CHƯƠNG 3

CÁC THUẬT TOÁN TÌM KIẾM TRÊN ĐỒ THỊ VÀ ỨNG DỤNG

(tuần 3 & 4: Tổng cộng có 4 tiết lý thuyết và 4 tiết hướng dẫn bài tập/thực hành)

1. TÌM KIẾM THEO CHIỀU SÂU TRÊN ĐỒ THỊ

Ý tưởng chính của thuật toán có thể trình bày như sau. Ta sẽ bắt đầu tìm kiếm từ một đỉnh v_0 nào đó của đồ thị. Sau đó chọn u là một đỉnh tuỳ ý kề với v_0 và lặp lại quá trình đối với u. Ở bước tổng quát, giả sử ta đang xét đỉnh v. Nếu như trong số các đỉnh kề với v tìm được đỉnh w là chưa được xét thì ta sẽ xét đỉnh này (nó sẽ trở thành đã xét) và bắt đầu từ nó ta sẽ bắt đầu quá trình tìm kiếm còn nếu như không còn đỉnh nào kề với v là chưa xét thì ta nói rằng đỉnh này đã duyệt xong và quay trở lại tiếp tục tìm kiếm từ đỉnh mà trước đó ta đến được đỉnh v (nếu $v=v_0$, thì kết thúc tìm kiếm). Có thể nói nôm na là tìm kiếm theo chiều sâu bắt đầu từ đỉnh v được thực hiện trên cơ sở tìm kiếm theo chiều sâu từ tất cả các đỉnh chưa xét kề với v. Quá trình này có thể mô tả bởi thủ tục đệ qui sau đây

```
void DFS(v);
(*tim kiem theo chieu sau bat dau tu dinh v; cac bien chuaxet, Ke la bien toan cuc*)
{
        tham_dinh(v);
        chuaxet[v]=0;
        for u∈ ke(v)
        If (chuaxet[u]) DFS(u);
} (*dinh v da duyet xong*)
```

Khi đó, tìm kiếm theo chiều sâu trên đồ thị được thực hiện nhờ thuật toán sau:

```
void main()
{
        (*Initialization*)
        for v∈ V chuaxet[v]=1;
        for v ∈ V
        if (chuaxet[v]) DFS(v);
}
```

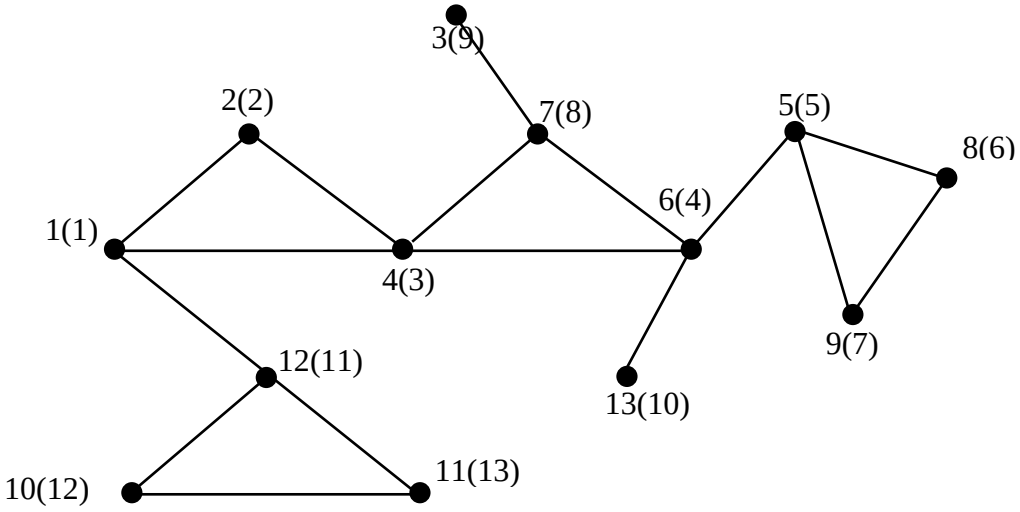
Rõ ràng lệnh gọi DFS(v) sẽ cho phép đến thăm tất cả các đỉnh thuộc cùng thành phần liên thông với đỉnh v, bởi vì sau khi thăm đỉnh là lệnh gọi đến thủ tục DFS đối với tất cả các đỉnh kề với nó. Mặt khác, do mỗi khi thăm đỉnh v xong, biến chuaxet[v] được đặt lại giá trị

false nên mỗi đỉnh sẽ được thăm đúng một lần. Thuật toán lần lượt sẽ tiến hành tìm kiếm từ các đỉnh chưa được thăm , vì vậy, nó sẽ xét qua tất cả các đỉnh của đồ thị (không nhất thiết phải là liên thông).

