CHƯƠNG 6

BÀI TOÁN ĐƯỜNG ĐI NGẮN NHẤT

1. Các khái niệm

Xét đồ thị có hướng G =(V,E), |V|=n, |E|=m. Mỗi cung (u,v) đặt tương ứng với một số thực a(u,v) gọi là trọng số của cung, 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 đường đi là tổng của các trọng số trên các cung:

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

(nếu gán trọng số cho tất cả cung bằng 1, thì độ dài của đường đi là số cung của đường đi.)

Đường đi ngắn nhất từ đỉnh s đến đỉnh t là đường đi có độ dài nhỏ nhất từ s đến t, và độ dài nhỏ nhất (còn gọi là khoảng cách) ký hiệu là d(s,t). Nếu không có đường đi từ s đến t thì đặt d(s,t)= ∞ .

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 như sau:

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] , cận trên d[v] sẽ được làm tốt lên với giá trị mới: d[v]= d[u] + a[u,v].

Quá trình sẽ kết thúc khi không làm tốt thêm được bất kỳ cận trên d[v] nào. Khi đó, giá trị của mỗi d[v] sẽ cho khoảng cách từ đỉnh s đến đỉnh v.

Để tính khoảng cách từ s đến t, 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 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.

2.1 Thuật toán Ford-Bellman

Xét đồ thị với trọng số các cung là tuỳ ý và không có chu trình âm. Tìm đường đi ngắn nhất từ đỉnh s đến tất cả các đỉnh còn lại.

***Ý tưởng**

Mỗi đỉnh v gán hai giá trị :

d[v]: Khoảng cách ngắn nhất từ s đến v

Trước[v] : đỉnh ngay trước v trên đường đi ngắn nhất từ s đến v.

Ban dầu gán ∀ v∈V; d[v] = trọng số cung (s,v), hoặc ∞ nếu không có cung (s,v) và truoc[v]=s;

done=false;

Trong khi done = false thực hiện các lệnh sau:

done=true;

Lần lượt xét các đỉnh v từ 1 đến n (v ≠ s)

Với mỗi đỉnh v, lần lượt cho đường đi từ s đến v qua tất cả đỉnh u (u ≠ v, u ≠ s). Nếu d[v]>d[u] + a[u,v] thì cập nhật: d[v]=d[u]+a[u,v]; và Truoc[v]=u; done=false;

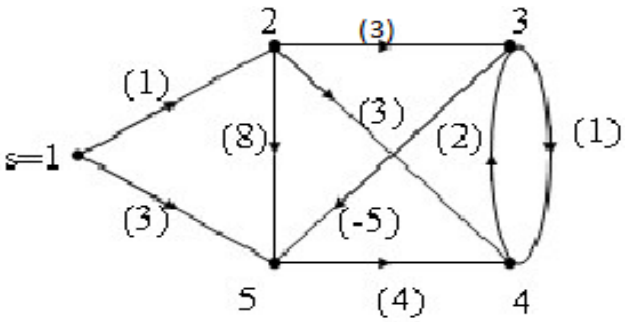
***Cài đặt**

```
const int vc=1000;
void FordBellman() {
	int d[max],truoc[max];
	bool done=false;
	for(int v=1;v<=n;v++){
		d[i]=a[s][v]; truoc[v]=s;
	}
	d[s]=0;
	while (! done) {
		done=true;
		for (int v=1;v<=n;v++)
			for (int u=1;u<=n;u++)
				if (u!=v&&a[u][v]!=vc&&d[v]>d[u]+a[u][v]){
					d[v]=d[u]+a[u][v];truoc[v]=u;done=false;
				}
	}
	cout<<endl<<"\nCac duong di tu dinh "<<s<<":"<<endl;
	for (int i=1; i<=n; i++)
		if(i!=s){
			cout<<endl<<s<<"->"<<i<<": ";
			if(d[i]==vc) cout<<"Khong co duong di"<<endl;
			else{
				int stack[max],top=-1;
				stack[++top]=i;
				int t=i;
				do{
					t=truoc[t];
					stack[++top]=t;
				}while(t!=s);
				while(top!=-1){
```

```
                    cout<<stack[top--]<<" ";
                }
                cout<<"; CD="<<d[i]<<endl;
            }
        }
}
```

Nhận xét: Độ phức tạp của thuật toán là $O(n^3)$.

