CHƯƠNG 6

BÀI TOÁN ĐƯỜNG ĐI NGẮN NHẤT

(tuần 10,11: Tổng cộng có 4 tiết lý thuyết và 4 tiết hướng dẫn bài tập/thực hành)

Trong các ứng dụng thực tế, vài toán tìm đường đi ngắn nhất giữa hai đỉnh của một đồ thị liên thông có một ý nghĩa to lớn. Có thể dẫn về bài toán như vậy nhiều bài toán thực tế quan trọng. Ví dụ, bài toán chọn một hành trình tiết kiệm nhất (theo tiêu chuẩn hoặc khoảng cách hoặc thời gian hoặc chi phí) trên một mạng giao thông đường bộ, đường thủy hoặc đường không; bài toán chọn một phương pháp tiết kiệm nhất để đưa ra một hệ thống động lực từ trạng thái xuất phát đến trạng một trạng thái đích, bài toán lập lịch thi công các công các công đoạn trong một công trình thi công lớn, bài toán lựa chọn đường truyền tin với chi phí nhỏ nhất trong mạng thông tin, v.v… Hiện nay có rất nhiều phương pháp để giải các bài toán như vậy. Thế nhưng, thông thường, các thuật toán được xây dựng dựa trên cơ sở lý thuyết đồ thị tỏ ra là các thuật toán có hiệu quả cao nhất. Trong chương này chúng ta sẽ xét một số thuật toán như vậy.

1. CÁC KHÁI NIỆM MỞ ĐẦU

Trong chương này chúng ta chỉ xét đồ thị có hướng G =(V,E), |V|=n, |E|=m với các cung được gán trọng số, nghĩa là, mỗi cung (u,v) $\in$ E của nó được đặt tương ứng với một số thực a(u,v) gọi là trọng số của nó. Chúng ta sẽ đặt a(u,v) = ∞ , nếu (u,v) $\notin$ E. Nếu dãy

$v_0, v_1, \ldots, v_p$

là một đường đi trên G, thì độ dài của nó được định nghĩa là tổng sau

$$\sum_{i=1}^{p} a(v_{i-1}, v_i).$$

tức là, độ dài của đường đi chính là tổng của các trọng số trên các cung của nó. (Chú ý rằng nếu chúng ta gán trọng số cho tất cả cung đều bằng 1, thì ta thu được định nghĩa độ dài của đường đi như là số cung của đường đi giống như trong các chương trước đã xét).

Bài toán tìm đường đi ngắn nhất trên đồ thị dưới dạng tổng quát có thể phát biểu như sau: tìm đường đi có độ dài nhỏ nhất từ một đỉnh xuất phát s $\in$ V đến đỉnh cuối (đích) t $\in$ V. Đường đi như vậy ta sẽ gọi là đường đi ngắn nhất từ s đến t còn độ dài của nó ta sẽ ký hiệu là d(s,t) và còn gọi là khoảng cách từ s đến t (khoảng cách định nghĩa như vậy có thể là số âm). Nếu như không tồn tại đường đi từ s đến t thì ta sẽ đặt d(s,t)=∞ . Rõ ràng, nếu như mỗi chu trình trong đồ thị đều có độ dài dương, trong đường đi ngắn nhất không có đỉnh

nào bị lặp lại (đường đi không có đỉnh lặp lại sẽ gọi là đường đi cơ bản). Mặt khác nếu trong đồ thị có chu trình với độ dài âm (chu trình như vậy để gọi ngắn gọn ta gọi là chu trình âm) thì khoảng cách giữa một số cặp đỉnh nào đó của đồ thị có thể là không xác định, bởi vì, bằng cách đi vòng theo chu trình này một số đủ lớn lần, ta có thể chỉ ra đường đi giữa các đỉnh này có độ dài nhỏ hơn bất cứ số thực cho trước nào. Trong những trường hợp như vậy, có thể đặt vấn đề tìm đường đi cơ bản ngắn nhất, tuy nhiên bài toán đặt ra sẽ trở nên phức tạp hơn rất nhiều, bởi vì nó chứa bài toán xét sự tồn tại đường đi Hamilton trong đồ thị như là một trường hợp riêng.

