前缀和，差分，树状数组

前缀和

前缀和可以简单理解为「数列的前 n 项的和」，是一种重要的预处理方式，能大大降低查询的时间复杂度

例题：

[蓝桥杯 2022 省 A] 求和

题目描述

给定 n 个整数 $a_1, a_2, \cdots, a_n$, 求它们两两相乘再相加的和，即

$$S = a_1 \cdot a_2 + a_1 \cdot a_3 + \cdots + a_1 \cdot a_n + a_2 \cdot a_3 + \cdots + a_{n-2} \cdot a_{n-1} + a_{n-2} \cdot a_n + a_{n-1} \cdot a_n$$

输入格式

输入的第一行包含一个整数 n 。

第二行包含 n 个整数 $a_1, a_2, \cdots a_n$ 。

输出格式

输出一个整数 S，表示所求的和。请使用合适的数据类型进行运算。

样例 #1

样例输入 #1

```
4
1 3 6 9
```

样例输出 #1

```
117
```

提示

对于 30% 的数据, $1 \leq n \leq 1000, 1 \leq a_i \leq 100$ 。

对于所有评测用例, $1 \leq n \leq 2 \times 10^5, 1 \leq a_i \leq 1000$ 。

题解

非常简单的前缀和模版题，大家应该都会。

$$S = a_1 \cdot (a_2 + a_3 + \cdots + a_n) + a_2 \cdot (a_3 + a_4 + \cdots + a_n) + \cdots + a_k * (a_{k+1} + a_{k+2} + \cdots + a_n) + \cdots + a_{n-1} * a_n$$

对于任意的$\sum_{i=j}^{n} a_i [2 \leq j \leq n]$可以通过前缀和快速求出

$$\sum_{i=j}^{n} a_i = sum[n] - sum[j-1]$$

时间复杂度$O(n)$

代码：

```cpp
#include <iostream>
#define int long long
using namespace std;
const int N = 2e5+5;
int n;
int a[N],sum[N];

signed main(){
    cin>>n;
    for(int i = 1;i<=n;i++){
        cin>>a[i];
        sum[i] = sum[i-1] + a[i];
    }
    int ans = 0;
    for(int i = 1;i<=n;i++){
        ans += a[i] * (sum[n] - sum[i]);
    }
    cout<<ans;
}
```

树上前缀和

$sum[i]$表示i点到根节点的权值总和

求树上前缀和代码：

```cpp
int sum[N],a[N]//节点权值;
//预处理sum[root] = a[root]
void dfs(int u){
    for(int v : u的所有子节点){
        sum[v] = sum[u] + a[v];
        dfs(v);
    }
}
```

例题：

[蓝桥杯 2022 国 B] 机房

题目描述

这天，小明在机房学习。

他发现机房里一共有 n 台电脑，编号为 1 到 n，电脑和电脑之间有网线连接，一共有 $n-1$ 根网线将 n 台电脑连接起来使得任意两台电脑都直接或者间接地相连。

小明发现每台电脑转发、发送或者接受信息需要的时间取决于这台电脑和多少台电脑直接相连，而信息在网线中的传播时间可以忽略。比如如果某台电脑用网线直接连接了另外 d 台电脑, 那么任何经过这台电脑的信息都会延迟 d 单位时间 (发送方和接收方也会产生这样的延迟，当然如果发送方和接收方都是 同一台电脑就只会产生一次延迟)。

小明一共产生了 m 个疑问：如果电脑 u_i 向电脑 v_i 发送信息，那么信息从 u_i 传到 v_i 的最短时间是多少?

输入格式

输入共 $n+m$ 行，第一行为两个正整数 n, m 。

后面 $n-1$ 行，每行两个正整数 x, y 表示编号为 x 和 y 的两台电脑用网线 直接相连。

后面 m 行，每行两个正整数 u_i, v_i 表示小明的第 i 个疑问。

输出格式

输出共 m 行，第 i 行一个正整数表示小明第 i 个疑问的答案。

样例 #1

样例输入 #1

```
4 3
1 2
1 3
2 4
2 3
3 4
3 3
```

样例输出 #1

```
5
6
1
```

提示

【样例说明】

这四台电脑各自的延迟分别为 $2, 2, 1, 1$ 。

对于第一个询问, 从 2 到 3 需要经过 $2, 1, 3$, 所以时间和为 $2 + 2 + 1 = 5$。对于第二个询问，从 3 到 4 需要经过 $3, 1, 2, 4$，所以时间和为 $1 + 2 + 2 + 1 = 6$。

对于第三个询问，从 3 到 3 只会产生一次延迟, 所以时间为 1。

【评测用例规模与约定】

对于 30% 的数据，保证 $n, m \leq 1000$;

对于 100% 的数据，保证 $n, m \leq 10^5$。

题意

给定一颗n个节点的树，m个询问，求给定两个节点间的点权和，每个节点的点权为该点的度

题解

树上两点间有且只有一条路径，设两点为p和q，则路径可以表示为$p->lca(p,q)->q$,所以我们需要查询的是从p到$lca(p,q)$的点权和加上$lca(p,q)$到q的点权和。这两段区间的点权和显然可以通过前缀和预处理，设$sum[i]$表示从根节点到i的点权和，答案可以表示为：

$$ans = sum[p] - sum[lca(p,q)] + sum[q] - sum[lca(p,q)] + a[lca(p,q)];$$

两点的lca我们可以通过倍增,重链剖分,$tarjan$等方法求得
复杂度： $O(nlogn)$

如果你不会这些求lca的方法，那可以用暴力求lca

操作步骤：

求出每个结点的深度；
询问两个结点是否重合，若重合，则LCA已经求出；
否则，选择两个点中深度较大的一个，并移动到它的父亲。

实例：

```
int LCA(int x,int y)
{
        while(x!=y)
        {
                if(depth[x]>=depth[y]) x=fa[x];
                else y=fa[y];
        }
        return x;
}
```