Ví dụ: tìm đường đi ngắn nhất từ đỉnh 1 đến các đỉnh còn lại



Bước lặp	1	2	3	4	5
K/T	0,1	1,1	vc ,1	vc ,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

Các đường đi ngắn nhất từ đỉnh 1:

1->2: 1 -> 2 ; CD=1

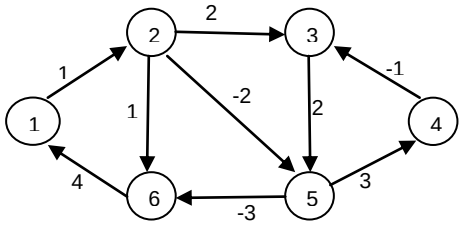
1->3: 1 ->2-> 3 ; CD=4

1->4: 1 ->2-> 3-> 5-> 4 ; CD=3

1->5: 1-> 2-> 3-> 5 ; CD=-1

Ví dụ:

Tìm đường đi ngắn nhất từ 1 đến các đỉnh khác trên đồ thị có MTTS như sau:



S=1	1	2	3	4	5	6
K/T	0,1	1,1	vc ,1	vc ,1	vc,1	vc,1
1	0,1	1,1	3,2	vc,1	-1,2	-4,5
2	0,1	1,1	3,2	2,5	-1,2	-4,5
3	0,1	1,1	1,4	2,5	-1,2	-4,5
4	0,1	1,1	1,4	2,5	-1,2	-4,5

2.2 Thuật toán Dijkstra

Trường hợp trọng số trên các cung là không âm thuật toán Dijkstra tốt hơn nhiều so với thuật toán Ford-Bellman.

***Ý tưởng**

Tìm đường đi ngắn nhất từ s đến tất cả các đỉnh còn lại

Mỗi đỉnh v gán hai giá trị:

d[v]: là khoảng cách ngắn nhất từ s đến v.

truoc[v]: là đỉnh ngay trước v trên đường đi ngắn nhất từ s đến v.

ban đầu gán: ∀ v∈V; d[v] = trọng số cung (s,v), hoặc ∞ nếu không có cung (s,v) và truoc[v]=s;

đánh dấu s chọn rồi

lần lượt lặp (n-1) lần, mỗi lần chọn một đỉnh.

đỉnh u được chọn là đỉnh u chưa chọn và có d[u] nhỏ nhất.

đánh dấu u chọn rồi

cập nhật giá trị cho các đỉnh v thỏa điều kiện: v chưa chọn, v kề u, d[v]>d[u]+a[u][v]. Cập nhật bằng cách gán d[v]=d[u]+a[u][v] và truoc[v]=u;