Để đánh giá độ phức tạp tính toán của thủ tục, trước hết nhận thấy rằng số phép toán cần thực hiện trong hai chu trình của thuật toán (hai vòng for ở chương trình chính) là cỡ n. Thủ tục DFS phải thực hiện không quá n lần. Tổng số phép toán cần phaỉ thực hiện trong các thủ tục này là O(n+m), do trong các thủ tục này ta phải xét qua tất cả các cạnh và các đỉnh của đồ thị. Vậy độ phức tạp tính toán của thuật toán là O(n+m).

Thí dụ 1.

Xét đồ thị cho trong hình 1 gồm 13 đỉnh, các đỉnh được đánh số từ 1 đến 13 như sau:



Hình 1

Chỉ số mới (trong ngoặc) của các đỉnh được đánh lại theo thứ tự chúng được thăm theo thuật toán tìm kiếm theo chiều sâu.

Thuật toán tìm kiếm theo chiều sâu trên đồ thị vô hướng trình bày ở trên dễ dàng có thể mô tả lại cho đồ thị có hướng. Trong trường hợp đồ thị có hướng, thủ tcụ DFS(v) sẽ cho phép thăm tất cả các đỉnh u nào mà từ v có đường đi đến u. Độ phức tạp tính toán của htuật toán là O(n+m).

2. TÌM KIẾM THEO CHIỀU RỘNG TRÊN ĐỒ THỊ

Để ý rằng trong thuật toán tìm kiếm theo chiều sâu đỉnh được thăm càng muộn sẽ càng sớm trở thành đã duyệt xong. Điều đó là hệ quả tất yếu của việc các đỉnh được thăm sẽ được kết nạp vào trong ngăn xếp (STACK). Tìm kiếm theo chiều rộng trên đồ thị, nếu nói

một cách ngắn gọn, được xây dựng trên cơ sở thay thế ngăn xếp (STACK) bởi hàng đợi (QUEUE). Với sự cải biên như vậy, đỉnh được thăm càng sớm sẽ càng sớm trở thành đã duyệt xong (tức là càng sớm dời khỏi hàng đợi). Một đỉnh sẽ trở thành đã duyệt xong ngay sau khi ta xét xong tất cả các đỉnh kề (chưa được thăm) với nó. Thủ tục có thể mô tả như sau:

```
void BFS(v);
(*Tim kiem theo chieu rong bat dau tu dinh v, cac bien chuaxet, Ke la bien cuc
bo*)
{
        QUEUE=∅;
        QUEUE  ⇐v; (*ket qua nap vao QUEUE*)
        Chuaxet[v]=0;
        While (QUEUE<>∅)
        {
                P ⇐QUEUE; (*lay p tu QUEUE:*)
                Tham_dinh(p);
                for u ∈ Ke(v)
                If (chuaxet[u])
                {
                        QUEUE ⇐u;
                        chuaxet[u]=0;
                }
        }
}
```

Khi đó, tìm kiếm theo chiều rộng trên đồ thị được thực hiện nhờ thuật toán sau:

