SAP 算法

关于网络流的定义之类可以到 nocow 寻找，这里不在阐述。

SAP 算法（by dd_engi):求最大流有一种经典的算法，就是每次找增广路时用 BFS 找，保证找到的增广路是弧数最少的，也就是所谓的 Edmonds-Karp 算法。可以证明的是在使用最短路增广时增广过程不超过 V*E 次，每次 BFS 的时间都是 O(E)，所以 Edmonds-Karp 的时间复杂度就是 O(V*E^2)。

如果能让每次寻找增广路时的时间复杂度降下来，那么就能提高算法效率了，使用距离标号的最短增广路算法就是这样的。所谓距离标号，就是某个点到汇点的最少的弧的数量（另外一种距离标号是从源点到该点的最少的弧的数量，本质上没什么区别）。设点 i 的标号为 D[i]，那么如果将满足 D[i]=D[j]+1 的弧(i,j)叫做允许弧，且增广时只走允许弧，那么就可以达到“怎么走都是最短路”的效果。每个点的初始标号可以在一开始用一次从汇点沿所有反向边的 BFS 求出，实践中可以初始设全部点的距离标号为 0，问题就是如何在增广过程中维护这个距离标号。

维护距离标号的方法是这样的：当找增广路过程中发现某点出发没有允许弧时，将这个点的距离标号设为由它出发的所有弧的终点的距离标号的最小值加一。这种维护距离标号的方法的正确性我就不证了。由于距离标号的存在，由于“怎么走都是最短路”，所以就可以采用 DFS 找增广路，用一个栈保存当前路径的弧即可。当某个点的距离标号被改变时，栈中指向它的那条弧肯定已经不是允许弧了，所以就让它出栈，并继续用栈顶的弧的端点增广。为了使每次找增广路的时间变成均摊 O(V)，还有一个重要的优化是对于每个点保存“当前弧”：初始时当前弧是邻接表的第一条弧；在邻接表中查找时从当前弧开始查找，找到了一条允许弧，就把这条弧设为当前弧；改变距离标号时，把当前弧重新设为邻接表的第一条弧，还有一种在常数上有所优化的写法是改变距离标号时把当前弧设为那条提供了最小标号的弧。当前弧的写法之所以正确就在于任何时候我们都能保证在邻接表中当前弧的前面肯定不存在允许弧。

还有一个常数优化是在每次找到路径并增广完毕之后不要将路径中所有的顶点退栈，而是只将瓶颈边以及之后的边退栈，这是借鉴了 Dinic 算法的思想。注意任何时候待增广的“当前点”都应该是栈顶的点的终点。这的确只是一个常数优化，由于当前边结构的存在，我们肯定可以在 O(n)的时间内复原路径中瓶颈边之前的所有边。

代码：

```
var
flag:boolean;
jl,min,flow,aug,j,m,n,tmp,a,b,c,i:longint;
his,pre,dis,vh,di:array[0..1024] of longint;
map:array[1..1024,1..1024] of longint;

begin
  assign(input,'ditch.in');reset(input);
  assign(output,'ditch.out');rewrite(output);
  readln(m,n);
  for i:=1 to m do
```

```
  begin
    readln(a,b,c);
    inc(map[a,b],c);
  end;
vh[0]:=n;
for i:=1 to n do di[i]:=1;{当前弧初始为第一条弧}
i:=1;{从 S 开始搜}
aug:=maxlongint;
  while dis[1]<n do
    begin
      his[i]:=aug;{标记，以便以后返回这个值}
      flag:=false;{这个是是否有找到允许弧的标志}
      for j:=di[i] to n do{从当前弧开始搜}
        begin
          if (map[i,j]>0)and(dis[j]+1=dis[i]) then{找到允许弧}
            begin
              flag:=true;
              di[i]:=j;{标记当前弧}
              if map[i,j]<aug then aug:=map[i,j];
              pre[j]:=i;{记录前驱}
              i:=j;
                if i=n then{找到增广路}
                  begin
                    inc(flow,aug);
                      while i<>1 do
                        begin
                          tmp:=i;
                          i:=pre[i];
                          dec(map[i,tmp],aug);
                          inc(map[tmp,i],aug);
                        end;
                      aug:=maxlongint;
                  end;
                break;{找到允许弧则退出查找}
            end;
          end;
        if flag then continue;{找到允许弧}
        min:=n-1;{没有允许弧了，需要重标号}
          for j:=1 to n do
            begin
              if (map[i,j]>0)and(dis[j]<min) then begin jl:=j;min:=dis[j];end;
            end;
        di[i]:=jl;
        dec(vh[dis[i]]);{GAP 优化}
```

```
        if vh[dis[i]]=0 then break;
        dis[i]:=min+1;
        inc(vh[dis[i]]);
        if i<>1 then begin i:=pre[i];{返回上一层}aug:=his[i];{知道之前记录这个值的用处了吧}end;
      end;
  write(flow);
  close(input);close(output);
end.
```

优化：

1.邻接表优化：

如果顶点多的话，往往 N^2 存不下，这时候就要存边：

存每条边的出发点，终止点和价值，然后排序一下，再记录每个出发点的位置。以后要调用从出发点出发的边时候，只需要从记录的位置开始找即可（其实可以用链表）。优点是时间加快空间节省，缺点是编程复杂度将变大，所以在题目允许的情况下，建议使用邻接矩阵。

2.GAP 优化：

如果一次重标号时，出现距离断层，则可以证明 ST 无可行流，此时则可以直接退出算法。

3.当前弧优化：

为了使每次找增广路的时间变成均摊 O(V)，还有一个重要的优化是对于每个点保存“当前弧”：初始时当前弧是邻接表的第一条弧；在邻接表中查找时从当前弧开始查找，找到了一条允许弧，就把这条弧设为当前弧；改变距离标号时，把当前弧重新设为邻接表的第一条弧。