CHƯƠNG 5
TÌM ĐƯỜNG ĐI NGẮN NHẤT

Trong thực tế, chúng ta thường gặp những tình huống như sau: để đi từ địa điểm A đến địa điểm B trong thành phố, có nhiều đường đi, nhiều cách đi; có lúc ta chọn đường đi ngắn nhất theo khoảng cách, có lúc lại cần chọn đường đi nhanh nhất theo thời gian và có lúc cần phải chọn đường đi ít chi phí nhất (theo nghĩa chi phí),…hoặc lập lịch thi công các công trình một cách tối ưu, xử lý trong truyền tin ... 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. Trong chương này chúng ta sẽ xét một số thuật toán để tìm ra đường đi "tốt nhất".

5.1. BÀI TOÁN TÌM ĐƯỜNG ĐI NGẮN NHẤT

Cho đồ thị có trọng số G=(V,E), mỗi cạnh (hoặc cung) (u,v) $\in$ E được gán một số thực a(u,v) gọi là trọng số. Trọng số của mỗi cạnh (cung) còn gọi là chiều dài của cạnh (cung) đó. Mỗi đường đi từ đỉnh u đến đỉnh v, có chiều dài là m(u,v), bằng tổng chiều dài các cạnh (cung) mà nó đi qua. Khoảng cách d(u,v) giữa hai đỉnh u và v là chiều dài đường đi ngắn nhất (theo nghĩa m(u,v) nhỏ nhất) trong các đường đi từ u đến v. Chúng ta sẽ đặt a(u,v) = ∞ , nếu (u,v) $\notin$ E.

Nếu dãy v_0, v_1, . . ., 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 cạnh (cung) của đường đi.

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. Nếu như không tồn tại đường đi từ s đến t thì ta sẽ đặt d(s,t)=∞.

Để tìm đường đi, chỉ cần để ý: đối với cặp đỉnh s, t $\in$ V tuỳ ý (s $\neq$ t) luôn tìm được đỉnh v sao cho

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

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), . . . Suy ra 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 các thuật toán tìm đường đi ngắn nhất từ s đến t.

Tóm lại: Mục đích của bài toán tìm đường đi ngắn nhất là tìm đường đi P từ s đến t mà có độ dài nhỏ nhất so với tất cả những đường đi có thể có từ s đến t.

Khi thực hiện bài toán này cần chú ý một số điểm sau:

a. Chúng ta có thể bỏ bớt đi các cạnh song song và chỉ chừa lại một cạnh có trọng số nhỏ nhất trong số các cạnh song song.

b. Đối với các khuyên có trọng số không âm thì cũng có thể bỏ đi mà không làm ảnh hưởng đến kết quả của bài toán. Đối với các khuyên có trọng số âm thì đây là bài toán tìm đường đi ngắn nhất không có lời giải.

c. Dữ liệu cho bài toán đường đi ngắn nhất thường là ma trận A được định nghĩa như sau:

$$A_{ij} = \begin{cases} \text{Trọng số cạnh (cung) nối từ i đến j (nếu có)} \\ \infty, \text{ nếu không có cạnh (cung) nối từ i đến j} \end{cases}$$

5.2. THUẬT TOÁN DIJKSTRA

Cho đơn đồ thị liên thông, có trọng số G=(V,E). Tìm khoảng cách $d(u_0,v)$ từ một đỉnh u_0 cho trước đến một đỉnh v bất kỳ của G và tìm đường đi ngắn nhất từ u_0 đến v.

Thuật toán do E. Dijkstra, nhà toán học người Hà Lan, đề xuất năm 1959. Phương pháp của thuật toán Dijkstra là: xác định tuần tự các đỉnh có khoảng cách đến u_0 từ nhỏ đến lớn.