因为复杂度达到了$O(nm)$的级别，所以只能混到三十分。

代码（倍增求lca）：

```cpp
#include <iostream>
#include <algorithm>
#include <cstring>
#include <vector>
#include <set>
#include <map>
#include <cmath>
#include <stack>
#include <queue>

using namespace std;
const int N =1e6+10;
vector<int> g[N<<1];
int sum[N],a[N],dep[N],fa[N][20],n,m;
void dfs(int u,int fath)
{
        dep[u]=dep[fath]+1;
        fa[u][0]=fath;
        for(int i=1;(1<<i)<=dep[u];i++)
                fa[u][i]=fa[fa[u][i-1]][i-1];
        for(int i=0;i<g[u].size();i++)
        {
                int v=g[u][i];
                if(v==fath)
                        continue;
                sum[v]=sum[u]+a[v];
                dfs(v,u);
        }
}
int lca(int x,int y)
{
        if(dep[x]<dep[y])
                swap(x,y);
        int d=dep[x]-dep[y];
        for(int i=0;i<=log2(n);i++)
                if((1<<i)&d)
                        x=fa[x][i];
        if(x==y)
                return x;
        for(int i=log2(n);i>=0;i--)
        {
                if(fa[x][i]!=fa[y][i])
                        x=fa[x][i],y=fa[y][i];
        }
        return fa[x][0];
}
signed main()
{
        cin>>n>>m;
        for(int i=1,u,v;i<n;i++)
                cin>>u>>v,g[u].push_back(v),g[v].push_back(u);
        for(int i=1;i<=n;i++)
                a[i]=g[i].size(),sum[i]=g[i].size();
        dfs(1,0);
        for(int i=1;i<=m;i++)
        {
                int u,v;
                cin>>u>>v;
                cout<<sum[u]+sum[v]-2*sum[lca(u,v)]+a[lca(u,v)]<<endl;
        }
}
```

暴力代码（30分）：

```cpp
#include <iostream>
#include <algorithm>
#include <cstring>
#include <vector>
#include <set>
#include <map>
#include <cmath>
#include <stack>
#include <queue>

using namespace std;
const int N =1e6+10;
vector<int> g[N<<1];
int sum[N],a[N],dep[N],fa[N],n,m;
void dfs(int u,int fath)
{
        dep[u]=dep[fath]+1;
        fa[u]=fath;
        for(int i=0;i<g[u].size();i++)
        {
                int v=g[u][i];
                if(v==fath)
                        continue;
                sum[v]=sum[u]+a[v];
                dfs(v,u);
        }
}
int lca(int x,int y)
{
        while(x!=y)
        {
                if(dep[x]>=dep[y]) x=fa[x];
                else y=fa[y];
        }
        return x;
}
signed main()
{
        cin>>n>>m;
        for(int i=1,u,v;i<n;i++)
                cin>>u>>v,g[u].push_back(v),g[v].push_back(u);
        for(int i=1;i<=n;i++)
                a[i]=g[i].size(),sum[i]=g[i].size();
        dfs(1,0);
        for(int i=1;i<=m;i++)
        {
                int u,v;
                cin>>u>>v;
                cout<<sum[u]+sum[v]-2*sum[lca(u,v)]+a[lca(u,v)]<<endl;
        }
}
```

二维前缀和

二维前缀和建立在一维前缀和之上，可以求一个矩阵内一个任意的子矩阵的数的和。
核心代码：

```cpp
    sum[i][j] = sum[i-1][j] + sum[i][j-1] - sum[i-1][j-1] + a[i][j];
```

[蓝桥杯 2018 国 B] 搭积木

题目描述

小明对搭积木非常感兴趣。他的积木都是同样大小的正立方体。

在搭积木时，小明选取 m 块积木作为地基，将他们在桌子上一字排开，中间不留空隙，并称其为第 0 层。

随后，小明可以在上面摆放第 1 层，第 2 层，……，最多摆放至第 n 层。摆放积木必须遵循三条规则：

规则 1：每块积木必须紧挨着放置在某一块积木的正上方，与其下一层的积木对齐；

规则 2：同一层中的积木必须连续摆放，中间不能留有空隙；

规则 3：小明不喜欢的位置不能放置积木。

其中，小明不喜欢的位置都被标在了图纸上。图纸共有 n 行，从下至上的每一行分别对应积木的第 1 层至第 n 层。每一行都有 m 个字符，字符可能是 `.` 或 `X`，其中 `X` 表示这个位置是小明不喜欢的。

现在，小明想要知道，共有多少种放置积木的方案。他找到了参加蓝桥杯的你来帮他计算这个答案。

由于这个答案可能很大，你只需要回答这个答案对 $1000000007(10^9+7)$ 取模后的结果。

注意：地基上什么都不放，也算作是方案之一种。

输入格式

输入数据的第一行有两个正整数 n 和 m，表示图纸的大小。

随后 n 行，每行有 m 个字符，用来描述图纸。每个字符只可能是 `.` 或 `X`。

输出格式

输出一个整数，表示答案对 10^9+7 取模后的结果。

样例 #1

样例输入 #1

```
2 3
..X
.X.
```

样例输出 #1

```
4
```

提示

【样例解释】

成功的摆放有（其中O表示放置积木）：

```
 1      2      3      4
..X    ..X    O.X    ..X
.X.    OX.    OX.    .XO
```

