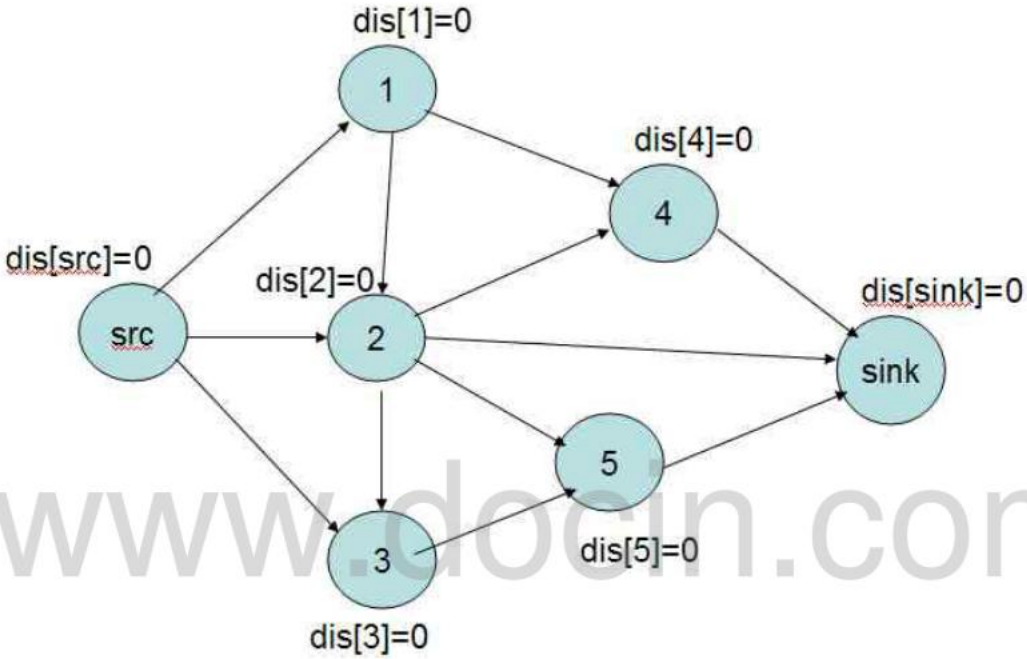
概述：

最短增广路算法(Shortest Augmenting Path Algorithm)，即每次寻找包含弧的个数最少的增广路进行增广，可以证明，此算法最多只需要进行 mn/2 次增广。并且引入距离标号的概念，可以在 O(n)的时间里找到一条最短增广路。最终的时间复杂度为 O(n^2m)，但在实践中，时间复杂度远远小于理论值（特别是加了优化之后），因此还是很实用的。

1）距离标号：

对于每个顶点 i 赋予一个非负整数值 dis(i)来描述 i 到 t 的“距离”远近。

初始化的时候，所有的顶点的 dis[i]的值均为 0；

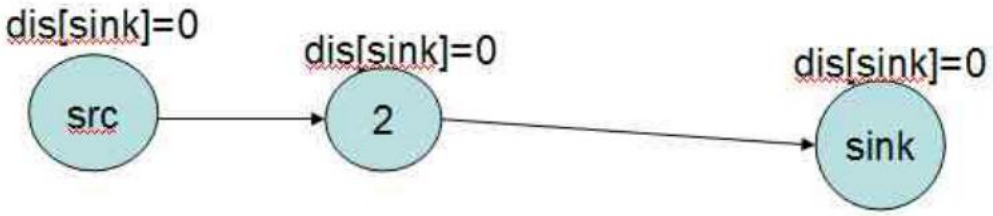


2）允许弧和允许路：

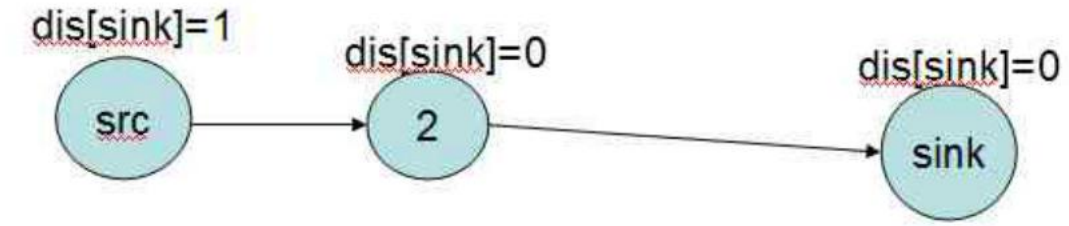
如果残留网络 G 中的一条弧(i,j)满足 $dis(i)=dis(j)+1$，我们称(i,j)是允许弧，由允许弧组成的一条 s-t 路径是允许路。显然，允许路是残留网络 G 中的一条最短增广路。当找不到允许路的时候，我们需要修改某些点的 dis(i)。