Trước tiên, đỉnh có khoảng cách đến u_0 nhỏ nhất chính là u_0, với $d(u_0,u_0)=0$. Trong các đỉnh $v \neq u_0$, tìm đỉnh có khoảng cách k_1 đến u_0 là nhỏ nhất. Đỉnh này phải là một trong các đỉnh kề với u_0. Giả sử đó là u_1. Ta có: $d(u_0,u_1) = k_1$.

Trong các đỉnh $v \neq u_0$ và $v \neq u_1$, tìm đỉnh có khoảng cách k_2 đến u_0 là nhỏ nhất. Đỉnh này phải là một trong các đỉnh kề với u_0 hoặc với u_1. Giả sử đó là u_2. Ta có: $d(u_0,u_2) = k_2$.

Tiếp tục như trên, cho đến bao giờ tìm được khoảng cách từ u_0 đến mọi đỉnh v của G. Nếu $V=\{u_0, u_1, ..., u_n\}$ thì:

$$0 = d(u_0,u_0) < d(u_0,u_1) < d(u_0,u_2) < ... < d(u_0,u_n).$$

- ***Mô tả thuật toán Dijkstra***

Trong trường hợp trọng số trên các cạnh (cung) là không âm, thuật toán Dijkstra làm việc rất hữu hiệu. 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 độ dài của đường đi ngắn nhất từ đỉnh s đến nó. Thuật toán được mô tả như sau:

```
/*
Đầu vào:
        Đồ thị có hướng G=(V,E) với n đỉnh,
        s ∈ V là đỉnh xuất phát,
        a[u,v], u,v ∈ V là 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 đỉnh 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} ;
        // 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 $O(n^2)$.

Chứng minh:

Trước hết ta chứng minh là thuật toán tìm được đường đi ngắn nhất từ đỉnh s đến các đỉnh còn lại của đồ thị. Giả sử ở một bước lặp nào đó các nhãn cố định cho ta độ dài các đường đi ngắn nhất từ s đến các đỉnh có nhãn cố định, ta sẽ chứng minh rằng ở lần gặp tiếp theo nếu đỉnh u* thu được nhãn cố định d(u*) chính là độ dài đường đi ngắn nhất từ s đến u*.

Ký hiệu S_1 là tập hợp các đỉnh có nhãn cố định còn S_2 là tập các đỉnh có nhãn tạm thời ở bước lặp đang xét. Kết thúc mỗi bước lặp nhãn tạm thời $d(u^*)$ cho ta độ dài của đường đi ngắn nhất từ s đến u* không nằm trong tập S_1, tức là nó đi qua ít nhất một đỉnh của tập S_2. Gọi $z \in S_2$ là đỉnh đầu tiên như vậy trên đường đi này. Do trọng số trên các cạnh (cung) là không âm, nên đoạn đường từ z đến u* có độ dài $L>0$ và $d(z) < d(u^*) – L < d(u^*)$.

Bất đẳng thức này là mâu thuẫn với cách xác định đỉnh u* là đỉnh có nhãn tạm thời nhỏ nhất. Vậy đường đi ngắn nhất từ s đến u* phải nằm trọn trong S_1, và vì thế, d[u*] là độ dài của nó. Do ở lần lặp đầu tiên $S_1 = \{s\}$ và sau mỗi lần lặp ta chỉ thêm vào một đỉnh u* nên giả thiết là d(v) cho độ dài đường đi ngắn nhất từ s đến v với mọi $v \in S_1$ là đúng với bước lặp đầu tiên. Theo qui nạp suy ra thuật toán cho ta đường đi ngắn nhất từ s đến mọi đỉnh của đồ thị.

Bây giờ ta sẽ đánh giá số phép toán cần thực hiện theo thuật toán. Ở mỗi bước lặp để tìm ra đỉnh u cần phải thực hiện O(n) phép toán, và để gán nhãn lại cũng cần thực hiện một số lượng phép toán cũng là O(n). Thuật toán phải thực hiện n-1 bước lặp, vì vậy thời gian tính toán của thuận toán cỡ $O(n^2)$. Định lý được chứng minh.

Khi tìm được độ dài của đường đi ngắn nhất d[v] thì đường đi này có thể tìm dựa vào nhãn Truoc[v], $v \in V$.

- ***Gợi ý cài đặt thuật toán Dijkstra***