Trước hết cần chú ý rằng nếu biết khoảng cách từ s đến t, thì đường đi ngắn nhất từ s đến t, trong trường hợp trọng số không âm, có thể tìm được một cách dễ dàng. Để tìm đường đi, chỉ cần để ý là đối với cặp đỉnh s, t $\in$ V tuỳ ý (s <> t) luôn tìm được đỉnh v sao cho

$$d(s,t) = d(s,v) + a(v,t).$$

Thực vậy, đỉnh v như vậy chính là đỉnh đi trước đỉnh t trong đường đi ngắn nhất từ s đến t. Tiếp theo ta lại có thể tìm được đỉnh u sao cho d(s,v) = d(s,u) + a(u,v), . . . Từ giả thiết về tính không âm của các trọng số dễ dàng suy ra rằng dãy t, v, u, . . . không chứa đỉnh lặp lại và kết thúc ở đỉnh s. Rõ ràng dãy thu được xác định (nếu lật ngược thứ tự các đỉnh trong nó) đường đi ngắn nhất từ s đến t. Từ đó ta có thuật toán sau đây để tìm đường đi ngắn nhất từ s đến t khi biết độ dài của nó.

```
void Find_Path()
(*
Đầu vào:
        D[v] - khoảng cách từ đỉnh s đến tất cả các đỉnh còn lại v∈V;
        - đỉnh đích;
        a[u,v], u, v ∈V –ma trận trọng số trên các cung.
Đầu ra:
        Mảng Stack chứa dãy đỉnh xác định đường đi ngắn nhất từ s đến t
*)
        stack=∅; stack⇐t; v=t;
        while (v <> s)
        {
        u=đỉnh thoả mãn d[v]=d[u]+a[u,v];
        stack⇐u;
        v=u;
        }
```

}

Chú ý rằng độ phức tạp tính toán của thuật toán là $O(n^2)$, do để tìm đỉnh u ta phải xét qua tất cả các đỉnh của đồ thị. Tất nhiên, ta cũng có thể sử dụng kỹ thuật ghi nhận đường đi đã trình bày trong chương 3: dùng biến mảng Truoc[v], v $\in$V, để ghi nhớ đỉnh đi trước v trong đường đi tìm kiếm.

Cũng cần lưu ý thêm là trong trường hợp trọng số trên các cạnh là không âm, bài toán tìm đường đi ngắn nhất trên đồ thị vô hướng có thể dẫn về bài toán trên đồ thị có hướng, bằng cách thay đổi mỗi cạnh của nó bởi nó bởi hai cung có hướng ngược chiều nhau với cùng trọng số là trọng số của các cạnh tương ứng. Tuy nhiên, trong trường hợp có trọng số âm, việc thay như vậy có thể dẫn đến chu trình âm.

2.ĐƯỜNG ĐI NGẮN NHẤT XUẤT PHÁT TỪ MỘT ĐỈNH

Phần lớn các thuật toán tìm khoảng cách giữa hai đỉnh s và t được xây dựng nhờ kỹ thuật tính toán mà ta có thể mô tả tổng quan như sau: từ ma trận trọng số a[u][v], u,v $\in$ V, ta tính cận trên d[v] của khoảng cách từ s đến tất cả các đỉnh v $\in$V. Mỗi khi phát hiện

$$d[u] + a[u][v] < d[v] \quad (1)$$

cận trên d[v] sẽ được làm tốt lên: d[v] + a[u][v].

Quá trình đó sẽ kết thúc khi nào chúng ta không làm tốt thêm được bất kỳ cận trên nào. Khi đó, rõ ràng giá trị của mỗi d[v] sẽ cho khoảng cách từ đỉnh s đến đỉnh v. Khi thể hiện kỹ thuật tính toán này trên máy tính, cận trên d[v] sẽ được gọi là **nhãn của đỉnh** v, còn việc tính lại các cận này sẽ được gọi là **thủ tục gán**. Nhận thấy rằng để tính khoảng cách từ s đến t, ở đây, ta phải tính khoảng cách từ s đến tất cả các đỉnh còn lại của đồ thị. *Hiện nay vẫn chưa biết thuật toán nào cho phép tìm đường đi ngắn nhất giữa hai đỉnh làm việc thực sự hiệu quả hơn những thuật toán tìm đường đi ngắn nhất từ một đỉnh đến tất cả các đỉnh còn lại.*

Sơ đồ tính toán mà ta vừa mô tả còn chưa xác định, bởi vì còn phải chỉ ra thứ tự các đỉnh u và v để kiểm tra điều kiện (1). Thứ tự chọn này có ảnh hưởng rất lớn đến hiệu quả của thuật toán.