【数据约定】

对于 10% 的数据，$n=1$，$m\le 30$；

对于 40% 的数据，$n\le 10$，$m\le 30$；

对于 100% 的数据，$n\le 100$，$m\le 100$。

时限 1 秒, 256M。蓝桥杯 2018 年第九届国赛

题意

给定一个$n*m$的图纸，代表有n层每层m块的房子，需要按你放置防止积木，第$i+1$层第j块能放置积木当且仅当第i层第j块有积木且图纸上该块不为X,且同一层放置积木应该连续,求总的放置积木的方案树。

题解

该题可通过二维前缀和优化的动态规划求解。

设$dp[i][j][k]$为积木放置到了从上到下第i行，积木摆放在从第j块到第k块的方案数；
如果第i层的第j块到第k块可以摆放积木，那么第$i+1$层的从第$l[1<=l<=j]$到$r[k<=r<=m]$的块必须有积木放置,由此可得转移方程：

$$dp[i][j][k]=\sum_{l=1}^{j}\sum_{r=k}^{m}dp[i+1][l][r]$$

对于$\sum_{l=1}^{j}\sum_{r=k}^{m}dp[i+1][l][r]$ 可以通过二维前缀和预处理

最终答案为

$$ans=1+\sum_{i=1}^{n}\sum_{i=1}^{m}\sum_{j=l}^{m}dp[i][j][k]$$

其中1为什么也不放的方案数
时间复杂度： $O(nm^2)$

代码：

```
#include<bits/stdc++.h>
using namespace std;
const int mod=1e9+7;
int num[110][110];char s[110];
long long dp[110][110][110],sum[110][110];
int main(){
        int n,m;scanf("%d%d",&n,&m);
        for(int i=1;i<=n;i++){
                scanf("%s",s+1);
                for(int j=1;j<=m;j++)
                        num[i][j]=num[i][j-1]+(s[j]=='X');//用前缀和来记录区间内是否有 X
        }
        long long ans=1;//一个积木都不摆放的方案数
        for(int l=1;l<=m;l++)
                for(int r=l;r<=m;r++){
                        dp[n][l][r]=(num[n][r]-num[n][l-1]==0);
                        ans=(ans+dp[n][l][r])%mod;//注意初始值也要加到 ans 上
                }
        for(int i=n-1;i>=1;i--){
                for(int l=1;l<=m;l++)
                        for(int r=1;r<=m;r++)
                                sum[l][r]=(dp[i+1][l][r]+sum[l][r-1]+sum[l-1][r]-sum[l-1][r-1])%mod;//用前缀和预处理来优化时间复杂度
                for(int l=1;l<=m;l++)
                        for(int r=l;r<=m;r++) if(num[i][r]-num[i][l-1]==0){
                                dp[i][l][r]=(sum[l][m]-sum[0][m]-sum[l][r-1]+sum[0][r-1])%mod;
                                ans=(ans+dp[i][l][r])%mod;
                        }
        }
        printf("%lld",(ans+mod)%mod); //由于上面加加减减又要取余，可能出现负数，所以是 (ans+mod)%mod 而不是 ans
        return 0;
}
```

差分

例题：

[ABC338D] Island Tour

Problem Statement

The AtCoder Archipelago consists of N islands connected by N bridges. The islands are numbered from 1 to N, and the i-th bridge ($1 \leq i \leq N-1$) connects islands i and $i+1$ bidirectionally, while the N-th bridge connects islands N and 1 bidirectionally. There is no way to travel between islands other than crossing the bridges.

On the islands, a **tour** that starts from island X_1 and visits islands $X_2, X_3, \ldots, X_M$ in order is regularly conducted. The tour may pass through islands other than those being visited, and the total number of times bridges are crossed during the tour is defined as the **length** of the tour.

More precisely, a **tour** is a sequence of $l+1$ islands $a_0, a_1, \ldots, a_l$ that satisfies all the following conditions, and its **length** is defined as l:

- For all j $(0 \leq j \leq l-1)$, islands a_j and a_{j+1} are directly connected by a bridge.
- There are some $0 = y_1 < y_2 < \cdots < y_M = l$ such that for all k $(1 \leq k \leq M)$, $a_{y_k} = X_k$.

Due to financial difficulties, the islands will close one bridge to reduce maintenance costs. Determine the minimum possible length of the tour when the bridge to be closed is chosen optimally.

Constraints

- $3 \leq N \leq 2 \times 10^5$
- $2 \leq M \leq 2 \times 10^5$
- $1 \leq X_k \leq N$
- $X_k \neq X_{k+1}$ $(1 \leq k \leq M-1)$
- All input values are integers.

Sample Input 1

```
3 3
1 3 2
```

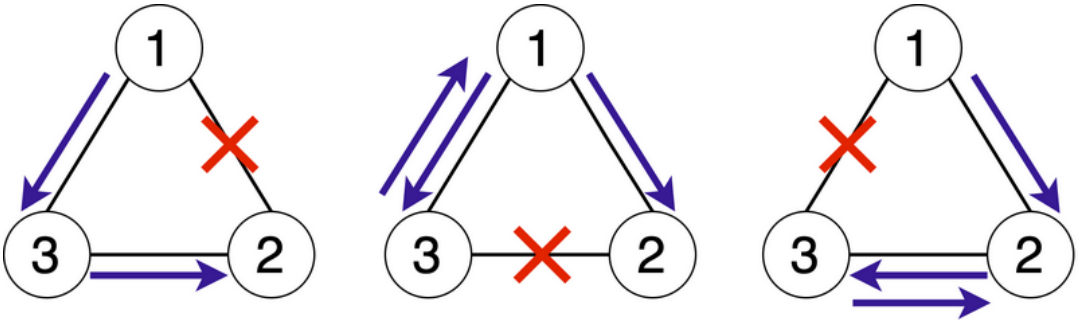
Sample Output 1

```
2
```

- If the first bridge is closed: By taking the sequence of islands $(a_0, a_1, a_2) = (1, 3, 2)$, it is possible to visit islands $1, 3, 2$ in order, and a tour of length 2 can be conducted. There is no shorter tour.
- If the second bridge is closed: By taking the sequence of islands $(a_0, a_1, a_2, a_3) = (1, 3, 1, 2)$, it is possible to visit islands $1, 3, 2$ in order, and a tour of length 3 can be conducted. There is no shorter tour.
- If the third bridge is closed: By taking the sequence of islands $(a_0, a_1, a_2, a_3) = (1, 2, 3, 2)$, it is possible to visit islands $1, 3, 2$ in order, and a tour of length 3 can be conducted. There is no shorter tour.

