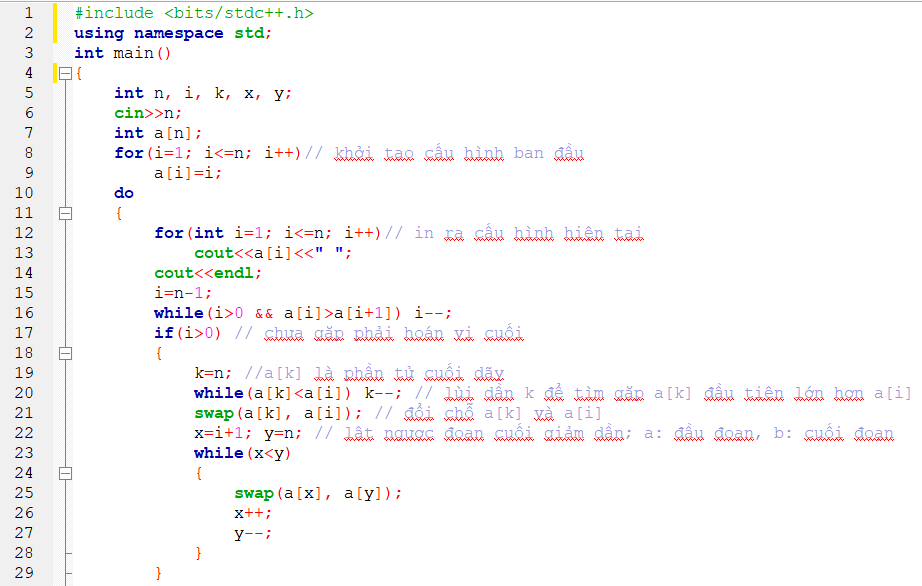
* *Trong đoạn cuối giảm dần, tìm phần tử xk nhỏ nhất thỏa mãn điều kiện xk > xi. Do đoạn cuối giảm dần, điều này thực hiện bằng cách tìm từ cuối dãy lên đầu gặp chỉ số k đầu tiên thỏa mãn xk > xi (có thể dùng tìm kiếm nhị phân).*

k := n;

while xk < xi do k := k-1;

* *Đổi chổ xk và xi, lật ngược thứ tự đoạn cuối giảm dần (từ xi+1 đến xk) trở thành tăng dần.*

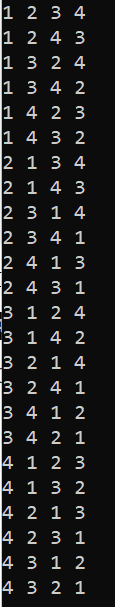
Chương trình liệt kê các hoán vị:



INPUT:



OUTPUT:



1. **Sinh tổ hợp**

Ta sẽ lập chương trình liệt kê các tập con k phần tử của tập {1, 2, ..., n} theo thứ tự từ điển

Vi dụ: với n = 5, k = 3, ta phải liệt kê đủ 10 tập con:

1.{1, 2, 3} 2.{1, 2, 4} 3.{1, 2, 5} 4.{1, 3, 4} 5.{1, 3, 5}

6.{1, 4, 5} 7.{2, 3, 4} 8.{2, 3, 5} 9.{2, 4, 5} 10.{3, 4, 5}

Như vậy tập con đầu tiên (cầu hình khởi tạo) là {1, 2, ..., k}.

Cấu hình kết thúc là {n - k + 1, n - k + 2, ..., n}.

Nhận xét: Ta sẽ in ra tập con bằng cách in ra lần lượt các phần tử của nó theo thứ tự tăng dần. Từ đó, ta có nhận xét nếu x = {x1, x2, …, xk} và x1 < x2 < … < xk thì giới hạn trên (giá trị lớn nhất có thể nhận) của xk là n, của xk - 1 là n - 1, của xk - 2 là n - 2…

Cụ thể: **Giới hạn trên của xi = n - k + i;**

Còn tất nhiên, **giới hạn dưới của xi (giá trị nhỏ nhất xi có thể nhận) là xi -1 + 1.**

Như vậy nếu ta đang có một dãy x đại diện cho một tập con, nếu x là cấu hình kết thúc có nghĩa là tất cả các phần tử trong x đều đã đạt tới giới hạn trên thì quá trình sinh kết thúc, nếu không thì ta phải sinh ra một dãy x mới tăng dần thỏa mãn vừa đủ lớn hơn dãy cũ theo nghĩa không có một tập con k phần tử nào chen giữa chúng khi sắp thứ tự từ điển.

*Ví dụ: n = 9, k = 6. Cấu hình đang có x = {1, 2, 6, 7, 8, 9}. Các phần tử từ x3 đến x6 đã đạt tới giới hạn trên nên để sinh cấu hình mới ta không thể sinh bằng cách tăng một phần tử trong số các x6, x5, x4, x3 lên được, ta phải tăng x2 = 2 lên thành x2 = 3. Được cấu hình mới là x = {1, 3, 6, 7, 8, 9}. Cấu hình này đã thỏa mãn lớn hơn cấu hình trước nhưng chưa thỏa mãn tính chất* ***vừa đủ lớn*** *muốn vậy ta lại thay x3, x4, x5, x6, bằng các giới hạn dưới của nó. Tức là:*

• *x3 := x2 + 1 = 4*

• *x4 := x3 + 1 = 5*

• *x5 := x4 + 1 = 6*

• *x6 := x5 + 1 = 7*

*Ta được cấu hình mới x = {1,* ***3,*** *4, 5, 6, 7} là cấu hình kế tiếp. Nếu muốn tìm tiếp, ta lại nhận thấy rằng x6 = 7 chưa đạt giới hạn trên, như vậy chỉ cần tăng x6 lên 1 là được x = {1, 3, 4, 5, 6,* ***8*** *}.*

***Vậy kỹ thuật sinh tập con kế tiếp từ tập đã có x có thể xây dựng như sau:***

Tìm từ cuối dãy lên đầu cho tới khi gặp một phần tử xi chưa đạt giới hạn trên n - k + i.

i := n;

While (i > 0) and (xi = n - k + i) do i := i - 1;

Nếu tìm thấy:

if i > 0 then

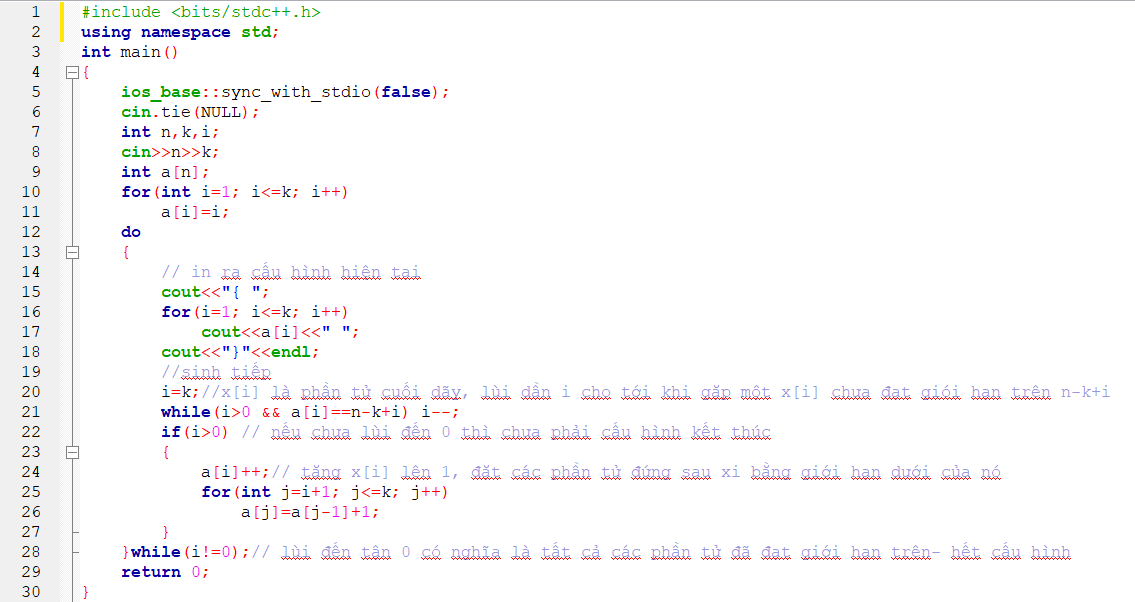
Tăng xi đó lên 1.

xi = xi + 1;

Đặt tất cả các phần tử phía sau xi bằng giới hạn dưới:

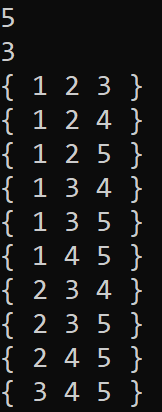
for j := i + 1 to k do xj := xj-1 + 1;

Chương trình liệt kê tập con k phần tử của tập n phần tử {1, 2, 3, …, n}



Input: n = 5, k = 3

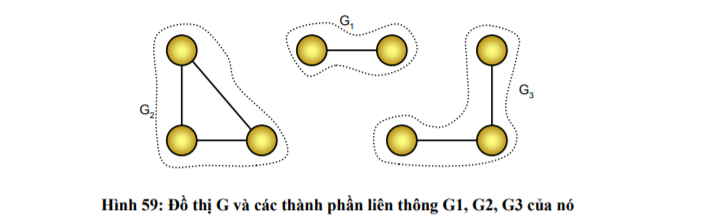
OutPut:



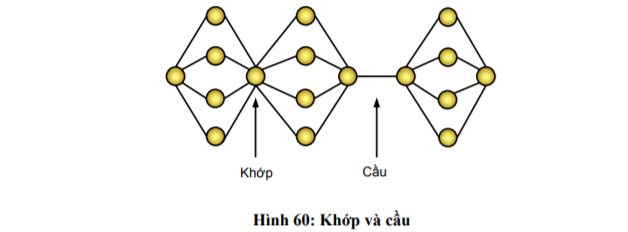
**PHẦN II: KIỂM TRA TÍNH LIÊN THÔNG CỦA MỘT ĐỒ THỊ ĐƠN**

**1. Đối với đồ thị vô hướng G = (V, E)**

G gọi là **liên thông** (connected) nếu luôn tồn tại đường đi giữa mọi cặp đinh phân biệt của đồ thị. Nếu G không liên thông thì chắc chắn nó sẽ là hợp của hai hay nhiều đồ thị con liên thông, các đồ thị con này đôi một không có đỉnh chung. Các đồ thị con liên thông rời nhau như vậy được gọi là các thành phần liên thông của đồ thị đang xét (Xem ví dụ).