Bây giờ ta sẽ mô tả thuật toán Ford-Bellman tìm đường đi ngắn nhất từ đỉnh s đến tất cả các đỉnh còn lại của đồ thị. Thuật toán làm việc trong trường hợp trọng số của các cung là tuỳ ý, nhưng giả thiết rằng trong đồ thị không có chu trình âm.

void Ford_Bellman()

*(**

Đầu vào:

Đồ thị có hướng G = (V,E) với n đỉnh, s ∈ V là đỉnh xuất phát, a[u][v], u, v ∈ V, ma trận trọng số;

Giả thiết: Đồ thị không có chu trình âm.

Đầu ra:

Khoảng cách từ đỉnh s đến tất cả các đỉnh còn lại d[v], v ∈ V.

Trước[v], v ∈ V, ghi nhận đỉnh đi trước v trong đường đi ngắn nhất từ s đến v.

**)*

```
(* Khởi tạo *)
for v ∈ V
{
d[v]=a[s][v];
Truoc[v]=s;
};
d[s]=0;
for k=1 to n-2
for v ∈ V\{ s}
for u ∈ V
if (d[v] > d[u] +a[u][v] )
{
        d[v]=d[u]+a[u][v];
        Truoc[v]=u;
}
}
```

Tính đúng đắn của thuật toán có thể chứng minh trên cơ sở trên nguyên lý tối ưu của quy hoạch động. Rõ ràng là độ phức tạp tính toán của thuật toán là $O(n^3)$. Lưu ý rằng chúng ta có thể chấm dứt vòng lặp theo k khi phát hiện trong quá trình thực hiện hai vòng lặp trong không có biến d[v] nào bị đổi giá trị. Việc này có thể xảy ra đối với k<n-2, và điều đó làm tăng hiệu quả của thuật toán trong việc giải các bài toán thực tế. Tuy nhiên, cải tiến đó không thực sự cải thiện được đánh giá độ phức tạp của bản thân thuật toán. Đối với đồ thị thưa tốt hơn là sử dụng danh sách kề Ke(v), v ∈ V, để biểu diễn đồ thị, khi đó vòng lặp theo u cần viết lại dưới dạng

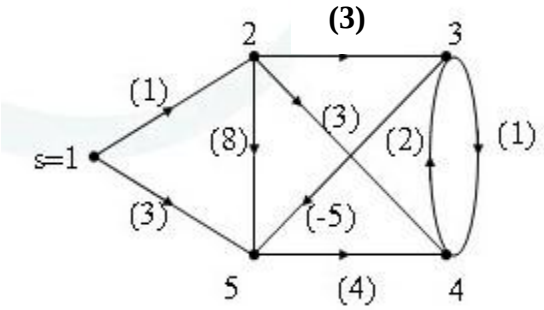
for u ∈Ke(v)

```
if (d[v] > d[u] + a[u][v] )
{
        d[v]=d[u]+a[u][v];
        Truoc[v]=u;
}
```

Trong trường hợp này ta thu được thuật toán với độ phức tạp O(n,m).

Thí dụ 1.