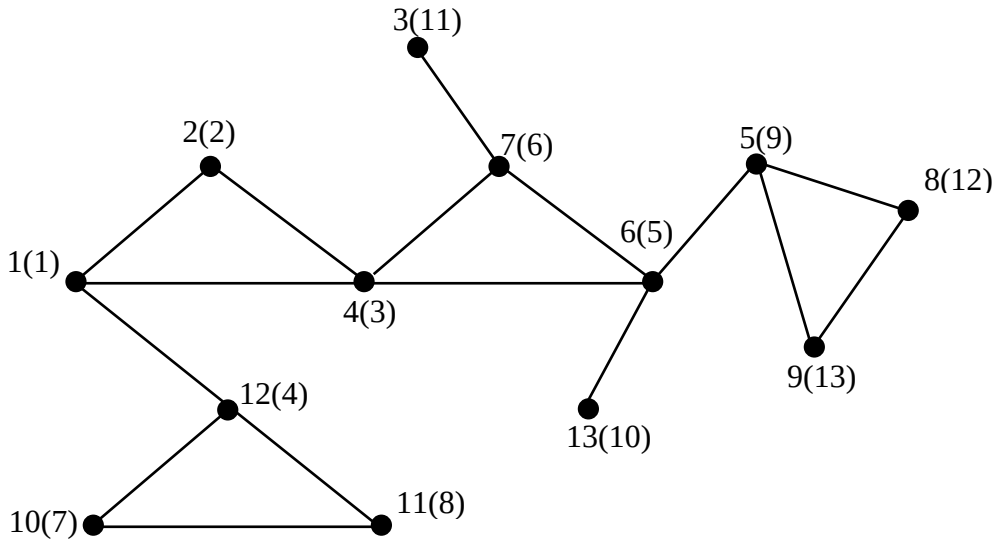
```
void main()
{
        (*Initialization*)
        for f ∈ V do chuaxet[v]=1;
        for v ∈ V
        if (chuaxet[v]) BFS(v);
}
```

Lập luận tương tự như trong thủ tục tìm kiếm theo chiều sâu, có thể chỉ ra được rằng lệnh gọi BFS(v) sẽ cho phép thăm đến tất cả các đỉnh thuộc cùng thành phần liên thông với đỉnh

v, và mỗi đỉnh của đồ thị sẽ được thăm đúng một lần. Độ phức tạp tính toán của thuật toán là O(m+n).

Thí dụ 2.

Xét đồ thị xét trong hình 2. Thứ tự thăm đỉnh của đồ thị theo thuật toán tìm kiếm theo chiều rộng được ghi trong ngoặc.



Hình 2

3. TÌM ĐƯỜNG ĐI VÀ KIỂM TRA TÍNH LIÊN THÔNG

Trong mục này ta xét ứng dụng các thuật toán tìm kiếm mô tả trong các mục trước vào việc giải bài toán cơ bản trên đồ thị: bài toán về tìm đường đi và bài toán về xác định tính liên thông của đô thị.

a) Bài toán tìm đường đi giữa hai đỉnh:

Giả sử s và t là hai đỉnh nào đó của đồ thị. Hãy tìm đường đi từ s đến t.

Như trên đã phân tích, thủ tục DFS(s) (BFS(s)) sẽ cho thăm tất cả các đỉnh thuộc cùng một thành phần liên thông với s. vì vậy, sau khi thực hiện xong thủ tục, nếu chuaxet[t]=true(1), thì điều đó có nghĩa là không có đường đi từ s đến t, còn nếu chuaxet[t]=false(0) thì t thuộc cùng thành phần liên thông với s, hay nói một cách khác: tồn tại đường đi từ s đến t. Trong trường hợp tồn tại đường đi, để ghi nhận đường đi, ta dùng thêm biểu thức Truoc[v] để ghi nhận đỉnh đi trước đỉnh v trong đường đi tìm kiếm tứ s đến v. Khi đó, đối với thủ tục DFS(v) cần sửa đổi câu lệnh ì trong nó như sau:

```
If (chuaxet[u])
{
        Truoc[u]=v;
```

```
            DFS(u);
        }
```

Còn đối với thủ tục BFS(v) cần sửa đổi câu lện if trong nó như sau:

```
    If (chuaxet [u])
    {
        QUEUE ⇐u;
        chuaxet[u]=0;
        Truoc[u]=p;
    }
```

Chú ý:

Đường đi tìm được theo thuật toán tìm kiếm theo chiều rộng là đường đi ngắn nhất (theo số cạnh) từ s đến t. Điều này suy trực tiếp từ thứ tự thăm đỉnh theo thuật toán tìm kiếm theo chiều rộng.

b) Tìm các thành phần liên thông của đồ thị:

Hãy cho biết đồ thị gồm bao nhiêu thành phần liên thông và từng thành phần liên thông của nó là gồm những đỉnh nào.

