Floyd-Warshall 算法 DP 流程詳解

Floyd-Warshall 算法，簡稱 Floyd 算法，用於求解任意兩點間的最短距離，時間複雜度為 O(n^3)。我們平時所見的 Floyd 算法的一般形式如下：

```
1 void Floyd(){
2     int i,j,k;
3     for(k=1;k<=n;k++)
4         for(i=1;i<=n;i++)
5             for(j=1;j<=n;j++)
6                 if(dist[i][k]+dist[k][j]<dist[i][j])
7                     dist[i][j]=dist[i][k]+dist[k][j];
8 }
```

注意下第 6 行這個地方，如果 dist[i][k]或者 dist[k][j]不存在，程序中用一個很大的數代替。

最好寫成 if(dist[i][k]!=INF && dist[k][j]!=INF && dist[i][k]+dist[k][j]<dist[i][j])，從而防止溢出所造成的錯誤。

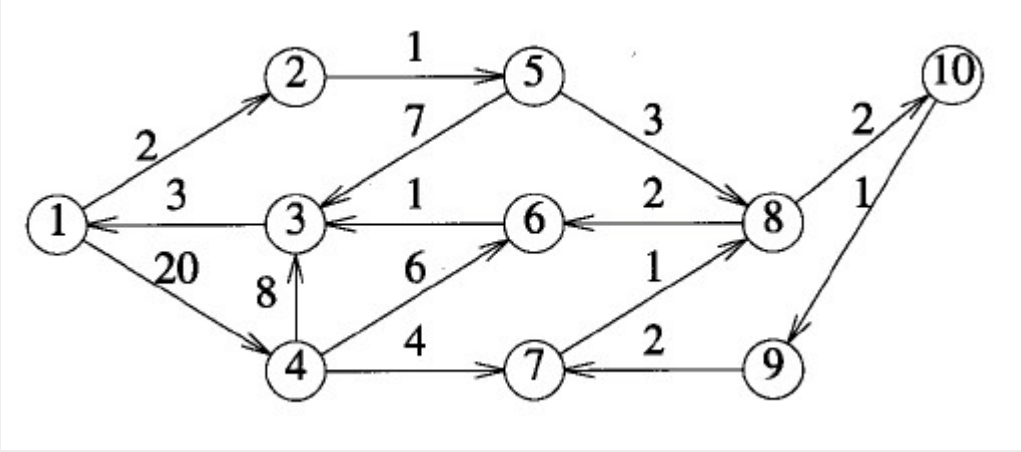
上面這個形式的算法其實是 Floyd 算法的精簡版，而真正的 Floyd 算法是一種基於 DP(Dynamic Programming)的最短路徑算法。

設圖 G 中 n 個頂點的編號為 1 到 n。令 c [i, j, k]表示從 i 到 j 的最短路徑的長度，其中 k 表示該路徑中的最大頂點，也就是說 c[i,j,k]這條最短路徑所通過的中間頂點最大不超過 k。因此，如果 G 中包含邊<i, j>，則 c[i, j, 0] =邊<i, j> 的長度；若 i= j ，則 c[i,j,0]=0；如果 G 中不包含邊<i, j>，則 c (i, j, 0)= +∞。c[i, j, n] 則是從 i 到 j 的最短路徑的長度。

對於任意的 k>0，通過分析可以得到：中間頂點不超過 k 的 i 到 j 的最短路徑有兩種可能：該路徑含或不含中間頂點 k。若不含，則該路徑長度應為 c[i, j, k-1]，否則長度為 c[i, k, k-1] +c [k, j, k-1]。c[i, j, k]可取兩者中的最小值。