Xét đồ thị trong hình 1. Các kết quả tính toán theo thuật toán được mô tả trong bảng dưới đây



$$A = \begin{matrix} \infty & 1 & \infty & \infty & 3 \\ \infty & \infty & 3 & 3 & 8 \\ \infty & \infty & \infty & 1 & -5 \\ \infty & \infty & 2 & \infty & \infty \\ \infty & \infty & \infty & 4 & \infty \end{matrix}$$

Hình 1. *Minh họa thuật toán Ford_Bellman*

	d[1] Truoc[1]	d[2] Truoc[2]	d[3] Truoc[3]	d[4] Truoc[4]	d[5] Truoc[5]
	0,1	1,1	∞ ,1	∞ ,1	3,1
1	0,1	1,1	4,2	4,2	-1,3
2	0,1	1,1	4,2	3,5	-1,3
3	0,1	1,1	4,2	3,5	-1,3

Bảng kết quả tính toán theo thuật toán Ford_Bellman

Trong các mục tiếp theo chúng ta sẽ xét một số trường hợp riêng của bài toán tìm đường đi ngắn nhất mà để giải chúng có thể xây dựng những thuật toán hiệu quả hơn thuật toán Ford_Bellman. Đó là khi trọng số của tất cả các cung là các số không âm hoặc là khi đồ thị không có chu trình.

3. TRƯỜNG HỢP MA TRẬN TRỌNG SỐ KHÔNG ÂM - THUẬT TOÁN DIJKSTRA

Trong trường hợp trọng số trên các cung là không âm thuật toán do Dijkstra đề nghị làm việc hữu hiệu hơn rất nhiều so với thuật toán trình bày trong mục trước. Thuật toán được xây dựng dựa trên cơ sở gán cho các đỉnh các nhãn tạm thời. Nhãn của mỗi đỉnh cho biết cận của độ dài đường đi ngắn nhất từ s đến nó. Các nhãn này sẽ được biến đổi theo một thủ tục lặp, mà ở mỗi bước lặp có một nhãn tạm thời trở thành nhãn cố định. Nếu nhãn của một đỉnh nào đó trở thành một nhãn cố định thì nó sẽ cho ta không phải là cận trên mà là độ dài của đường đi ngắn nhất từ đỉnh s đến nó. Thuật toán được mô tả cụ thể như sau.

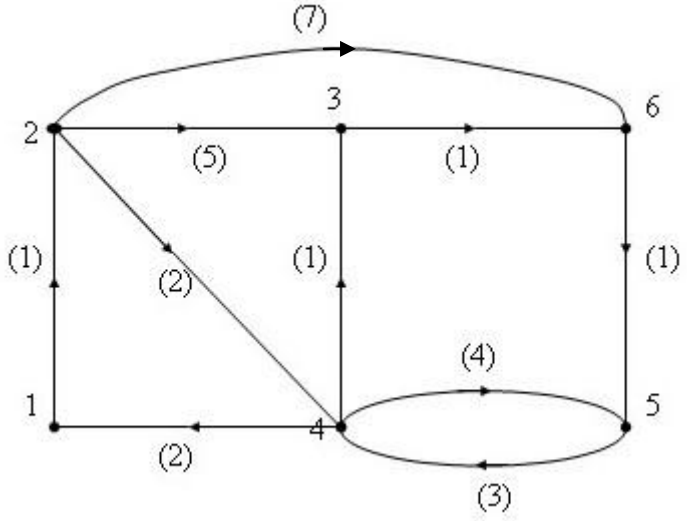
```
void Dijkstra();
(*
Đầu vào:
        Đồ thị có hướng G=(v,E) với n đỉnh,
        s ∈ V là đỉnh xuất phát, a[u,v], u,v ∈ V, ma trận trọng số;
Giả thiết: a[u,v] ≥0, u,v ∈ V.
Đầu ra:
        Khoảng cách từ đỉnh s đến tất cả các đỉnh còn lại d[v], v ∈ V.
        Truoc[v], v ∈ V, ghi nhận đỉnh đi trước v trong đường đi ngắn nhất từ s đến
        v
*)
        (* Khởi tạo *)
        for v ∈ V
        {
                d[v]=a[s][v];
                Truoc[v]=s;
        }
        d[s]=0; T=V\{ s} ; (* T là tập các đỉnh cá nhãn tạm thời *)
        (* Bước lặp *)
        while T <> ∅
```

```
        {
            tìm đỉnh u ∈ T thoả mãn d[u]=min{ d[z]:z ∈ T} ;
            T=T\{ u} ; (* Cố định nhãn của đỉnh u*)
            For v∈ T
            if (d[v]>d[u]+a[u,v] )
            {
                d[v]=d[u]+a[u,v];
                Truoc[v]=u;
            }
        }
    }
```

Định lý 1.