```
void Dijkstra (int a[][Max], int n, int s)
{
   int Chuaxet[Max], i, j, u, k, min;
   for (i=1; i<=n; i++)
   {
        d[i]=a[s][i];
```

```
            Truoc[i]=s;
            Chuaxet[i]=1;
      }
      d[s]=0;
      Chuaxet[s]=0;
      k=n-1;
      while (k != 0)
      {
            min = vocung;
            for (j = 1; j <= n; j++)
            {
                  if (Chuaxet[j]==0)
                        continue;
                  if (d[j] < min)
                  {
                        min = d[j];
                        u = j;
                  }
            }
            Chuaxet[u] = 0;
            k--;
            for (v = 1; v <= n; v++)
            {
                  if (Chuaxet[v] == 0)
                        continue;
                  if (d[v] > d[u] + a[u][v])
                  {
                        d[v] = d[u] + a[u][v];
                        truoc[v] = u;
                  }
            }
      }
  }
```

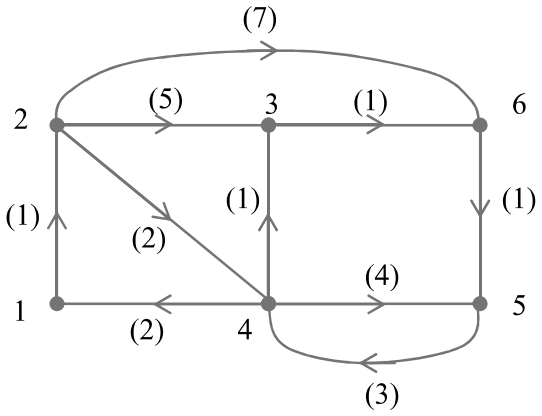
Ví dụ:

Dùng thuật toán Dijkstra để tìm đường đi ngắn nhất từ đỉnh 1 đến các đỉnh còn lại của đồ thị, biết rằng ma trận trọng số của đồ thị được cho như sau:

$$A=\begin{pmatrix} \infty & \mathbf{1} & \infty & \infty & \infty & \infty \\ \infty & \infty & \mathbf{5} & \mathbf{2} & \infty & \mathbf{7} \\ \infty & \infty & \infty & \infty & \infty & \mathbf{1} \\ \mathbf{2} & \infty & \mathbf{1} & \infty & \mathbf{4} & \infty \\ \infty & \infty & \infty & \mathbf{3} & \infty & \infty \\ \infty & \infty & \infty & \infty & \mathbf{1} & \infty \end{pmatrix}$$

Giải:

Dựa vào ma trận trọng số đã cho, ta có được đồ thị như hình 5.1



Hình 5.1. Đồ thị tương ứng với ma trận trọng số

Lập bảng tính theo giải thuật, ta được:

Bảng 5.1. Bảng minh họa ví dụ thuật toán Dijkstra

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	-	1,1*	∞,1	∞,1	∞,1	∞,1
2	-	-	6,2	3,2*	∞,1	8,2
3	-	-	4,4*	-	7,4	8,2