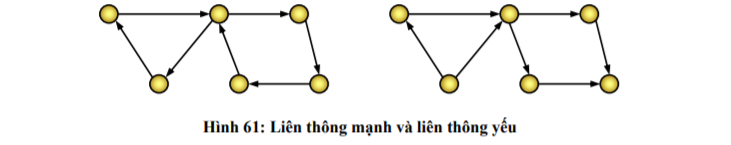
Đôi khi, việc xoá đi một đỉnh và tất cả các cạnh liên thuộc với nó sẽ tạo ra một đồ thị con mới có nhiều thành phần liên thông hơn đồ thị ban đầu, các đỉnh như thế gọi là **đỉnh cắt** hay **điểm khớp**. Hoàn toàn tương tự, những cạnh mà khi ta bỏ nó đi sẽ tạo ra một đồ thị có nhiều thành phần liên thông hơn so với đồ thị ban đầu được gọi là một **cạnh cắt** hay một **cầu.**

****

**2. Đối với đồ thị có hướng G = (V, E)**

Có hai khái niệm về tính liên thông của đồ thị có hướng tuỳ theo chúng ta có quan tâm tới hướng của các cung không.

G gọi là **liên thông mạnh** (Strongly connected) nếu luôn tồn tại đường đi (theo các cung định hướng) giữa hai định bất kỳ của đồ thị, G gọi là **liên thông yếu** (weakly connected) nếu đồ thị vô hướng nền của nó là liên thông



**3. TÍNH LIÊN THÔNG TRONG ĐỒ THỊ VÔ HƯỚNG**

Một bài toán quan trọng trong lý thuyết đồ thị là bài toán kiểm tra tính liên thông của đồ thị vô hướng hay tổng quát hơn: Bài toán liệt kê các thành phần liên thông của đồ thị vô hướng.

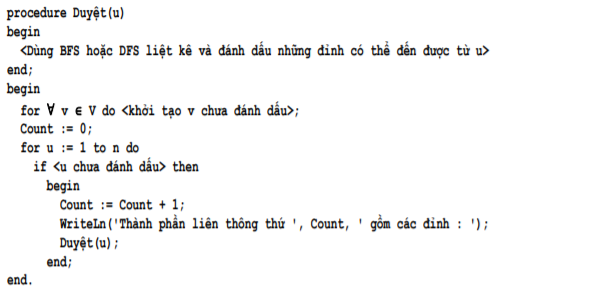
Giả sử đồ thị vô hướng G = (V, E) có n đỉnh đánh số 1, 2, ..., n.

Để liệt kê các thành phần liên thông của G phương pháp cơ bản nhất là:

Đánh dấu đỉnh 1 và những đỉnh có thể đến từ 1, thông báo những đinh đó thuộc thành phần liên thông thứ nhất.

Nếu tất cả các đỉnh đều đã bị đánh dấu thì G là đồ thị liên thông, nếu không thì sẽ tồn tại một đỉnh v nào đó chưa bị đánh dấu, ta sẽ đánh dấu v và các đinh có thể đến được từ v, thông báo những đỉnh đó thuộc thành phần liên thông thứ hai.

Và cứ tiếp tục như vậy cho tới khi tất cả các đỉnh đều đã bị đánh dấu



Với thuật toán liệt kê các thành phần liên thông như thế này, thì độ phức tạp tính toán của nó đúng bằng độ phức tạp tính toán của thuật toán tìm kiếm trên đồ thị trong thủ tục Duyệt.

1. **Thuật toán WARSALL**

Thuật toán Warshall - gọi theo tên của Stephen Warshall, người đã mô tả thuật toán này vào năm 1960, đôi khi còn được gọi là thuật toán Roy-Warshall vì Roy cũng đã mô tả thuật toán này vào năm 1959. Thuật toán đó có thể mô tả rất gọn:

Từ ma trận kề A của đơn đồ thị vô hướng G (aij = True nếu (i, j) là cạnh của G) ta sẽ sửa đổi A để nó trở thành ma trận kề của bao đóng bằng cách: Với mọi đỉnh k xét theo thứ tự từ 1 tới n, ta xét tất cả các cặp đỉnh (u, v): nếu có cạnh nối (u, k) (auk = True) và có cạnh nối (k, v) (akv = True) thì ta tự nối thêm cạnh (u, v) nếu nó chưa có (đặt auv := True). Tư tưởng này dựa trên một quan sát đơn giản như sau: Nếu từ u có đường đi tới k và từ k lại có đường đi tới v thì tất nhiên từ u sẽ có đường đi tới v.

Với n là số đỉnh của đồ thị, ta có thể viết thuật toán Warshall như sau:

for k := 1 to n do for u := 1 to n do

if a[u, k] then

for v := 1 to n do

if a[k, v] then a[u, v] := True;

hoặc

for k := 1 to n do for u := 1 to n do

for v := 1 to n do

a[u, v] := a[u, v] or a[u, k] and a[k, v];

Việc chứng minh tính đúng đắn của thuật toán đòi hỏi phải lật lại các lý thuyết về bao đóng bắc cầu và quan hệ liên thông, ta sẽ không trình bày ở đây. Có nhận xét rằng tuy thuật toán Warshall rất dễ cài đặt nhưng độ phức tạp tính toán của thuật toán này khá lớn (O(n3)).

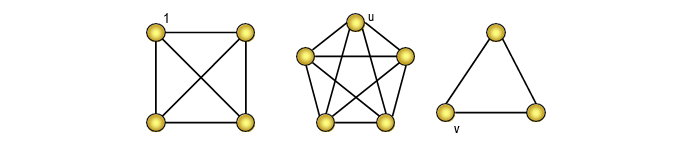
Dưới đây, ta sẽ thử cài đặt thuật toán Warshall tìm bao đóng của đơn đồ thị vô hướng sau đó đếm số thành phần liên thông của đồ thị:

Việc cài đặt thuật toán sẽ qua những bước sau:

Nhập ma trận kề A của đồ thị (Lưu ý ở đây A[v, v] luôn được coi là True với với mọi v)

Dùng thuật toán Warshall tìm bao đóng, khi đó A là ma trận kề của bao đóng đồ thị

Dựa vào ma trận kề A, đỉnh 1 và những đỉnh kề với nó sẽ thuộc thành phần liên thông thứ nhất; với đỉnh u nào đó không kề với đỉnh 1, thì u cùng với những đỉnh kề nó sẽ thuộc thành phần liên thông thứ hai; với đỉnh v nào đó không kề với cả đỉnh 1 và đỉnh u, thì v cùng với những đỉnh kề nó sẽ thuộc thành phần liên thông thứ ba v.v…

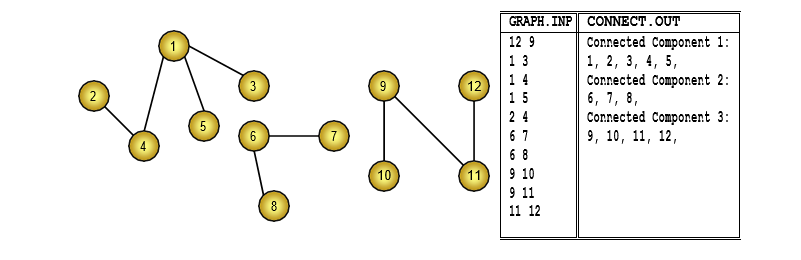


Input: file văn bản GRAPH.INP

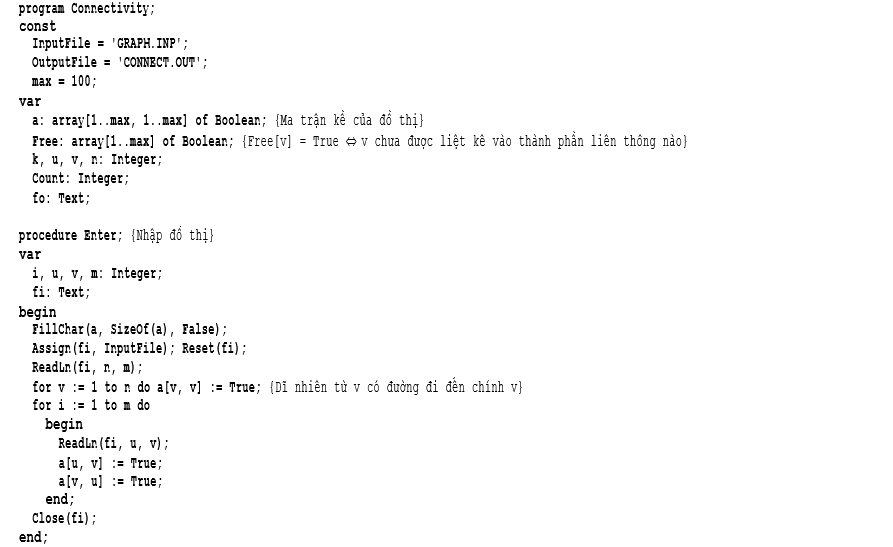
• Dòng 1: Chứa số đỉnh n (n <= 100) và số cạnh m của đồ thị cách nhau ít nhất một dấu cách