sử dụng mảng trước in ra đường đi ngắn nhất từ s đến tất cả các đỉnh khác s

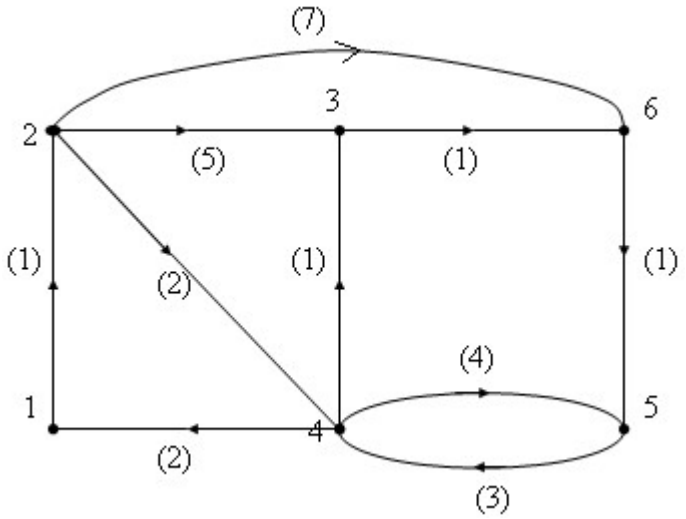
***Cài đặt**

```
int a[max][max], n;
void Dijkstra(int s){
    int tham[max],d[max],truoc[max];
    for(int i=1;i<=n;i++){
        d[i]=a[s][i]; truoc[i]=s; tham[i]=0;
```

```
	}
	tham[s]=1;
	for(int i=1;i<=n-1;i++){
		int u=0,j,temp=vc;
		for (j=1;j<=n;j++)
			if(tham[j]==0&&d[j]<temp){
				u=j;temp=d[j];
			}
		if(u==0) break;
		tham[u]=1;
		for (int v=1;v<=n;v++)
			if (tham[v]==0&&d[v]>d[u]+a[u][v]){
				d[v]=d[u]+a[u][v];truoc[v]=u;
			}
	}
	cout<<endl<<"\nCac duong di tu dinh "<<s<<":"<<endl;
	for (int i=1; i<=n; i++)
		if(i!=s){
			cout<<endl<<s<<"->"<<i<<": ";
			if(d[i]==vc) cout<<"Khong co duong di"<<endl;
			else{
				int stack[max],top=-1;
				stack[++top]=i;
				int t=i;
				do{
					t=truoc[t];
					stack[++top]=t;
				}while(t!=s);
				while(top!=-1){
					cout<<stack[top--]<<" ";
				}
				cout<<"; CD="<<d[i]<<endl;
			}
		}
}
```

Ví dụ: tìm ddnn từ 1 đến các đỉnh còn lại

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



Cac duong di tu dinh 1:

1->2: 1 - 2 ; CD=1

1->3: 1- 2- 4- 3 ; CD=4

1->4: 1 -2- 4 ; CD=3

1->5: 1- 2– 4– 3- 6- 5 ; CD=6

1->6: 1-2 -4 -3 -6 ; CD=5

Chú ý:

- Độ phức tạp của thuật toán là $O(n^2)$.
- Nếu chỉ cần tìm đường đi ngắn nhất từ s đến một đỉnh t nào đó thì có thể kết thúc thuật toán khi thăm được t.

2.3 Thuật toán Critical Path

2.3.1 Định lý (chấp nhận)

Giả sử G là đồ thị có hướng, trong số tuỳ ý, không có chu trình Khi đó các đỉnh có thể đánh số lại sao cho mỗi cung của đồ thị luôn hướng từ đỉnh có chỉ số nhỏ đến đỉnh có chỉ số lớn.

2.3.2 Thuật toán Numbering

***Ý tưởng:**

Tính bán bậc vào của tất cả các đỉnh i, cất vào mảng vao[i]

Cất các đỉnh có bán bậc vào bằng 0 vào hàng đợi.

k=0;

Trong khi hàng đợi còn khác rỗng thì thực hiện các lệnh sau:

Tăng k lên 1

Lấy một đỉnh i trong hàng đợi, đánh số đỉnh i là v[k].

Tìm các đỉnh j kề i, giảm vao[j] đi 1, nếu vao[j]=0 thì cất j vào hàng đợi.

Nếu k<n (còn đỉnh chưa được đánh số) thì đồ thị chứa chu trình nên không đánh số lại được.

***Cài đặt**

```
int vc = 30000;
int a[max][max],n,v[max];
int Numbering(){
	int vao[max]={0},queue[max], d=0, c=0, num=0 ,i, j;
	for (j=1;j<=n;j++)
		for (i=1;i<=n;i++)
			if (i!=j && a[i][j]!=vc) vao[j]++;
	for (i=1;i<=n;i++)
		if (vao[i]==0) queue[c++]= i;
	while (d!=c){
		i=queue[d++]; v[++num]=i;
		for (j=1;j<=n; j++)
			if(i!=j && a[i][j] != vc){
				vao[j] --;
				if (vao[j]==0) queue[c++]=j;
			}
	}
	if (num==n) return 1; else return 0;//tra ve 0 la danh so khong thanh cong
}
```

Dựa vào thuật toán đánh số ta có thể xây dựng thuật toán **Critical Path** tìm chiều dài đường đi ngắn nhất từ đỉnh nguồn đến các đỉnh còn lại trên đồ thị có trọng số tùy ý, không có chu trình như sau:

2.3.3 Thuật toán critical path

Giả sử đồ thị đã được đánh số lại là v1,…,vn. Hãy tìm chiều dài đđnn từ v1 đến các đỉnh v2,…,vn.