4	-	-	-	-	7,4	5,3*
5	-	-	-	-	6,6*	-

Vậy kết quả đường đi từ đỉnh 1 đến các đỉnh còn lại của đồ thị như sau:

Độ dài đường đi ngắn nhất từ đỉnh 1 đến 2 là: 1

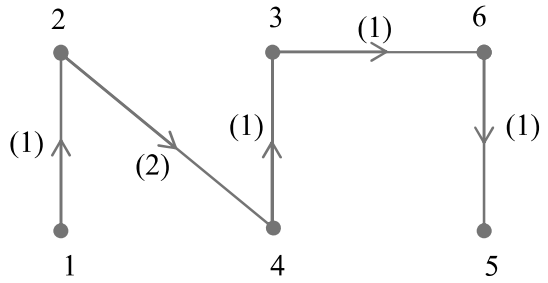
Độ dài đường đi ngắn nhất từ đỉnh 1 đến 3 là: 4

Độ dài đường đi ngắn nhất từ đỉnh 1 đến 4 là: 3

Độ dài đường đi ngắn nhất từ đỉnh 1 đến 5 là: 6

Độ dài đường đi ngắn nhất từ đỉnh 1 đến 6 là: 5

Đường đi tìm được:



Hình 5.2. Đường đi tìm được theo thuật toán Dijkstra

5.3. THUẬT TOÁN FORD-BELLMAN

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 như sau: từ ma trận trọng số a[u,v], u,v ∈ V, ta tính cận trên d[v] của khoảng cách từ s đến tất cả các đỉnh v ∈ V. Mỗi khi phát hiện d[u] + a[u,v] < d[v] thì cận trên d[v] sẽ được làm tốt lên: d[u] + a[u,v].

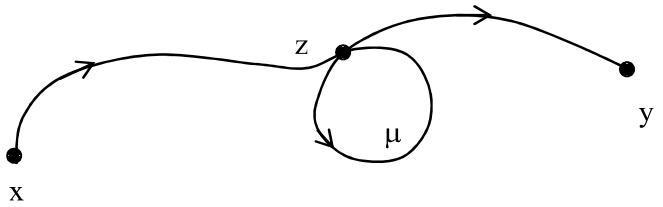
Quá trình đó sẽ kết thúc khi nào không làm tốt thêm được bất kỳ cận trên nào. Khi đó, rõ ràng giá trị của mỗi d[v] sẽ cho khoảng cách từ đỉnh s đến đỉnh v. Khi thể hiện kỹ thuật tính toán này trên máy tính, cận trên d[v] sẽ được gọi là nhãn của đỉnh v, còn việc tính lại các cận này sẽ được gọi là thủ tục gán.

Bây giờ, ta sẽ tìm hiểu 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 cạnh (cung) là tuỳ ý, nhưng giả thiết rằng trong đồ thị không có chu trình âm.

- ***Điều kiện tồn tại lời giải***

Gọi P là đường đi từ x đến y, giả sử P có chứa một chu trình μ. Có 2 trường hợp sau đây:

a. Nếu độ dài của chu trình μ >= 0 thì có thể cải tiến đường đi P bằng cách bỏ đi chu trình μ.
b. Nếu độ dài của chu trình μ < 0 thì không tồn tại đường đi ngắn nhất từ đỉnh x đến đỉnh y vì nếu lặp lại chu trình μ càng nhiều thì độ dài đường đi P càng lúc càng nhỏ đi, nghĩa là độ dài đường đi P → -∞.



Hình 5.3. Điều kiện tồn tại lời giải

- ***Mô tả thuật toán 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 là 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.

**/*