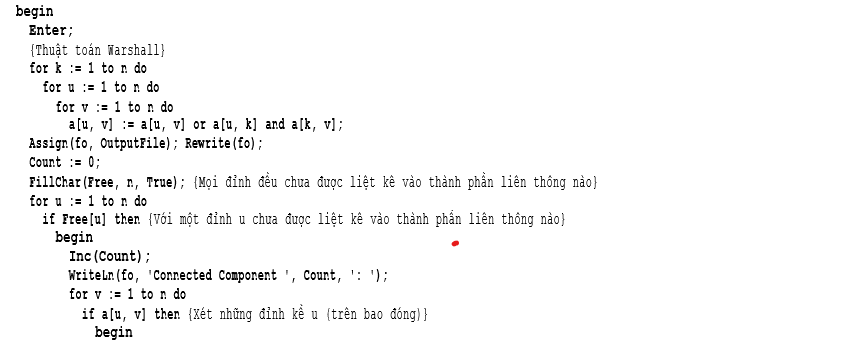
• m dòng tiếp theo, mỗi dòng chứa một cặp số u và v cách nhau ít nhất một dấu cách tượng trưng cho một cạnh (u, v)

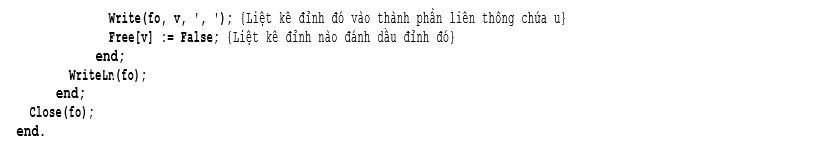
Output: file văn bản CONNECT.OUT, liệt kê các thành phần liên thông



Chương trình: (ngôn ngữ Pascal)







**III.THUẬT TOÁN FLEURY TÌM CHU TRÌNH EULER**

1.Đối với đồ thị vô hướng liên thông, mọi đỉnh đều có bậc chẵn.

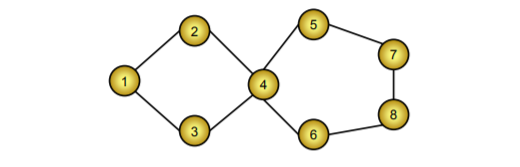
Xuất phát từ một đỉnh, ta chọn một cạnh liên thuộc với nó để đi tiếp theo hai nguyên lý sau:

Xóa bỏ cạnh đã đi qua

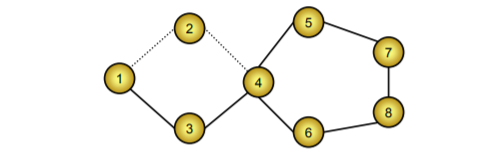
Chỉ đi qua cầu khi không còn cạnh nào khác để chọn

Và ta cứ chọn cạnh đi một cách thoải mái như vậy cho tới khi không đi tiếp được nữa, đường đi tìm được là chu trình Euler.

Ví dụ: Với đồ thị ở Hình sau:



Nếu xuất phát từ đỉnh 1, có hai cách để đi tiếp: hoặc sang 2 hoặc sang 3, giả sử ta sẽ sang 2 và xóa cạnh (1, 2) vừa đi qua. Từ 2 chỉ có cách duy nhất là sang 4 nên cho dù (2, 4) là cầu ta cũng phải đi sau đó xóa luôn cạnh (2, 4).Đến đây, các cạnh còn lại của đồ thị có thể vẽ như Hình sau bằng nét liền, các cạnh đã xóa được vẽ bằng nét đứt.



Bây giờ đang đứng ở đỉnh thứ 4 thì ta có 3 cách đi tiếp: sang 3, sang 5 hoặc sang 6.Vì (4,3) là cầu nên ta sẽ không đi theo cạnh (4, 3) mà sẽ đi (4, 5) hoặc (4, 6). Nếu đi theo (4, 5) và cứ tiếp tục đi như vậy, ta sẽ được chu trình Euler là (1, 2, 4, 5, 7, 8,6 ,4, 3, 1). Còn đi theo (4, 6) sẽ tìm được chu trình Euler là: (1, 2, 4, 6, 8, 7, 5, 4, 3, 1).

2.Đối với đồ thị có hướng liên thong yếu, mọi đỉnh đều có bán bậc ra bằng bán bậc vào.

Bằng cách “lạm dụng thuật ngữ”, ta có thể mô tả được thuật toán tìm chu trình Euler cho cả đồ thị có hướng cũng như vô hướng.

Thứ nhất, dưới đây nếu ta nói cạnh (u,v) thì hiểu là cạnh nối đinh u và đỉnh v trên đồ thị vô hướng, hiểu là cung nối từ đỉnh u tới đỉnh v trên đồ thị có hướng.

Thứ hai, ta gọi cạnh (u, v) là “một đi không trở lại” nếu như từ u ta đi tới v theo cạnh đó, sau đó xóa cạnh đó đi thì không có cạnh nào từ v quay lại u.

Vậy thì thuật toán Fleury tìm chu trình Euler có thể mô tả như sau:

Xuất phát từ một đỉnh, ta đi một cách tùy ý theo các cạnh tuân theo hai nguyên tắc : Xóa bỏ cạnh vừa đi qua vì chỉ chọn cạnh “một đi không trở lại” nếu như không còn cạnh nào khác để chọn.

CÀI ĐẶT

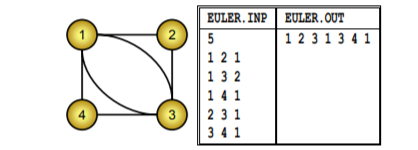
Ta sẽ cài đặt thuật toán Fleury trên một đa đồ thị vô hướng. Để đơn giản, ta coi đồ thị này đã có chu trình Euler, công việc của ta là tìm ra chu trình đó thôi. Bởi việc kiểm tra tính liên thong cũng như kiểm tra mọi đỉnh đều có bậc chẵn đến giờ có thể coi là chuyện nhỏ.

Input: file văn bản EULER.INP

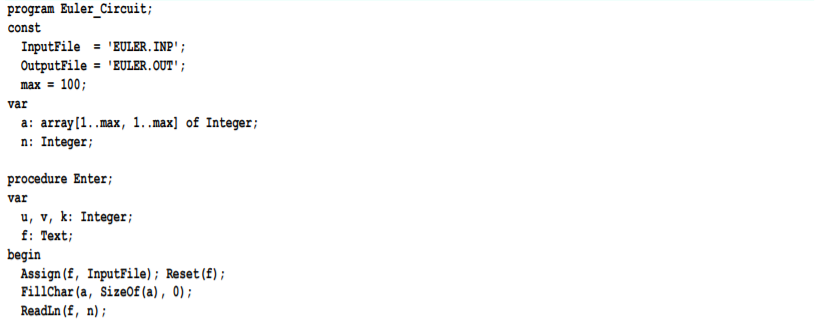
-Dòng 1: Chứa số đỉnh n của đồ thị(n<=100)

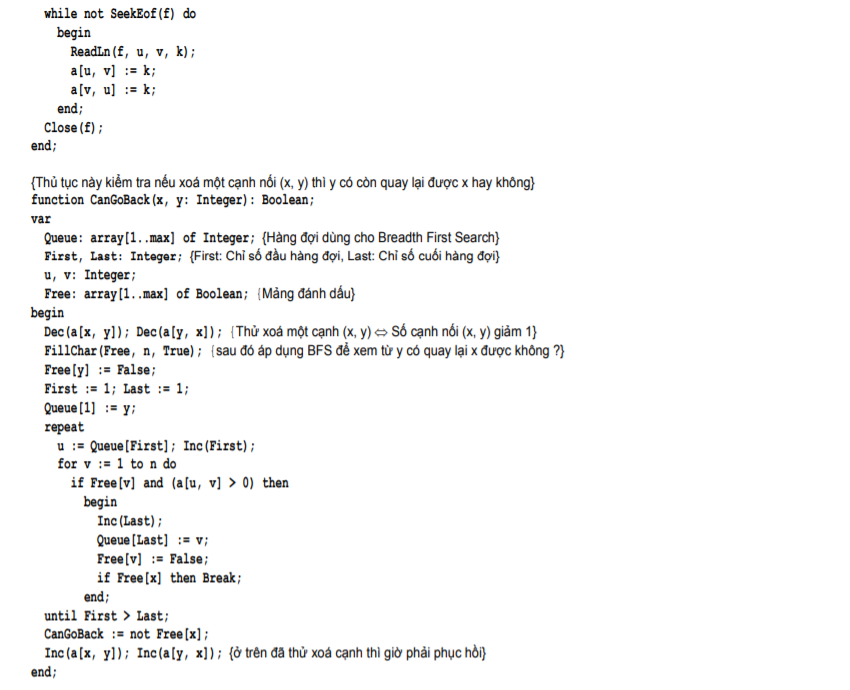
-Các dòng tiếp theo, mỗi dòng chứa 3 số nguyên dương cách nhau ít nhất 1 dấu cách có dạng: u v k cho biết giữa đỉnh u và đỉnh v không có cạnh nối