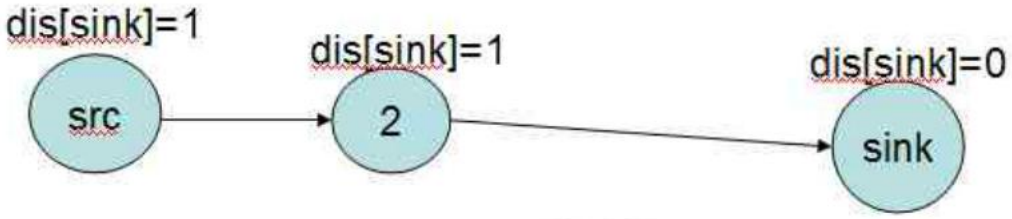
eg：增光路径：src->2->sink(以下的说明只用这三个顶点说明如果找到增广路)

1:dis[src]=0; 没有增广路径 修改 dis[src]=1;

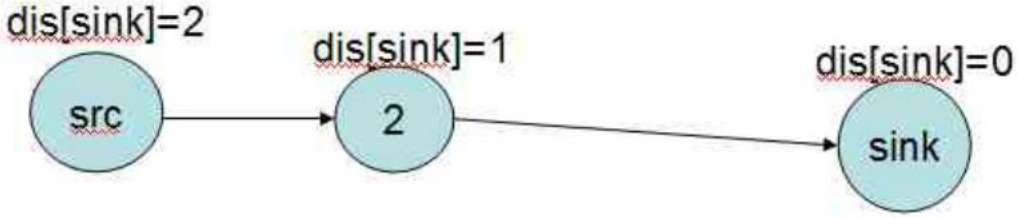


2:dis[src]=1; dis[src]=dis[2]+1; dis[2]=0,没有增广路径，修改：dis[2]=1;





3: dis[src]=1; 接着在修改的值 dis[src]=dis[2]+1=2;



4：这样就找到例子的增广路径了：src->2->sink;

dis：2　　1　　0 （严格按照允许弧的定义）

特别说明：你只需要知道这条路径是怎样找出来的时候就可以了，具体如何实现我在后面将用结合代码图片讲解。

3）Gap 优化：（这个数组的作用判断残留网络来还有没有增广路径，相当于搜索的剪枝）

Gap[x]=y:说明残留网络中 dis[i]=x 的个数为 y，一样好好理解这句话，不然后面的好难看懂

我们可以注意到由于残留网络的修改只会使 dis(i)越来越大（因为修改前 dis(i)<dis(j)+1，而修改后会存在 dis(i)=dis(j)+1，因此变大了），所以说 dis(i)是单调递增的，这就提示我们，如果 dis 函数出现了“断层”，即没有 dis(i)=k，而有 dis(i)=k±1，这时候必定无法再找到增广路径。我们可以这么想，现在的 i 满足 dis(i)=k+1，发现没有一个 dis(j)为 k，因此就会尝试去调整 dis(i)，但是 dis(i)是单调递增的，只会越来越大，所以 k 这个空缺便永远不会被补上，也就是说无法再找到增广路径。（这个地方一定要想明白，本算法经典所在）。

本人用的是递归实现 sap 算法的，从理解和代码量来说都比非递归简单,缺陷就是时间长了那么一点点：但是我想说的是，咱们有的是时间，不差时间！其实 sap 递归算法优化的好的话，其实也蛮快的。