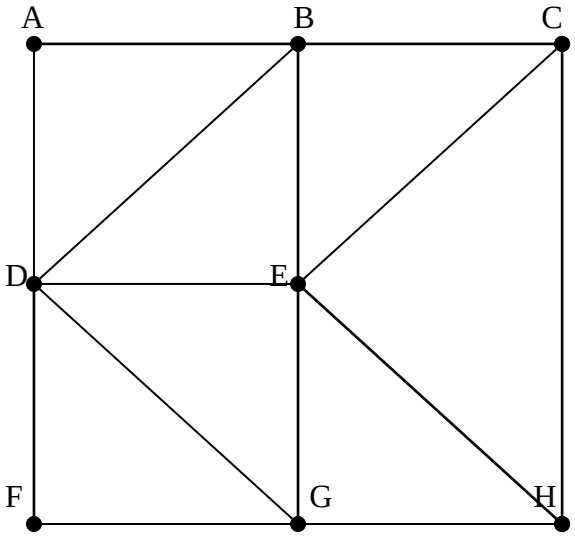
Do thủ tục DFS(v) (BFS(s)) cho phép thăm tất cả các đỉnh thuộc cùng một thành phần liên thông với s, nên số thành phần liên thông cỉa đồ thị bằng số lần gọi đến thủ tục này. Vấn đề còn lại là cách ghi nhận các đỉnh trong từng thành phần liên thông. Ta dùng thêm biến Index[v] đê ghi nhận chỉ số của thành phần liên thông chứa đỉnh v, và dùng thêm biến Inconnect để đếm số thành phần liên thông (biến này cần khởi tạo giá trị 0). Thủ tục Tham_dinh(v) trong các thủ tục DFS(v) và BFS(v) có nhiệm vụ gán: Index[v]=connect, còn câu lện if trong các chương trình chính gọi đến các thủ tục này cần được sửa lại như sau:

```
        Inconnect=0;
        If (chuaxet[v] )
        {
            Inconnect=Inconnect+1;
            DFS(v); (*BFS(v)*)
        }
```

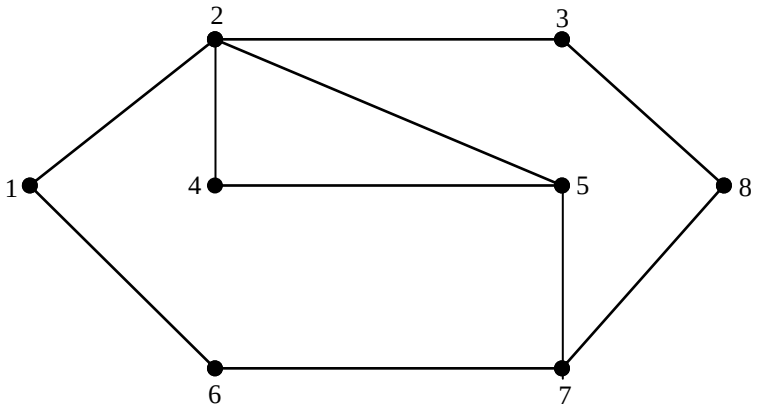
Kết thúc vòng lặp thứ hai trong chương trình chính, Inconnect cho số thành phần liên thông của đồ thị, còn biến mảng Index[v], v ∈ V cho phép liệt kê các đỉnh thuộc cùng một thành phần liên thông.

Bài tập lý thuyết:

3-1.Hãy liệt kê các đỉnh của đồ thị được duyệt theo phương pháp tìm kiếm theo chiều sâu, tìm kiếm theo chiều rộng. Tìm đường đi từ đỉnh A đến đỉnh H.



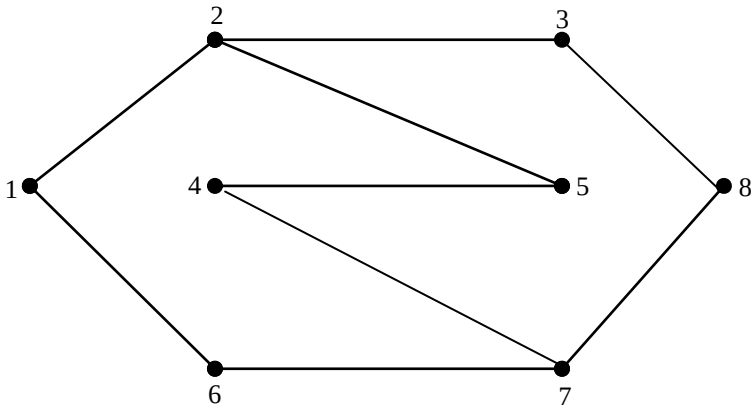
3-2. Cho đồ thị vô hướng liên thông G như hình vẽ bên.



a.Hãy liệt kê danh sách các đỉnh của G theo thuật toán tìm kiếm theo chiều sâu (DFS), theo thuật toán tìm kiếm theo chiều rộng (BFS) bắt đầu từ đỉnh 1.

b.Hãy tìm một đường đi từ đỉnh 1 đến đỉnh 6 trên G theo thuật toán DFS và từ đỉnh 1 đến đỉnh 7 theo thuật toán BFS.