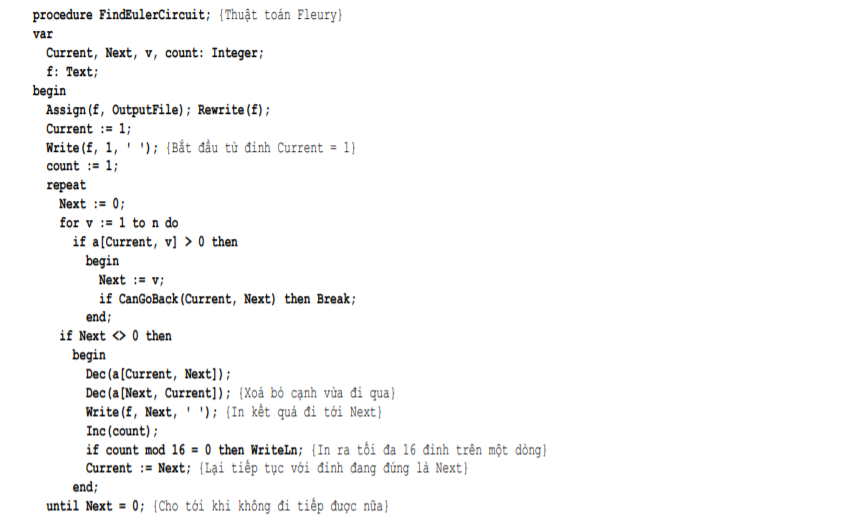
Output: file văn bản EULER.OUT, ghi chu trình EULER



Chương trình: (ngôn ngữ Pascal)



****

****

****

**PHẦN IV: KIỂM TRA CHU TRINH HAMILTON**

**1. ĐỊNH NGHĨA**

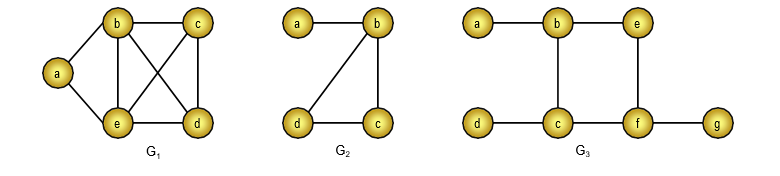
Cho đồ thị G = (V, E) có n đỉnh

Chu trình (x1, x2, ..., xn, x1) được gọi là chu trình Hamilton nếu xi xj với 1 ≤ i < j ≤ n

Đường đi (x1, x2, ..., xn) được gọi là đường đi Hamilton nếu xi xj với 1 ≤ i < j ≤ n

Có thể phát biểu một cách hình thức: Chu trình Hamilton là chu trình xuất phát từ 1 đỉnh, đi thăm tất cả những đỉnh còn lại mỗi đỉnh đúng 1 lần, cuối cùng quay trở lại đinh xuất phát. Đường đi Hamilton là đường đi qua tất cả các đinh của đồ thị, mỗi đinh đúng 1 lần. Khác với khái niệm chu trình Euler và đường đi Euler, một chu trình Hamilton không phải là đường đi Hamilton bởi có đỉnh xuất phát được thăm tới 2 lần.

Ví dụ: Xét 3 đơn đồ thị G¡, G2, G3 như trong Hình sau:



Đồ thị G1 có chu trình Hamilton (a, b, c, d, e, a). G2 không có chu trình Hamilton vì deg(a) = 1 nhưng có đường đi Hamilton (a, b, c, d). G3 không có cả chu trình Hamilton lẫn đường đi Hamilton.

**2. ĐỊNH LÝ**

Đồ thị vô hướng G, trong đó tồn tại k đỉnh sao cho nếu xoá đi k đỉnh này cùng với những cạnh liên thuộc của chúng thì đồ thị nhận được sẽ có nhiều hơn k thành phần liên thông. Thì khẳng định là G không có chu trình Hamilton. Mệnh đề phản đảo của định lý này cho ta điều kiện cần để một đồ thị có chu trình Hamilton

Định lý Dirac (1952): Đồ thị vô hướng G có n đỉnh (n ≥ 3). Khi đó nếu mọi đinh v của G đều có deg(v) ≥ n/2 thì G có chu trình Hamilton. Đây là một điều kiện đủ để một đồ thị có chu trình Hamilton.

Đồ thị có hướng G liên thông mạnh và có n đỉnh. Nếu deg+(v) ≥ n/2 và deg-(v) ≥ n/2 với mọi đỉnh v thì G có chu trình Hamilton.

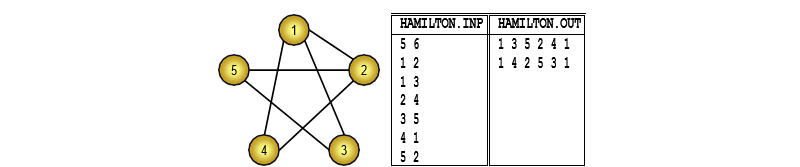
**7.3. CÀI ĐẶT**

Dưới đây ta sẽ cài đặt một chương trình liệt kê tất cả các chu trình Hamilton xuất phát từ đỉnh 1, các chu trình Hamilton khác có thể có được bằng cách hoán vị vòng quanh. Lưu ý rằng cho tới nay, người ta vẫn **chưa tìm ra** một phương pháp nào thực sự hiệu quả hơn phương pháp quay lui để tìm dù chi một chu trình Hamilton cũng như đường đi Hamilton trong trường hợp đồ thị tổng quát.

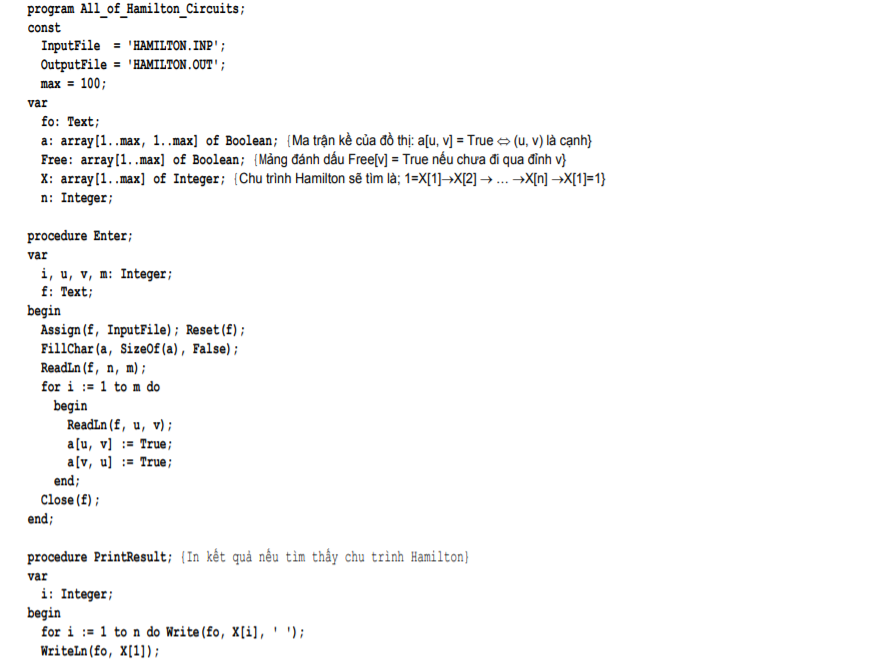
**Input:** file văn bản HAMILTON.INP

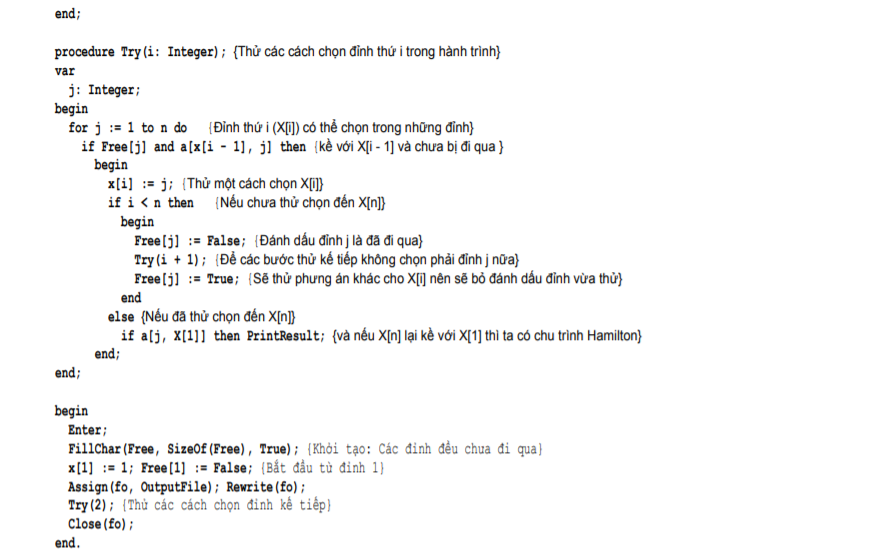
* Dòng 1 ghi số đỉnh n (2 ≤ n ≤ 100) và số cạnh m của đồ thị cách nhau 1 dấu cách
* m dòng tiếp theo, mỗi dòng có dạng hai số nguyên dương u, v cách nhau 1 dấu cách, thể hiện u, v là hai đinh kề nhau trong đồ thị

**Output:** file văn bản HAMILTON.OUT liệt kê các chu trình Hamilton

****

Chương trình: (ngôn ngữ Pascal)





**PHẦN V: TÌM ĐƯỜNG ĐI NGẮN NHẤT VỚI THUẬT TOÁN DIJKSTRA**

Trong trường hợp trọng số trên các cung không âm, thuật toán do Dijkstra đề xuất dưới đây hoạt động hiệu quả hơn nhiều so với thuật toán Ford-Bellman. Ta hãy xem trong trường hợp này, thuật toán Ford-Bellman thiếu hiệu quả ở chỗ nào:

Với đỉnh v V, Gọi d[v] là độ dài đường đi ngắn nhất từ S tới v. Thuật toán Ford-Bellman khởi gán d[S] = 0 và d[v] = + với v S, sau đó tối ưu hoá dần các nhãn d[v] bằng cách sửa nhãn theo công thức: d[v] := min(d[v], d[u] + c[u, v]) với u, v V. Như vậy nếu như ta dùng đỉnh u sửa nhãn đỉnh v, sau đó nếu ta lại tối ưu được d[u] thêm nữa thì ta cũng phải sửa lại nhãn d[v] dẫn tới việc d[v] có thể phải chỉnh đi chỉnh lại rất nhiều lần. Vậy nên chăng, tại mỗi bước **không phải ta xét mọi cặp đỉnh (u, v)** để dùng định u sửa nhãn đỉnh v mà sẽ **chọn đỉnh u là đỉnh mà không thể tối ưu nhãn d[u]** thêm được nữa.