```
for ( v ∈ V )              // Khởi tạo nhãn cho các đỉnh
{
        d[v]=a[s,v];
        Truoc[v]=s;
 }
d[s]=0;
// Thực hiện đúng n-2 lần lặp
 for ( k=1; k<=n-2; k++)
        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;
                    }
```

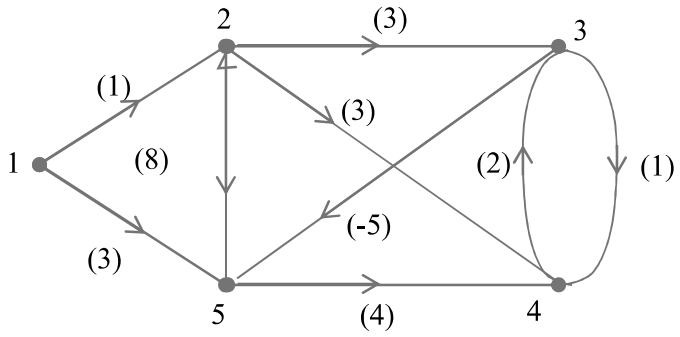
- ***Gợi ý cài đặt thuật toán Ford-Bellman***

```
void Ford-Bellman (int a[][Max], int n, int s)
{
   int i, u, v, k, min;
```

```
      for(i=1; i<=n; i++)
      {
            d[i]=a[s][i];
            truoc[i]=s;
      }
      d[s]=0;
      for(k=1; k<=n-2; k++)
            for (v=1; v<=n, v!=s; v++)
                  for (u=1; u<=n; u++)
                        if (d[v] > d[u] +a[u][v])
                        {
                              d[v]= d[u]+a[u][v];
                              Truoc[v]= u;
                        }
   }
```

Ví dụ:

Dùng thuật toán Ford-Bellman để tìm đường đi ngắn nhất từ đỉnh 1 đến các đỉnh còn lại của đồ thị được cho như hình 5.4 dưới đây:



Hình 5.4. Đồ thị minh họa thuật toán Ford-Bellman

Giải:

Lập bảng tính theo giải thuật, ta được:

Bảng 5.2. Thuật toán Ford-Bellman

Bước lặp	Đỉnh 1	Đỉnh 2	Đỉnh 3	Đỉnh 4	Đỉnh 5
Khởi tạo	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

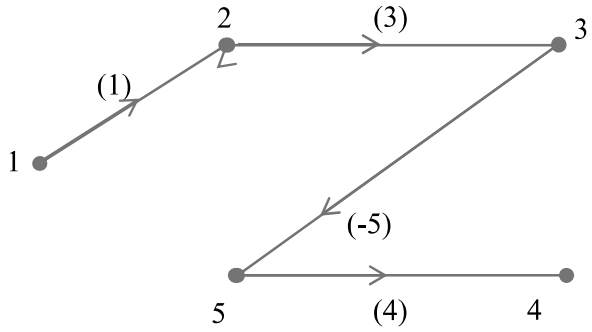
Kết quả đường đi từ đỉnh 1 đến các đỉnh còn lại của đồ thị như sau:

Độ dài đường đi ngắn nhất từ đỉnh 1 đến 2 là: 1

Độ dài đường đi ngắn nhất từ đỉnh 1 đến 3 là: 4

Độ dài đường đi ngắn nhất từ đỉnh 1 đến 4 là: 3

Độ dài đường đi ngắn nhất từ đỉnh 1 đến 5 là: -1



Hình 5.5. Đường đi tìm được theo thuật toán Ford-Bellman

5.4. THUẬT TOÁN FLOYD

Ta hoàn toàn 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.

- ***Mô tả thuật toán Floyd***

Ký hiệu :

A : ma trận đường đi ngắn nhất, được gán giá trị ban đầu như sau :

$$\begin{cases} 0 & \text{nếu } i = j \\ A[i,j] = a(i, j) & \text{nếu } (i, j) \in E \\ \infty: & \text{nếu } (i, j) \notin E \end{cases}$$

P : ma trận các đỉnh trước, được gán giá trị ban đầu: P[i,j] = i

Kết thúc thuật toán, ta có : P[i,j] là đỉnh trước của j trên đường đi ngắn nhất từ đỉnh i đến đỉnh j, với chiều dài tương ứng là A[i,j].

/*

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 A[i,j] cho độ dài đường đi ngắn nhất giữa các cặp đỉnh*

Ma trận P[i,j] ghi nhận đường đi ngắn nhất giữa các cặp đỉnh.