3-3. Cho đồ thị vô hướng liên thông G như hình vẽ bên.



a.Hãy biểu diễn đồ thị G bằng ma trận kề.

b.Hãy liệt kê danh sách các đỉnh của G theo thuật toán tìm kiếm theo chiều sâu (DFS), theo thuật toán tìm kiếm theo chiều rộng (BFS) bắt đầu từ đỉnh 1.

Bài tập thực hành

3-4.Một khóa học gồm N môn học, môn học i phải học trong ti ngày. Giữa các môn học có mối quan hệ trước/sau: có môn học chỉ học được sau khi đã học một số môn học khác. Mối quan hệ đó được thể hiện bởi một mảng hai chiều A[i, j];

i, j = 1, …, N trong đó A[i, j] = 1/0 và A[i, i] bằng 0 với mọi i, A[i,j] = 1 khi và chỉ khi môn học i phải được dạy xong trước khi học môn j (ngày kết thúc môn i phải trước ngày bắt đầu môn j). Môn học i phải dạy trước môn học j nếu có một dãy môn học i_1, i_2, …, ik sao cho a[it, it_{+1}] = 1, 1 <= t <= k-1, i_1=i và ik=j. Nếu có một nhóm các môn học từng đôi một không có quan hệ trước/sau thì trong mỗi ngày, về nguyên tắc, ta có thể học đồng thời tất cả những môn học này (nếu không vi phạm quan hệ với các môn học khác). Mảng A[i, j] được gọi là bế tắc nếu có một dãy các môn học i_1, i_2,…, ik, k > 1, mà môn i_1 phải dạy trước môn i_2, môn i_2 phải dạy trước môn i_3, …, môn ik_{-1} phải dạy trước môn ik, môn ik phải dạy trước môn i_1.

Hãy viết chương trình với tên KT3.CPP làm các việc sau:

Hãy xét xem mảng A có bế tắc hay không.

Nếu mảng A không bế tắc, hãy tính xem khóa học có thể kết thúc trong thời gian nhanh nhất là bao nhiêu ngày.

Theo các học bảo đảm thời gian hoàn thành ngắn nhất ở câu 2, hãy tính xem một học sinh trong quá trình học phải học đồng thời trong một ngày nhiều nhất bao nhiêu môn.

Dữ liệu vào được cho bởi file text có tên MH.DAT trong đó số N ghi ở dòng thứ nhất, trong nhóm N dòng tiếp theo, dòng thứ i ghi N số A[i, 1], …, A[i, N] dòng cuối cùng ghi N số nguyên dượng ti không lớn hơn 30, 1 <= i <= N; N <= 30.

Kết quả ghi ra file TKB.DAT như sau: dòng thứ nhất ghi số 1/0 tùy theo mảng A bế tắc / không bế tắc. Nếu dòng thứ nhất ghi số 0, ta mới ghi tiếp kết quả câu 2 và 3.

Kết quả câu 2 ghi tiếp vào file TKB.DAT N+1 dòng như sau: dòng dầu ghi số T là số ngày tối thiểu có thể hoàn thành khóa học, tiếp theo là N dòng trong đó dòng thứ i ghi 2 số X, Y với ý nghĩa môn học thứ i học từ ngày thứ X đến ngày thứ Y (chú ý rằng$Y–X=t_{i–1}$).

Kết quả câu 3 ghi tiếp vào file TKB.DAT như sau: dòng thứ nhất ghi 2 số Z, W với ý nghĩa trong ngày Z phải học W môn (W là số nhiều nhất các môn học phải học đồng thời trong một ngày), tiếp theo là một dòng ghi tên các môn học phải học đồng thời trong ngày Z.

Trong các câu 2 và 3, có thể có nhiều lời giải tương đương chỉ cầu đưa ra một lời giải.