**Thuật toán Dijkstra (E.Dijkstra - 1959) có thể mô tả như sau:**

**Bước 1: Khởi tạo**

Với đỉnh v V, gọi nhãn d[v] là độ dài đường đi ngắn nhất từ S tới v. Ta sẽ tính các d[v]. Ban đầu d[v] được khởi gán như trong thuật toán Ford-Bellman (d[S] = 0 và d[v] = ∞ với v S). Nhãn của mỗi đỉnh có hai trạng thái tự do hay cố định, nhãn tự do có nghĩa là có thể còn tối ưu hơn được nữa và nhãn cố định tức là d[v] đã bằng độ dài đường đi ngắn nhất từ S tới v nên không thể tối ưu thêm. Để làm điều này ta có thể sử dụng kỹ thuật đánh dấu: Free[v] = TRUE hay FALSE tuỳ theo d[v] tự do hay cố định. Ban đầu các nhãn đều tự do.

**Bước 2: Lặp**

Bước lặp gồm có hai thao tác:

1. **Cố định nhãn: Chọn trong các đỉnh có nhãn tự do, lấy ra đỉnh u là đỉnh có d[u] nhỏ nhất, và cố định nhãn đỉnh u.**
2. **Sửa nhãn: Dùng đỉnh u, xét tất cả những đỉnh v và sửa lại các d[v] theo công thức:**

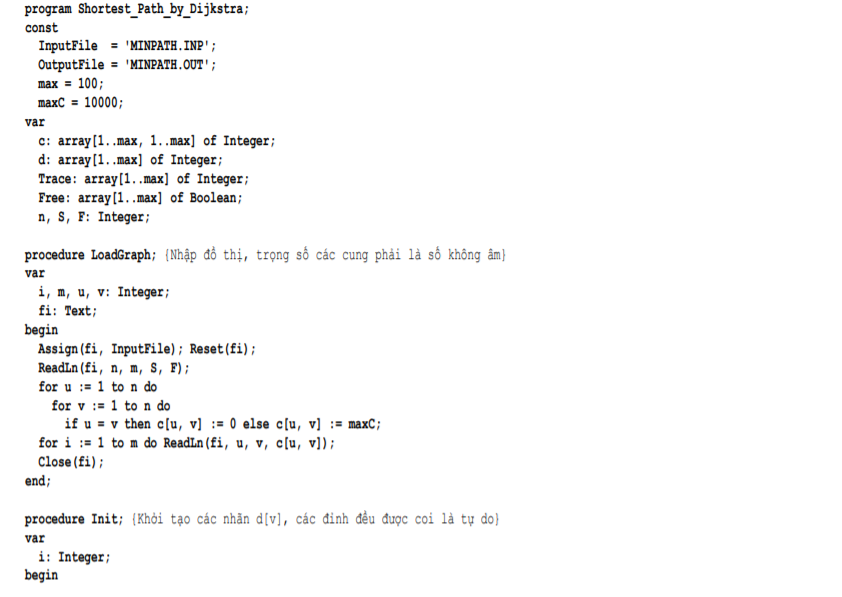
**d[v] := min(d[v], d[u] + c[u, v])**

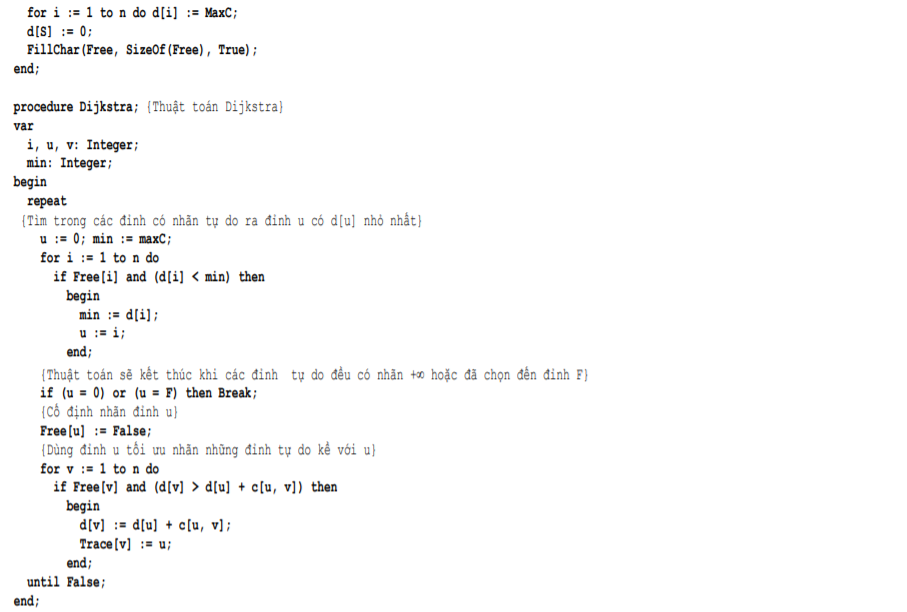
**Bước lặp sẽ kết thúc khi mà đỉnh đích F được cố định nhãn (tìm được đường đi ngắn nhất từ S tới F); hoặc tại thao tác cố định nhãn, tất cả các đỉnh tự do đều có nhãn là + (không tồn tại đường đi).**

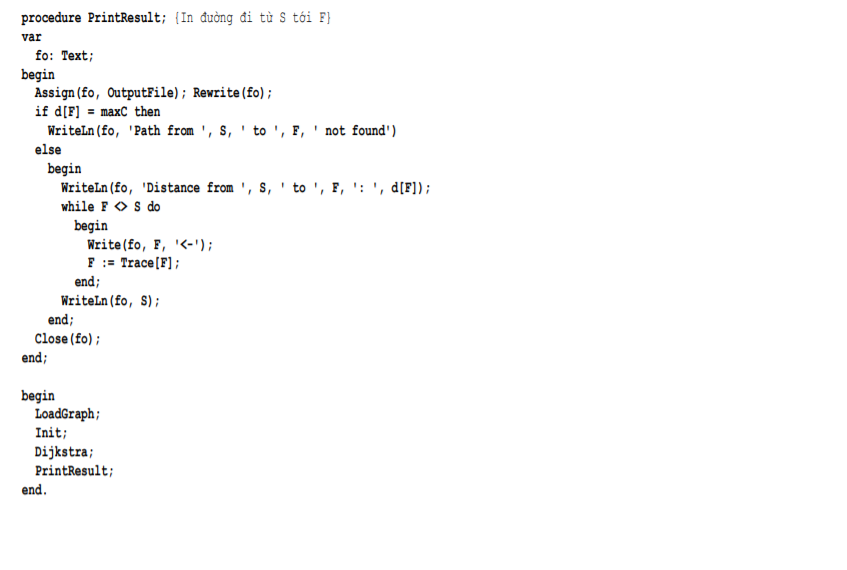
Có thể đặt câu hỏi, ở thao tác 1, tại sao định u như vậy được cố định nhãn, giả sử d[u] còn có thể tối ưu thêm được nữa thì tất phải có một đỉnh t mang nhãn tự do sao cho d[u] > d[t] + c[t, u]. Do trọng số c[t, u] không âm nên d[u] > d[t], trái với cách chọn d[u] là nhỏ nhất. Tất nhiên trong lần lặp đầu tiên thì S là định được cố định nhãn do d[S] = 0.

**Bước 3:** Kết hợp với việc lưu vết đường đi trên từng bước sửa nhãn, thông báo đường đi ngắn nhất tìm được hoặc cho biết không tồn tại đường đi (d[F] = +∞).

Chương trình: (ngôn ngữ Pascal)