Therefore, the minimum possible length of the tour when the bridge to be closed is chosen optimally is 2.

The following figure shows, from left to right, the cases when bridges $1, 2, 3$ are closed, respectively. The circles with numbers represent islands, the lines connecting the circles represent bridges, and the blue arrows represent the shortest tour routes.



Sample Input 2

```
4 5
2 4 2 4 2
```

Sample Output 2

```
8
```

The same island may appear multiple times in $X_1, X_2, \ldots, X_M$.

Sample Input 3

```
163054 10
62874 19143 77750 111403 29327 56303 6659 18896 64175 26369
```

Sample Input 3

```
163054 10
62874 19143 77750 111403 29327 56303 6659 18896 64175 26369
```

题意

Atcoder 国里有 n 个小岛，第 i 个岛（$1 <= i < n$）和第 $i+1$ 个岛有一座桥连接，第 1 个岛和第 n 个岛有桥连接。
你想按照 $s_1, s_2, s_3, \ldots s_m$ 的顺序依次游览 m 座岛。只能通过桥从一个岛到另一个岛。
但现需要断开一座桥。问你任意选择一座桥断开的情况下，完成游览最少要通过多少次桥。
如果一座桥被多次计算，需要重复计算次数。

题解

这是一道带差分知识点的一个思维题。
从第s_i座岛到第s_{i+1}座岛有两条路径可选，一个是顺时针走一个是逆时针走。
当我们选择顺时针走，那么逆时针方向上的路径可以被拆除，因此逆时针方向上路径上的每一座桥加上顺时针走路径的长度。同理，顺时针上路径上每一座桥也要加上逆时针路径上的长度。通过这些操作，我们可以知道拆除每一座桥的答案。最终的答案就是值最小的桥。我们可以通过拆分计算每一座桥的值。

时间复杂度：$O(m)$

代码：

```cpp
#include <iostream>
#define endl "\n"
#define int long long
using namespace std;

const int N = 2e5+5;
int n,t;
int a[N],c[N];

void add(int l,int r,int k){
    c[l] += k;
    c[r + 1] -= k;
}

void solve(){
    int m;
    cin>>n>>m;
    for(int i = 1;i<=m;i++){
        cin>>a[i];
    }
    for(int i = 1;i<m;i++){
        int x = a[i],y = a[i+1];
        if(x > y) swap(x,y);
        int d = y - x;
        add(1,x-1,d);
        add(x,y-1,n-d);
        add(y,n,d);
    }
    int ans = 0x3f3f3f3f3f3f3f3f;
    for(int i = 1;i<=n;i++){
        c[i] += c[i-1];
        ans = min(ans,c[i]);
    }
    cout<<ans<<endl;
}

signed main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    solve();
}
```

[蓝桥杯 2022 国 C] 打折

题目描述

小蓝打算采购 n 种物品，每种物品各需要 1 个。

小蓝所住的位置附近一共有 m 个店铺，每个店铺都出售着各种各样的物品。

第 i 家店铺会在第 s_i 天至第 t_i 天打折，折扣率为 p_i，对于原件为 b 的物品，折后价格为 $\lfloor \frac{b \cdot p_i}{100} \rfloor$。其它时间需按原价购买。

小蓝很忙，他只能选择一天的时间去采购这些物品。请问，他最少需要花多少钱才能买到需要的所有物品。

题目保证小蓝一定能买到需要的所有物品。

输入格式

输入的第一行包含两个整数 n, m，用一个空格分隔，分别表示物品的个数和店铺的个数。

接下来依次包含每个店铺的描述。每个店铺由若干行组成，其中第一行包含四个整数 s_i, t_i, p_i, c_i，相邻两个整数之间用一个空格分隔，分别表示商店优惠的起始和结束时间、折扣率以及商店内的商品总数。之后接 c_i 行，每行包含两个整数 a_j, b_j，用一个空格分隔，分别表示该商店的第 j 个商品的类型和价格。商品的类型由 1 至 n 编号。

输出格式

输出一行包含一个整数表示小蓝需要花费的最少的钱数。

样例 #1

样例输入 #1

```
2 2
1 2 89 1
1 97
3 4 77 1
2 15
```

样例输出 #1

```
101
```

提示

对于 40% 的评测用例，$n, m \leq 500$，$s_i \leq t_i \leq 100$，$\sum c_i \leq 2000$；

对于 70% 的评测用例，$n, m \leq 5000$，$\sum c_i \leq 20000$；

对于所有评测用例，$1 \leq n, m \leq 10^5$，$1 \leq c_i \leq n$，$\sum c_i \leq 4 \times 10^5$，$1 \leq s_i \leq t_i \leq 10^9$，$1 < p_i < 100$，$1 \leq a_j \leq n$，$1 \leq b_j \leq 10^9$。