***Ý tưởng**

Sử dụng thuật toán đánh số để đánh số lại các đỉnh là v[1],…,v[n]

Gọi d[v[i]] là khoảng cách ngắn nhất từ v[1] đến v[i], với mọi i từ 1 đến n

Ban đầu gán d[v[1]]=0; và d[v[i]]=a [v[1]] [v[i]] với mọi i từ 2 đến n

Lần lượt xét các đỉnh trung gian v[i] (i từ 2 đến n-1)

Xét tất cả các đỉnh v[j] kề với v[i] (j từ i+1 đến n)

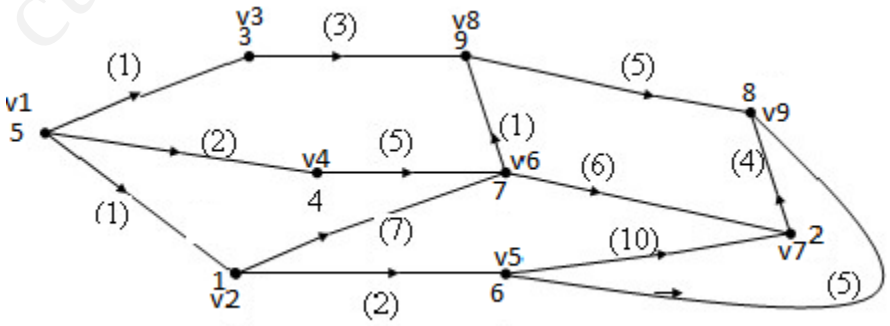
Nếu d[v[j]]>d[v[i]]+a[v[i]][v[j]] nghĩa là từ v[1] đến v[j] qua v[i] ngắn hơn khoảng cách hiện có thì cập nhật d[v[j]] = d[v[i]]+a[v[i]][v[j]]

*** Cài đặt**

```
 //d[v[i]] là kcnn từ v[1] đến v[i]
void  CriticalPath(){
        int i, j, d[max];
        if(Numbering()==0){
                cout<<"co chu trinh"<<endl; return;
        }
        d[v[1]]=0;
        for (i=2; i<=n; i++) d[v[i]]=a[v[1]][v[i]];
        for (i=2;i<n;i++)
                for (j=i+1;j<=n;j++) //xet cac dinh v[j] ke voi v[i]
                        if( a[v[i]][v[j]]!=vc && d[v[j]]>d[v[i]]+a[v[i]][v[j]])
                                d[v[j]] = d[v[i]]+a[v[i]][v[j]];
        for(i=2;i<=n;i++)
                cout<<v[1]<<"->"<<v[i]<<"="<<d[v[i]]<<endl;
}
```

Nhận xét: Độ phức tạp của thuật toán là O(m)

Ví dụ: Đánh số lại các đỉnh đồ thị sau và tìm đđ nn từ v1 đến các đỉnh v2,..,v9.



+ Đánh số: v1=5, v2=1, v3=3, v4=4, v5= 6, v6=7, v7=2, v8=9, v9=8

+ Cập nhật kcnn từ v_1 đến v_j khi lần lượt cho qua v_i với i=2,…,8

	1	3	4	6	7	2	9	8
$v_i \setminus v_j$	v2	v3	v4	v5	v6	v7	v8	v9
K/T	1	1	2	vc	vc	vc	vc	vc
v2	1	1	2	3	8	vc	vc	vc
v3	1	1	2	3	8	vc	4	vc
v4	1	1	2	3	7	vc	4	vc
v5	1	1	2	3	7	13	4	8
v6	1	1	2	3	7	13	4	8
v7	1	1	2	3	7	13	4	8
v8	1	1	2	3	7	13	4	8

+ Kcnn từ v1=5 đến các đỉnh còn lại:

5->1=1; 5->3=1; 5->4=2; 5->6=3; 5->7=7; 5->2=13; 5->9=4; 5->8=8

2.4 Thuật toán PERT (Project Evaluation and Review Technique) hay CPM (Critical path Method).

Một dự án gồm nhiều công việc, có những công việc cần làm trước công việc khác. t[i] là thời gian hoàn thành cv[i]. Giả sử thời điểm bắt đầu dự án là 0, hãy tìm thời điểm bắt đầu thực hiện mỗi công việc sao cho dự án được hoàn thành trong thời gian sớm nhất.