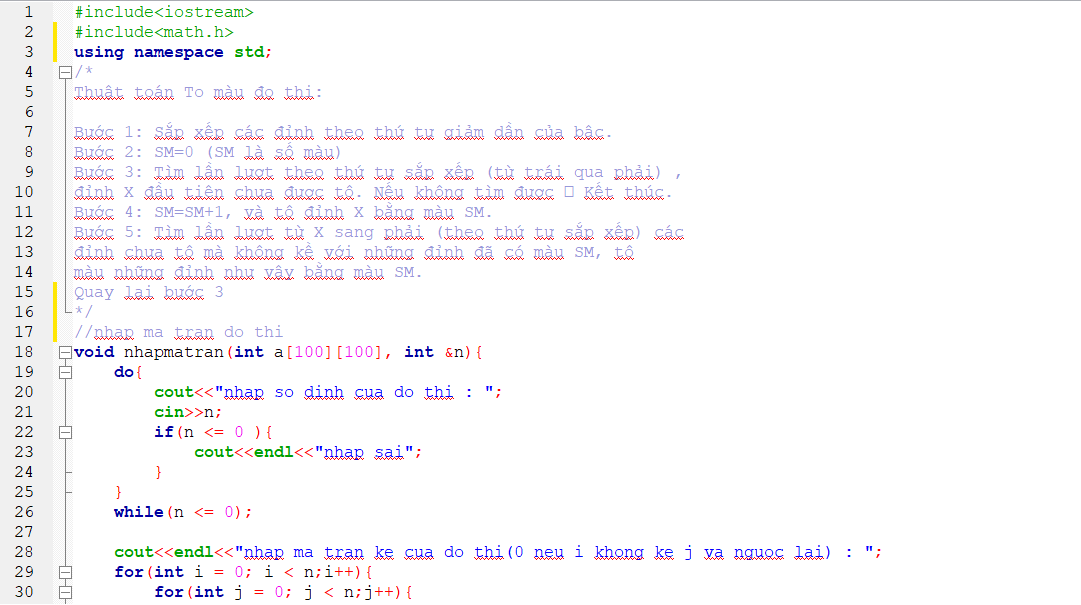


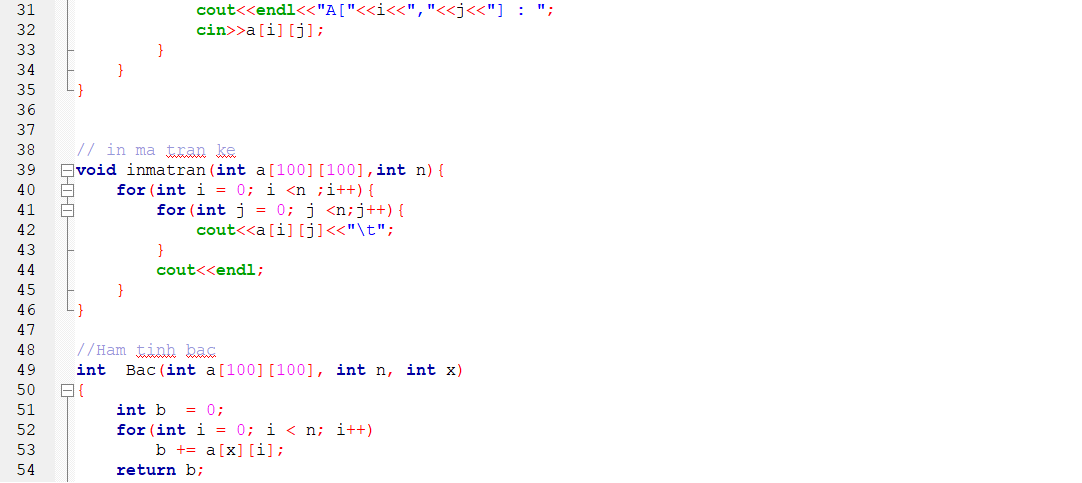


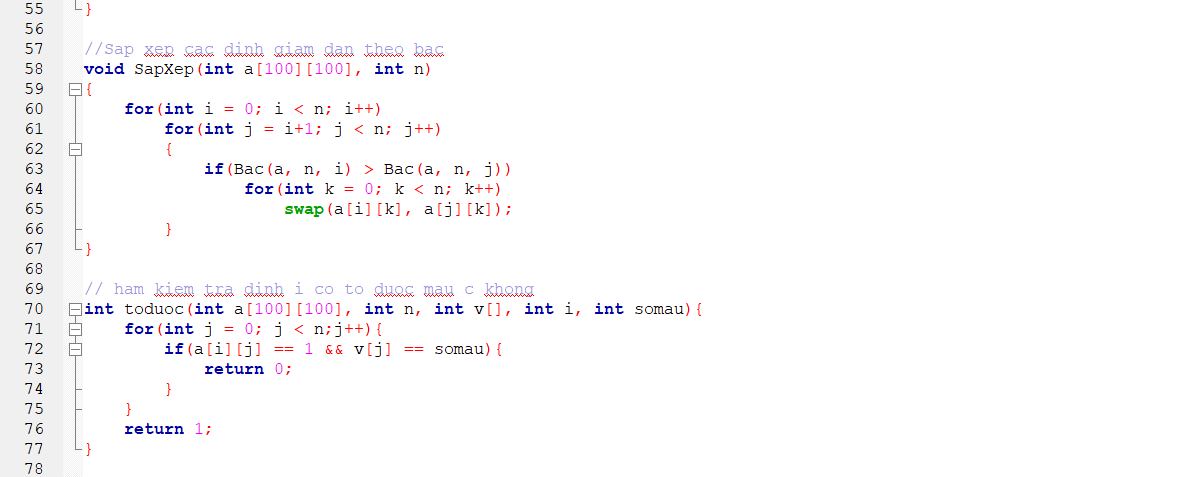


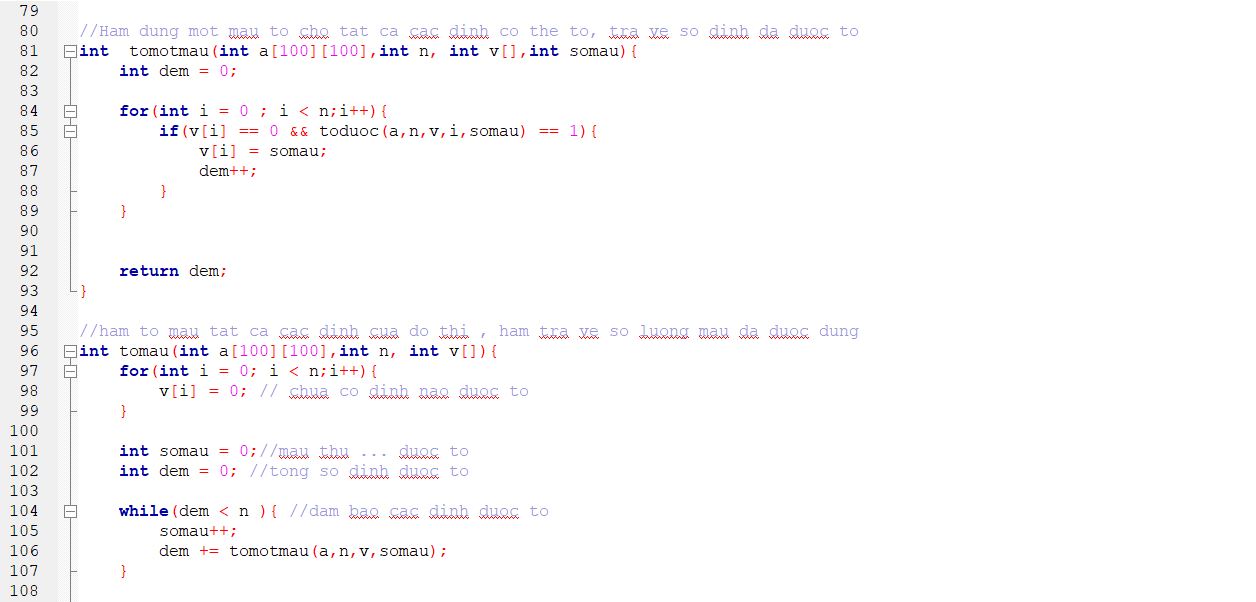
**PHẦN VI: THUẬT TOÁN TÔ MÀU ĐỒ THỊ**

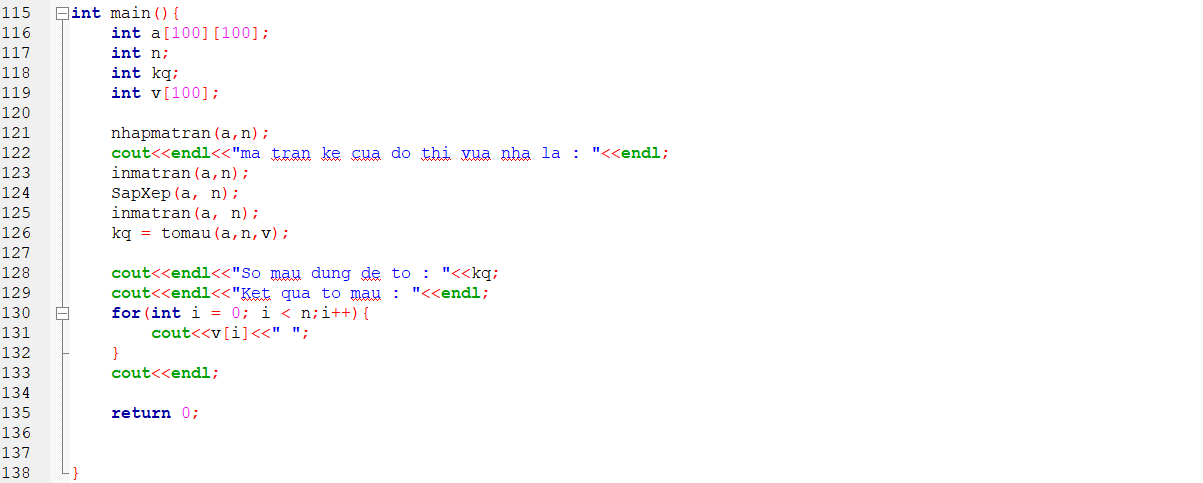
Chương trình: (ngôn ngữ C++)





