



# 反悔贪心



作者

Asukaa\_ ✓

发布时间

2025-11-19 12:41

分类

个人记录

反悔贪  
第一次...  
——关于  
题目  
分析  
代码  
总结

## 第一次周报

### ——关于双周赛E题的总结

#### 题目

E 社团招新分配

#### 题目描述

小 L 是学校算法协会的成员。在今年的学校社团招新中，小 L 一共招收了 n 个新成员，其中 n 为偶数。现在小 L 希望将他们分到协会不同的部门。

算法协会共设有三个部门，其中第 i ( $1 \leq i \leq n$ ) 个新成员对第 j ( $1 \leq j \leq 3$ ) 个部门的满意度为  $v$ 。定义一个分配方案的满意度为所有新成员对分配到的部门的满意度之和，也就是说，若将第 i ( $1 \leq i \leq n$ ) 个新成员分配到了第 d ( $i \in \{1, 2, 3\}$ ) 个部门，则该分配方案的满意度为  $\sum_{i=1}^n v_{i,d}$ 。

小 L 不希望某一个部门的新成员数量过多。具体地，他要求在分配方案中，不存在一个部门被分配多于  $2n$  个新成员。你需要帮助小 L 求出，满足他要求的分配方案的满意度的最大值。

**输入格式** 本题包含多组测试数据。

输入的第一行包含一个正整数 t，表示测试数据组数。

接下来依次输入每组测试数据，对于每组测试数据：

第一行包含一个正整数 n，表示新成员的数量。第  $i+1$  ( $1 \leq i \leq n$ ) 行包含三个非负整数  $a_{i,1}, a_{i,2}, a_{i,3}$ ，分别表示第 i 个新成员对第 1, 2, 3 个部门的满意度。输出格式 对于每组测试数据，输出一行一个非负整数，表示满足小 L 要求的分配方案的满意度的最大值。

#### 分析

本题一眼看去好像可以使用贪心算法，但普通的贪心却又难以做出本题。

不妨换个思路，我们先使用贪心算出**不符合限制条件**的最优解，最后再朝着限制条件**修改前面的选择**，每次修改取对最优解影响最小的，直到修改到符合题目约束条件。

这就是**反悔贪心**的思路

#### 代码

**不按题目约束贪心选择最优**



```

int best;
if(stu[i][1] >= stu[i][2] && stu[i][1] >= stu[i][3]) best = 1;
else if(stu[i][2] >= stu[i][3] && stu[i][2] >= stu[i][1]) best = 2;
else best = 3;
sum += stu[i][best];
cnt[best]++;
rec[best].push_back(i); //记录每组被谁选择了
}

```

## 如果不符合约束 则进行反悔 取亏损最小

```

vector<int> loss_;
int k = 0; //一定要初始化
for(int i = 1; i <= 3; i++)
{
    if(cnt[i] > n/2) //如果超出n/2
    {
        k = cnt[i] - n/2; //超出人数
        for(int j = 0; j < cnt[i]; j++)
        {
            int loss = INT_MAX;
            int x = rec[i][j]; //第几个 stu
            for(int u = 1; u <= 3; u++)
            {
                if(u == i) continue;
                loss = min(stu[x][i] - stu[x][u], loss); //记录返回后亏损
            }
            loss_.push_back(loss);
        }
    }
}
sort(loss_.begin(), loss_.end()); //排序 好选择最小的几个进行反悔
for(int i = 0; i <= k - 1; i++) sum -= loss_[i]; //有几个不符合 就减去最小的几个

```

## 完整代码

```

#include<bits/stdc++.h>
using namespace std;
int stu[100005][4];
long long sum = 0;
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    int t;
    cin >> t;
    while(t--)
    {
        sum = 0;
        vector<int> cnt(4, 0);
        vector<vector<int>> rec(4);
        int n;
        cin >> n;
        for(int i = 1; i <= n; i++)
        {
            cin >> stu[i][1] >> stu[i][2] >> stu[i][3];
        }
        for(int i = 1; i <= n; i++)
        {

```



```

        else best = 3;
        sum += stu[i][best];
        cnt[best]++;
        rec[best].push_back(i);
    }

vector<int> loss_;
int k = 0;//一定要初始化
for(int i = 1;i <= 3;i++)
{
    if(cnt[i] > n/2)//如果超出n/2
    {
        k = cnt[i] - n/2;//超出人数
        for(int j = 0;j < cnt[i];j++)
        {
            int loss = INT_MAX;
            int x = rec[i][j];//第几个 stu
            for(int u = 1;u <= 3;u++)
            {
                if (u == i) continue;
                loss = min(stu[x][i] - stu[x][u] , loss);//记录返回后亏损
            }
            loss_.push_back(loss);
        }
    }
}
sort(loss_.begin(),loss_.end());//排序 好选择最小的几个进行反悔
for(int i = 0;i <= k - 1;i++) sum -= loss_[i];//有几个不符合 就减去最小的几个
cout << sum << endl;
}
}

```

## 总结

反悔贪心能解决什么样的问题？反悔贪心通常用来解决一类特殊的优化问题，尤其是调度问题和选择问题。这类问题通常具有以下一个或多个特征：

### 1、带截止时间的任务调度问题：

典型问题：有若干个任务，每个任务有一个截止时间和完成收益。在任意时刻只能做一个任务，问如何安排能获得最大总收益。

为什么需要反悔：一开始我们可能选择做收益高但截止时间早的任务，但后来发现一个收益更高但截止时间晚的任务。如果没有反悔机制，我们就无法替换掉之前的次优选择。

### 2、在限制条件下选择元素，使总和最大/最小：

典型问题：从一堆数中选出至多k个数，使得它们的和最大。（这是一个简单例子，实际上反悔贪心用于更复杂的限制条件）。

为什么需要反悔：我们可能先选了几个大的正数，但后来发现如果放弃其中一个正数，就可以同时选两个更大的正数（或者放弃一个正数，可以避免选一个绝对值很大的负数）。

### 3、问题可以建模为“不断修正一个可能解”的过程：



总而言之，反悔贪心最适合解决 “**在做出不可撤销的承诺之前，有机会修正过去决策**” 的优化问题。

作者：Asukaa\_ 创建时间：2025-11-19 12:41:03



1



1



不推荐



编辑

## 评论区

### 发表评论

发表一条友善的评论吧！

发表

0条评论

默认排序



[关于洛谷](#) · [帮助中心](#) · [用户协议](#) · [联系我们](#) · [小黑屋](#) · [陶片放逐](#) · [社区规则](#) · [招贤纳才](#)

© 2013-2025 洛谷. All rights reserved.

增值电信业务经营许可证 沪B2-20200477

[沪ICP备18008322号](#)