Thuật toán Dijkstra tìm được đường đi ngắn nhất trên đồ thị sau thời gian cỡ $O(n^2)$.

Thí dụ 2. Tìm đường đi ngắn nhất từ 1 đến các đỉnh còn lại của đồ thị ở hình 2.



Hình 2. Minh họa thuật toán Dijkstra

Kết quả tính toán theo thuật toán được trình bày theo bảng dưới đây. Qui ước viết hai thành phần của nhãn theo thứ tự: d[v]. Đỉnh được đánh dấu * là đỉnh được chọn để cố định nhãn ở bước lặp đang xét, nhãn của nó không biến đổi ở các bước tiếp theo, vì thế ta đánh dấu -.

Bước lặp	Đỉnh 1	Đỉnh 2	Đỉnh 3	Đỉnh 4	Đỉnh 5	Đỉnh 6
Khởi tạo	0,1	1,1*	∞ ,1	∞ ,1	∞ ,1	∞ ,1
1	-	-	6,2	3,2*	∞ ,1	8,2
2	-	-	4,4*	-	7,4	8,2
3	-	-		-	7,4	5,3*
4	-	-		-	6,6*	-
5						

Chú ý:

Nếu chỉ cần tìm đường đi ngắn nhất từ s đến một đỉnh t nào đó thì có thể kết thúc thuật toán khi đỉnh t trở thành có nhãn cố định.

4. ĐƯỜNG ĐI NGẮN NHẤT GIỮA TẤT CẢ CÁC CẶP ĐỈNH

Rõ ràng ta có thể giải bài toán tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh của đồ thị bằng cách sử dụng n lần thuật toán mô tả ở mục trước, trong đó ta sẽ chọn s lần lượt là các đỉnh của đồ thị. Rõ ràng, khi đó ta thu được thuật toán với độ phức tạp $O(n^4)$ (nếu sử dụng thuật toán Ford_Bellman) hoặc $O(n^3)$ đối với trường hợp trọng số không âm hoặc đồ thị không có chu trình. Trong trường hợp tổng quát, sử dụng thuật toán Ford_Bellman n lần không phải là cách làm tốt nhất. Ở đây ta sẽ mô tả một thuật toán giải bài toán trên với độ phức tạp tính toán $O(n^3)$: thuật toán Floyd. Thuật toán được mô tả trong thủ tục sau đây.

void Floyd()

(Tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh*

Đầu vào: Đồ thị cho bởi ma trận trọng số a[i,j], i, j =1, 2,. . ,n.

Đầu ra:

Ma trận đường đi ngắn nhất giữa các cặp đỉnh

d[i][j]=, i,j = 1, 2. . .,n,

trong đó d[i,j] cho độ dài đường đi ngắn nhất từ đỉnh i đến đỉnh j.

Ma trận ghi nhận đường đi

p[i][j], i, j = 1, 2.. . , n,

trong đó p[i][j] ghi nhận đỉnh đi trước đỉnh j trong đường đi ngắn nhất từ i đến j.

```
*)
        (* Khởi tạo *)
        for i=1 to n
        for j=1 to n
        {
                d[i][j]=a[i][j];
                p[i][j]=i;
        }
        (* Bước lặp *)
        for k=1 to n
        for i=1 to n
        for j=1 to n
        if (d[i][j]>d[i][k]+d[k][j])
        {
                d[i][j]=d[i][k]+d[k][j];
                p[i][j]=p[k][j];
        }
}
```

Rõ ràng độ phức tạp tính toán của thuật toán là $O(n^3)$.