```
*/
// Khởi tạo
for (i=1; i<=n; i++)
        for (j=1; j<=n; j++)
        {
              A[i,j]=a[i,j];
              P[i,j]=i;
         }
// Bước lặp
for (k=1; k<=n; k++)
        for (i=1; i<=n; i++)
              for (j=1; j<=n; j++)
                    if (A[i,j]>A[i,k]+A[k,j])
                    {
                          A[i,j]=A[i,k]+A[k,j];
                         P[i,j]=P[k,j];
                    }
```

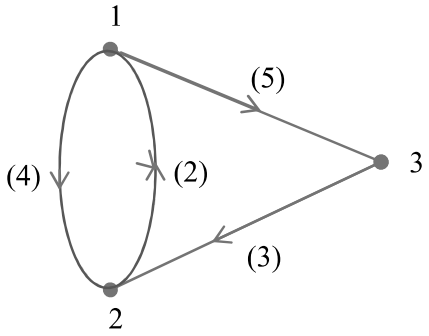
- ***Gợi ý cài đặt thuật toán Floyd***

```
void Floyd (int a[][Max], int n)
{
   int i, j, k;
   for(i=1; i<=n; i++)
         for(j=1; j<=n; j++)
         {
               A[i][j]= a[i][j];
               P[i][j]= i;
         }
   for (k=1; k<=n; k++)
         for(i=1; i<=n; i++)
               for(j=1; j<=n; j++)
                     if (A[i][j] > A[i][k]+ A[k][j])
                           {
```

