最快最好用的——spfa算法

求单源最短路的SPFA算法的全称是：Shortest Path Faster Algorithm。

SPFA算法是西南交通大学段凡丁于1994年发表的。

从名字我们就可以看出，这种算法在效率上一定有过人之处。

很多时候，给定的图存在负权边，这时类似Dijkstra等算法便没有了用武之地，而Bellman-Ford算法的复杂度又过高，SPFA算法便派上用场了。有人称spfa算法是最短路的万能算法。

简洁起见，我们约定有向加权图G不存在**负权回路**，即最短路径一定存在。当然，我们可以在执行该算法前做一次拓扑排序，以判断是否存在负权回路。

我们用数组dis记录每个结点的最短路径估计值，可以用邻接矩阵或邻接表来存储图G，推荐使用邻接表。

spfa的算法思想（动态逼近法）：

设立一个先进先出的队列q用来保存待优化的结点，优化时每次取出队首结点u，并且用u点当前的最短路径估计值对离开u点所指向的结点v进行**松弛操作**，如果v点的最短路径估计值有所调整，且v点不在当前的队列中，就将v点放入队尾。这样不断从队列中取出结点来进行松弛操作，直至队列空为止。

松弛操作的原理是著名的定理：“三角形两边之和大于第三边”，在信息学中我们叫它三角不等式。所谓对结点i,j进行松弛，就是判定是否dis[j]>dis[i]+w[i,j]，如果该式成立则将dis[j]减小到dis[i]+w[i,j]，否则不动。

下面举一个实例来说明SFFA算法是怎样进行的：

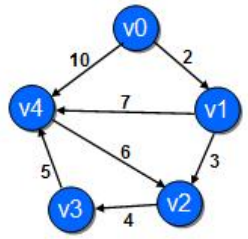


图1 有向图G，求源点v0到各点的最短路

图2 spfa算法的执行步骤及队列、dis数组变化情况

和广搜bfs的区别：

SPFA 在形式上和广度(宽度)优先搜索非常类似，不同的是bfs中一个点出了队列就不可能重新进入队列，但是SPFA中一个点可能在出队列之后再次被放入队列，也就是一个点改进过其它的点之后，过了一段时间可能本身被改进(**重新入队**)，于是再次用来改进其它的点，这样反复迭代下去。

算法的描述：

```
void  spfa(s);  //求单源点s到其它各顶点的最短距离
    for i=1 to n do { dis[i]=∞; vis[i]=false; }   //初始化每点到s的距离，不在队列
    dis[s]=0;  //将dis[源点]设为0
    vis[s]=true; //源点s入队列
    head=0; tail=1; q[tail]=s; //源点s入队，头尾指针赋初值
    while head<tail do {
       head+1;  //队首出队
       v=q[head];  //队首结点v
       vis[v]=false;  //释放对v的标记，可以重新入队
       for 每条边(v,i)  //对于与队首v相连的每一条边
          if (dis[i]>dis[v]+a[v][i])  //如果不满足三角形性质
                dis[i] = dis[v] + a[v][i]   //松弛dis[i]
                if (vis[i]=false) {tail+1; q[tail]=i; vis[i]=true;} //不在队列，则加入队列
    }
```

最短路径本身怎么输出？

在一个图中，我们仅仅知道结点A到结点E的最短路径长度，有时候意义不大。这个图如果是地图的模型的话，在算出最短路径长度后，我们总要说明“怎么走”才算真正解决了问题。如何在计算过程中记录下来最短路径是怎么走的，并在最后将它输出呢？

我们定义一个path[]数组，path[i]表示源点s到i的最短路程中，结点i之前的结点的编号(父结点)，我们在借助结点u对结点v松弛的同时，标记下path[v]=u，记录的工作就完成了。