Ví dụ: Xét đồ thị G sau:

Ap dụng giải thuật Floyd , ta tìm được (các ô trống là ∞)

W=W$_0$ =

	7		2		
		4		1	
					3
	4				
2		2			
	1				

	7		2		
		4		1	
					3

W=W$_1$ =

	4				
2	9	2	4		
	1				

W=W$_2$ =

	7	11	2	8	
		4		1	
					3
	4	8		5	
2	9	2	4	10	
	1	5		2	

W=W$_3$ =

	7	11	2	8	14
		4		1	7
					3
	4	8		5	11
2	9	2	4	10	5
	1	5		2	8

W=W$_4$=

	6	10	2	7	13
		4		1	7
					3
	4	8		5	11
2	9	2	4	10	5
	1	5		2	8

9	6	9	2	7	12
3	9	3	5	1	6

W=W_5

					3
7	4	7	9	5	10
2	8	2	4	9	5
4	1	4	6	2	7

W*=W_6=

9	6	9	2	7	12
3	9	3	5	1	6
7	4	7	9	5	3
7	4	7	9	5	10
2	6	2	4	7	5
4	1	4	6	2	7

Bài tập thực hành:

6-1.Di chuyển giữa các đảo

Trên một đảo quốc, có N hòn đảo. Giả sử tất cả các đảo đều có hình dạng là hình chữ nhật nằm ngang. Trên mỗi hòn đảo có thể có sân bay nằm ở trung tâm đảo, có thể có cảng nằm ở 4 góc đảo. Trên mỗi đảo đều có tuyến đường xe buýt nối 4 góc đảo với nhau và với trung tâm đảo. Giữa 2 đảo có thể đi lại bằng máy bay nếu cả 2 đảo đều có sân bay và có thể đi lại bằng tàu nếu cả 2 đảo đều có cảng.

Giả sử rằng:

Các tuyến đường (bộ, không, thủy) đều là đường thẳng.

Chi phí cho mỗi km và tốc độ của mỗi loại phương tiện là:

Phương tiện	**Tốc độ (km/h)**	**Chi phí (đ/km)**
Máy bay	1000	1000
Xe buýt	70	100
Tàu thủy	30	50

Hãy viết chương trình xác định tuyến đường và cách di chuyển giữa 2 hòn đảo trong đảo quốc sao cho:

Thời gian di chuyển ít nhất.

Chi phí di chuyển ít nhất.

Thời gian di chuyển ít nhất nhưng với một số tiền chi phí không quá Đ đồng.

Chi phí di chuyển ít nhất nhưng với thời gian di chuyển không vượt quá T giờ.

6-2.Tìm hành trình tốn ít xăng nhất

Trên một mạng lưới giao thông, một người muốn đi từ điểm A đến điểm B bằng xe máy. Xe chứa được tối đa 3 lít xăng và chạy 100km hết 2,5 lít. Các trạm xăng chỉ được đặt ở các điểm dân cư, không đặt ở giữa đường và người này không mang theo bất kỳ thùng chứa xăng nào khác. Hãy viết chương trình nhập vào mạng lưới giao thông và xác định giúp người này tuyến đường đi từ A đến B sao cho ít tốn xăng nhất.

6-3.Tìm đường ngắn nhất

Giả sử X là tập các khu dân cư, U là tập các đường sá nối liền các khu đó. Ta giả sử mọi chỗ giao nhau của các con đường đều thuộc X. Với con đường u, số l(u) là độ dài của u tính bằng km. Hãy chỉ ra tuyến đường đi từ một khu i sang khu j sao cho tổng chiều dài là nhỏ nhất.