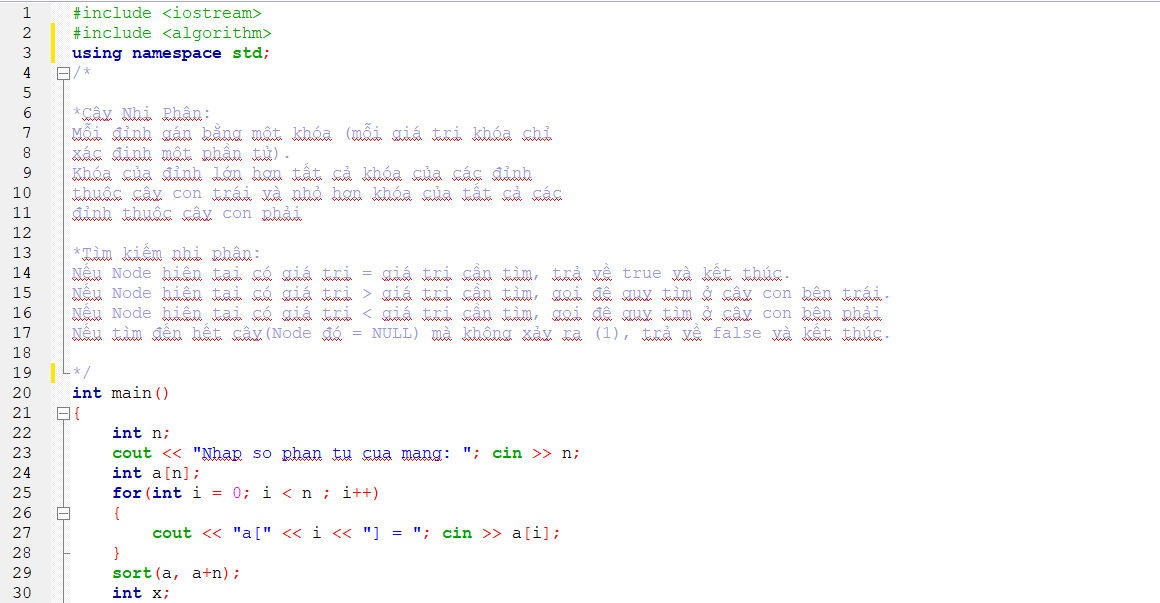


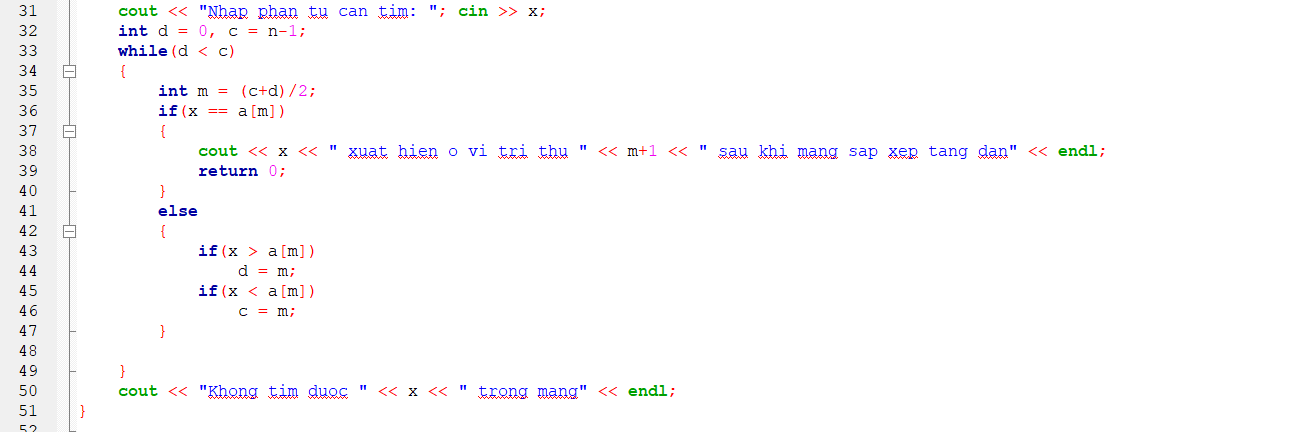




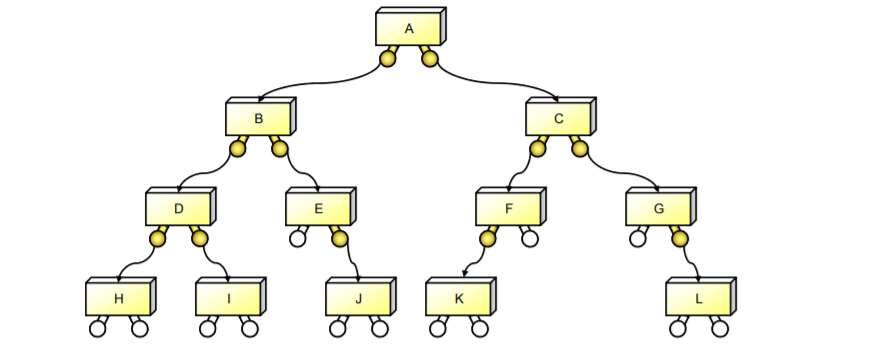
**PHẦN VII: THUẬT TOÁN TÌM KIẾM NHỊ PHÂN**

Chương trình: (ngôn ngữ C++)





**PHẦN VIII: BA PHƯƠNG PHÁP DUYỆT CÂY**

****

**1. PHÉP DUYỆT CÂY NHỊ PHÂN**

Phép xử lý các nút trên cây mà ta gọi chung là phép thăm (Visit) các nút một cách hệ thốngsao cho mỗi nút chỉ được thăm một lần gọi là phép duyệt cây.

Giả sử rằng nếu như một nút không có nút con trái (hoặc nút con phải) thì liên kết Left (Right) của nút đó được liên kết thẳng tới một nút đặc biệt mà ta gọi là NIL (hay NULL), nếu cây rỗng thì nút gốc của cây đó cũng được gán bằng NIL. Khi đó có ba cách duyệt cây hay được sử dụng:

**1.1. Duyệt theo thứ tự trước (preorder traversal)**

Trong phép duyệt theo thứ tự trước thì giá trị trong mỗi nút bắt kỳ sẽ được liệt kê trước giá trị lưu trong hai nút con của nó, có thể mô tả bằng thủ tục đệ quy sau:

procedure Visit(N); {Duyệt nhánh cây nhận N là nút gốc của nhánh đó}

begin

if N != ni1 then

begin.

<Output trường Info của nút N>

Visit(Nút con trái của N)

Visit(Nút con phải của N)

end;

Quá trình duyệt theo thứ tự trước bắt đầu băng lời gọi Visit(nút gốc).

Như cây ở Hình trên, nếu ta duyệt theo thứ tự trước thì các giá trị sẽ lần lượt được liệt kê theo thứ tự:

A B D H I E J C F K G L

**1.2. Duyệt theo thứ tự giữa (inorder traversal)**

Trong phép duyệt theo thứ tự giữa thì giá trị trong mỗi nút bắt kỳ sẽ được liệt kê sau giá trị lưu ở nút con trái và được liệt kê trước giá trị lưu ở nút con phải của nút đó, có thẻ mô tả bằng thủ tục đệ quy sau:

procedure Visit(N); {Duyệt nhánh cây nhận N là nút gốc của nhánh đó}

begin

if N!= ni1 then

begin

Visit(Nút con trái của N);

<Output trường Tnfo của nút N>

Visit(Nút con phải của N) ;

end;

end;

Quá trình duyệt theo thứ tự giữa cũng bắt đầu bằng lời gọi Visit(nút gốc).

Như cây ở Hình trên, nếu ta duyệt theo thứ tự giữa thì các giá trị sẽ lần lượt được liệt kê theo thứ tự:

H D I B E I J I A K F C G L

**1.3. Duyệt theo thứ tự sau (postorder traversal)**

Trong phép duyệt theo thứ tự sau thì giá trị trong mỗi nút bắt kỳ sẽ được liệt kê sau giá trị lưu ở hai nút con của nút đó, có thể mô tả bằng thủ tục đệ quy sau:

procadure Visit(N); (Duyệt nhánh cây nhận N là nút gốc của nhánh đó)

begin

if N!= nil then

begin

Visit(Nút con trái của N) ;

Visit(Nút con phải của N) ;

<Output trường Info của nút N>

end;

procedure visit(N);{Duyệt nhánh cây nhận N là nút gốc của nhánh đó}

begin

if N!= ni1 then

begin

Visit(Nút con trái của N) ;

Visit(Nút con phải của N) ;

<Output trường Info của nút N>

end;

end;

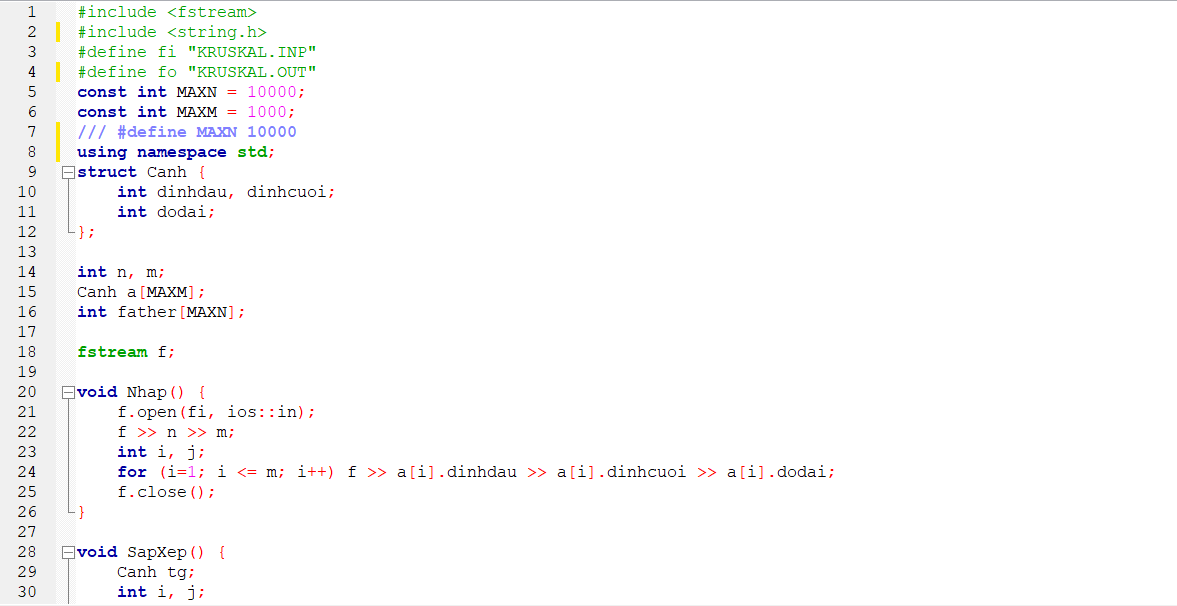
Quá trình duyệt theo thứ tự sau cũng bắt đầu bằng lời gọi Visit(nút gốc).

