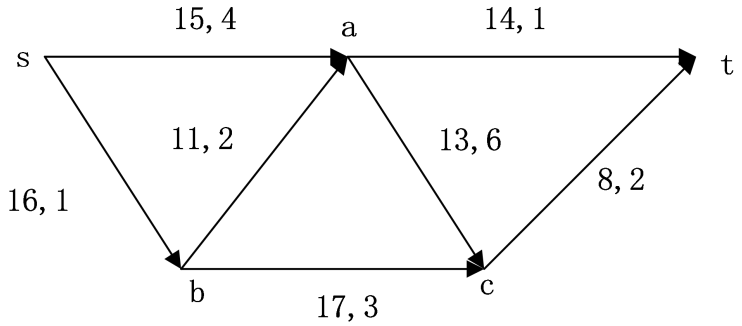
1. 主要算法

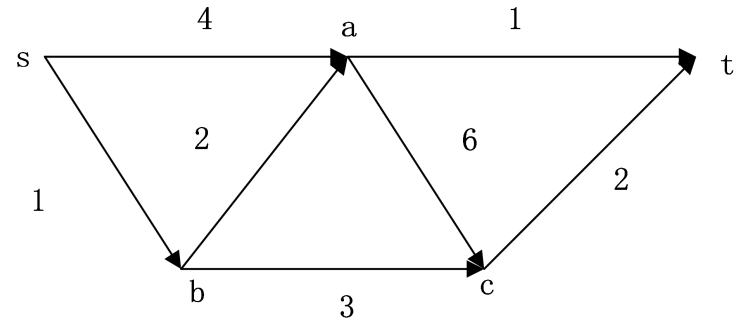
1.1 最大流最小费用算法

问题描述：给定一个有向图$G(V,E)$，设其中任意一条边(u,v)都有$w(u,v)$、$c(u,v)$、$f(u,v)$分别表示边的单位流量费用，边的容量以及边的流量。在给定起点s与终点t的情况下，从s尽可能传输流量至t，同时保证费用最小。

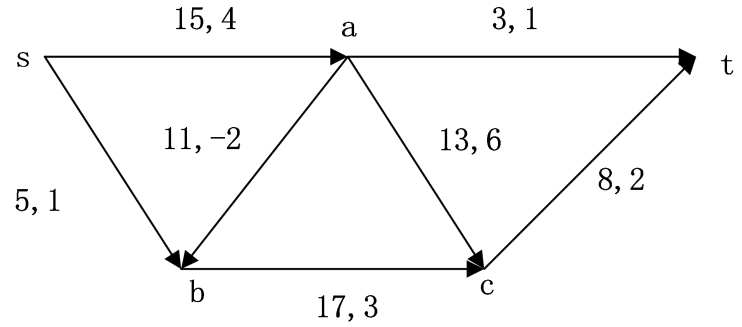
思路：将边的单位费用作为权值构建有向图，搜索从起点至终点的最短路径，完成后可以确定一条路径，路径上的流量为该线路上边的容量最小值，接下来更新图。之前线路上由于消耗了流量因此容量减少，若容量减至0，则删除该条边，同时加入一条反向边，反向边的容量为原始边的容量但是单位费用是原始边单位费用的相反数。继续搜索最短路径，重复上述步骤直至找不到最短路径。接下来以一个实际的算例说明该算法的执行步骤：



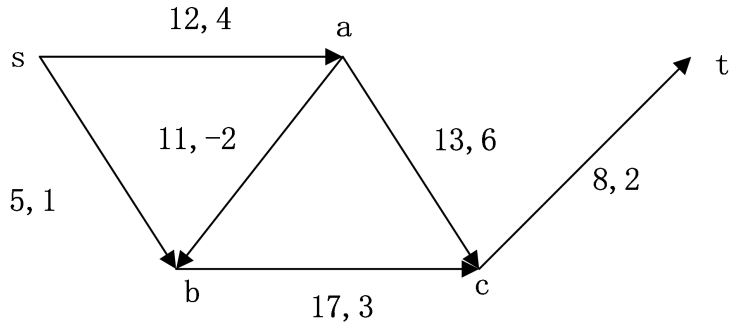
上图是一个简单的有向图，其中每条边上的两个数字分别表示边的容量和边的单价。首先以单价为权值可以得到：



