A. 可以粉碎的数

按照题意，如果所给的数x不含有其它的素因子则输出 `YES`；反之输出 `NO`。

所以我们只需要枚举所给的素数集不断试除，若最终$x = 1$，则说明原本的x不含有其它的素因子。

参考代码

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

int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  int n, Q; cin >> n >> Q;
  vector<int> p(n);
  for (int &x : p) cin >> x;
  for (int x; Q--; ) {
    cin >> x;
    for (const int &pi : p) {
      while (x % pi == 0) x /= pi;
    }
    cout << (x == 1 ? "YES" : "NO") << '\n';
  }
  return 0;
}
```

B. 良心签到题

参考代码

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

int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  cout << "shenlilinghua yyds!" << '\n';
  return 0;
}
```

C. 成对销售

首先对商品按照价格从小到大排序，然后遍历一遍数组，查找$a_i \times x$是否存在。

由于需要删去已经匹配过的商品，我们使用 `STL` 中的 `map` 来记录某个价格所对应的商品的个数。

（也可以使用离散化预处理一遍，不使用 `map` ）

参考代码

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

void solve() {
  int n; cin >> n;
  long long x; cin >> x;

  map<long long, int> cnt;
  vector<long long> a(n);
  for (auto &&ai : a) {
    cin >> ai;
    cnt[ai] ++;
  }

  long long CostBefore = accumulate(a.begin(), a.end(), 0LL);
  long long CostAfter = CostBefore;
  sort(a.begin(), a.end());
  for (const auto &ai : a) {
    if (cnt[ai] == 0) continue;
    cnt[ai] --;
    if (cnt[ai * x]) cnt[ai * x] --;
    else {
      CostAfter += ai % x ? ai * x : ai / x;
    }
  }

  cout << (CostAfter * 0.7 < CostBefore ? "YES" : "NO") << '\n';
}

int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  int T; cin >> T;
  while (T--) solve();
  return 0;
}
```

D. 小白算不对

等比数列可以通过**首项**和**公比**唯一确定，通项$a_n = a_0 \times q^n$。

对于本题，由于公比q已经确定，所以只要找使得修改次数最少的首项。

对于数组中的任意的a_i，则只要首项$a_0 = \frac{a_i}{q^i}$，该项就无需修改。

显然，使得修改次数最少的首项a_0就是出现次数最多的$\frac{a_i}{q^i}$。

参考代码

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

void solve() {
  int n; cin >> n;
  double q; cin >> q;

  vector<double> a(n);
  for (auto &&x : a) cin >> x;

  int ans = n;
  map<double, int> cnt;
  double power = 1;
  for (int i = 0; i < n; i++, power *= q) {
    ans = min(ans, n - ++cnt[a[i] / power]);
  }
  cout << ans << '\n';
}

int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  int T; cin >> T;
  while (T--) solve();
  return 0;
}
```

E. 嘉然的小数

分数$\frac{1000}{998999}$的小数位，可以通过模拟除法来获得。所以直接预处理其小数点后45位，而后一次回答询问即可。

PS: 出题人提供的代码（关键部分）如下：

```
void solve(){
    int k;
    a[1]=1;
    for(int i=2;i<=15;i++)a[i]=a[i-1]+a[i-2];
    scanf("%d",&t);
    while(t--){
        scanf("%d",&k);
        k=(k+2)/3;
        printf("%03d\n",a[k]);
    }
}
```

这是由于，该分数为斐波那契数列的生成函数$G(x)=\frac{x}{1-x-x^2}$取$x=\frac{1}{1000}$得到，即$\frac{1000}{998999}=\sum_{n=0}^{\infty}F_n\times(\frac{1}{1000})^n$，其中$F_n$为斐波那契数列的第$n$项。

参考代码

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

vector<int> bit;

void init() {
  int a = 1000, b = 998999;
  while ((int) bit.size() < 45) {
    bit.push_back(a * 10 / b);
    a = (a * 10) % b;
  }
}

void solve() {
  int k; cin >> k; k = (k - 1) / 3 * 3;
  cout << bit[k] << bit[k + 1] << bit[k + 2] << '\n';
}

