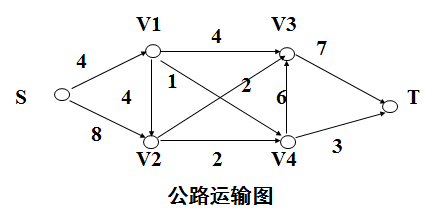
# ZKW费用流

在介绍zkw费用流之前，我们先来看它的简化版sap网络流（可以理解为所有边权为1的费用流）（sap动态演示）https://wenku.baidu.com/view/cc583f06a6c30c2259019e4e.html

 现在想将一些物资从S运抵T，必须经过一些中转站。连接中转站的是公路，每条公路都有最大运载量。

         每条弧代表一条公路，弧上的数表示该公路的最大运载量。最多能将多少货物从S运抵T？

  
         这是一个典型的网络流模型。为了解答此题，我们先了解网编流的有关定义和概论。

         若有向图G=(V,E)满足下列条件：

1.      有且仅有一个顶点S，它的入度为零，即d-(S)=0，这个顶点S便称为源点，或称为发点。

2.      有且仅有一个顶点T，它的出度为零，即d+(T)=0，这个顶点T便称为汇点，或称为收点。

3.      每一条弧都有非负数，叫做这条边的容量。边(vi,vj)的容量用cij表示。

则称之为网络流图，记为G=(V,E,C)。

**可行流**

对于网络流图G，每一条弧(i,j)都给定一个非负数fij，这一组数满足下列三条件时称为这网络的可行流，用f表示它。

1.      每一条弧(i,j)都有fij<Cij

2.      流量平衡

除了源点S和汇点T之外的所有点vi，恒有：∑j(fij)=∑k(fjk)，该等式说明中间点vi的流量守恒，输入与输出量相等。

3.      对于源点S和汇点T有，∑i(fSi)=∑j(fjT)=V(*f*)

**可增广路**

         给定一个可行流f={fij}。若fij=Cij，称<vi,vj>为饱和弧；否则称<vi,vj>为非饱和弧。若fij=0，称<vi,vj>为零流弧；否则称<vi,vj>为非零流弧。

         先定义一条道路P，起点是S，终点是T。把P上所有与P方向一致的弧定义为正向弧，正向弧的全体记为P+；把P上所有与P方向相悖的弧定义为反向孤，反向弧的全体记为P-。

         譬如在图中，P={S,V1,V2,V4,T}，那么P+={<S,V1>,<V1,V2>,<V2,V3>,<V4,T>}，P-={<V4,V3>}

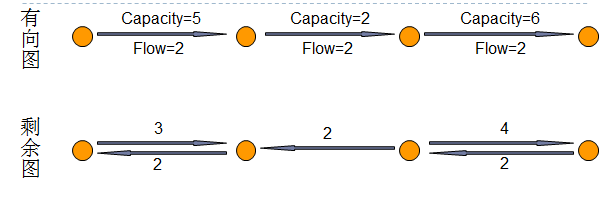
         给定一个可行流f，P是从S到T的一条道路，如果满足：fij是非饱和流，并且<i,j>∈ P+，fij是非零流，并且<i,j>∈P-，那么就称P是f的一条可增广路。之所以称作“可增广”，是因为可改进路上弧的流量通过一定的规则修改，可以令整个流量放大。

**剩余图**

剩余图G’=(V,E’)

流量网络=(V,E)中，对于任意一条边(a,b)，若flow(a,b)<capacity(a,b) or flow(b,a)>0，则(a,b)∈E’。（可以沿着aàb方向增广。