蓝桥杯 2022 国赛 C 组 I 题。

代码

```cpp
#include <bits/stdc++.h>
using namespace std;

// 2023 OneWan

const int MAXM = 100000 + 5;
int s[MAXM], t[MAXM], p[MAXM], c[MAXM]; // 对应题目输入的各数组
multiset<long long> st[MAXM]; // st[i] 为 物品 i 在所有商店的价格
vector<vector<pair<int, int>>> v(MAXM); // v[i][j] 为 商店 i 出售的 第 j 个物品

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    int n, m;
    cin >> n >> m;
    vector<int> time; // 用于离散化 时间
    for (int i = 0 ; i < m ; i++) {
        cin >> s[i] >> t[i] >> p[i] >> c[i];
        time.emplace_back(s[i]);
        time.emplace_back(t[i] + 1); // 打折是闭区间 所以需要+1才是没有打折截止
        for (int j = 0 ; j < c[i] ; j++) {
            int a, b; // 物品编号及原价
            cin >> a >> b;
            v[i].emplace_back(a, b);
        }
    }
    sort(time.begin(), time.end()); // 排序时间
    time.resize(unique(time.begin(), time.end()) - time.begin()); // 离散化时间
    auto get = [&](int t) {
        return lower_bound(time.begin(), time.end(), t) - time.begin();
    }; // 获取离散化后的下标
    int len = time.size();
    vector<vector<pair<int, int>>> startD(len), endD(len);
    // startD[i] 为 第 i 天 开始打折的物品编号 和 打折后的价格
    // endD[i] 为 第 i 天 结束打折的物品编号 和 打折后的价格
    for (int i = 0 ; i < m ; i++) {
        int starts = get(s[i]), ends = get(t[i] + 1); // 获取打折开始与结束时间离散化后的下标
        for (auto& [x, y] : v[i]) {
            int t = 1LL * y * p[i] / 100; // 打折后的价格
            st[x].insert(y); // 把物品原价放入
            startD[starts].emplace_back(x, t);
            endD[ends].emplace_back(x, t);
        }
    }
    long long temp = 0; // 用于存每天购买所有物品所用的价格
    for (int i = 1 ; i <= n ; i++) temp += *st[i].begin(); // 计算不进行打折时购买所有物品所用的价格
    long long ans = temp;
    for (int i = 0 ; i < len ; i++) {
        long long k = 0; // 打折与不打折对价格的贡献
        for (auto& [x, y] : startD[i]) { // 遍历当天所有开始打折的物品 打折前价格最小值为a, 打折后价格最小值为b, 贡献为b - a
            k -= *st[x].begin();
            st[x].insert(y);
            k += *st[x].begin();
        }
        for (auto& [x, y] : endD[i]) { // 遍历当天所有结束打折的物品 打折前价格最小值为a, 打折后价格最小值为b, 贡献为b - a
            k -= *st[x].begin();
            int t = st[x].count(y);
            st[x].erase(y);
            for (int j = 1 ; j < t ; j++) st[x].insert(y);
            k += *st[x].begin();
        }
        temp += k; // 加上贡献, 由前一段转移到后一段
        ans = min(ans, temp); // 找花费最小
    }
    cout << ans;
    return 0;
```

```
}
```

树状数组

例题

[蓝桥杯 2014 省 B] 小朋友排队

题目描述

n 个小朋友站成一排。现在要把他们按身高从低到高的顺序排列，但是每次只能交换位置相邻的两个小朋友。

每个小朋友都有一个不高兴的程度。开始的时候，所有小朋友的不高兴程度都是 0。

如果某个小朋友第一次被要求交换，则他的不高兴程度增加 1，如果第二次要求他交换，则他的不高兴程度增加 2（即不高兴程度为 3），依次类推。当要求某个小朋友第 k 次交换时，他的不高兴程度增加 k。

请问，要让所有小朋友按从低到高排队，他们的不高兴程度之和最小是多少。

如果有两个小朋友身高一样，则他们谁站在谁前面是没有关系的。

输入格式

输入的第一行包含一个整数 n，表示小朋友的个数。

第二行包含 n 个整数 $H_1, H_2 \cdots H_n$，分别表示每个小朋友的身高。

输出格式

输出一行，包含一个整数，表示小朋友的不高兴程度和的最小值。

样例 #1

样例输入 #1

```
3
3 2 1
```

样例输出 #1

```
9
```

提示

【样例说明】

首先交换身高为 3 和 2 的小朋友，再交换身高为 3 和 1 的小朋友，再交换身高为 2 和 1 的小朋友，每个小朋友的不高兴程度都是 3，总和为 9。

【数据规模与约定】

对于 10% 的数据，$1 \leq n \leq 10$；

对于 30% 的数据，$1 \leq n \leq 1000$；

对于 50% 的数据，$1 \leq n \leq 10000$；

对于 100% 的数据，$1 \leq n \leq 100000$，$0 \leq H_i \leq 1000000$。

时限 1 秒, 256M。蓝桥杯 2014 年第五届省赛

题解

对于每个小朋友，他们会和在他之前并且身高比他高的人换位子，也会和在他之后身高比他低的换，所以交换次数就等于两者的和。
设第i个小朋友的交换次数为cnt_i，则答案为

