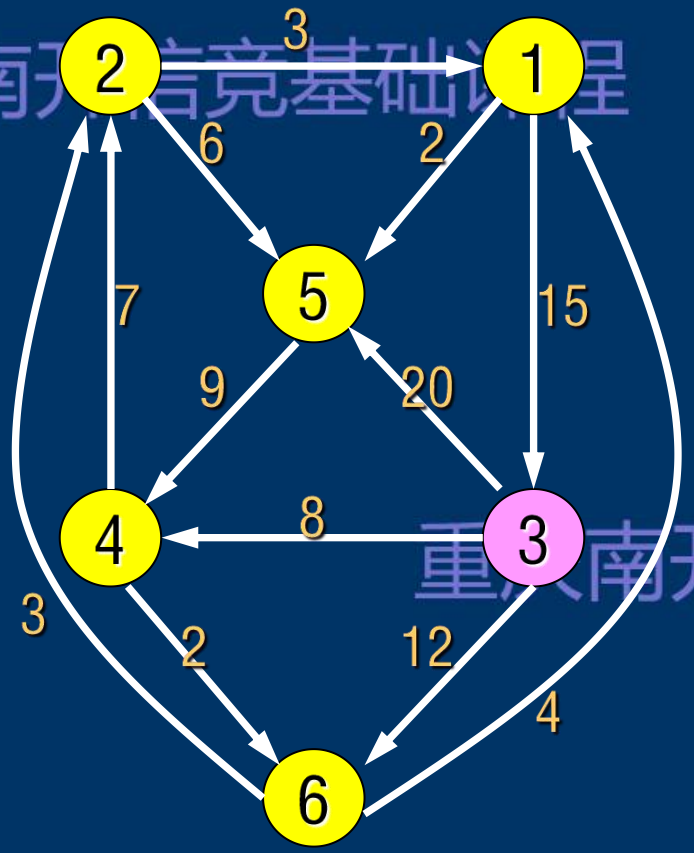
SPFA

Shortest Path Faster Algorithm



地图上有6座城市（编号1到6），它们通过很多单行道连接，请找出从3号城市出发到其他所有城市的最短距离

下面数组存每个点到起点（3号点）的最短距离：

	3	1	2	4	5	6
dis	0	14	13	8	26	10

队列：

3	4	5	6	2	1	5		

SPFA

SPFA是Bellman-Ford算法的一种队列实现，减少了不必要的冗余计算。

算法流程

用一个队列来进行维护。初始时将起点加入队列。每次从队列中取出一个元素，并对所有与他相邻的点进行松弛，若某个相邻的点松弛成功（到起点距离缩短），则将其入队。直到队列为空时算法结束。

简单的说就是队列优化的bellman-ford,利用了每个点不会更新次数太多的特点

SPFA的时间复杂度是O(kE) k一般取2左右（k是增长很快的函数ackermann的反函数,2^{65536}次方也就5以下 ），可以处理负边。

SPFA的实现甚至比Dijkstra或者 Bellman_Ford还要简单。

```
queue<int>q;
int dis[maxn]; // q为队列,dis记录节点到起点的距离
bool f[maxn]; //用于标记节点是否在队列中

void spfa(int s) //s为起点,求s到图中所有点的距离
{
        int i,x,
        for(i=1;i<=n;i++)dis[i]=99999999;        //将距离初始化为一个很大的值
        q.push(s);       f[s]=true;      dis[s]=0;          //初始化起点，将其入队

        while(q.size()>0)
        {
              x=q.front(); //x存队首元素的编号
              q.pop();                f[x]=false; //队首元素出队，将其标记为不在队中
              for(i=1;i<=n;i++)                   //讨论n个点中与x相连的点
                    if (dis[x]+map[x][i]<dis[i])      //若经过x号点，起点到i的距离缩短，则更新距离
                    {
                          dis[i]=dis[x]+map[x][i];
                          if (f[i]==false) //如果i点松弛成功且不在队中，入队
                          {
                                q.push(i);      f[i]=true;     //i号点进队，标记为已在队列中
                          }
                    }
         }
}
```

用边存储改进SPFA


```
void SPFA(int s)
……
for(i=1;i<=n;i++)dis[i]=inf;
q.push(s);    f[s]=true;     dis[s]=0;
while(q.empty()==false)
{
      x=q.front();   q.pop();   f[x]=false;
      t=last[x];
      while(t!=0)
     {
            y=end[t];
            if (dis[x]+len[t]<dis[y])
            {
                  dis[y]=dis[x]+len[t];
                  if(f[y]==false)
                  {
                        f[y]=true;
                        q.push(y);
                  }
            }
            t=Next[t];
      }
}
```


采用链式存储可大大减少讨论的次数

USACO 3.2.6

用SPFA判定负权回路

想一想：

在SPFA算法中，每个点最多进队多少次?

N-1次，因为对于一个点x,最坏情况下是其余N-1个点都可以将其松弛。

所以，若一个点进队次数超过了N次，我们就可以认定，该图中存在负权回路，可以果断结束SPFA了。

```
queue<int>q;
int dis[maxn];                          //  q为队列,dis记录节点到起点的距离
bool f[maxn];                           //用于标记节点是否在队列中
int cnt[maxn];                          //申明一个数组,用于统计每个点进队的次数

void spfa(int s)                        //s为起点,求s到图中所有点的距离
{
        int i,x,
        for(i=1;i<=n;i++)dis[i]=99999999;     //将距离初始化为一个很大的值
        q.push(s);      f[s]=true;      dis[s]=0;   //初始化起点,将其入队

        while(q.size()>0)
        {
                x=q.front();                        //x存队首元素的编号
                q.pop();                f[x]=false; //队首元素出队,将其标记为不在队中
                for(i=1;i<=n;i++)                   //讨论n个点中与x相连的点
                     if (dis[x]+map[x][i]<dis[i])   //若经过x号点,起点到i的距离缩短,则更新距离
                     {
                             dis[i]=dis[x]+map[x][i];
                             if (f[i]==false)//如果i点松弛成功且不在队中,入队
                             {
                                  q.push(i);      f[i]=true;//i号点进队,标记为已在队列中
                                  cnt[i]++;       //记录i号点进队的次数
                                  if(cnt[i]==n){   cout<<"有负权环"; return;   }
                             }
                     }
         }
}
```

spfa 小结

1.时间复杂度O(kE)

k指每个点的平均进队次数，一般为2

E指边的总数，所以期望时间复杂度为O(2E)

2.可用于负权

3.可判断负权回路

每个点的进队次数不超过N

最长路怎么求?

怎样卡掉SPFA?

课后习题

- Nkoj 1560 1976 2225 2226