通用思路：

1） 读入：Q，D_client_moment，server[ ].band_limit，邻接表和邻接矩阵，拷贝原始需求到 DO_client_moment（永远只读不改）

2）计算 percent95，percent5=T-percent95=每个服务器的免费机会次数

3）将所有 client 的 neighbor 队列排序，通过比较 band_limit 或者 demand 或者 degree 等进行预分配：将所有服务器按一定顺序排序排序

4）对一个服务器 s_id，统计它的邻居需求之和最大的 percent5 个时刻，就在这些时刻中使用免费机会。
对于选出的一个时刻 t，创建一个堆，来维护所有邻居客户的剩余需求。循环：{将一个小片资源分给剩余需求最大的邻居 c_id，如果 c_id 的需求仍不为 0，则继续入堆。} 退出条件：要么堆为空（所有邻居的剩余需求都为 0），要么 s_id 已经满载
当一个服务器的 percent5 次免费机会都处理完之后，更新计费序列，轮到下一个服务器进行预分配

5）拷贝预分配完成之后的 D 到 D1，X 到 X1，保留现场，为二次规划做准备

6）初次分配：按照一定顺序排列（时间，需求，空间等）
将所有客户排序，neighbor[0] 的容量大的客户在前，相同则度小的在前。

```
for(排好序的 T 个时刻)
{
    for(排好序的 M/N 个客户/服务器)
    {
        对于一个客户 c_id
        先使用邻居的免费空间，邻居没有则推流（把远亲的免费空间传递过来）
        c_id 仍有需求，按排好的 neighbor 顺序找它的责任人，能放得下就放
    }
    维护所有服务器的计费序列，注意在预分配时已经入队的不可再入，且 0 永远不能入队
}
```

7）二次规划：在第一次规划时，不能提前知道每个服务器的 p95 免费空间，而是一点点突破 95 成本，并更新免费空间大小。
二次规划时，利用 D1，X1 等保留好的现场，仿佛回到初次分配的时候，只是已经知道了初次分配得到的每个服务器的 p95 成本。相当于在提前知道服务器免费空间大小的前提下，重新来一次“初次分配”

8）迁移算法：随机枚举服务器进行流量迁移，同时维护计费序列并选择可以降低费用的方案进行迁移
多次调用迁移算法，直到接近 TLE

启发式预分配和正式分配：

在实际问题中，对于成本影响幅度最大的因素是主体分配方案，由于 np 问题无法通过固定算法寻得最优解，因此在实际比赛中，由于数据集特征的不同，算法鲁棒性和算法效果往往不可兼得，所以为了在有限的尝试次数中获得较好效果，需要设计多种解决方案。

1） 计费队列 fare_sequence：
前提：
根据题目要求，需要在分配过程中实时计算 95/90 百分位流量，以及 96/91 百分位以上的队列剩余空间，和 94/89 百分位在不超过 95/90 百分位流量情况下可分配的流量。

算法：
使用优先队列存储所有时刻所分配的流量，同时动态更新以上数据

2） 预分配
前提：
在正式进行分配前（指分配结果计入成本前），可对所有服务器的 5%的时刻进行预分配，使其尽可能多承载流量，这部分流量不计入成本

算法：
对于每个服务节点，先在每个时刻模拟演练分配，再根据每个时刻的装入效果，决定真正在哪些时刻进行预分配
对服务器 S 进行排序（此处将排序算法 cmp_server_for_pre_allocate 设为超参数，可进行调整），确定服务器进行预分配的顺序。

遍历排序好的服务器 S
对于当前服务器 Si，遍历所有时刻 T
对于当前时刻 t，令此服务器将尽可能多的流量分配给相连的客户节点 C
根据此客户节点 Cj 当前流量需求对其流量种类 P（片）进行排序（此处排序算法 cmp_Node_piece_for_pre_allocate 设为超参数，可进行调整）
遍历此客户机要求的每种流量 Pk
分配流量 Si,Cj,Pk
记录分配结果，同时更新保存场景上下文
所有时刻分配结束后，根据分配结果，选取分配流量最大的 5%作为结果保留，回滚其余分配结果