int main() {
  init();
  cin.tie(nullptr)->sync_with_stdio(false);
  int T; cin >> T;
  while (T--) solve();
  return 0;
}
```

F. 购物

可知，当买家支付的钱到达一个界限后，超出的部分会如数找回。但是问题在于，如何确定一个较好的上界？

~~从经验出发，直接取 `1e6` 级别的数去试不失为一种办法，取到 `2e6` 就能AC。~~

本题可以取上界为$max_v \times max_v + T$，证明如下：

假设存在一种最优支付方案，支付多于$max_v \times max_v + T$的钱，则商店找零会多于$max_v \times max_v$的钱，这些硬币的个数大于max_v。假设这些硬币的面值分别为v_i，则根据鸽巢原理，在硬币序列种至少存在两个子序列，这两个子序列的和都可以被max_v整除。若这届用长度更小的序列换算为面值为max_v的硬币某整数个，再去替换原序列，就可以用更少的硬币买到商品，这与最优支付方案矛盾。

上界确定的实际就是“容积”，现在本题已经变成一个十分经典的**背包问题（数的拆分）**，其中对店家找钱为**完全背包**，对于买方付钱则为**多重背包（需要优化）**。

参考代码

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

int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  int N, T; cin >> N >> T;
  vector<int> v(N), c(N);
  for (int &x : v) cin >> x;
  for (int &x : c) cin >> x;

  const int LIM = (int) 2e6 + 10, inf = (int) 1e9;
  vector<int> payment(LIM, inf), change(LIM, inf);
  payment[0] = change[0] = 0;
  for (int i = 0; i < N; i++) {
    for (int V = v[i]; V < LIM; V++) {
      change[V] = min(change[V], change[V - v[i]] + 1);
    }
  }
  for (int i = 0; i < N; i++) {
    int k = 1;
    while (k <= c[i]) {
      int vk = v[i] * k;
      for (int V = LIM - 1; V >= vk; V--) {
        payment[V] = min(payment[V], payment[V - vk] + k);
      }
      c[i] -= k, k <<= 1;
    }
    if (c[i]) {
      int vk = v[i] * c[i];
      for (int V = LIM - 1; V >= vk; V--) {
        payment[V] = min(payment[V], payment[V - vk] + c[i]);
      }
    }
  }
```

```
35
36    int ans = inf;
37    for (int i = 0; T + i < LIM; i++) {
38      ans = min(ans, change[i] + payment[T + i]);
39    }
40    cout << (ans >= inf ? -1 : ans) << '\n';
41    return 0;
42  }
```

G. 关于愿望的愿望

第一个操作一定是$+1$，之后就一直乘以2就好了（指数爆炸显然）。

注意特判没有修改机会$n = 0$的情况。

参考代码

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  int main() {
5    cin.tie(nullptr)->sync_with_stdio(false);
6    int n; cin >> n;
7    cout << max(0, 1 << (n - 1)) << '\n';
8    return 0;
9  }
```

H. 国足upupup

先考虑一种十分容易想到的DP：

按照时间顺序，如果i可以通过j到，则可以转移结果。

```
1  bool check(int i, int j) {
2    return abs(t[i] - t[j]) * v >= abs(p[i] - p[j]);
3  }
4  for (int i = 1; i <= n; i++) {
5    for (int j = 1; j < i; j++) {
6      if (check(i, j)) dp[i] = max(dp[i], dp[j] + 1);
7    }
8  }
```

通过化简 `check` 的表达式，我们可以把条件转化成如下的式子：

$$\begin{cases} p_i - t_i \times v \le p_j - t_j \times v \\ p_j + t_j \times v \le p_i + t_i \times v \end{cases} \tag{1}$$

之后，问题变成一个普通的偏序问题，使用处理偏序问题的基本手段就能解决。

PS: 还有一种思路是转化成LIS，本质相同但是写法更简单。

> 按时间先后对所有踢球排序
>
> 设$x_i = v * t_i - p_i, y_i = v * t_i + p_i$
>
> $x_i \le x_j \iff vt_i - p_i \le v_tj - p_j \iff p_j - p_i \le v(t_j - t_i)$
>
> $y_i \le y_j \iff vt_i + p_i \le vt_j + p_j \iff p_i - p_j \le v(t_j - t_i)$
>
> 可以从p_i到$p_j \iff t_j > t_i$ 且 $v * (t_j - t_i) >= abs(p_j - p_i) \iff x_i \le x_j$ 且 $y_i \le y_j$
>
> 对x和y构成的二元组序列按x升序排序后y的最长上升子序列即为答案。

参考代码

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

struct FenwickTree {
  vector<long long> s;
  FenwickTree(int n) : s(n) {}
  void update(int pos, long long dif) {
    for (; pos < (int)s.size(); pos |= pos + 1) {
      s[pos] = max(s[pos], dif);
    }
  }
  long long query(int pos) { // query [0, pos)
    long long res = 0;
    for (; pos > 0; pos &= pos - 1) {
      res = max(res, s[pos - 1]);
    }
    return res;
  }
};

int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  int n; cin >> n;
  long long v; cin >> v;

  vector<long long> t(n), p(n);
  for (auto &&x : t) cin >> x;
  for (auto &&x : p) cin >> x;

  vector<long long> rank;
```

```cpp
  vector<pair<long long, long long> > P;
  for (int i = 0; i < n; i++) {
    long long dis = t[i] * v;
    if (dis < llabs(p[i])) continue;
    P.emplace_back(p[i] - dis, p[i] + dis);
    rank.push_back(P.back().second);
  }

  sort(rank.begin(), rank.end());
  rank.erase(unique(rank.begin(), rank.end()), rank.end());

  FenwickTree dp(rank.size());
  sort(P.begin(), P.end(), [](const auto &A, const auto &B) {
    return B.first == A.first ? B.second > A.second : B.first < A.first;
  });
  for (const auto &Px : P) {
    int pos = lower_bound(rank.begin(), rank.end(), Px.second) -
rank.begin();
    dp.update(pos, 1 + dp.query(pos + 1));
  }
  cout << dp.query((int)rank.size()) << '\n';
  return 0;
}
```