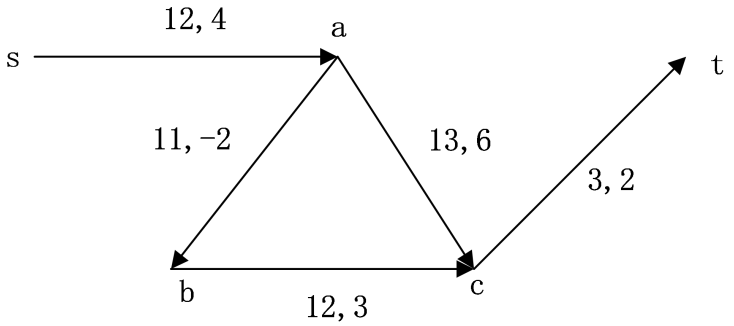
从起点s搜索至终点t的最短路径（最短路径的搜索算法见后文）得到$s \to b \to a \to t$，路径上的最小容量为11，更新原始图：



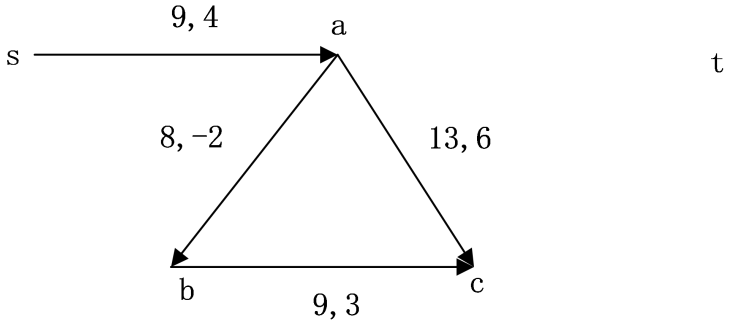
继续以单价为权值搜索最短路径得到$s \to a \to t$，路径上最小容量为3，更新：



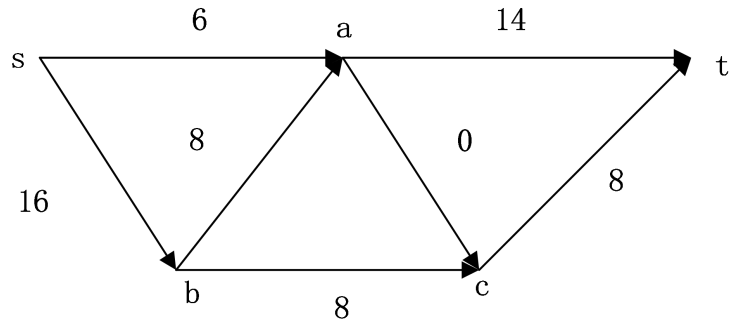
搜索最短路径 $s \to b \to c \to t$，路径最小容量为 5，更新：



搜索最短路径 $s \to a \to b \to c \to t$，路径最小容量为 3，更新：

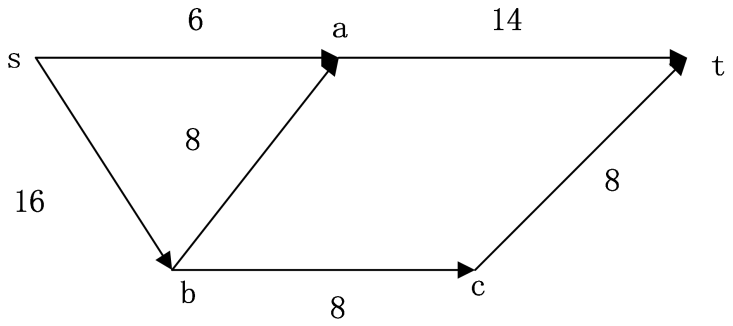


搜索不到路径了，算法结束，最终的流量分配如下：

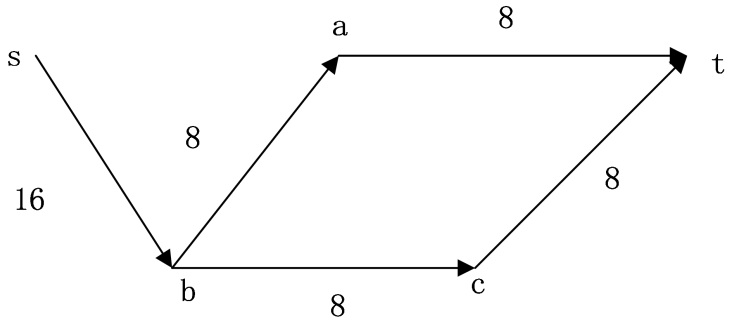


该算法中有些细节需要讨论：1.最短路径算法的选择，由于更新图的过程中会产生负权值的边，因此最短路径算法要适用于存在负权值的情况；2.与起点和终点相连的边一旦容量耗尽，不需要添加负权值的边，直接将边删除即可（针对起点与终点的反向边毫无意义）；3.该算法的适用对象为有向图，单源单汇的流网络，对于无向图的处理在编程时未认真考虑，解决办法具体可参考《算法导论》416 页的解决方式，对于多源多汇的情况，增加超级源点