Ta xây dựng đồ thị có hướng n đỉnh, mỗi đỉnh ứng với một công việc, nếu công việc i phải được thực hiện trước công việc j thì trên đồ thị có cung (i,j) với trọng số t[i].

Thêm vào đồ thị hai đỉnh 0 và n+1: đỉnh 0 tương ứng với công việc lễ khởi công, nó phải được thực hiện trước tất cả các công việc khác, và đỉnh n+1 tương ứng với công việc cắt băng khánh thành, nó phải được thực hiện sau tất cả các công việc khác, với t[0]=t[n+1]=0. Ta nối đỉnh 0 với tất cả các đỉnh có bán bậc vào bằng 0 và nối tất cả các đỉnh có bán bậc ra bằng 0 với đỉnh n+1.

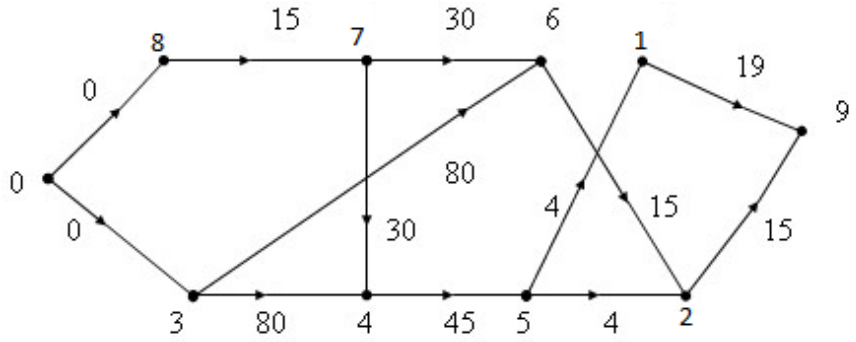
Bài toán quản lý dự án trở thành bài toán tìm đường đi dài nhất từ đỉnh 0 đến tất cả các đỉnh còn lại trên đồ thị có hướng, không chứa chu trình, nên để giải bài toán có thể áp dụng thuật toán **critical path**, chỉ cần đổi tìm Min thành tìm Max.

Khi kết thúc thuật toán **critical path**, ta có d[v] là độ dài đường đi dài nhất từ đỉnh 0 đến đỉnh v. Khi đó d[v] là thời điểm sớm nhất có thể bắt đầu thực hiện công việc v, d[n+1] là thời điểm sớm nhất có thể cắt băng khánh thành, tức là thời điểm sớm nhất có thể hoàn thành toàn bộ dự án.

Lưu ý: tìm đđ dài nhất chứ không phải tìm đđ ngắn nhất, ví dụ: cv1:4; cv2:5;cv3:6; biết 1->2,1->3, 2->3

Ví dụ

CV	t[i]	CV làm trước
1	19	5
2	15	5, 6
3	80	Không có
4	45	7, 3
5	4	4
6	15	7, 3
7	30	8
8	15	Không có



+Các đỉnh được đánh số lại

v0=0; v1=3; v2=8; v3=7; v4=4; v5=6; v6=5; v7=1; v8=2; v9=9;

+Thời điểm sớm nhất thực hiện công việc i =1,..,8:

1: 129; 2: 129; 3: 0; 4:80; 5:125; 6:80; 7:15; 8:0

+Thời điểm sớm nhất hoàn thành dự án: 148

+Cập nhật kcdn từ v_0 đến v_j khi lần lượt cho qua v_i với i=1,…,8

	3	8	7	4	6	5	1	2	9
$v_i \setminus v_j$	v1	v2	v3	v4	v5	v6	v7	v8	v9
K/t	0	0	-vc	-vc	-vc	-vc	-vc	-vc	-vc
v1	0	0	-vc	80	80	-vc	-vc	-vc	-vc
v2	0	0	15	80	80	-vc	-vc	-vc	-vc
v3	0	0	15	80	80	-vc	-vc	-vc	-vc
v4	0	0	15	80	80	125	-vc	-vc	-vc
v5	0	0	15	80	80	125	-vc	95	-vc
v6	0	0	15	80	80	125	129	129	-vc
v7	0	0	15	80	80	125	129	129	148
v8	0	0	15	80	80	125	129	129	148