$$ans = \sum_{i=1}^{n}(cnt_i * (cnt_i + 1)/2)$$

如果按照以上的思路直接暴力，复杂度为$O(n^2)$,居然可以得60分。代码很简单就不放了。

我们发现，这道题本质上是求逆序对，对于逆序对，我们可以用树状数组或归并排序在$O(logn)$的复杂度内解决。我们只需要正着求一遍在反着求一遍即可知道每个小朋友需要交换的次数。复杂度$O(nlogn)$,完全可以通过本题。
代码(记得开long long)：

```
#include <iostream>
#include <algorithm>
#include <cstring>
#include <vector>
#include <set>
#include <map>
#include <cmath>
#include <stack>
#include <queue>
#define endl '\n'
#define int long long
using namespace std;
typedef pair<int,int> PII;
const int N = 1e5+5,M = 1e6+5;
int n,maxx;
int t[M],a[N],cnt[N];

int lowbit(int x){
    return x&-x;
}

void add(int i,int x){
    while(i<=M) t[i]+=x,i+=lowbit(i);
}
int sum(int i){
    int res=0;
    while(i>0) res+=t[i],i-=lowbit(i);
    return res;
}

signed main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cin>>n;
    for(int i = 1;i<=n;i++){
        cin>>a[i];
        a[i] += 1;
    }
    for(int i = 1;i<=n;i++){
        cnt[i] += i - sum(a[i]) - 1;
        add(a[i],1);
    }
    memset(t,0,sizeof t);
    for(int i = n;i>=1;i--){
        cnt[i] += sum(a[i]-1);
        add(a[i],1);
    }
    int ans = 0;
    for(int i = 1;i<=n;i++){
        ans += (cnt[i] + 1)*cnt[i]/2;
    }
    cout<<ans<<endl;
}
```

三元上升子序列

题目描述

Erwin 最近对一种叫 `thair` 的东西巨感兴趣。。。

在含有 n 个整数的序列 $a_1, a_2, \ldots, a_n$ 中，三个数被称作 `thair` 当且仅当 $i < j < k$ 且 $a_i < a_j < a_k$。

求一个序列中 `thair` 的个数。

输入格式

开始一行一个正整数 n,

以后一行 n 个整数 $a_1, a_2, \ldots, a_n$。

输出格式

一行一个整数表示 `thair` 的个数。

样例 #1

样例输入 #1

```
4
2 1 3 4
```

样例输出 #1

```
2
```

样例 #2

样例输入 #2

```
5
1 2 2 3 4
```

样例输出 #2

```
7
```

提示

样例2 解释

7 个 `thair` 分别是:

- 1 2 3
- 1 2 4
- 1 2 3
- 1 2 4
- 1 3 4
- 2 3 4
- 2 3 4

数据规模与约定

- 对于 30% 的数据 保证 $n \leq 100$;
- 对于 60% 的数据 保证 $n \leq 2000$;
- 对于 100% 的数据 保证 $1 \leq n \leq 3 \times 10^4$，$1 \leq a_i \leq 10^5$。

题解

这道题最简单直接的想法可以直接暴力枚举三个下标统计个数，时间复杂度$O(n^3)$，能得30分。

那我们能不能优化掉一维呢？我们可以只枚举下标j和k，用树状数组取维护所有下标j前小于$a[j]$的数的个数，时间复杂度$O(n^2logn)$，能得60分。

我们在进一步想，发现以下标k结尾的三元上升子序列来自所有以j结尾的二元上升子序列且$a[j]<a[k]$，这自然让我们想到了dp。

设 $dp[i][j]$ 为以j结尾的所有i元上升子序列，转移方程为：

$$dp[i][k]=\sum_{j<k,a[j]<a[k]} dp[i-1][j]$$

那怎么找到 $\sum_{j<k,a[j]<a[k]} dp[i-1][j]$ 呢，用树状数组在下标为$a[j]$的地方维护$dp[i-1][j]$就行了。
时间复杂度$O(nlogn)$
代码：

```
#include <iostream>
#include <algorithm>
#include <cstring>
#include <vector>
#include <set>
#include <map>
#include <cmath>
#include <queue>
#define int long long
using namespace std;
const int N = 3e4 + 5,M = 1e5 + 5;
int n,a[N],dp[4][N],t[M],ans;

int lowbit(int x){
    return x& -x;
}
void add(int i,int x){
    while(i<=M-1) t[i]+=x,i+=lowbit(i);
}
int sum(int i){
    int res=0;
    while(i>0) res+=t[i],i-=lowbit(i);
    return res;
}

signed main() {
        cin >> n;
        for(int i = 1; i <= n; i++) cin >> a[i];
        for(int i = 1; i <= n; i++) dp[1][i] = 1;
        for(int i = 2; i <= 3; i++) {
                memset(t, 0, sizeof t);
                for(int j = 1; j <= n; j++) {
                        dp[i][j] = sum(a[j]-1);
                        add(a[j], dp[i-1][j]);
                }
        }
        for(int i = 1; i <= n; i++) ans += dp[3][i];
        cout << ans << endl;
        return 0;
}
```

并查集

例题：

[蓝桥杯 2019 省 A] 修改数组

题目描述

给定一个长度为 N 的数组 $A=[A_1,A_2,\cdots A_N]$，数组中有可能有重复出现的整数。

现在小明要按以下方法将其修改为没有重复整数的数组。小明会依次修改 $A_2,A_3,\cdots,A_N$。

当修改 A_i 时，小明会检查 A_i 是否在 $A_1\sim A_{i-1}$ 中出现过。如果出现过，则小明会给 A_i 加上 1；如果新的 A_i 仍在之前出现过，小明会持续给 A_i 加 1，直到 A_i 没有在 $A_1\sim A_{i-1}$ 中出现过。

当 A_N 也经过上述修改之后，显然 A 数组中就没有重复的整数了。

现在给定初始的 A 数组，请你计算出最终的 A 数组。

输入格式

第一行包含一个整数 N。

第二行包含 N 个整数 $A_1, A_2, \cdots, A_N$。

输出格式

输出 N 个整数，依次是最终的 $A_1, A_2, \cdots, A_N$。

样例 #1

样例输入 #1

```
5
2 1 1 3 4
```

样例输出 #1

```
2 1 3 4 5
```

提示

对于 80% 的评测用例，$1 \leq N \leq 10000$。

对于所有评测用例，$1 \leq N \leq 10^5$，$1 \leq A_i \leq 10^6$。

蓝桥杯 2019 年省赛 A 组 H 题。

题解

首先我们想暴力模拟这个过程。当该元素及其之上的一段连续的元素前面均出现过时，将当前元素不断向上加是非常耗时的。最差的情况下这样的时间复杂度会达到$O(n^2)$级别。当然这差不多可以得80分。