如何输出呢？我们记录的是每个点前面的点是什么，输出却要从最前面到后面输出，这很好办，递归就可以了：

```
c++ code:
void printpath(int k){
        if (path[k]!=0) printpath(path[k]);
        cout << k << ' ';
}

pascal code:
procedure printpath(k:longint);
  begin
    if path[k]<>0 then printpath(path[k]);
    write(k,' ');
  end;
```

spfa算法模板(邻接矩阵)：

```
c++ code:
void spfa(int s){
        for(int i=0; i<=n; i++) dis[i]=99999999; //初始化每点i到s的距离
        dis[s]=0; vis[s]=1; q[1]=s;  队列初始化,s为起点
        int i, v, head=0, tail=1;
        while (head<tail){   队列非空
                head++;
                v=q[head];  取队首元素
                vis[v]=0;   释放队首结点，因为这节点可能下次用来松弛其它节点，重新入队
                for(i=0; i<=n; i++)  对所有顶点
                   if (a[v][i]>0 && dis[i]>dis[v]+a[v][i]){
                               dis[i] = dis[v]+a[v][i];   修改最短路
                               if (vis[i]==0){  如果扩展结点i不在队列中，入队
                                       tail++;
                                       q[tail]=i;
                                       vis[i]=1;
                               }
                   }

        }
}

pascal code:
procedure spfa(s:longint);
  var i,j,v,head,tail:longint;
  begin
    for i:=0 to n do dis[i]:=99999999;
    dis[s]:=0; vis[s]:=true; q[1]:=s;
    head:=0;tail:= 1;
    while head<tail do
       begin
```

```
        inc(head);
        v:=q[head];
        vis[v]:=false;
        for i:=0 to n do
          if dis[i]>dis[v]+a[v,i] then
            begin
              dis[i]:= dis[v]+a[v,i];
              if not vis[i] then
                begin
                  inc(tail);
                  q[tail]:=i;
                  vis[i]:=true;
                end;
            end;

    end;
end;
```

【程序1】畅通工程 (laoj1138)

某省自从实行了很多年的畅通工程计划后，终于修建了很多路。不过路多了也不好，每次要从一个城镇到另一个城镇时，都有许多种道路方案可以选择，而某些方案要比另一些方案行走的距离要短很多。这让行人很困扰。