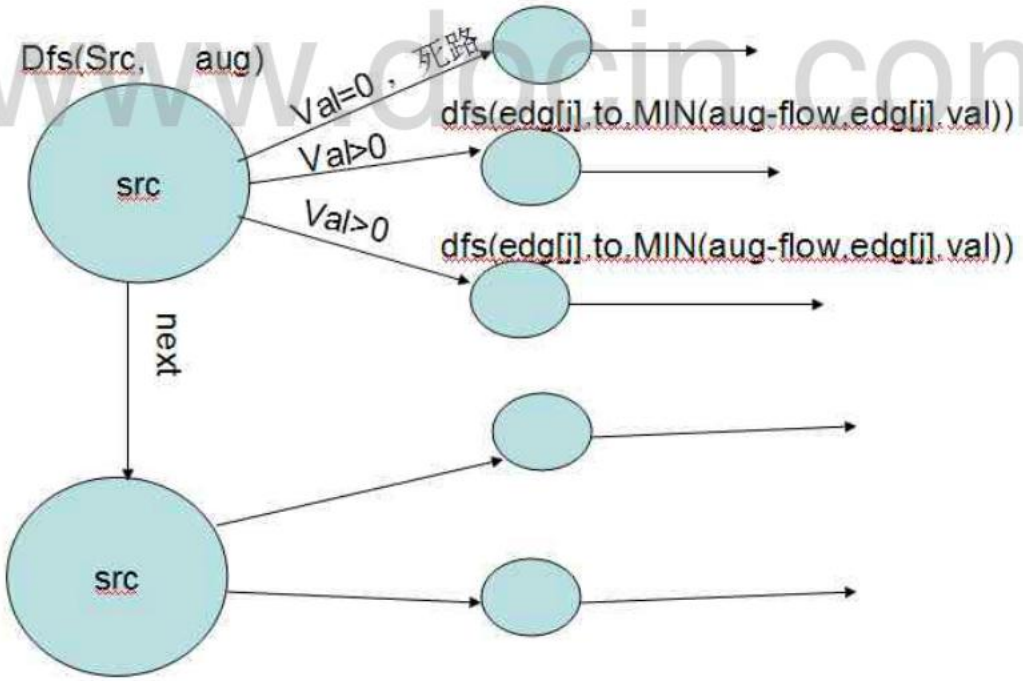
就拿这个光着身子的例题开刀：
http://acm.hdu.edu.cn/showproblem.php?pid=3549

```
int maxflow_sap(int src,int ed)  //传递参数为源点和汇点
{
    int ans(0);
    clr(dis); clr(gap);  //define clr(p)  memst(p,0,sizeof(p))
    gap[0]=Vs=ed;        //这个地方 gap[0]和 Vs 分别是顶点的个数，有
                         //候是 ed+1…… ，反正就等于顶点的个数
                         //gap[0]=Vs:说明此事有 Vs 个 dis[i]=0
    source=src; sink=ed;
    while(dis[source]<Vs)
        ans+=dfs(source,inf);  //厉害的就是这个 dfs 函数，从源点出发，
                               //第二个参数就是流量，从源点出发，相当
                               //一个水库出发
    return ans;
}
```

```
//再次强调 gap[0]和 Vs 的赋值


/*
struct E
{
    int to,val,next;
        to:下个节点是什么（可以从深度上理解，可以理解 to 的都是子节点）
        val：当前节点的流量
        next：下一个节点（可以从广度上理解，可以理解 next 的都是兄弟节
 点）
        比如：
};*/
```

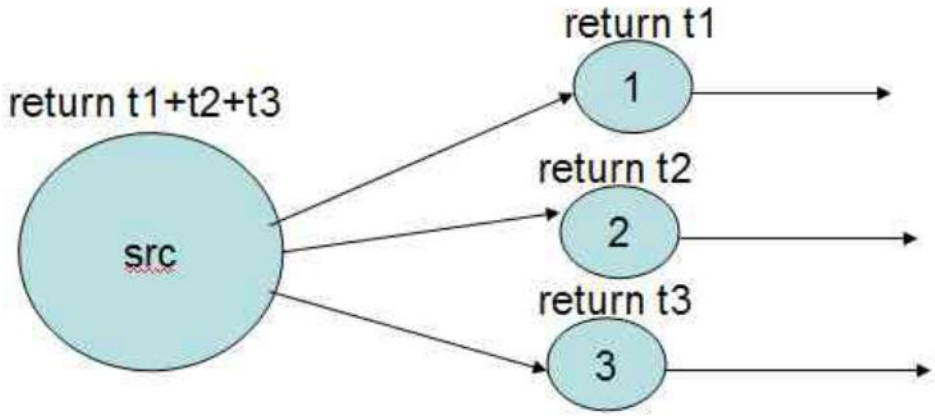
```
int dfs(int src,int aug)  //函数返回这条增广路径的流量
{
    if(src==sink)  return aug;
    int flow=0,min_d=Vs-1; //flow 这条路径的所有增流量（注意是所有的）
                           //min_d 就和这条边相连接的所有顶点的最小
                           //dis 的值
    for(int j=list[src]; j!=-1;j=edg[j].next)
       if(edg[j].val)
       {
           if(dis[src]==dis[edg[j].to]+1)   //允许弧
           {
```




```
int t=dfs(edg[j].to,MIN(aug-flow,edg[j].val))
```

```
        //t 保存了这条路径的增量，
        // MIN(aug-flow,edg[j].val)：网络流特性决定的


        edg[j].val-=t;          //更新网络流
        edg[j^1].val+=t;
        flow+=t;                 //把所有的增量加起来
```




```
        if(dis[source]>=Vs) return flow;
         //如果出现这种情况，说明已经没有增光路径了，返回此时的
         //流量就可以了
        if(aug==flow)  break;
         //这是一个减枝的作用，如果 src 这个管子的 aug 都用完了，
        //后面就算有增广路径，增加的只是一个 0，索性跳出循环
        //节省了时间
    }
    min_d=MIN(min_d,dis[edg[j].to]);
```

```
            //找到和 src 链接的所有点中 dis 最小的值;如果从 src 出发有增
            //量的话，则不需要更新 dis 函数值，这怎么办呢？看后面
        }
        if(!flow)    //看到这个判断了没有，flow 其实有两个作用：
                     //1:保存路径的所有流量；
                     //2：判断从 src 出发能不能找到路径
        {
            //这说明 flow=0，这有说明了什么了？不解释
           if(!(--gap[dis[src]]))
            //分开写：--gap[dis[src]；因为我要修改 dis[src]的值，
            //                    显然要 gap[dis[src]]要减去 1
            // if(!gap[dis[src]])出现断层的现象了，不需要再找了，
                 dis[source]=Vs;  //这个赋值就是算法结束的标志
             ++gap[dis[src]=min_d+1]; //不明白，自己展开
        }
        return flow;
}
```