Ví dụ 1

MH.DAT	TKB.DAT
4	1

0 1 0 0 0 0 1 0 0 0 0 1 1 0 0 0 1 1 1 1	

Ví dụ 2

MH.DAT	TKB.DAT
7	0
0 1 0 0 0 0 0	22
0 0 0 1 0 0 0	1 2
0 0 0 1 0 0 0	3 4
0 0 0 0 1 1 0	1 8
0 0 0 0 0 0 0	9 12
0 0 0 0 0 0 1	13 22
0 0 0 0 0 0 0	13 14
2 2 8 4 10 2 3	15 17
	1 2
	1 3

3-5. Cho một mạng N (N <= 20) máy tính được đánh số từ 1 đến N. Sơ đồ mạng được cho bởi hệ gồm M kênh (đoạn) nối trực tiếp giữa một số cặp máy trong mạng, m kênh tương ứng với m cặp. Cho biết chi phí truyền 1 đơn vị thông tin theo mỗi kênh của mạng.

Người ta cần chuyển một bức thông điệp từ máy s đến máy t. Để đảm bảo an toàn, người ta chuyển bức thông điện này theo hai đường truyền tin khác nhau (tức không có kênh nào) của mạng được sử dụng trong cả hai đường truyền tin; cho phép hai đường truyền tin cùng đi qua một số máy tính). Chi phí của một đường truyền được hiểu là tổng chi phí trên các kênh của nó. Đơn giá đường truyền từ máy s sang máy t được tính như sau:

Với hai máy s và t, cùng bức thông điệp có độ dài là 1 đơn vị thông tin, đơn giá truyền cho cặp (s, t) được tính bằng tổng chi phí chuyển thông điệp an toàn (bằng tổng chi phí của hai đường truyền tin) là nhỏ nhất.

Người ta mong muốn mạng máy tính (mạng truyền tin nói trên thỏa mãn tính chất an toàn theo nghĩa là từ một máy bất kỳ luôn truyền được (một cách an toàn) thông điệp tới một

máy bất kỳ khác. Khi một mạng an toàn, người ta tính được đơn giá của mạng là tổng đơn giá mọi đường truyền từ một máy bất kỳ tới một máy bất kỳ khác.

Ma trận đơn giá của mạng là mảng hai chiều A có N dòng và N cột, mà giá trị phần tử A[i,j] chính là đơn giá từ máy i sang máy j.

3-6.Truyền tin trên mạng

Có một nhóm gồm N lập trình viên được đánh số từ 1 tới N, một số người trong họ có biết địa chỉ email của nhau. Khi biết một thông tin nào mới họ gửi thông tin đó cho nhau. Bạn là một người rất quan trọng và bạn biết tất cả các mối quan hệ của họ cũng như bạn có một thông tin rất đặc biệt mà muốn cho tất cả họ đều biết. Hãy lập trình chỉ ra một số ít nhất các lập trình viên cần cho họ biết thông tin sao cho những người đó có thể thông báo cho tất cả những người còn lại thông tin của bạn.

Dữ liệu cho trong file văn bản với tên INFOR.INP trong đó dòng đầu chức số N (N <= 1000), dòng thứ I trong N dòng tiếp theo chứa danh sách các lập trình viên mà người I biết địa chỉ email của họ. Nếu người thứ I không biết địa chỉ của bất cứ ai thì dòng này là dòng trống.

Kết quả ghi ra file văn bản với tên INFOR.OUT trong đó dòng đầu ghi số K là số người cần cho họ biết thông tin. Dòng thứ hai ghi ra chỉ số của những người đó.

Ví dụ:

INFOR.INP	INFOR.OUT
6 2 3 1 1 5 4	3 1 4 6

Cài đặt một số thuật toán căn bản quan trọng

1.Tìm kiếm theo chiều sâu trên đồ thị

(toàn văn chương trình)

DFS.CPP

```
1.  #include <conio.h>
2.  #include <iostream.h>
3.  #include <stdio.h>
4.  int daxet[100];
5.  int a[100][100];
6.  int n;
7.  void tham_dinh(int v)
8.  {
9.  cout<<v<<" ";
10. }
11. void dfs(int v)
12. {
13. int u;
14. tham_dinh(v);
15. daxet[v]=1;
16. for (u=1;u<=n;u++)
17. if (a[v][u]!=0 && !daxet[u])
18. dfs(u);
19. }
20. void readfile()
21. {
22. FILE *f;
23. int u,v;
24. clrscr();
25. f=fopen("d:\\dothi\\dfs.inp","rt");
26. fscanf(f,"%d",&n);
27. for (u=1;u<=n;u++)
28. for (v=1;v<=n;v++)
29. fscanf(f,"%d",&a[u][v]);
30. fclose(f);
```