```
                                    A[i][j]= A[i][k]+ A[k][j];
                                    P[i][j]=P[k][j];
                                }
    }
```

Ví dụ:

Tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh trong đồ thị dưới đây. Sau đó cho biết kết quả đường đi ngắn nhất từ đỉnh 2 đến đỉnh 3.



Hình 5.6. Đồ thị minh họa thuật toán Floyd

Giải:

Ma trận A, P ban đầu

Bảng 5.3. Bảng khởi tạo các giá trị ban đầu

A_0=

0	4	5
2	0	∞
∞	3	0

P_0=

1	1	1
2	2	2
3	3	3

Các bước lặp:

Bảng 5.4. Bảng thể hiện các bước lặp

A_1=

0	4	5
2	0	**7**
∞	3	0

P_1=

1	1	1
2	2	**1**
3	3	3

A_2=

0	4	5
2	0	**7**
5	3	0

P_2=

1	1	1
2	2	**1**
2	3	3

A_3=

0	4	5
2	0	**7**
5	3	0

P_3=

1	1	1
2	2	**1**
2	3	3

Cách nhận biết đường đi ngắn nhất.

Để nhận được đường đi ngắn nhất từ i đến j, ta sử dụng dòng thứ i của ma trận P.

Đường đi ngắn nhất từ đỉnh 2 đến đỉnh 3, ta tham khảo ma trận P như sau:

P[2,3] = 1 → 2 là đỉnh trước của đỉnh 3;

P[2,1] = 2 → đỉnh 2 là đỉnh trước của 1.

Cuối cùng, kết quả đường đi là: 2 → 1→3.

Độ dài đường đi ngắn nhất từ đỉnh 2 đến đỉnh 3, thì dựa vào ma trận A:

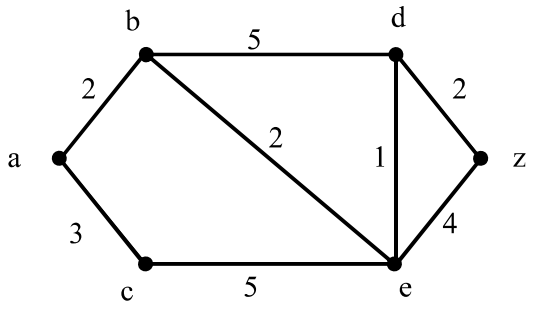
A[2,3] = 7.

Vậy đường đi ngắn nhất từ đỉnh 2 đến 3 có độ dài 7.

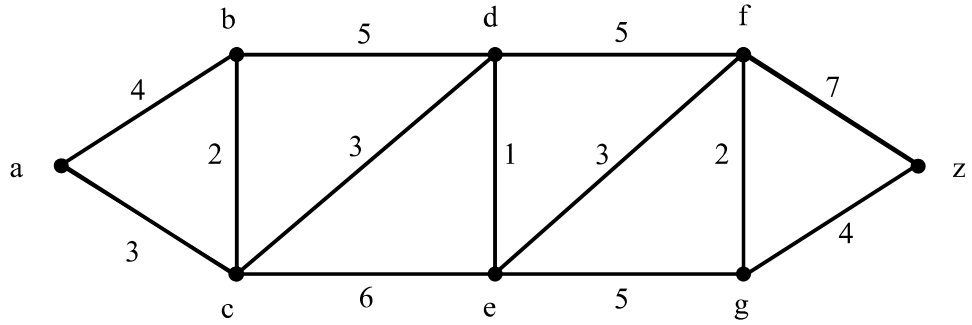
BÀI TẬP CHƯƠNG 5

1. Tìm đường đi ngắn nhất từ đỉnh a đến đỉnh z trong đồ thị bằng thuật toán Dijkstra và Ford-Bellman.

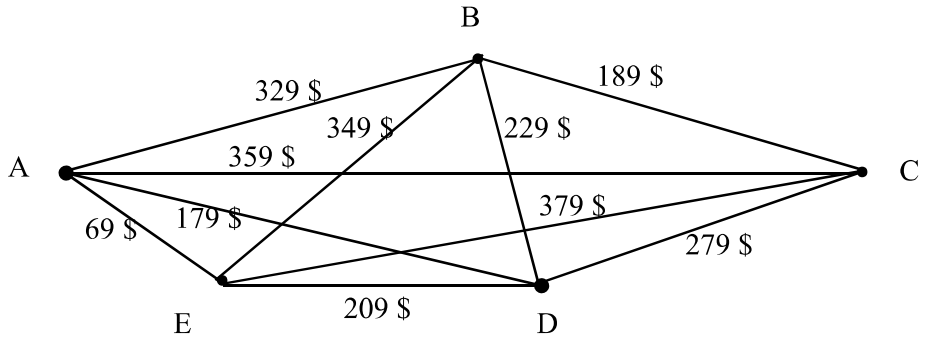
a.



b.

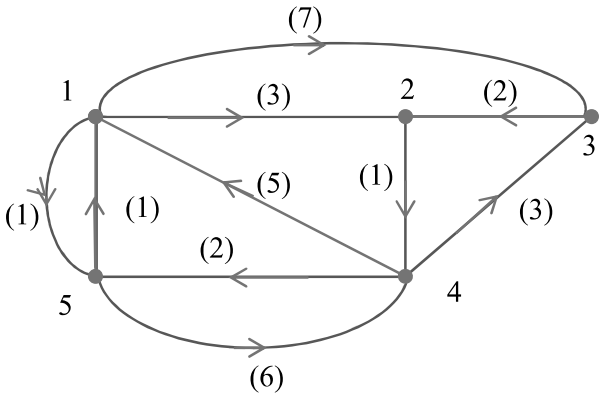


