2022年第五届广西大学生程序设计竞赛（正式赛）题解

题目PDF

难度	签到题	普通题	中等题	难题
题号	A E H L	B G I	D F J	C K
状态	✔	✔	○	✘

> 参考资料：第五届GXCPC广西大学生程序设计竞赛 部分题解（无CDK）

签到题

A Arrangement

题目大意：

自定义排序。

最终难度低的放前面，难度一样则思维难度低放前面，都一样就题目序号小的放前面

解题思路：

自定义函数来进行排序

参考代码c++

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

bool cmp(vector<int>a, vector<int>b) {
    if (a[0] != b[0])
        return a[0] < b[0];
    if (a[1] != b[1])
        return a[1] < b[1];
    return a[2] < b[2];
}

int main() {
    int n;
    cin >> n;
    vector<int>a(n), b(n);
    vector<vector<int>>c(n);
    for (int i = 0; i < n; i++)
        cin >> a[i];
    for (int i = 0; i < n; i++) {
        cin >> b[i];
        c[i] = {a[i] *b[i], a[i], i};
```

```
    }
    sort(c.begin(), c.end(), cmp);
    for (int i = 0; i < n; i++)
        cout << (char)(c[i][2] + 'A') << " ";
    return 0;
}
```

E Envy-freeness

题目大意：

这题题目很长，但看完题目就能发现这是个签到。

相同商品在双方眼中价值不同。需要对前i件商品中，自己东西的价值 和 对方东西的价值 进行比较。

记录一下两边自己物品在自己眼里的最大值和在对面眼里的最大值，以及对面认为我的价值最大和最小的两个物品。

- 只要两边都认为自己的东西价值最大，就输出EF。
- 如果对面物品减去最小的货物后价值比我的小，输出EFX。
- 如果对面物品减去最大的货物后价值比我的小，输出EF! 。
- 其它情况输出E。

解题思路：

按照题目要求进行处理即可

c++要注意 输入、输出 和 换行 优化，不然过不去。所以还是直接用c的写法方便。

一开始想到的是使用pair处理，但后来发现没必要。

参考代码c++

```
#include <iostream>
using namespace std;
//xa、ya表示自己拥有的自己眼里物品总价值，xb、yb表示自己拥有的对面眼里物品总价值
//maxx、maxy表示对面认为我的价值最高的物品，minx、miny表示对面认为我的价值最低的物品
long xa, xb, ya, yb, maxx, maxy, minx = 1e6, miny = 1e6, n, c, e, b;

int main() {
    cin >> n;
    while (n--) {
        scanf("%d%d%d", &c, &e, &b);
        if (!b) {
            xa += c;
            xb += e;
            maxx = max(maxx, e);
            minx = min(minx, e);
        } else {
            ya += c;
            yb += e;
            maxy = max(maxy, c);
            miny = min(miny, c);
```

```
        }
        if (xa >= ya && yb >= xb)
            printf("EF\n");
        else if ((ya - miny <= xa) && (xb - minx <= yb))
            printf("EFX\n");
        else if ((ya - maxy <= xa) && (xb - maxx <= yb))
            printf("EF1\n");
        else
            printf("E\n");
    }
    return 0;
}
```

H Homework

题目大意：

有很多份作业要写，每份作业字数（所需时间）不同。但是可以先写完一份，然后其他的抄这一份，当然不同作业抄是需要花不同时间的。

解题思路：

首先我们肯定至少要写最多的那一份的数量的字，其余的我们都可以用复制粘贴来解决。

但不是说我们先把最多的那一份作业写完，比如三个作业子数是100，200，300对应的复制时间是99，199，1。那么显然我们先写完200的字，再复制给作业1和3，最后补足作业三缺少的字更省时间。

既然我们要写的字数已经固定，那初步时间就可以知道了，然后只有第一份是不能粘贴的，所以我们第一份作业选的就是复制时间用时最多的那一个。

即总时间是：最大字数+除去一份最大复制时间后其它复制时间的总和。

参考代码c++

```
#include<iostream>
using namespace std;
long n,cnt,ans,a,b;