31. }

32. void find()

33. {

34. int v;

35. for (v=1;v<=n;v++)

36. daxet[v]=0;

37. for (v=1;v<=n;v++)

38. if (!daxet[v])

39. dfs(v);

40. getch();

41. }

42. void main()

43. {

44. readfile();

45. find();

46. }

Dfs.inp

13

0 1 0 1 0 0 0 0 0 0 0 1 0

1 0 0 1 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 1 0 0 0 0 0 0

1 1 0 0 0 1 1 0 0 0 0 0 0

0 0 0 0 0 1 0 1 1 0 0 0 0

0 0 0 1 1 0 1 0 0 0 0 0 1

0 0 1 1 0 1 0 0 0 0 0 0 0

0 0 0 0 1 0 0 0 1 0 0 0 0

0 0 0 0 1 0 0 1 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 1 1 0

0 0 0 0 0 0 0 0 0 1 0 1 0

1 0 0 0 0 0 0 0 0 1 1 0 0

0 0 0 0 0 1 0 0 0 0 0 0 0

2.Tìm kiếm theo chiều rộng trên đồ thị

(toàn văn chương trình)

BFS.CPP

```
1.  #include <conio.h>
2.  #include <iostream.h>
3.  #include <stdio.h>
4.  int daxet[100];
5.  int a[100][100];
6.  int Queue[100];
7.  int n;
8.  void tham_dinh(int v)
9.  {
10. cout<<v<<" ";
11. }
12. void BFS(int u)
13. {
14. int w,v;
15. int dauQ,cuoiQ;
16. dauQ=1;cuoiQ=1;
17. Queue[dauQ]=u;
18. daxet[u]=1;
19. while (dauQ<=cuoiQ)
20. {
21. v=Queue[dauQ];
22. dauQ=dauQ+1;
23. tham_dinh(v);
24. for (w=1;w<=n;w++)
25. if (a[v][w]==1 && !daxet[w])
26. {
27. cuoiQ=cuoiQ+1;
28. Queue[cuoiQ]=w;
29. daxet[w]=1;
30. }
31. }
```

```
32. }
33. void find()
34. {
35. int v;
36. for (v=1;v<=n;v++)
37. daxet[v]=0;
38. for (v=1;v<=n;v++)
39. if (!daxet[v])
40. BFS(v);
41. getch();
42. }
43. void readfile()
44. {
45. FILE *f;
46. int u,v;
47. clrscr();
48. f=fopen("d:\\ dothi\\bfs.inp","rt");
49. fscanf(f,"%d",&n);
50. for (v=1;v<=n;v++)
51. for (u=1;u<=n;u++)
52. fscanf(f,"%d",&a[u][v]);
53. fclose(f);
54. }
55. void main()
56. {
57. readfile();
58. find();
59. }
```

Bfs.inp

13

0 1 0 1 0 0 0 0 0 0 0 1 0

1 0 0 1 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 1 0 0 0 0 0 0

1 1 0 0 0 1 1 0 0 0 0 0 0

0 0 0 0 0 1 0 1 1 0 0 0 0

0 0 0 1 1 0 1 0 0 0 0 0 1

0 0 1 1 0 1 0 0 0 0 0 0 0

0 0 0 0 1 0 0 0 1 0 0 0 0

0 0 0 0 1 0 0 1 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 1 1 0

0 0 0 0 0 0 0 0 0 1 0 1 0

1 0 0 0 0 0 0 0 0 1 1 0 0

0 0 0 0 0 1 0 0 0 0 0 0 0

3.Tim đường đi giữa 2 đỉnh

(toàn văn chương trình)