**剩余图的权值代表能沿边增广的大小**



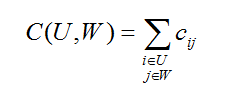
剩余图中，每条边都可以沿其方向增广

剩余图中，从源点到汇点的每一条路径都对应着一条增广路

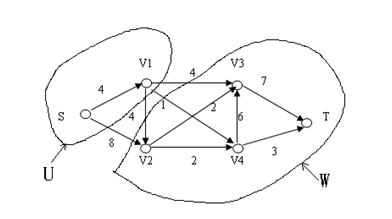
**割切**

G={V,E,C}是已知的网络流图，设U是V的一个子集，W=V\U，满足S∈U，T∈W。即U、W把V分成两个不相交的集合，且源点和汇点分属不同的集合。

对于弧尾在U，弧头在W的弧所构成的集合称之为割切，用(U,W)表示。把割切(U,W)中所有弧的容量之和叫做此割切的容量，记为C(U,W)，即：



割切示例



         上例中，令U={S，V1}，则W={V2，V3，V4，T}，那么，C(U,W)=<S,V2>+<V1,V2>+<V1,V3>+<V1,V4>=8+4+4+1=17

**流量算法的基本理论**

定理1：对于已知的网络流图，设任意一可行流为f，任意一割切为(U,W)，必有：V(f)<C(U,W).

定理2：可行流f是最大流的充分必要条件是：f中不存在可改进路。

定理3：整流定理。

         如果网络中所有的弧的容量是整数，则存在整数值的最大值。

定理4：最大流最小割定理

         最大流等于最小割，即maxV(f)=minC(U,W)。

**SAP()**

使用距离sap标号的最短增广路算法就是这样的。所谓距离标号，也可以理解为最短路，就是某个点到汇点的最小的弧的数量定义为dist[i]。

根据最短路的性质有 (1) 对任一条边(u,v)都有dis[u]<=dis[v]+w(u,v)

(2) 最短路上的边(u,v)必有 dis[u]=dis[v]+w(u,v)

设点i的标号为dist[i]，那么如果将满足dist[i]=dist[j]+1的弧称为允许弧，且增广时只走允许弧，那么就可以达到“怎么走老师最短路”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的反向边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。因为流着流着可能有点边不通了（满流了），那么这条边不能走，虽然性质(1)仍然满足，性质（2）就不一定了.

         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**。这样这个点又可以继续走下去了。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还是允许弧了，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均摊成O(V)，还有一个重要的优化是对于每个点保存“当前弧”：初始是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的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间，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。

         还有一种常数优化是在每次找到路径并增广完毕之后还要将路径中的所有的顶点退栈，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。注意任何时间待增广的“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)的时候内复原路径中瓶颈边之前的所有边。

 优化：

1.      邻接表优化：如果顶点多，往往n^2存不下，这时候就要存边：存每条边的出发点，终点点和价值，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就可以（其实可以用链表）。优化是时间快空间节省，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：为了使每次打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，找到了一条弧，就把这条弧设为当前弧；改变距离标号时，把当前弧设为邻接表的第一条弧。

void init() {

for(int i=0;i<MaxN;i++){

aug[i]=gap[i]=dis[i]=0;

cur[i]=head[i];

}

aug[st]=INF; gap[0]=n; mx\_flow=0;

}

int augment(int &point){//修改找到的增广路上的边的容量，当前点修改为起点。

for(int i=ed;i!=st;i=edge[path[i]].u) {

int pair=path[i]^1;

edge[ path[i] ].flow+=aug[ed];

edge[ pair ].flow-=aug[ed];

}

point=st;

return aug[ed];

}

int solve(){

int u=st;

while(dis[st]<n){

if(u==ed) mx\_flow+=augment(u);

bool flag=1;

for(int i=head[u];i!=-1;i=edge[i].pre) {

int v=edge[i].v;

if(edge[i].cap-edge[i].flow>0&&dis[u]==dis[v]+1) {

path[v]=i; cur[u]=i;

aug[v]=min(aug[u],edge[i].cap-edge[i].flow);

u=v;

flag=0; break;

}

}

if(flag){

if(--gap[dis[u]]==0) return mx\_flow;

dis[u]=MaxN;

for(int i=head[u];i!=-1;i=edge[i].pre) {

int v=edge[i].v;

if(edge[i].cap-edge[i].flow>0) dis[u]=min(dis[u],dis[v]+1);