int main(){
    cin>>n;
    for(int i=0;i<n;i++){
        cin>>a;
        cnt=max(cnt,a);
    }
    for(int i=0;i<n;i++){
        cin>>b;
        cnt+=b;
        ans=max(ans,b);
    }
    cout<<cnt-ans;
    return 0;
}
```

L Lowbit

题目大意：

对一个数（二进制）进行操作，询问使其变成0的最短操作步骤。
操作方式：x+=lowbit(x) 或者 x−=lowbit(x)

解题思路：

经过观察
单独的 “1” 只需经过一次操作即可转为 “0”，即x−=lowbit(x)
连续的 “1” 只需经过两次次操作即可转为 “0”，比如
1111 -> x+=lowbit(x) -> 10000 -> x−=lowbit(x) -> 0

为了更好处理（避免进位到最前面一位时需要前补0）

- 可以先进行翻转（reverse），并在末尾加上两个字符 "00"
- 或者不需要翻转，直接加几个前导0就行。

然后

1. 从左往右扫描字符串。如果扫描到一个字符为 ‘1’，就检查它右边的字符。
2. 如果右边的字符是 ‘0’，说明这是一个独立的 “1”，计数加一；
3. 如果右边的字符也是 ‘1’，说明这是一个连续的 “11”，就将这个连续段的第一个 ‘1’ 改为 ‘0’，并将前面的 ‘1’ 变成 ‘0’，直到遇到一个 ‘0’ 为止，再将最后一个 ‘0’ 变成 ‘1’。然后回退一格继续扫描。

参考代码c++

```
#include<iostream>
using namespace std;
int ans,sum;
string s;

int main(){
    cin>>s;
    s="00"+s;
    for (int i=s.size()-1;i>=0;i--){
        if (s[i]=='1') sum++;
        else if (s[i]=='0'&&sum){
            ans++;
            if (sum>1) s[i]='1',sum=1;
            else sum=0;
        }
    }
    cout<<ans<<endl;
}
```

普通题

B Bus

题目大意：

给出 总人数 和 一辆车的乘客数，判断需要多少量车才能载完所有人，并按指定格式输出。

解题思路：

简单的除法，注意只用一个bus的时候输出的语句不同。

参考代码c++

```
#include <iostream>
using namespace std;

int main() {
    int n, m;
    cin >> n >> m;
    if (n <= m) {
        cout << "We need one bus.";
    } else { // 一辆车坐不下时，给出三种写法
        cout << "We need " << (n % m == 0 ? n / m : n / m + 1) << " buses." <<
endl;

//      cout << "We need " << ceil(n * 1.0 / m) << " buses." << endl;

//      int t = n / m;
//      if (n % m != 0) t++;
//      cout << "We need " << t << " buses." << endl;
    }
}
```

G Gambling

题目大意：

根据题目规则，Colin 参加了英语口语考试，有 n 道题目。规则如下：

- Eva 老师会随机选择一道题目，每道题目被选中的概率相等。
- Colin 只需要回答被选中的题目，并且他只准备了其中的一道题。
- 如果 Colin 准备了选中的题目，他可以得到满分100分。
- 如果 Colin 没准备选中的题目，他只能得到50分。
- Eva 老师给了 Colin 一次重新抽题的机会，但是这次抽到的题目得分会以80%的比例计算。

Colin 在最优决策下能够获得的最大期望得分是多少，将该得分乘以 n 输出。

解题思路：

如果我们不换题，那么就有：

- 抽到会的题，概率1/n，期望分数就是：1/n*100.
- 没抽到会的题，概率（n-1）/n，期望分数就是：（n-1）/n*50.

总期望是：(50+50*n)/n 或写成 50+50/n

如果我们换题，那么有：

- 一次抽到会的题，概率1/n，期望分数：1/n*100.
- 没抽到会的题，换题后抽到，期望分数：（n-1)/n*1/(n-1) *100 *0.8
- 没抽到会的题，换题后也没抽到，期望分数：(n-1)/n*(n-2)/(n-1) *50 *0.8.

总期望是（100+40*n)/n 或写成 40+100/n

我们只要根据n，看是**50+50*n大还是100+40 *n大**（如果直接比较50+50/n和40+100/n，那高精度小数判断比较繁琐，所以还是比较(50+50*n)/n和（100+40*n)/n方便，因为同分母，所以直接比较50+50*n和100+40 *n）。

参考代码c++

```
#include <iostream>
using namespace std;

int main() {
    int n;
    cin >> n;
    cout << max(50 * n + 50, 40 * n + 100);
    return 0;
}
```

I Infinity

题目大意：

伊娃会在开始时给科林三个整数 a、b、n 和一个变量 x 。

在科林开始练习之前， 他可以选择 0 或 1 作为 x 的初始值 。

然后科林将 有两个操作来选择：

1. 乘：x <-- x * a
2. 加：x <-- x + b

伊娃会要求他通过这两个操作的有限的步骤从 x 发展到 n 。 例如， 当一个 = 5， b=3, n = 40 时， 有一个合法进展：

- 步骤 1： x <-- x * a , × 等于 5 。
- 步骤 2： x <-- x + b , × 等于 8 。
- 步骤 3： × <--x * a , × 等于 40 。

但并不是每个正整数都有可能以这种方式生成 。 当 a = 2， b = 4 时， 没有办法将 x 变成 3 。

对于一对(a, b)， Eva想知道是否存在一个正整数N，满足所有正数 能够生成大于N的整数，这样她就可以轻松地为Colin创建无限个问题。从字面上看，定义 n ⇦ (a,b) ，表示n可以由 (a,b)生成，然后给出一对(a, b)， Eva想要知道是否满足

∃ N , Vn > N , n ⇦ (a,b)

你可以帮她吗

解题思路：

如果x可以被表示，那么x+b也能被表示。

如果a^k%b可以得到{2,b-1}的所有数（不用0和1因为初始就是0和1），那么就可以通过a^k+b*n得到所有的数。

由于题目范围只有 10^6，所以直接枚举 $k \in [1, b)$，判断 $a^k \bmod b$ 是否遍历 $2 \cdots b-1$ 即可。

参考代码c++