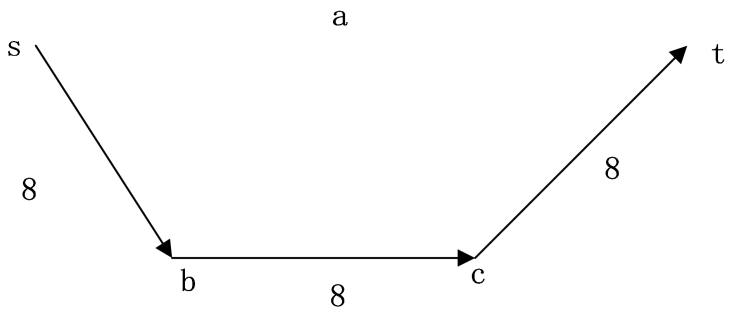
与超级汇点，从超级点到普通点的边容量为∞，费用为 0；4.完整的流量路径输出。算法运行完成之后，如果需要输出从起点到终点的流量路径，具体解决方式运用了图的深度优先遍历算法，以最终的流量分配图为基础，删除流量分配为 0 的边：



从 s 开始进行深度优先遍历至 t 得到 $s \to a \to t$，流量分配为 6，更新图：



重复深度优先遍历得到 $s \to b \to a \to t$，流量分配为 8，更新图：



最终 $s \to b \to c \to t$，流量分配为 8，结束。

1.2 最短路径搜索算法

最短路径搜索问题描述如下：给定图 $G(V,E)$，设其中任意一条边 (u,v) 都有 $w(u,v)$ 表示边的权值，给定起点 s，输出起点到其余点的最短路径。

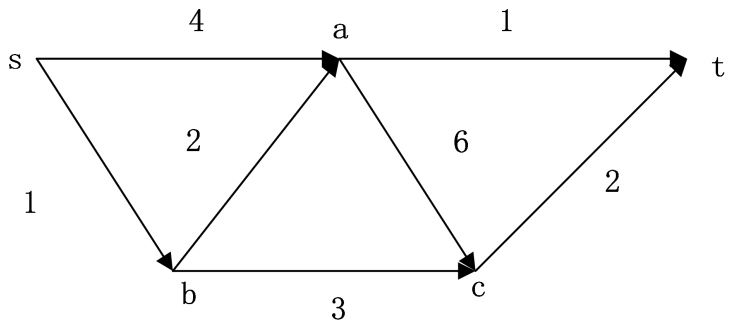
1.2.1 Dijkstra 算法

针对有向无环图且非负权值的算法，采用的思想是贪婪算法，假设起点 s 到特定点 v 的最短路径已知，设 u 为该路径上的一个顶点，那么此路径从起点 s 到顶点 v 的路径一定是最短路径，否则已知的从 s 到 v 最短路径不可能是最短的。因此根据贪婪策略，保证每一次寻找的路径均为最短路径。

算法的执行过程与图的广度优先遍历类似，从起点开始针对与起点相邻的点更新路径，

选出其中路径最短的点，继续更新，直到所有的点更新完成。

还是以费用流的有向图为例：



设置一个额外的表格保存中间结果：

顶点	d	known	p
s	0	F	-1
a	∞	F	-1
b	∞	F	-1
c	∞	F	-1
t	∞	F	-1

表格中 d 表示最短距离，$known$ 表示该点的最短路径是否已知，p 表示最短路径中该点的前一个顶点。初始化表格时，除起点外所有顶点的距离 d 置为 ∞，$known$ 置为 F，p 置为-1，起点的距离为 0。算法执行时从所有 $known$ 为 F 的顶点中选出距离最短的点 v，将 $known$ 置为 T，计算所有与该点相邻的且 $known$ 为 F 的点 u 的距离 $d_v + w(v,u)$，若这个值小于 d_u 则更新 $d_u = d_v + w(v,u)$，并且 $p_u = v$，反之则不更新。

还是以上图为例，首先起点 d 最小，取出，更新表格：

顶点	d	known	p
s	0	T	-1
a	4	F	s
b	1	F	s
c	∞	F	-1
t	∞	F	-1

然后 b 取出，更新表格：

顶点	d	known	p
s	0	T	-1
a	3	F	b
b	1	T	s
c	4	F	b
t	∞	F	-1

a 取出，更新表格：

顶点	d	known	p
s	0	T	-1
a	3	T	b
b	1	T	s
c	4	F	b
t	4	F	a

c 取出更新，t 取出更新，算法完成：

顶点	d	known	p
s	0	T	-1
a	3	T	b
b	1	T	s
c	4	T	b
t	4	T	a

从终点 t 进行回溯可以得到 s 到 t 的最短路径 $s \to b \to a \to t$，距离为 4。其中需要讨论的细节最重要的是保证每次取出的都是 $known$ 为 F 且 d 最小的顶点，优化的方式是使用最小堆（还有更先进的斐波那契堆）。由于赛题中主要寻找的是点到点的最短路径，因此终点 t 的 $known$ 一旦置为 T 算法就可以结束了。没有路径可找的标志是最小堆为空。