狀態轉移方程：c[i, j, k]=min{c[i, j, k-1], c [i, k, k-1]+c [k, j, k-1]}，k>0。

這樣，問題便具有了最優子結構性質，可以用動態規劃方法來求解。



為了進一步理解，觀察上面這個有向圖：若 k=0, 1, 2, 3，則 c[1,3,k]= +∞；c[1,3,4]= 28；若 k = 5, 6, 7，則 c [1,3,k] = 10；若 k=8, 9, 10，則 c[1,3,k] = 9。因此 1 到 3 的最短路徑長度為 9。

下面通過程序來分析這一 DP 過程，對應上面給出的有向圖：

```
#include <iostream>
using namespace std;

const int INF = 100000;
int n=10,map[11][11],dist[11][11][11];
void init(){
    int i,j;
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            map[i][j]=(i==j)?0:INF;
    map[1][2]=2,map[1][4]=20,map[2][5]=1;
    map[3][1]=3,map[4][3]=8,map[4][6]=6;
    map[4][7]=4,map[5][3]=7,map[5][8]=3;
    map[6][3]=1,map[7][8]=1,map[8][6]=2;
    map[8][10]=2,map[9][7]=2,map[10][9]=1;
}
void floyd_dp(){
    int i,j,k;
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            dist[i][j][0]=map[i][j];
    for(k=1;k<=n;k++)
        for(i=1;i<=n;i++)
            for(j=1;j<=n;j++){   // dist[i][j][k]=min(dist[i][k][k-1]+dist[k][j][k-1], dist[i][j][k-1]);
                dist[i][j][k]=dist[i][j][k-1];
                if(dist[i][k][k-1]+dist[k][j][k-1]<dist[i][j][k])
                    dist[i][j][k]=dist[i][k][k-1]+dist[k][j][k-1];
            }
}
int main(){
    int k,u,v;
    init();
    floyd_dp();
    while(cin>>u>>v,u||v){
        for(k=0;k<=n;k++){
            if(dist[u][v][k]==INF) cout<<"+∞"<<endl;
            else cout<<dist[u][v][k]<<endl;
        }
    }
    return 0;
}
```

輸入 1 3

輸出 +∞

+∞

+∞

+∞

28

10

10

10

9

9

9

Floyd-Warshall 算法不僅能求出任意 2 點間的最短路徑，還可以保存最短路徑上經過的節點。下面用精簡版的 Floyd 算法實現這一過程，程序中的圖依然對應上面的有向圖。

```
#include <iostream>
using namespace std;

const int INF = 100000;
int n=10,path[11][11],dist[11][11],map[11][11];
void init(){
    int i,j;
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            map[i][j]=(i==j)?0:INF;
    map[1][2]=2,map[1][4]=20,map[2][5]=1;
    map[3][1]=3,map[4][3]=8,map[4][6]=6;
    map[4][7]=4,map[5][3]=7,map[5][8]=3;
    map[6][3]=1,map[7][8]=1,map[8][6]=2;
    map[8][10]=2,map[9][7]=2,map[10][9]=1;
}
void floyd(){
    int i,j,k;
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            dist[i][j]=map[i][j],path[i][j]=0;
    for(k=1;k<=n;k++)
        for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)
                if(dist[i][k]+dist[k][j]<dist[i][j])
                    dist[i][j]=dist[i][k]+dist[k][j],path[i][j]=k;
}
void output(int i,int j){
```

```
29    if(i==j) return;
30    if(path[i][j]==0) cout<<j<<' ';
31    else{
32        output(i,path[i][j]);
33        output(path[i][j],j);
34    }
35 }
36 int main(){
37    int u,v;
38    init();
39    floyd();
40    while(cin>>u>>v,u||v){
41        if(dist[u][v]==INF) cout<<"No path"<<endl;
42        else{
43            cout<<u<<' ';
44            output(u,v);
45            cout<<endl;
46        }
47    }
48    return 0;
49 }
```

輸入 1 3

輸出 1 2 5 8 6 3

PS：原創系列 轉載注明出處。

posted on 2009-04-21 00:38 極限定律 閱讀(3504)所屬分類: ACM/ICPC