```
#include <iostream>
#include <unordered_map>
using namespace std;

unordered_map<int, int>mymap;

int main() {
    int n, m, num = 1;
    cin >> n >> m;
    int cnt = m - 2;
    for (int i = 1; i <= m; i++) {
        num *= n;
        num %= m;
        if (num != 0 && num != 1 && mymap[num] == 0) {
            mymap[num] = 1;
            cnt--;
        }
    }
    if (cnt <= 0)
        cout << "YES";
    else
        cout << "NO";
    return 0;
}
```

中等题

F Finding Stars

题目大意：

解题思路：

我们可以用线段树的单点修改+区间查询模板来写。

每个线段树都有一个bitset<1000>，如果一个数x的出现次数是奇数，那么bitset上第x个位置我们设为1，偶数设为0。

那么查询一个区间有多少出现次数为偶数，只要看bitset<1000>有多少个1就行。至于合并两个bitset，我们可以用异或运算（全为1或全为0时为0，一边为0一边为1时为1，正好符合我们的要求）。

参考代码c++

```
#include <iostream>
#include <bitset>
using namespace std;

const int N = 2e5 + 50, MOD = 1e9 + 7;
bitset<1001>f[4 * N];
int a[N], n, m;

void build_tree(int k, int l, int r) {
    if (l == r) {
        f[k].set(a[l], 1);
        return;
    }
    int mid = (l + r) / 2;
    build_tree(k + k, l, mid);
    build_tree(k + k + 1, mid + 1, r);
    f[k] = f[k + k] ^ f[k + k + 1];
}

void revise(int k, int l, int r, int x, int y) {
    if (l == r) {
        f[k].reset();
        f[k].set(y, 1);
        return;
    }
    int mid = (l + r) / 2;
    if (x <= mid)
        revise(k + k, l, mid, x, y);
    else
        revise(k + k + 1, mid + 1, r, x, y);
    f[k] = f[k + k] ^ f[k + k + 1];
}

bitset<1001> query(int k, int l, int r, int x, int y) {
    if (l == x && r == y)
        return f[k];
    int mid = (l + r) / 2;
    if (y <= mid)
        return query(k + k, l, mid, x, y);
    else if (x > mid)
        return query(k + k + 1, mid + 1, r, x, y);
```

```
    else
        return query(k + k, l, mid, x, mid) ^ query(k + k + 1, mid + 1, r, mid +
1, y);
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(nullptr);
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    build_tree(1, 1, n);
    int st, x, y;
    while (m--) {
        cin >> st >> x >> y;
        if (st == 0) {
            revise(1, 1, n, x, y);
            a[x] = y;
        } else if (st == 1) {
            auto c = query(1, 1, n, x, y);
            if (c.none() || c.flip().none())
                cout << 1 << endl;
            else
                cout << 0 << endl;
        } else {
            auto c = query(1, 1, n, x, y);
            cout << c.count() << endl;
        }
    }
    return 0;
}
```

J Jumping Game

题目大意：

解题思路：

初步想法，对所有数质因数分解，比如42分解成2、3、6，那只要在前面找所有含有质因数2或3或6的位置，把他们的方案数加起来就是我们42的方案数了。但是这样的话时间复杂度是n^2。

所以要用到容斥定理，在这里简单来说就是，加满足奇数条件的-满足偶数条件的。

用一个sum数组，sum[i]表示含有因数i的位置的方案数。

例如序列：2 3 5 9 10 18 30 120，sum[3]=f[2]+f[4]+f[6]+f[7]+f[8] (f表示到达第i个位置的方案数)

那么f[7]=sum[2]+sum[3]+sum[5]-sum[2*3]-sum[2 *5]-sum[3 *5]+sum[2 * 3 *5]。

可以先求出所有数的质因数，然后根据他们的质因数数量用01枚举来求出所有可能的组合数（最多2^5种），然后根据用了奇数个质因数或偶数个质因数来决定是加还是减。

每次算完f后要更新sum，只要是f能影响到的sum全部都要更新。

参考代码c++