2. Cho biết sơ đồ các thành phố và chi phí để di chuyển (bằng máy bay) giữa các thành phố, giả sử một công ty du lịch đặt tại thành phố C muốn tổ chức các tour du lịch cho khách hàng từ C tới các thành phố khác. Hãy chỉ đường cho công ty này sao cho các tour là ít tốn kém về tiền vé nhất.

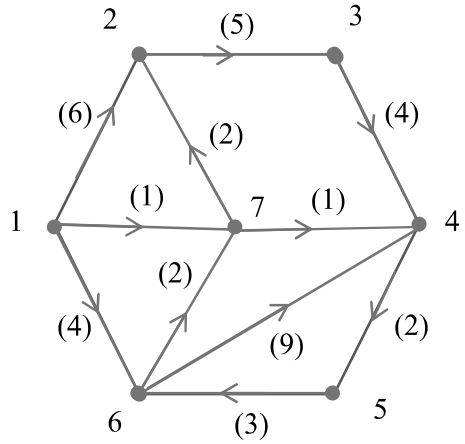


3. Tìm đường đi ngắn nhất từ đỉnh 1 đến các đỉnh còn lại của đồ thị bằng thuật toán Dijkstra và Ford-Bellman.

a.



b.



4. Dùng các đồ thị ở câu 3, tìm đường đi ngắn nhất từ đỉnh 2 đến các đỉnh còn lại bằng thuật toán Dijkstra và Ford-Bellman.

5. Tìm đường đi ngắn nhất từ đỉnh B đến các đỉnh khác của đồ thị được cho bởi ma trận trọng số sau:

	A	B	C	D	E	F	G
A	∞	**3**	**6**	∞	∞	∞	∞
B	**3**	∞	**2**	**4**	∞	∞	∞
C	**6**	**2**	∞	**1**	**4**	**2**	∞
D	∞	**4**	**1**	∞	**2**	∞	**4**
E	∞	∞	**4**	**2**	∞	**2**	**1**
F	∞	∞	**2**	∞	**2**	∞	**4**
G	∞	∞	∞	**4**	**1**	**4**	∞

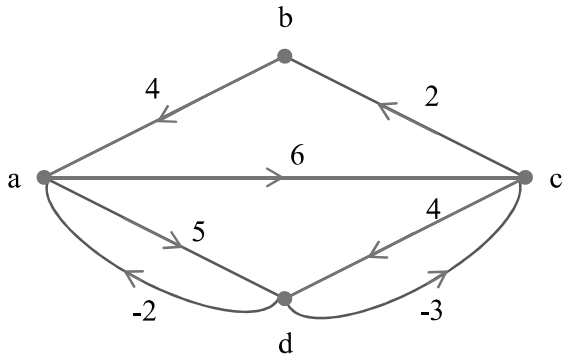
6. Tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh bằng thuật toán Floyd. Sau đó cho biết kết quả của các đường đi sau:

a. Từ đỉnh a đến đỉnh b

b. Từ đỉnh b đến đỉnh c

c. Từ đỉnh c đến đỉnh a

d. Từ đỉnh d đến đỉnh b



7. Tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh bằng thuật toán Floyd. Sau đó cho biết kết quả đường đi sau:

a. Từ đỉnh a đến đỉnh d

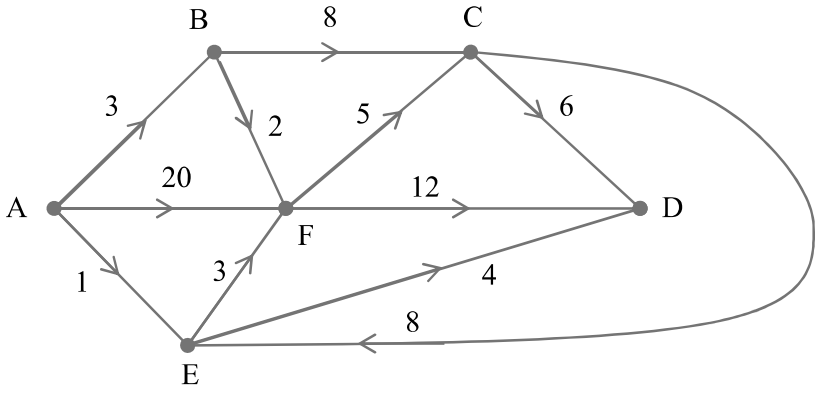
b. Từ đỉnh b đến đỉnh e

c. Từ đỉnh c đến đỉnh f

d. Từ đỉnh d đến đỉnh c

e. Từ đỉnh e đến đỉnh b

f. Từ đỉnh f đến đỉnh a



8. Viết chương trình minh họa thuật toán Dijkstra.

9. Viết chương trình minh họa thuật toán Ford-Bellman.

10. Viết chương trình minh họa thuật toán Floyd.