3. Đường Đi Ngắn Nhất Giữa Tất Cả Các Cặp Đỉnh

***Ý tưởng**

Mỗi cặp đỉnh (i,j) gán hai giá trị:

d[i][j] : độ dài đường đi ngắn nhất từ đỉnh i đến đỉnh j.

p[i][j] : đỉnh ngay trước đỉnh j trên đường đi ngắn nhất từ i đến j.

Ban đầu gán: ∀(i,j) d[i][j]=a[i][j] hoặc ∞ nếu không có cung (i,j) và p[i][j]=i;

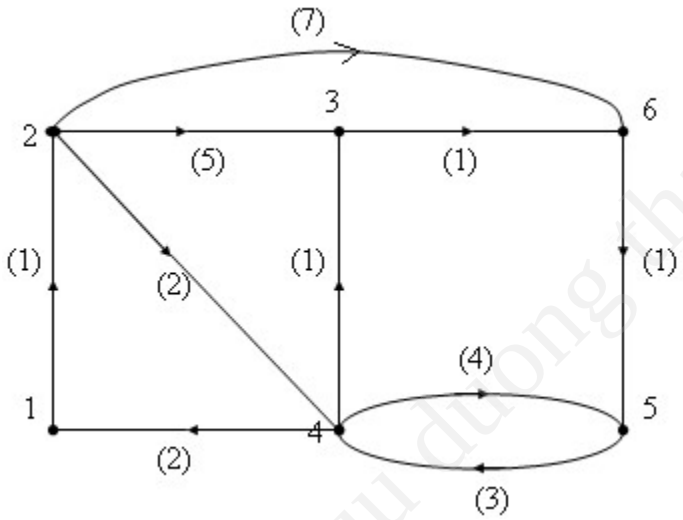
Lần lượt xét các đỉnh trung gian k=1,..,n và cho tất cả các cặp đỉnh (i,j) qua k

Nếu (d[i][j]>d[i][k]+d[k][j]) thì cập nhật d[i][j]=d[i][k]+d[k][j]; và p[i][j]=p[k][j];

***Cài đặt**

```
void Floyd() {
        for (i=1;i<=n;i++)
                for (j=1;j<=n;j++){
                        d[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 (d[i][j]>d[i][k]+d[k][j]) {
                                        d[i][j]=d[i][k]+d[k][j]; p[i][j]=p[k][j];
                                }
}
```

Ví dụ: Hãy tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh bằng thuật toán Floyd.



	1	2	3	4	5	6
1	0,1	1,1	∞,1	∞,1	∞,1	∞,1
2	∞,2	0,2	5,2	2,2	∞,2	7,2
3	∞,3	∞,3	0,3	∞,3	∞,3	1,3
4	2,4	∞,4	1,4	0,4	4,4	∞,4
5	∞,5	∞,5	∞,5	3,5	0,5	∞,5
6	∞,6	∞,6	∞,6	∞,6	1,6	0,6

Khởi tạo

	1	2	3	4	5	6
1	0,1	1,1	∞,1	∞,1	∞,1	∞,1
2	∞,2	0,2	5,2	2,2	∞,2	7,2
3	∞,3	∞,3	0,3	∞,3	∞,3	1,3
4	2,4	**3,1**	1,4	0,4	4,4	∞,4
5	∞,5	∞,5	∞,5	3,5	0,5	∞,5
6	∞,6	∞,6	∞,6	∞,6	1,6	0,6

k=1

	1	2	3	4	5	6
1	0,1	1,1	**6,2**	**3,2**	∞,1	**8,2**
2	∞,2	0,2	5,2	2,2	∞,2	7,2
3	∞,3	∞,3	0,3	∞,3	∞,3	1,3
4	2,4	3,1	1,4	0,4	4,4	**10,2**
5	∞,5	∞,5	∞,5	3,5	0,5	∞,5
6	∞,6	∞,6	∞,6	∞,6	1,6	0,6