PATHDFS.CPP

```
1.  #include<iostream.h>
2.  #include<conio.h>
3.  #include<stdio.h>
4.  int daxet[100];
5.  int a[100][100];
6.  int truoc[100];
7.  int n,s,t;
8.  void dfs(int v)
9.  {
10. int u;
11. daxet[v]=1;
12. for (u=1;u<=n;u++)
13. if (a[v][u]!=0 && !daxet[u])
14. {
15. dfs(u);
16. truoc[u]=v;
17. }
18. }
19. void readfile()
20. {
21. FILE *f;
22. int u,v;
23. clrscr();
24. f=fopen("d:\\ dothi\\path.inp","rt");
25. fscanf(f,"%d",&n);
26. cout<<"dinh s :";  cin>>s;
27. cout<<"dinh t :";  cin>>t;
28. for (u=1;u<=n;u++)
29. for (v=1;v<=n;v++)
30. fscanf(f,"%d",&a[u][v]);
31. fclose(f);
```

```
}

void print_way(int i)
{
if (i!=s)
{
print_way(truoc[i]);
cout<<"-->"<<i;
}
}
void main()
{
readfile();
int v;
for (v=1;v<=n;v++)
daxet[v]=0;
dfs(s);
if (daxet[t]==1)
{
cout<<s<<" ";
print_way(t);
}
else
cout<<" khong ton tai duong di";
getch();
}
```

Path.inp

13

0 1 0 1 0 0 0 0 0 0 0 1 0

1 0 0 1 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 1 0 0 0 0 0 0

1 1 0 0 0 1 1 0 0 0 0 0 0

0 0 0 0 0 1 0 1 1 0 0 0 0

0 0 0 1 1 0 1 0 0 0 0 0 1

0 0 1 1 0 1 0 0 0 0 0 0 0

0 0 0 0 1 0 0 0 1 0 0 0 0

0 0 0 0 1 0 0 1 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 1 1 0

0 0 0 0 0 0 0 0 0 1 0 1 0

1 0 0 0 0 0 0 0 0 1 1 0 0

0 0 0 0 0 1 0 0 0 0 0 0 0

4.Kiểm tra tính liên thông/xác định thành phần liên thông của đồ thị.

(toàn văn chương trình)

Lien thong

```
1.  #include <conio.h>
2.  #include <iostream.h>
3.  #include <stdio.h>
4.  int  daxet[100];
5.  int a[100][100];
6.  int lienthong[100];
7.  int n,i,j,connect;
8.  void  tham_dinh(int u)
9.  {
10. lienthong[u]=connect;
11. }
12. void dfs(int v)
13. {
14. int u;
15. tham_dinh(v);
16. daxet[v]=1;
17. for (u=1;u<=n;u++)
18. if (a[v][u]==1 && !daxet[u])
19. dfs(u);
20. }
21. void find()
22. {
23. int v;
24. for (v=1;v<=n;v++)
25. daxet[v]=0;
26. connect=0;
27. for (v=1;v<=n;v++)
28. if (!daxet[v])
29. {
30. connect++;
31. dfs(v);
```

```
}
}
void readfile()
{
FILE *f;
int  u,v;
clrscr();
f=fopen("d:\\ dothi\\li_thong.inp","rt");
fscanf(f,"%d",&n);
for (v=1;v<=n;v++)
for (u=1;u<=n;u++)
fscanf(f,"%d",&a[u][v]);
fclose(f);
}
void main()
{
readfile();
find();
if (connect==1)
cout<<"do thi lien thong";
else
{
cout<<"do thi gom "<<connect<<" cac thanh phan lien thong sau day";
for (i=1;i<=connect;i++)
{
for (j=1;j<=n;j++)
if (lienthong[j]==i)
cout<<j<<"   ";
cout<<endl;
}
}
getch();
}
```

Li_thong.inp

14

0 0 0 0 0 1 0 1 0 0 1 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 1 0 0 0 0

0 0 0 0 1 0 1 0 1 0 0 0 0 0

0 0 0 1 0 0 1 0 0 0 0 0 0 0

1 0 0 0 0 0 0 1 0 0 1 1 0 0

0 0 0 1 1 0 0 0 1 0 0 0 0 1

1 0 0 0 0 1 0 0 0 0 1 0 1 0

0 0 0 1 0 0 1 0 0 0 0 0 0 0

0 0 1 0 0 0 0 0 0 0 0 0 0 0

1 0 0 0 0 1 0 1 0 0 0 0 0 0

0 0 0 0 0 1 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 1 0 0 0 0 0 0

0 0 0 0 0 0 1 0 0 0 0 0 0 0