现在，已知起点和终点，请你计算出要从起点到终点，最短需要行走多少距离。

输入格式：

第一行包含两个正整数N和M(0<N<200,0<M<1000)，分别代表现有城镇的数目和已修建的道路的数目。城镇分别以0～N-1编号。

接下来是M行道路信息。每一行有三个整数A,B,X(0<=A,B<N,A!=B,0<X<10000),表示城镇A和城镇B之间有一条长度为X的双向道路。

再接下一行有两个整数S,T(0<=S,T<N)，分别代表起点和终点。

输出格式：

输出最短需要行走的距离。如果不存在从S到T的路线，就输出-1。

样例输入1：

3 3

0 1 1

0 2 3

1 2 1

0 2

样例输入2：

3 1

0 1 1

1 2

样例输出1：

2

样例输出2：

-1

【分析】 注意本题可能有结点为0的顶点，我就在这上面wa了很多次。并且有可能两个城镇之间有多条道路，我们要保留最小的那条。

```
pascal code(邻接矩阵):
var i,n,m,s,t,x,y,z:longint; s:起点;t:终点
    a,b:array[0..201,0..201] of longint; b[x,c]存与x相连的第c个边的另一个结点y
    q:array[0..10001] of integer; 队列
```

```
    vis:array[0..201] of boolean; 是否入队的标记
    dis:array[0..201] of longint; 到起点的最短路
procedure spfa(s:longint);
  var i,j,v,head,tail:longint;
  begin
    fillchar(q,sizeof(q),0);
    fillchar(vis,sizeof(vis),false);
    for i:=0 to n do dis[i]:=99999999;
    dis[s]:=0; vis[s]:=true; q[1]:=s; 队列的初始状态,s为起点
    head:=0;tail:= 1;
    while head<tail do 队列不空
       begin
         inc(head);
         v:=q[head]; 取队首元素
         vis[v] := false; 释放结点，一定要释放掉，因为这节点有可能下次用来松弛其它节点
         for i:=1 to b[v,0] do
           if dis[b[v,i]]>dis[v]+a[v,b[v,i]] then
             begin
               dis[b[v,i]]:=dis[v]+a[v,b[v,i]]; 修改最短路
               if not vis[b[v,i]] then 扩展结点入队
                 begin
                   inc(tail);
                   q[tail]:=b[v,i];
                   vis[b[v,i]]:=true;
                 end;
             end;

      end;
  end;
begin
  read(n, m);  //n结点数;m边数
  fillchar(a,sizeof(a),0);
  for i:=1 to m do
    begin
      readln(x,y,z); x,y一条边的两个结点;z这条边的权值
      if (a[x,y]<>0)and(z>a[x,y]) then continue;如果两顶点间有多条边，保留最小的一条
      inc(b[x,0]);b[x,b[x,0]]:=y;a[x,y]:=z; b[x,0]以x为一个结点的边的条数
      inc(b[y,0]);b[y,b[y,0]]:=x;a[y,x]:=z;
    end;
  readln(s,t); 读入起点与终点
  spfa(s);
  if dis[t]<>99999999 then writeln(dis[t]) else writeln(-1);
end.

C++ code(邻接矩阵)：
#include <iostream>
using namespace std;
int q[10001], dis[201], a[201][201], b[201][201];
bool vis[201];
int n, m, s, t;
void spfa(int s){
        for(int i=0; i<=n; i++) dis[i]=99999999;
        dis[s]=0; vis[s]=1; q[1]=s;  队列的初始状态,s为起点
        int i, v, head=0, tail=1;
        while (head<tail){    队列不空
                head++;
                v=q[head];   取队首元素
                vis[v]=0;   释放结点，一定要释放掉，因为这节点有可能下次用来松弛其它节点
                for(i=1; i<=b[v][0]; i++)
                   if (dis[b[v][i]] > dis[v]+a[v][b[v][i]]){
                                dis[b[v][i]] = dis[v]+a[v][b[v][i]];   修改最短路
                                if (vis[b[v][i]]==0){   扩展结点入队
                                        tail++;
                                        q[tail]=b[v][i];
                                        vis[b[v][i]]=1;
                                }
                   }

        }
}
int main(){
        int x, y, z;
        cin >> n >> m;   //n结点数;m边数
        for(int i=0; i<m; i++){
                cin >> x >> y >> z;   x,y一条边的两个结点;z这条边的权值
                if (a[x][y]!=0 && z>a[x][y]) continue;如果两顶点间有多条边，保留最小的一条
                b[x][0]++; b[x][b[x][0]]=y; a[x][y]=z;   b[x,0]以x为一个结点的边的条数
                b[y][0]++; b[y][b[y][0]]=x; a[y][x]=z;
        }
        cin >> s >> t;   读入起点与终点
        spfa(s);
        if (dis[t]!=99999999) cout << dis[t] << endl;
        else cout << -1 << endl;
        return 0;
}
```

spfa优化——深度优先搜索dfs

在上面的spfa标准算法中，每次更新（松弛）一个结点u时，如果该结点不在队列中，那么直接入队。

但是有负环时，上述算法的时间复杂度退化为O(nm)。能不能改进呢？

那我们试着使用深搜，核心思想为**每次从更新一个结点u时，从该结点开始递归进行下一次迭代。**

使用dfs优化spfa算法：

```
pascal code:
procedure spfa(s:longint);
  var i:longint;
  begin
    for i:=1 to b[s,0] do  //b[s,0]是从顶点s发出的边的条数
          if dis[b[s,i]]>dis[s]+a[s,b[s,i]] then  //b[s,i]是从s发出的第i条边的另一个顶点
            begin
              dis[b[s,i]]:=dis[s]+a[s,b[s,i]];
              spfa(b[s,i]);
            end;
  end;

C++ code:
void spfa(int s){
        for(int i=1; i<=b[s][0]; i++)  //b[s,0]是从顶点s发出的边的条数
           if (dis[b[s][i]>dis[s]+a[s][b[s][i]]){  //b[s,i]是从s发出的第i条边的另一个顶点
               dis[b[s][i]=dis[s]+a[s][b[s][i]];
               spfa(b[s][i]);
           }
}
```