```
#include <iostream>
#include <vector>
using namespace std;

const int N = 1e5 + 50, MOD = 1e9 + 7;
vector<int>prime[N];
int sum[N], f[N];

void get_prime(int x) {
    int num = x;
    for (int i = 2; i <= x / i; i++) {
        if (num % i == 0) {
            prime[x].push_back(i);
            while (num % i == 0)
                num /= i;
        }
    }
    if (num > 1)
        prime[x].push_back(num);
}

void get_f(int x, int pos) {
    //x有n个质因数
    int n = prime[x].size(), num = 1 << n;
    for (int i = 1; i < num; i++) {
        int cnt = 0, res = 1;
        for (int j = 0; j < n; j++) {
            if ((i >> j) & 1) {
                cnt++;
                res *= prime[x][j];
            }
        }
        //用了奇数个质因数，按照容斥定理，要加
        if (cnt % 2 == 1)
            f[pos] = (f[pos] + sum[res]) % MOD;
        //偶数是减
        else
            f[pos] = (f[pos] - sum[res] + MOD) % MOD;
    }
}

void get_sum(int x, int pos) {
    int n = prime[x].size(), num = 1 << n;
    for (int i = 1; i < num; i++) {
        int res = 1;
        for (int j = 0; j < n; j++) {
            if ((i >> j) & 1) {
                res *= prime[x][j];
            }
        }
        //看x能影响到的sum全部都要更新
```

```
        sum[res] = (sum[res] + f[pos]) % MOD;
    }
}

signed main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    int n;
    cin >> n;
    vector<int>v(n);
    for (int i = 0; i < n; i++) {
        cin >> v[i];
        if (prime[v[i]].empty())
            get_prime(v[i]);
    }
    f[0] = 1;
    for (int i = 0; i < n; i++) {
        if (i > 0)
            get_f(v[i], i);
        get_sum(v[i], i);
    }
    cout << f[n - 1] << endl;
    return 0;
}
```

以下为E题的翻译：

E 不嫉妒

题目描述

本题是经济学和计算机科学中一个基本的问题：如何公平地分配物品给竞争的代理人。

问题假设有一个包含 m 件物品的集合 M，目标是以公平的方式将这些物品分配给 n 个代理人。

"公平" 的概念之一是不嫉妒。具体来说，如果代理人 i 对于代理人 j 拥有的物品组合 Xj 的价值高于其自身拥有的物品组合 Xi 的价值，则称代理人 i 嫉妒代理人 j。

其中，每个代理人都对每个物品 j 感兴趣，并对其有一个价值 wi，还对一组物品 S 感兴趣，并将其中的所有物品的价值求和表示为 Wi(S)。

分配是将 M 分成不相交的子集 X1, ..., Xn 的过程，其中 Xi 是分配给代理人 i 的物品集合。

本题涉及到三种分配方式：EF（嫉妒零），EFX（嫉妒任意物品），EF1（嫉妒一件物品）。在第一个最优先且最好的分配方式 EF 中，不存在一个代理人会嫉妒另一个代理人；在第二个分配方式 EFX 中，代理人 i 可以妒嫉代理人 j，但是只要从代理人 j 拥有的一组物品中移除任何一个物品，这种妒嫉就会消失；在第三个分配方式 EF1 中，代理人 i 也可以嫉妒代理人 j，但是只要从代理人j所拥有的一组物品中移除任何一个物品，这种妒嫉就会消失。

在本题中，只考虑两个代理人 Colin 和 Eva 的情况。开始时，他们都没有任何物品。然后进行 m 次操作，每次操作提供三个值 ci, ei 和 bi，分别表示该物品在 Colin 和 Eva 视角下的价值，以及该物品分配给 Colin 还是 Eva。每次操作之后，需要判断当前分配方式的优先级。最高的是 EF，其次是 EFX，再次是 EF1，如果都不满足，则最差的是嫉妒。

输入描述

第一行包含一个整数 m (1 <= m <= 10^6)。

接下来的 m 行中，每行都包含三个整数 ci, ei （1 <= ci,ei <= 10^6） 和 bi （bi∈{0,1}） ，表示 Colin 和 Eva 对于该物品的喜好值以及是否将该物品分配给 Colin 或 Eva。

输出描述

在每次操作之后，输出一行：

- 如果当前分配方式是 "EF"，则输出 "EF"；
- 否则如果当前分配方式是 "EFX"，则输出 "EFX"；
- 否则如果当前分配方式是 "EF1"，则输出 "EF1"；
- 否则输出 "E"。

示例1

输入

```
5
5 2 0
5 2 1
2 2 0
9 2 1
9 2 1
```

输出

```
EFX
EF
EFX
EF1
E
```