我们可以想一想如何更快的找到下一个之前未出现过的数。可以想到我们可以把这一串之前出现过的连续数字较大值向比它小一的数连边，这是一颗树，根节点加一就是我们所要找的数。这很像并查集。所以我们试着用并查集写，最初每个数的父节点都是自己，当这个数被用过之后将其父节点变为父节点加一，这样之后再遇到相同的数就可以通过$find$函数直接找到所需数。用路径压缩并查集最坏情况下时间复杂度为$O(nlogn)$。
代码：

```cpp
#include <iostream>
#include <algorithm>
#include <cstring>
#include <vector>
#include <set>
#include <map>
#include <cmath>
#include <queue>
using namespace std;
int n;
int fa[100100];
int find(int x)
{
        if(x==fa[x]) return x;
        return fa[x]=find(fa[x]);
}
int main()
{
        int n;
        cin>>n;
        for (int i=1;i<100100;i++)
        {
                fa[i]=i;
        }
        int a;
        for (int i=1;i<=n;i++)
        {
                cin>>a;
                a=find(a);
                fa[a]=find(a)+1;
                cout<<a<<" ";
        }
        return 0;
}
```

[蓝桥杯 2020 省 AB1] 网络分析

题目描述

小明正在做一个网络实验。

他设置了 n 台电脑，称为节点，用于收发和存储数据。

初始时，所有节点都是独立的，不存在任何连接。

小明可以通过网线将两个节点连接起来，连接后两个节点就可以互相通信了。两个节点如果存在网线连接，称为相邻。

小明有时会测试当时的网络，他会在某个节点发送一条信息，信息会发送到每个相邻的节点，之后这些节点又会转发到自己相邻的节点，直到所有直接或间接相邻的节点都收到了信息。所有发送和接收的节点都会将信息存储下来。一条信息只存储一次。

给出小明连接和测试的过程，请计算出每个节点存储信息的大小。

输入格式

输入的第一行包含两个整数 n，m，分别表示节点数量和操作数量。节点从 1 至 n 编号。

接下来 m 行，每行三个整数，表示一个操作。

如果操作为 `1 a b`，表示将节点 a 和节点 b 通过网线连接起来。当 $a = b$ 时，表示连接了一个自环，对网络没有实质影响。

如果操作为 `2 p t`，表示在节点 p 上发送一条大小为 t 的信息。

输出格式

输出一行，包含 n 个整数，相邻整数之间用一个空格分割，依次表示进行完上述操作后节点 1 至节点 n 上存储信息的大小。

样例 #1

样例输入 #1

```
4 8
1 1 2
2 1 10
2 3 5
1 4 1
2 2 2
1 1 2
1 2 4
2 2 1
```

样例输出 #1

```
13 13 5 3
```

提示

对于 30% 的评测用例，$1 \leq n \leq 20$，$1 \leq m \leq 100$。

对于 50% 的评测用例，$1 \leq n \leq 100$，$1 \leq m \leq 1000$。

对于 70% 的评测用例，$1 \leq n \leq 1000$，$1 \leq m \leq 10000$。

对于所有评测用例，$1 \leq n \leq 10000$，$1 \leq m \leq 10^5$ ，$1 \leq t \leq 100$。

蓝桥杯 2020 第一轮省赛 A 组 J 题（B 组 J 题）。

题解

这题是一道比较明显的并查集题，操作一相当于将两个并查集合并，操作二相当于给并查集根节点加一个值，并查集内其他点$find$后加上根节点的值即可。

这题的重点是在合并时要按秩合并，要用层数大的并查集合并层数小的并查集，以此来优化时间复杂度。因为洛谷上此题数据较水，时间复杂度$O(nm\alpha(n))$依然可以通过。
代码：

```cpp
#include <iostream>
#include <algorithm>
#include <cstring>
#include <vector>
#include <set>
#include <map>
#include <cmath>
#include <queue>
using namespace std;
const int N=10005;
int f[N+10],dep[N+10];
int val[N+10],ans[N+10];

int find(int x) {
        if(f[x]==x) return x;
        return f[x]=find(f[x]);
}

int main () {
        int n,m;
        cin>>n>>m;
        for(int i=1; i<=n; ++i) f[i]=i,dep[i]=1;
        for(int i=1; i<=m; i++) {
                int op,x,y;
                cin>>op>>x>>y;
                if(op==1) {
                        x=find(x);
                        y=find(y);
                        if(x!=y) {
                                for(int i=1; i<=n; ++i) ans[i]+=val[find(i)];
                                for(int i=1; i<=n; ++i) val[i]=0;
                                if(dep[x]>dep[y])       swap(x,y);
                                f[x]=y,dep[y]+=dep[x];
                        }
                }
                if(op==2) x=find(x),val[x]+=y;
        }
        for(int i=1; i<=n; i++) cout<<ans[i]+val[find(i)]<<" ";
}
```

[CF1927F] Microcycle

题目描述

给定一个无向加权图，图中有 n个顶点和 m 条边，不含重边和自环。该图不一定连通。
如果图中的循环不经过同一顶点两次，也不包含相同的边两次，则该循环称为简单循环。
记一个简单环的权值是环上最小的边权，你需要找到任意一个全图中权值最小的一个简单环并输出它。

输入

The first line of the input contains a single integer t ($1 \le t \le 10^4$) — the number of test cases. Then follow the descriptions of the test cases.