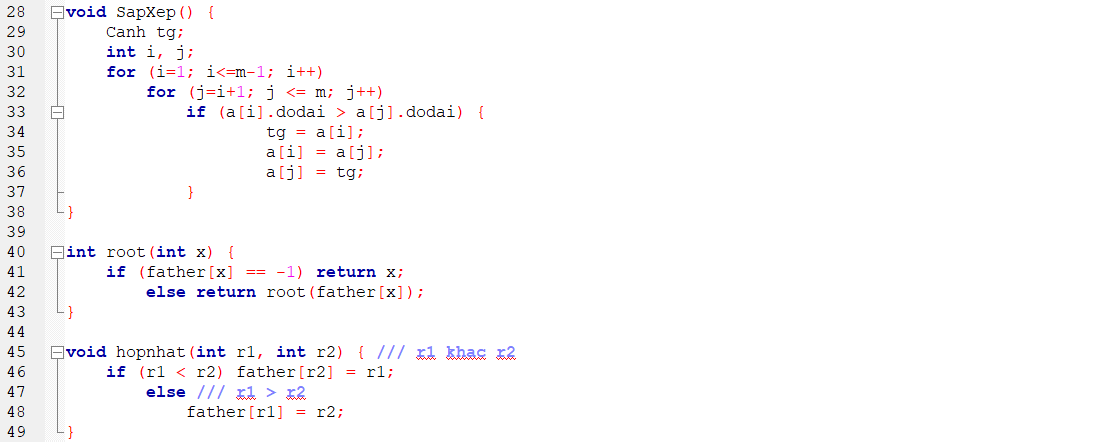
Cũng với cây ở Hình trên, nếu ta duyệt theo thứ tự sau thì các giá trị sẽ lần lượt được liệt kê theo thứ tự:

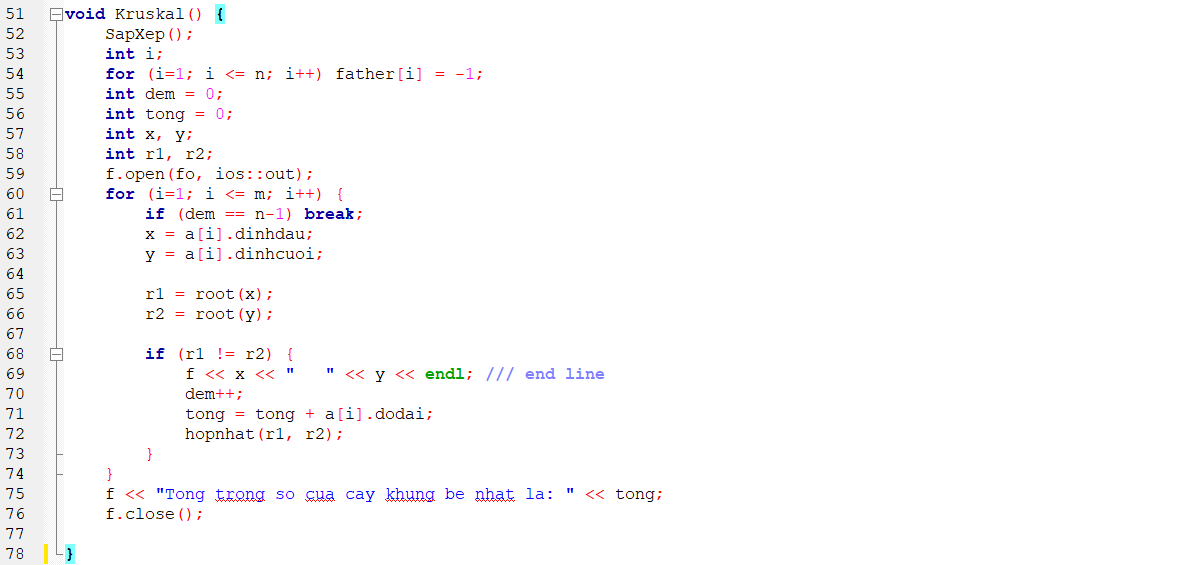
H I D J E B K F L G C A

**PHẦN IX: THUẬT TOÁN TÌM CÂY KHUNG NHỎ NHẤT**

Chương trình: (ngôn ngữ C++)









**PHẦN X: KỸ THUẬT QUAY LUI GIẢI BÀI TOÁN 8 HẬU**

**1. Bài toán tổng quát**

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:

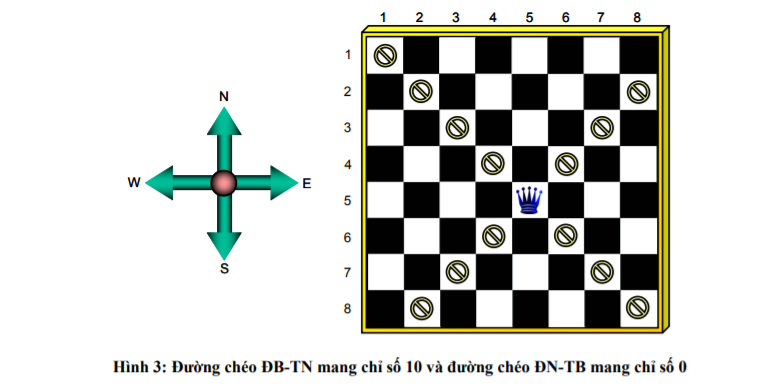


**2. Phân tích**

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 ≤ C ≤ 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 ≤ C ≤ 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.



**Cài đặt:**

**Ta có 3 mảng logic để đánh dấu:**

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

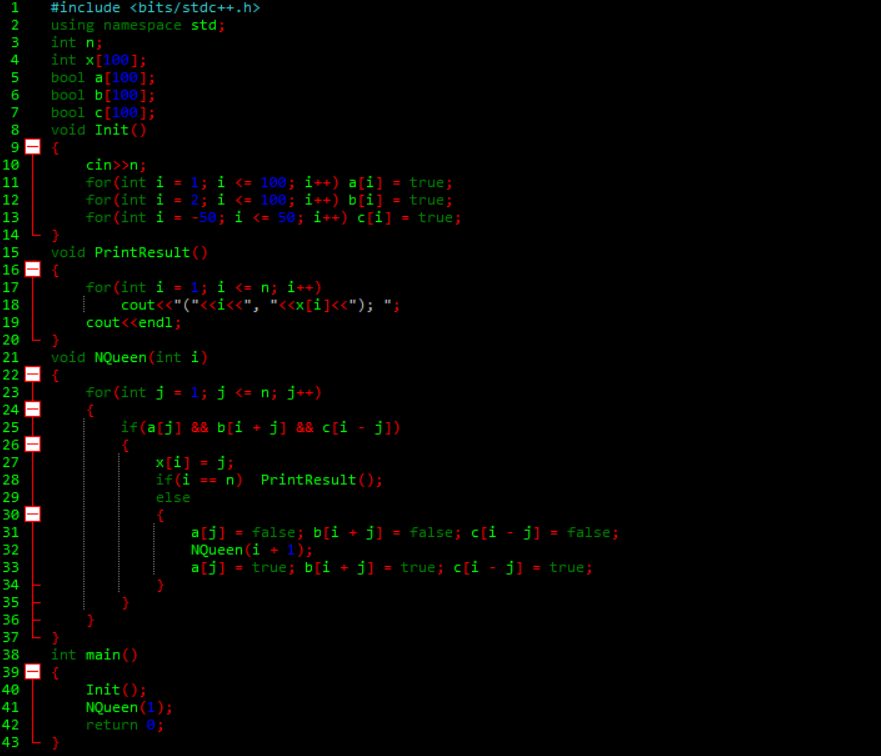
**Thuật toán quay lui:**

* 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 (aj = bi+j = cj-i = 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 (aj = bi+j = Ci-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 (aj = bi+j = ci-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

*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 đảnh 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*

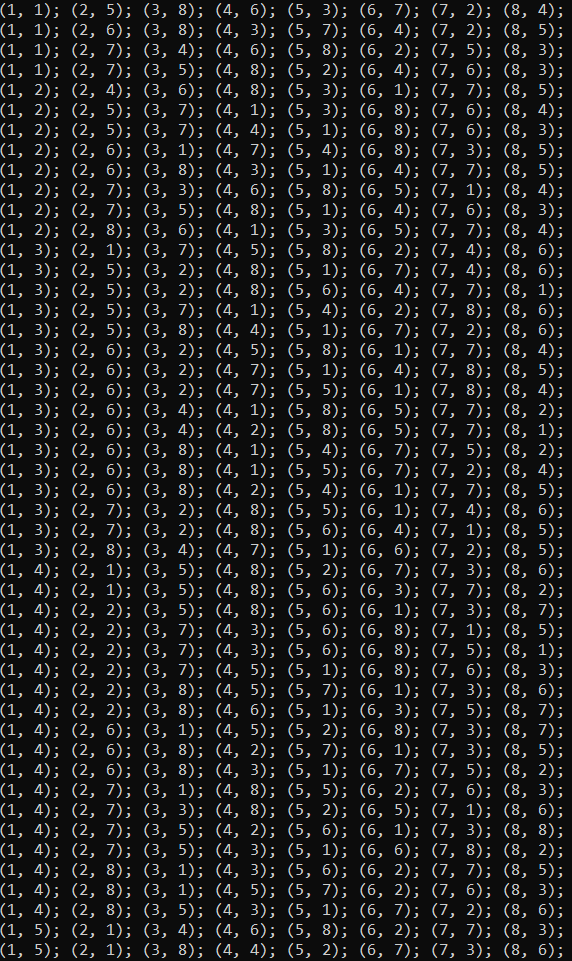
• **Input:** file văn bản QUEENS.INP chứa số nguyên dương n ≤ 12

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



**InPut: 8**

**OutPut:**

****