到此：shortest augmenting path 算法介绍完毕（以我的理解就是这样的）。目前我做的题目用这个算法经本通过了,或许还有考虑不周到的地方,如果发现,或者对我的讲解有什么不清楚的地方，可与鄙人联系：381018143.

最后希望在校 HYNUACMer 对算法的学习，我们一起努力，来年争取为校增光！

最后附上完整的代码：

http://acm.hdu.edu.cn/showproblem.php?pid=3549

```
#include <stdio.h>
```

```
#include <string.h>

#define MAXV   16

#define MAXN   500

#define inf    0x3fffffff

#define MIN(a,b) a>b?b:a

#define clr(p) memset(p,0,sizeof(p))


struct E

{

    int to,val,next;

};

E edg[MAXN];

int G[MAXV][MAXV],dis[MAXV],gap[MAXV],list[MAXV],nodes;

int source,sink,Vs;


void addedg(int from,int to,int val)

{

    edg[nodes].to=to; edg[nodes].val=val; edg[nodes].next=list[from];
list[from]=nodes++;

    edg[nodes].to=from; edg[nodes].val=0; edg[nodes].next=list[to];
list[to]=nodes++;

}
```

```
int dfs(int src,int aug)
{
    if(src==sink)  return aug;

    int flow=0,min_d=Vs-1;
    for(int j=list[src]; j!=-1;j=edg[j].next)
       if(edg[j].val)
       {
           if(dis[src]==dis[edg[j].to]+1)
           {
               int t=dfs(edg[j].to,MIN(aug-flow,edg[j].val));
               edg[j].val-=t;
               edg[j^1].val+=t;
               flow+=t;

               if(dis[source]>=Vs) return flow;
               if(aug==flow)  break;
           }
           min_d=MIN(min_d,dis[edg[j].to]);
       }
       if(!flow)
       {
           if(!(--gap[dis[src]]))  dis[source]=Vs;
```

```
            ++gap[dis[src]=min_d+1];
        }
        return flow;
}

int maxflow_sap(int src,int ed)
{
    int ans(0);
    clr(dis); clr(gap);
    gap[0]=Vs=ed;
    source=src; sink=ed;
    while(dis[source]<Vs)
        ans+=dfs(source,inf);
    return ans;
}
int main()
{
    int CS,c; scanf("%d",&CS);
    for(c=1;c<=CS;++c)
    {
        clr(G);
        memset(list,-1,sizeof(list)); nodes=0;
        int n,m; scanf("%d %d",&n,&m);
```

```
        int i,j;
        for(i=1;i<=m;i++)
        {
            int x,y,c; scanf("%d %d %d",&x,&y,&c);
            G[x][y]+=c;
        }

        for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)
           {
               if(G[i][j])
                  addedg(i,j,G[i][j]);
           }

        printf("Case %d: %d\n",c,maxflow_sap(1,n));
    }
    return 0;
}
```

在这道题目上没有体现出优势，可是在别的题目算法还是比较快的。

参考：http://hi.baidu.com/oimaster/blog/item/08145cd6b484972606088bc1.html

WhiteCloude(原创)