k=2

	1	2	3	4	5	6
1	0,1	1,1	6,2	3,2	∞,1	**7,3**
2	∞,2	0,2	5,2	2,2	∞,2	**6,3**
3	∞,3	∞,3	0,3	∞,3	∞,3	1,3
4	2,4	3,1	1,4	0,4	4,4	**2,3**
5	∞,5	∞,5	∞,5	3,5	0,5	∞,5
6	∞,6	∞,6	∞,6	∞,6	1,6	0,6

k=3

	1	2	3	4	5	6
1	0,1	1,1	**4,4**	3,2	**7,4**	**5,3**
2	**4,4**	0,2	**3,4**	2,2	**6,4**	**4,3**
3	∞,3	∞,3	0,3	∞,3	∞,3	1,3
4	2,4	3,1	1,4	0,4	4,4	2,3
5	**5,4**	**6,1**	**4,4**	3,5	0,5	**5,3**
6	∞,6	∞,6	∞,6	∞,6	1,6	0,6

k=4

	1	2	3	4	5	6
1	0,1	1,1	4,4	3,2	7,4	5,3
2	4,4	0,2	3,4	2,2	6,4	4,3
3	∞,3	∞,3	0,3	∞,3	∞,3	1,3
4	2,4	3,1	1,4	0,4	4,4	2,3
5	5,4	6,1	4,4	3,5	0,5	5,3
6	**6,4**	**7,1**	**5,4**	**4,5**	1,6	0,6

k=5

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

k=6

KQ:

1-> 2 (CD= 1): 2 <- 1
1-> 3 (CD= 4): 3 <- 4 <- 2 <- 1
1-> 4 (CD= 3): 4 <- 2 <- 1
1-> 5 (CD= 6): 5 <- 6 <- 3 <- 4 <- 2 <- 1
1-> 6 (CD= 5): 6 <- 3 <- 4 <- 2 <- 1

2-> 1 (CD= 4): 1 <- 4 <- 2
2-> 3 (CD= 3): 3 <- 4 <- 2
2-> 4 (CD= 2): 4 <- 2
2-> 5 (CD= 5): 5 <- 6 <- 3 <- 4 <- 2
2-> 6 (CD= 4): 6 <- 3 <- 4 <- 2

3-> 1 (CD= 7): 1 <- 4 <- 5 <- 6 <- 3
3-> 2 (CD= 8): 2 <- 1 <- 4 <- 5 <- 6 <- 3
3-> 4 (CD= 5): 4 <- 5 <- 6 <- 3
3-> 5 (CD= 2): 5 <- 6 <- 3
3-> 6 (CD= 1): 6 <- 3

4-> 1 (CD= 2): 1 <- 4
4-> 2 (CD= 3): 2 <- 1 <- 4
4-> 3 (CD= 1): 3 <- 4
4-> 5 (CD= 3): 5 <- 6 <- 3 <- 4
4-> 6 (CD= 2): 6 <- 3 <- 4

5-> 1 (CD= 5): 1 <- 4 <- 5
5-> 2 (CD= 6): 2 <- 1 <- 4 <- 5
5-> 3 (CD= 4): 3 <- 4 <- 5
5-> 4 (CD= 3): 4 <- 5
5-> 6 (CD= 5): 6 <- 3 <- 4 <- 5

6-> 1 (CD= 6): 1 <- 4 <- 5 <- 6
6-> 2 (CD= 7): 2 <- 1 <- 4 <- 5 <- 6
6-> 3 (CD= 5): 3 <- 4 <- 5 <- 6
6-> 4 (CD= 4): 4 <- 5 <- 6
6-> 5 (CD= 1): 5 <- 6

***Bài tập chương 6**

Bài 1: Cài đặt thuật toán Ford-Bellman

Bài 2: Cài đặt thuật toán Dijkstra

Bài 3: Cài đặt thuật toán Critical Path

Bài 4: Cài đặt thuật toán PERT

Bài 5: Cài đặt thuật toán Floyd