}

gap[dis[u]]++;

cur[u]=head[u];

if(u!=st) u=edge[path[u]].u;

}

}

return mx\_flow;

}

## ZKW费用流

根据最短路的性质有 (1) 对任一条边(u,v)都有dis[u]<=dis[v]+w(u,v)

(2) 最短路上的边(u,v)必有 dis[u]=dis[v]+w(u,v)

会发现上面那个不等式，有点类似KM，然后就想到这样照着KM做：

增广的时候 只有当 边(u,v) 满足 dis[v]+w(u,v)=dis[u]才去从源点增广. 如果发现增广不到汇点，则修改dis值.

修改dis值，即是将所有在增广路上的点u的dis加上一个delt，而delt=min(dis[v]+w(u,v)-dis[u]) 其中u在增广路上，v不在. 和KM 一样，这样至少会多使一条边满足(2)可增广，且不会破坏(1).

算法的步骤就是初始标号设为 $0$ , 不断增广, 如果不能增广, 修改标号继续增广, 直到彻底不能增广: 源点的标号已经被加到了无穷大,即怎么也到不了T了(vis[T]==0). 这个算法不能直接用于有任何负权边的图. 更不能用于负权圈的情况，这两种情形貌似我还没找到什么好办法。

bool spfa(){

memset(mark,0,sizeof(mark));

for(int i=0;i<=T;i++)d[i]=-1;

int head=0,tail=1;

q[0]=T;mark[T]=1;d[T]=0;

while(head!=tail){

int now=q[head];head++;if(head==605)head=0;

for(int i=last[now];i;i=e[i].next)

if(e[i^1].v&&d[now]+e[i^1].c>d[e[i].to]){

d[e[i].to]=d[now]+e[i^1].c;

if(!mark[e[i].to]){

mark[e[i].to]=1;

q[tail++]=e[i].to;

if(tail==605)tail=0;

}

}

mark[now]=0;

}

return d[0]!=-1;

}

int dfs(int x,int f){

mark[x]=1;

if(x==T)return f;

int w,used=0;

for(int i=last[x];i;i=e[i].next)

if(d[e[i].to]==d[x]-e[i].c&&e[i].v&&!mark[e[i].to]){

w=f-used;

w=dfs(e[i].to,min(w,e[i].v));

ans+=w\*e[i].c;

e[i].v-=w;e[i^1].v+=w;

used+=w;if(used==f)return f;

}

return used;

}

int zkw(){

while(spfa()){

mark[T]=1;

while(mark[T]){

memset(mark,0,sizeof(mark));

dfs(0,inf);

}

}

return ans;

}

**Poj 2135**

**题目大意**

    一个图上有N个顶点，从1到N标号，顶点之间存在一些无向边，边有长度，要求从顶点1走到顶点N，再从顶点N走回顶点1，其中不必要经过每个顶点，但是要求走的路径上的边只能经过一次。求出从1--->N-->1的路径的长度最小值。

**题目分析**

    每条无向边最多只能走一次，可以视为这些边的容量只有1。题目中要求从顶点1走到N再走回顶点1，其中经过的边只能走一次，其实可以看做从顶点1出发的两条不同的路径（路径的边不能有重合）到达顶点N。那么就可以视为，从顶点1出发到达顶点N的总流量为2. 题目要求总路径长度最小值，可以将路径长度视为网络流费用，则问题转化为求解最小费用最大流。   
    引入源点ss和汇点tt，ss引入一条边到顶点1，容量为2，费用为0；从顶点N引入一条边到tt，容量为2，费用为0。则问题就成了求从ss出发到达tt的最小费用最大流。   
    由于边为无向边，因此在添加边的时候，u-->v和v-->u都要添加，且添加相应的反向边（即实际图中(u，v)边在网络流图中上有4条边对应）。