1.2.2 SPFA 算法（Shortest Path Fast Algorithm）（要写的话又要画图加画表，不想写了）

该算法用于处理带负权值的最短路径搜索问题，其中的松弛操作在我的理解中依然是使用广度优先，因为该算法的实现离不开队列这种数据结构，通过对队列的改进可以进一步提升该算法的效率，与 Dijkstra 算法相比唯一的优势就在于解决负权值问题，理论上的时间复杂度比不上 Dijkstra 算法，在最大流最小费用算法中使用必须遍历到整张图，而 Dijkstra 算法只需要确定目标点的最短路径就可以提前结束，因此稳定性上 SPFA 算法被 Dijkstra 算法完爆。在实验过程中可以明显感觉到 SPFA 算法执行时会出现卡顿现象。

1.3 启发式搜索

启发式搜索算法用于搜索服务器部署的位置，待选的算法如下：模拟退火、遗传、蚁群、粒子群、禁忌搜索。前两者的实现较为简单。启发式搜索的主要问题在于初始情况与参数的设置。特别是遗传算法的参数较多，例如种群的规模、交叉概率、变异概率、选择方式等。同时遗传算法极有可能陷入局部最优解，因此参数可以随着世代增加而改变，即自适应的遗传算法，遗传算法主要是选择、交叉、变异三个过程的不断循环，其中的细节非常之多，参数和方式的设置也没有固定的思路，这也是启发式搜索面临的问题。

2. Java 实现细节

2.1 Collection 类

该类是使用非常频繁的类，List 相当于长度可变的数组，Map 存储键值对，List 还有队列和优先队列的实现，Collection 的 sort 方法用于排序。可以说几乎所有的数据结构 Java 已经实现好了。但是 List 又分为 ArrayList、LinkedList，Map 分为 HashMap、TreeMap。虽然操作大同小异，不同的数据结构对应相同操作的效率差别很大，这一点只能在使用过程中具体摸索，现在已经清楚的是可以把 List 看作线性表，ArrayList 理解为顺序表，LinkedList 理解为链表，这两者的差异套用数据结构的知识就行了，需要说明的是，如果能够用数组解决问题那么尽量使用数组，数组的效率比上述所有对象的效率都高。Map 和 Set 的区别只能够在以后用到时具体分析。

2.2 序列化(Serializable)深复制（这部分你更清楚，你可以完善一下）

这是 Java 里面深复制的一种简易方式，对于小型对象非常好用，具体实现时要注意序列化的类成员变量如果是对象也必须实现序列化，常用的 Collection 类实现了序列化。但是当对象较大并且需要多次深复制时，重写 clone 方法是更好的选择。

3. 展望一下

44 名感觉还行，反正准备明年再来一次，从刚开始上不了榜到之后想冲进 32，这个过程还是很有意思的。感谢凡哥不停调试与提交代码。最后没有 hold 住的原因总结如下：1.编程语言的问题，Java 跑的就是比 C++慢，最大规模一次费用流可能需要 500ms，C++讲道理只需要 50ms，启发式搜索的次数如果是别人的 1/10 那劣势就很大了。2.费用流算法的问题，在给出最佳服务器分布的情况下得不到极限费用，其中的原因至今都未找到，最短路径算法是一个因素，费用流算法本来是应用于有向图，但是我们强制应用到无向图，本身的正确性没有验证，这可能是一个隐患。3.费用流算法是绝对能够改进的，最大流问题有更好的算法，算法导论 26 章中称为推送-重贴标签算法，甚至还有前置重贴标签算法，我们使用的图顶点数目远小于边的数目，因此推送-重贴标签算法在解决最大流问题上更具优势，以下是截取算法导论中的原文：

到目前为止，许多渐近效率很高的最大流算法都是推送-重贴标签算法，最大流算法的最快实现也是基于推送-重贴标签方法。推送-重贴标签方法还能有效解决其他流问题，如最小成本流问题。

有理由猜想最小费用流问题有更快捷并且更加准确的实现方法，有的队伍可以在 10ms 内完成一次费用流问题，这也佐证了猜想。（推送-重贴标签算法值得研究并且具体实现一下）