Cài đặt một số thuật toán căn bản quan trọng

Dijkstra.cpp

```
#include <conio.h>
#include <iostream.h>
#include <stdio.h>

#define max 100
int n,s,t;
int truoc[max],d[max],a[max][max],final[max];

void nhapsolieu()
{
FILE *f;
clrscr();
f=fopen("d:\\dothi\\dijstra2.inp","rt");
fscanf(f,"%d%d%d",&n,&s,&t);
for (int i=1;i<=n;i++)
for (int j=1;j<=n;j++)
fscanf(f,"%d",&a[i][j]);
fclose(f);
}

void insolieu()
{
cout<<"Dinh nguon la:"<<s<<endl;
cout<<"Dinh dich la :" <<t<<endl;
for (int i=1;i<=n;i++)
{
for (int j=1;j<=n;j++)
cout<<a[i][j]<<"  ";
cout<<endl;
}
}
```

```
29. void inketqua()
30. {
31. cout<<" duong di ngan nhat tu "<< s <<" den "<<t <<": ";
32. cout<<t<<"<=";
33. int i=truoc[t];
34. while (i!=s)
35. {
36. cout<<i<<"<=";
37. i=truoc[i];
38. }
39. cout<<s<<endl;
40. cout<<"Do dai duong di ngan nhat la: "<<d[t]<<endl;
41. cout<<"Khoang cach tu dinh "<<s<<"den cac dinh khac la :";
42. for (i=1;i<=n;i++)
43. cout<<d[i]<<"  ";
44. getch();
45. }

46. void dijkstra()
47. {
48. int u,minp;
49. for (int v=1;v<=n;v++)
50. {
51. final[v]=0;
52. truoc[v]=s;
53. d[v]=max;   //a[s,v];s
54. }

55. d[s]=0;
56. for (int i=1;i<=n-1;i++)
57. {
58. minp=32767;
59. for (v=1;v<=n;v++)
60. if (!final[v] && minp>d[v])
```

```
61. {
62. u=v;
63. minp=d[v];
64. }
65. final[u]=1;

66. for (v=1;v<=n;v++)
67. if (!final[v])
68. if (a[u][v]>0)

69. if (d[v]>d[u]+a[u][v])
70. {
71. d[v]=d[u]+a[u][v];
72. truoc[v]=u;
73. }
74. }
75. }
```

76. void main()

```
77. {
78. nhapsolieu();
79. insolieu();
80. dijkstra();
81. inketqua();
82. }
```

Dijkstra1.inp

```
12 1 7
0 8 13 9 0 0 0 0 0 0 0 0
0 0 4 0 9 6 0 0 0 0 0 0
0 0 0 7 0 10 0 6 9 0 0 0
0 0 0 0 0 0 17 9 0 0 6 0
0 0 0 0 0 0 0 0 3 0 0 0
0 0 0 0 0 0 5 3 8 0 0 0
0 0 0 0 0 0 0 0 0 0 4 0
```

0 0 0 0 0 0 0 0 4 5 13 0

0 0 0 0 0 0 0 0 0 5 0 0

0 0 0 0 0 0 0 0 0 0 0 13

0 0 0 0 0 0 0 0 0 6 0 17

0 0 0 0 0 0 0 0 0 0 0 0

Dijkstra2.inp

6 1 5

0 1 0 0 0 0

0 0 5 2 0 7

0 0 0 0 0 1

2 0 1 0 4 0

0 0 0 3 0 0

0 0 0 0 1 0

Fordbellman.cpp

```
1. #include <conio.h>
2. #include <iostream.h>
3. #include <stdio.h>

4. #define max 50
5. #define trongsomax 1000
6. int  n,s,t;
7. int truoc[max],d[max],a[max][max];

8. void nhapsolieu()
9. {
10. FILE *f;
11. clrscr();
12. f=fopen("d:\\dothi\\fordbell.inp","rt");
13. fscanf(f,"%d%d%d",&n,&s,&t);
14. for (int i=1;i<=n;i++)
15. for (int j=1;j<=n;j++)
16. {
17. fscanf(f,"%d",&a[i][j]);
18. if (a[i][j]==0) a[i][j]=trongsomax;
19. }
20. fclose(f);
21. }

22. void insolieu()
23. {
24. cout<<"Dinh nguon la:"<<s<<endl;
25. cout<<"Dinh dich la :" <<t<<endl;
26. for (int i=1;i<=n;i++)
27. {
28. for (int j=1;j<=n;j++)
29. cout<<a[i][j]<<"  ";
```

```
30. cout<<endl;
31. }
32. }

33. void inketqua()
34. {
35. cout<<"Duong di ngan nhat tu "<< s <<" den "<<t <<": ";
36. cout<<t<<"<=";
37. int i=truoc[t];
38. while (i!=s)
39. {
40. cout<<i<<"<=";
41. i=truoc[i];
42. }
43. cout<<s<<endl;
44. cout<<"Do dai duong di ngan nhat la: "<<d[t]<<endl;
45. cout<<"Khoang cach tu dinh "<<s<<" den cac dinh khac la :";
46. for (i=1;i<=n;i++)
47. cout<<d[i]<<"  ";
48. getch();
49. }

50. void ford_bellman()
51. {
52. for (int v=1;v<=n;v++)
53. {
54. truoc[v]=s;
55. d[v]=a[s][v];
56. }
57. d[s]=0;
58. for (int k=1;k<=n-2;k++)
59. for (int v=1;v<=n;v++)
60. for (int u=1;u<=n;u++)
61. if ( d[v]>d[u]+a[u][v] && v!=s)
```

```
62. {
63. d[v]=d[u]+a[u][v];
64. truoc[v]=u;
65. }
66. }
67. void main()
68. {
69. nhapsolieu();
70. insolieu();
71. ford_bellman();
72. inketqua();
73. }
```