I. 困难的交流

我们令 `dif = s.length() - t.length()` 表明我们需要删除的字符数目。

可以知道，删除首部的字符只会让长度 `-1` ，删除中间部分的字符会让长度 `-2` 。那么，如果可以由s变成t串，则最小操作次数已经确定。

`dif` 为奇数时， `s` 的第一个字符（无论是什么）必须删去。

我们使用一个指针指向当前t串中待匹配的位置，在匹配过程中，如果当前字符不匹配则必然要**立即**删去，不然之后的串都无法匹配；而如果当前可以匹配上，可能可以删去（删去可能导致无解），由于保留下来并不会影响后续的匹配，故最优处理方案应该是保留。

参考代码

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

void solve() {
  int x; cin >> x;
  string s, t; cin >> s >> t;

  int lenS = s.length();
  int lenT = t.length();

```

```cpp
  if (lenS < lenT) return cout << "NO\n", void();
  int dif = lenS - lenT, ans = (dif - 1 + 2) / 2;
  if (ans > x) return cout << "NO\n", void();

  if (dif & 1) s = s.substr(1);
  int cur = 0, deleted = 0;
  for (const char &c : s) {
    if (deleted) { deleted = 0; continue; }

    if (c == t[cur]) cur++;
    else deleted = 1;

    if (cur == lenT) {
      cout << "YES\n" << ans << '\n';
      return;
    }
  }
  cout << "NO" << '\n';
}

int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  int T; cin >> T;
  while (T--) solve();
  return 0;
}
```

J. 机关解谜

显然，对于同一份答案，打击顺序并不影响结果，并且，对于同一个块击打4次不会改变该块的状态，所以有效的操作总共只有4^n种。

考虑到n最大只有10，直接枚举（使用 `dfs` 或者 `bfs` 或者其他搜索方法均可）所有可能的答案同时判断即可。

出题人提供了一个$\mathcal{O}(n)$的做法：

> 我们用$O(n)$的复杂度思考一下：
>
> 通过改变第i位，我们可以改变第$i-1, i+1$位，在不考虑最终答案的情况下，容易证明通过改变第2，3，$\ldots$，$n-1$位能够使得前$n-2$位数相同，那么我们的任务就是保证最后两位相同即可。
>
> 反向思考一下，如果在保证最后两位数相同的前提下，将前$n-2$位数变为相同，那么我们就可以通过改变第n位来使最后两位数与前$n-2$位数相同。
>
> 所以首先我们将最后两位数变为相同的数。
>
> 那么对于我们需要将数组变为什么相同的数这个问题，我们只需要将$0-3$遍历一遍即可，从第二位开始改变数组，直到前$n-2$位数相同，常数较小，可以接受

> 由于从第二位开始改变不能保证遍历到所有情况，所以我们还需要从第一位开始也遍历一次，将所有情况遍历到位即可

参考代码

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

int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  int n; cin >> n;
  vector<int> a(n);
  for (int &x : a) cin >> x;

  const auto get = [](int BASE, int n) -> vector<int> {
    vector<int> res(n);
    for (int &x : res) {
      x = BASE & 3, BASE >>= 2;
    }
    reverse(res.begin(), res.end());
    return res;
  };
  for (int mask = 0, _ = 1 << (2 * n); mask < _; mask++) {
    auto v = get(mask, n), b = a;
    for (int i = 0; i < n; i++) {
      b[i] += v[i];
      if (i - 1 >= 0) b[i - 1] += v[i];
      if (i + 1 < n) b[i + 1] += v[i];
    }
    for (int &x : b) x %= 4;
    if (count(b.begin(), b.end(), b.front()) == n) {
      cout << accumulate(v.begin(), v.end(), 0) << '\n';
      for (int i = 0; i < n; i++) {
        while (v[i]--) cout << i + 1 << ' ';
      }
      return 0;
    }
  }
  assert(false);
  return 0;
}
```

K. 送你一道签到题

当人数最多的教室的人数大于其他教室人数之和时：无论按怎样的顺序访问都会导致人数最多的教室中可能存在有人评阅了自己的试卷。

其余情况：按照教室人数从多到少访问就是一个安全的访问顺序（安全的访问顺序可能不唯一）。

参考代码

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

int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  int n; cin >> n;
  vector<int> a(n);
  for (int &x : a) cin >> x;
  int Sum = accumulate(a.begin(), a.end(), 0);
  int Max = *max_element(a.begin(), a.end());
  cout << (Max <= Sum - Max ? "YES" : "NO") << '\n';
  return 0;
}
```