相比队列，深度优先搜索有着先天优势：在环上走一圈，回到已遍历过的结点即有负环。绝大多数情况下的时间复杂度为O(m)级别。

那我们试着使用深搜，核心思想为**每次从更新一个结点u时，从该结点开始递归进行下一次迭代。**

对于WorldRings(ACM-ICPC Centrual European 2005)这道题，676个点，100000条边，查找负环dfs仅仅需219ms。

一个简洁的数据结构和算法在一定程度上解决了大问题。

判断存在负环的条件：重新经过某个在当前搜索栈中的结点。

【程序1】畅通工程 laoj1138 spfa算法(dfs)：

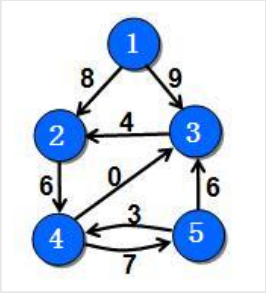
```
pascal code:
var i,n,m,s,t,x,y,z:longint;
    a,b:array[0..201,0..201] of longint;
    q:array[0..10001] of integer;
    vis:array[0..201] of boolean;
    dis:array[0..201] of longint;
procedure spfa(s:longint);
  var i:longint;
  begin
    for i:=1 to b[s,0] do
          if dis[b[s,i]]>dis[s]+a[s,b[s,i]] then
            begin
              dis[b[s,i]]:=dis[s]+a[s,b[s,i]];
              spfa(b[s,i]);
            end;
  end;
begin
  read(n, m);
  fillchar(a,sizeof(a),0);
  for i:=1 to m do
    begin
      readln(x,y,z);
      if (a[x,y]<>0)and(z>a[x,y]) then continue;
      inc(b[x,0]);b[x,b[x,0]]:=y;a[x,y]:=z;
      inc(b[y,0]);b[y,b[y,0]]:=x;a[y,x]:=z;
    end;
  readln(s,t);
  for i:=0 to n do dis[i]:=99999999;
  dis[s]:=0;
  spfa(s);
  if dis[t]<>99999999 then writeln(dis[t]) else writeln(-1);
end.

C++ code:
#include <iostream>
using namespace std;
int q[10001], dis[201], a[201][201], b[201][201];
bool vis[201];
int n, m, s, t;
void spfa(int s){
        for(int i=1; i<=b[s][0]; i++)
                if (dis[b[s][i]] > dis[s]+a[s][b[s][i]]){
                                dis[b[s][i]] = dis[s]+a[s][b[s][i]];
                                spfa(b[s][i]);
                }
}
int main(){
        int x, y, z;
        cin >> n >> m;
        for(int i=0; i<m; i++){
                cin >> x >> y >> z;
                if (a[x][y]!=0 && z>a[x][y]) continue;
                b[x][0]++; b[x][b[x][0]]=y; a[x][y]=z;
                b[y][0]++; b[y][b[y][0]]=x; a[y][x]=z;
        }
        cin >> s >> t;
        for(int i=0; i<=n; i++) dis[i]=99999999;
        dis[s]=0;
        spfa(s);
        if (dis[t]!=99999999) cout << dis[t] << endl;
        else cout << -1 << endl;
        return 0;
}
```

spfa优化——前向星优化

星形（star）表示法的思想与邻接表表示法的思想有一定的相似之处。对每个结点，它也是记录从该结点出发的所有弧，但它不是采用单向链表而是采用一个单一的数组表示。也就是说，在该数组中首先存放从结点1出发的所有弧，然后接着存放从节点2出发的所有孤，依此类推，最后存放从结点n出发的所有孤。对每条弧，要依次存放其起点、终点、权的数值等有关信息。这实际上相当于对所有弧给出了一个顺序和编号，只是从同一结点出发的弧的顺序可以任意排列。此外，为了能够快速检索从每个节点出发的所有弧，我们一般还用一个数组记录每个结点出发的弧的起始地址（即弧的编号）。在这种表示法中，可以快速检索从每个结点出发的所有弧，这种星形表示法称为前向星形（forward star）表示法。

例如，在下图中，仍然假设弧（1,2），（l,3），（2,4），（3,2），（4,3），（4,5），（5,3）和（5,4）上的权分别为8，9，6，4，0，7，6和3。此时该网络图可以用前向星形表示法表示如下：



前向星存储图：

```
#include <iostream>
using namespace std;
int first[10005];
struct edge{
        int point,next,len;
} e[10005];
void add(int i, int u, int v, int w){
                e[i].point = v;
                e[i].next = first[u];
                e[i].len = w;
                first[u] = i;
}
int n,m;
int main(){
        int u,v,w;
        cin >> n >> m;
        for (int i = 1; i <= m; i++){
                cin >> u >> v >> w;
                add(i,u,v,w);
        }  //这段是读入和加入
        for (int i = 0; i <= n; i++){
                cout << "from " << i << endl;
                for (int j = first[i]; j; j = e[j].next)  //这就是遍历边了
                        cout << "to " << e[j].point << " length= " << e[j].len << endl;
        }
}
```

欧拉回路

拓扑排序

© 六安一中信息学 @ laoj