在对所有服务器进行预分配结束后，对于每个服务器维护并更新计费序列 fare_sequence

Addition：
a. 在进行预分配前可以加上以先遍历客户节点 C 后遍历服务器 S 的模拟分配算法，可以先将客户节点的流量需求大片 P 进行模拟分配，查看某时刻各服务器的流量压力，不过这种方法并不是所有时刻都是正向优化效果
b. 在预分配遍历至服务器时可以对目前服务器可分配流量的需求片 CjPk 进行同质化排序来进行优化，避免一个客户节点在此时刻只分配需求给一个服务器

3） 正式分配

前提：
在预分配结束后，对于所有时刻剩余的所有客户节点的流量需求，进行分配，每次分配都有改变和不改变成本的结果。

算法：
delta_Cost_if_add_piece：计算此流量加入此服务器所导致的费用增加（需要运用此前已经维护并更新的 fare_sequence 的信息）

对于时间 T，服务器 S，客户节点 C 和流量需求 P（算作一类），在算法中有三层循环，对于主分配算法，这三层循环的顺序皆可调换，其结果根据数据集特征也不尽相同，我们依照类似预分配的遍历/排序算法，选取了以不同标准对服务器/客户节点/流量进行排序的结果进行初次分配，实际运用中，我们采取了两种方案：

第一种方案：将客户节点 C 算作循环，其中每个客户的需求 P 算作内循环，在分配完成后，观察到还有许多流量小片突破了服务器的 95%流量导致费用增加。
第二种方案：将客户节点的流量需求 P 同质化算作循环，同时按照可优化的比较算法进行排序，在大部分情况下相对第一种方法要更优。

推流算法：

为了保证在流量压力较大的情况下，使得流量分配有解，需要在分配过程中对于无法获得流量的用户节点进行搜索算法，达到可行解的效果

1）前提：
假设有一个服务器组 S，客户机组 C，对于某客户机 Ci 若有需求尚未满足，而此时存在服务器 Sj 仍有可分配空间，可以尝试用推流将 Ci 的需求间接分配给 Sj

2）算法：
在推流由客户机到服务器时，方法为：
a> 若客户机为有需求客户机 Ci，则分配给推流路径上的第一个服务器
b> 对于中途客户机 Ck（即以满足客户机），若对于当前遍历的服务器 Sk，Ck 未将全部需求分配给 Sk，则增大 Sk 分配给 Ck 的宽带

由服务器到客户机时，方法为：
a> 若服务器为有空余服务器 Sj，则增大其对推流路径上的第一个客户机的分配。
b> 对于中途服务器 Sk(即满载服务器)，若对于当前遍历的客户机 Ck，Sk 有分配宽带给 Ck，则减小分配给 Ck 的宽带

伪代码：
dfs 客户机到服务器：

标记当前客户机 u 为已访问
遍历其邻接服务器（设当前服务器为 v）
若 v 已经遍历过，continue
若 u 全部需求由 v 承担，continue
更改流为通道大小的最小值
执行 dfs 服务器到客户机，若返回 true：
增加 v 分配给 u 的宽带
若 u 为有需求客户机，更新需求
返回 true

dfs 服务器到客户机：
标记当前服务器 v 已访问
若当前服务器有剩余空间，则更新流最终大小，并更新服务器使用情况，返回 true
遍历其邻接客户机（设当前客户机为 u）
若 u 已经遍历过，continue
若服务器 v 没有分配宽带给 u，continue
更改流为通道大小的最小值
执行 dfs 客户机到服务器，若返回 true
减少 v 分配给 u 的宽带
返回 true

迁移算法：

为了在 np 问题中接近最优解，选择在启发式预分配和初次分配完成后进行随机化的流量迁移以降低总费用。

算法简介：
1）单次迁移：

打乱服务器顺序（随机）
记录服务器 n 在序列中的位置
按照打乱后的随即顺序开始遍历服务器
遍历此服务器相邻客户机
遍历片并找到可重分配片(并保证目标服务器在此服务器后面以达到剪枝效果并防止重复迁移)
同时选择不增大目标 95 成本的方案进行分配

2）二次分配：

设定迁移时间上限
初始化随机打乱顺序
在时间限定内重复执行迁移