Fordbell.inp

```
5 1 5
0 1 0 0 3
0 0 3 3 8
0 0 0 1 -5
0 0 2 0 0
0 0 0 4 0
```

Floyd.cpp

```
1. #include <conio.h>
2. #include <iostream.h>
3. #include <stdio.h>

4. #define max 50
5. #define trongsomax 1000
6. int  n;
7. int p[max][max],d[max][max],a[max][max];

8. void nhapsolieu()
9. {
10. FILE *f;
11. clrscr();
12. f=fopen("d:\\dothi\\floyd.inp","rt");
13. fscanf(f,"%d",&n);
14. for (int i=1;i<=n;i++)
15. for (int j=1;j<=n;j++)
16. {
17. fscanf(f,"%d",&a[i][j]);
18. if (a[i][j]==0) a[i][j]=trongsomax;
19. }
20. fclose(f);
21. }

22. void insolieu()
23. {    cout<<"Ma tran trong so cua do thi:"<<endl;
24. for (int i=1;i<=n;i++)
25. {
26. for (int j=1;j<=n;j++)
27. {
28. if (a[i][j]==trongsomax)
29. cout<<0<<"   ";
30. else
```

```
31. cout<<a[i][j]<<"   ";
32. }
33. cout<<endl;
34. }
35. }
```

36. void inketqua()

```
37. {    cout<<"Ma tran duong di ngan nhat giua tat ca cac tap dinh:"<<endl;
38. for (int i=1;i<=n;i++)
39. {
40. for (int j=1;j<=n;j++)
41. if (d[i][j]==trongsomax)
42. cout<<0<<"   ";
43. else
44. cout<<d[i][j]<<"   ";
45. cout<<endl;
46. }

47. cout<<"Ma tran ghi nhan duong di p[i][j]: di truoc dinh i trong ddnn la dinh j:"<<endl;
48. for (i=1;i<=n;i++)
49. {
50. for (int j=1;j<=n;j++)
51. cout<<p[i][j]<<"   ";
52. cout<<endl;
53. }

54. getch();
55. }
```

56. void floyd()

```
57. {
58. for (int i=1;i<=n;i++)
59. for (int j=1;j<=n;j++)
60. {
```

```
61. d[i][j]=a[i][j];
62. p[i][j]=i;
63. }
64. for (int k=1;k<=n;k++)
65. for (int i=1;i<=n;i++)
66. for (int j=1;j<=n;j++)
67. if ( d[i][j]>d[i][k]+d[k][j])
68. {
69. d[i][j]=d[i][k]+d[k][j];
70. p[i][j]=p[k][j];
71. }
72. }
73. void main()
74. {
75. nhapsolieu();
76. insolieu();
77. floyd();
78. inketqua();
79. }
```

Floyd.inp

```
6
0 7 0 2 0 0
0 0 4 0 1 0
0 0 0 0 0 3
0 4 0 0 0 0
2 0 2 0 0 0
0 1 0 0 0 0
```

Ma tran d

9	6	9	2	7	12
3	7	3	5	1	6
7	4	7	9	5	3
7	4	7	9	5	10
2	6	2	4	7	5
4	1	4	6	2	7

Ma tran p

5	4	5	1	2	3
5	6	5	1	2	3
5	6	5	1	2	3
5	4	5	1	2	3
5	6	5	1	2	3
5	6	5	1	2	3