The first line of each test case contains two integers n and m ($3 \le n \le m \le \min(\frac{n \cdot (n-1)}{2}, 2 \cdot 10^5)$) — the size of the graph and the number of edges.

The next m lines of the test case contain three integers u, v, and w ($1 \le u, v \le n$, $u \neq v$, $1 \le w \le 10^6$) — vertices u and v are connected by an edge of weight w.

It is guaranteed that there is at most one edge between each pair of vertices. Note that under the given constraints, there is always at least one simple cycle in the graph.

It is guaranteed that the sum of the values of m for all test cases does not exceed $2 \cdot 10^5$.

输出

For each test case, output a pair of numbers b and k, where:

- b — the minimum weight of the edge in the found cycle,

- k — the number of vertices in the found cycle.

On the next line, output k numbers from 1 to n — the vertices of the cycle in traversal order.

Note that the answer always exists, as under the given constraints, there is always at least one simple cycle in the graph.

样例 #1

样例输入 #1

```
5
6 6
1 2 1
2 3 1
3 1 1
4 5 1
5 6 1
6 4 1
6 6
1 2 10
2 3 8
3 1 5
4 5 100
5 6 40
6 4 3
6 15
1 2 4
5 2 8
6 1 7
6 3 10
6 5 1
3 2 8
4 3 4
5 3 6
2 6 6
5 4 5
4 1 3
6 4 5
4 2 1
3 1 7
1 5 5
4 6
2 3 2
1 3 10
1 4 1
3 4 7
2 4 5
1 2 2
4 5
2 1 10
3 1 3
4 2 6
1 4 7
2 3 3
```

样例输出 #1

```
1 3
1 2 3
3 3
6 4 5
1 5
4 2 1 6 3
1 4
1 4 3 2
3 3
2 3 1
```

题解

通过这道题主要学习一下如何通过并查集判环。

这道题让我们找到所有环里有最小边权的一个。很自然的我们可以想到按边权从大到小对边排序后，从1到m枚举可以成环的边，最后一个可以成环的边所在的环就是我们要找的环。那我们怎么判断一条边在不在环里呢，可以在枚举每条边时把两个端点所在的并查集合并，当一条边左右端点处在同一个并查集时，说明该边处于环内。

示例代码:

```
int find(int x) {
    if (x == fa[x]) return x;
    return fa[x] = find(fa[x]);
}

bool  merge(int x, int y) {
    int fx = find(x), fy = find(y);
    if (fx == fy) return 1;
    fa[fx] = fy;
    return 0;
}
int main{
        sort(a+1,a+1+m,[](edge x,edge y){return x.w > y.w;});
    int minx = 0x3f3f3f3f;
    int id = 0;
    for(int i = 1;i<=m;i++){
        if(merge(a[i].u,a[i].v) && a[i].w < minx){
            minx = a[i].w;
            id = i;
        }
        }
}
```

这样我们知道了那条在环中的权值最小的边，那怎么找到它所在的环呢?

其实很简单，我们只需要在建图时不把这条边加进去，从这条边的一个端点开始dfs,直到找到另一个端点，同时记录这个过程中走到了哪些点。
示例代码:

```
void dfs(int u,int v){
    vis[u] = 1;
    ans.push_back(u);
    if(u == v){
        cout<<ans.size()<<endl;
        for(int x : ans) cout<<x<<" ";
        cout<<endl;
        return;
    }
    for(int x : e[u]){
        if(!vis[x]) dfs(x,v);
    }
    ans.pop_back();
}
```

这样就能把这道题ac了

代码:

```cpp
#include <iostream>
#include <algorithm>
#include <cstring>
#include <vector>
#include <set>
#include <map>
#include <cmath>
#include <queue>
#define endl "\n"
using namespace std;

typedef pair<int,int> PII;
const int N = 2e5+5;
int n,t;
vector<int> e[N];
int vis[N];
struct edge{
    int u,v,w;
}a[N];
vector<int> ans;
int fa[N];

int find(int x) {
    if (x == fa[x]) return x;
    return fa[x] = find(fa[x]);
}

bool  merge(int x, int y) {
    int fx = find(x), fy = find(y);
    if (fx == fy) return 1;
    fa[fx] = fy;
    return 0;
}

void dfs(int u,int v){
    vis[u] = 1;
    ans.push_back(u);
    if(u == v){
        cout<<ans.size()<<endl;
        for(int x : ans) cout<<x<<" ";
        cout<<endl;
        return;
    }
    for(int x : e[u]){
        if(!vis[x]) dfs(x,v);
    }
    ans.pop_back();
}

void solve(){
    int m;
    cin>>n>>m;
    for(int i = 1;i<=n;i++){
        vis[i] = 0;
        fa[i] = i;
        e[i].clear();
    }
    ans.clear();
    for(int i = 1;i<=m;i++){
        int u,v,w;
        cin>>u>>v>>w;
        a[i] = {u,v,w};
    }
    sort(a+1,a+1+m,[](edge x,edge y){return x.w > y.w;});
    int minx = 0x3f3f3f3f;
    int id = 0;
    for(int i = 1;i<=m;i++){
        if(merge(a[i].u,a[i].v) && a[i].w < minx){
            minx = a[i].w;
```

```
            id = i;
        }
    }
    for(int i = 1;i<=m;i++){
        if(i == id) continue;
        e[a[i].u].push_back(a[i].v);
        e[a[i].v].push_back(a[i].u);
    }
    cout<<minx<<" ";
    dfs(a[id].u,a[id].v);
}

signed main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cin>>t;
    while(t--){
        solve();
    }
}
```