A Нечетный делитель

Если у числа x есть нечетный делитель, значит у него есть нечетный простой делитель. Для понимания этого факта, можно рассмотреть что происходит при умножении четных и нечетных чисел:

- четный $*$ четный $=$ четный;
- четный $*$ нечетный $=$ четный;
- нечетный $*$ четный $=$ четный;
- нечетный $*$ нечетный $=$ **нечетный**.

Существует только одно четное простое число — 2. Значит, если у числа нет нечетных делителей, то оно должно быть степенью двойки. Для проверки этого факта, можно например делить n на 2, пока оно делится. Если в конце мы получили 1, то n — степень двойки.

Бонус: для проверки также можно использовать следующее условие:

$$n\&(n-1)=0$$

. Если число — степень двойки, то оно содержит только одну единицу в двоичной записи. Тогда $(n-1)$ содержит единицы во всех позициях, кроме той, в которой находится единица в n. Значит их побитовое «И» не содержит единиц.

```cpp
#include <bits/stdc++.h>

using namespace std;
using ll = long long;
using ld = long double;

void solve() {
  ll n;
  cin >> n;
  if (n & (n - 1)) {
    cout << "YES\n";
  } else {
    cout << "NO\n";
  }
}

int main() {
  int t;
  cin >> t;
  while (t--) {
    solve();
  }
}
```

B Новогоднее число

Пусть x — количество чисел 2020, y — количество чисел 2021 ($x, y \geq 0$). Запишем искомое разложение числа n:

$$n = 2020 \cdot x + 2021 \cdot y = 2020 \cdot (x + y) + y$$

Тогда получаем, что $n - y$ делится на 2020. Возьмём y равный остатку от деления n на 2020. Тогда x однозначно определяется из формулы выше:

$$x = \frac{n - y}{2020} - y$$

Тогда, если в результате получится, что $x \geq 0$ ($y \geq 0$ потому что остаток от деления n на 2020 неотрицателен), тогда n можно представить в виде суммы некоторого количества чисел 2020 и некоторого количества чисел 2021;

```cpp
#include <bits/stdc++.h>

#include <utility>
using namespace std;

using pii = pair<int, int>;

int main() {
  int test;
  cin >> test;
  while (test-- > 0) {
    int n;
    cin >> n;
    int cnt2021 = n % 2020;
    int cnt2020 = (n - cnt2021) / 2020 - cnt2021;
    if (cnt2020 >= 0 && 2020 * cnt2020 + 2021 * cnt2021 == n) {
      cout << "YES\n";
    } else {
      cout << "NO\n";
    }
  }
  return 0;
}
```

C Различные делители

Hint 1: Число имеет 4 делителей, если оно равно pq или p^3 для некоторых простых p и q. В первом случае его делители равны 1, p, q, pq. Во втором случае его делителями являются 1, p, p^2, p^3.

Hint 2: Вместо поиска числа с хотя бы 4 делителями, попробуйте найти число с ровно 4 делителями.

Hint 3: Пусть p — наименьший простой делитель a. Тогда, $p \geq d + 1$ выполняется.

Решение:

Предположим, мы нашли a с больше, чем 4 делителями (и удовлетворяющее другому условию). Если у a есть хотя бы два различных простых делителя, мы можем избавиться от всех остальных простых делителей и получить меньшее число с хотя бы 4 делителями. Иначе, $a = p^k$ для некоторого $k > 3$. p^3 тоже будет иметь 4 делителя и будет меньше, чем a. Когда мы избавляемся от простого множителя, новые делители не появляются, поэтому разность между любыми двумя из них не станет меньше d.

Найдём наименьшие числа вида p^3 и pq $(p < q)$ независимо.

В первом случае, $p^3 - p^2 > p^2 - p > p - 1$, так как $p \geq 2$. Если мы найдём наименьшее $p \geq d + 1$ все условия будут выполнены.

Во втором случае, $1 < p < q < pq$. Найдём наименьшее $p \geq d + 1$ и наименьшее $q \geq d + p$. В этом случае легко видеть, что $p - 1 \geq d$ и $q - p \geq d$. Также, $pq - q = q(p - 1) \geq qd \geq d$ так как $q \geq 2$. Этого достаточно, чтобы получить `OK` даже без проверки случая $a = p^3$.

Также следует доказать, что не существует таких p' и q' что $p'q' < pq$. Это верно, так как p' должно быть больше или равно $d + 1$ и оно должно быть простым, но p является наименьшим простым, большим d, поэтому $p' \geq p$. Аналогично, q' должно быть хотя бы $p + d$ и можно доказать, что $q' \geq q$ таким же образом.

Время работы $\mathcal{O}(t \cdot gap \cdot primecheck)$, где gap — расстояние между простыми, равное $\mathcal{O}(\log a)$ в среднем и $primecheck$ это время работы алгоритма проверки числа на простоту, обычно равное $\mathcal{O}(\sqrt{a})$.

Ответ $d = 10000$ близок к $2 \cdot 10^8$, поэтому тривиальное решение с факторизацией каждого числа не будет работать, но им можно предподсчитать ответы для всех значений d за один запуск.

```cpp
#include <iostream>
#include <vector>

using namespace std;

void solve()
{
    int x;
    cin >> x;
    vector<int> p;
    for (int i = x + 1; ; i++)
    {
        int t = 1;
        for (int j = 2; j * j <= i; j++)
        {
            if (i % j == 0)
            {
                t = 0;
                break;
            }
        }
        if (t)
        {
            p.push_back(i);
            break;
        }
    }
    for (int i = p.back() + x; ; i++)
    {
        int t = 1;
        for (int j = 2; j * j <= i; j++)
        {
            if (i % j == 0)
            {
                t = 0;
                break;
            }
        }
        if (t)
        {
            p.push_back(i);
            break;
        }
    }
    cout << min(1ll * p[0] * p[1], 1ll * p[0] * p[0] * p[0]) << "\n";
}

int main()
{
    int t;
    cin >> t;
    while (t--)
    {
        solve();
    }
}
```

D Бал в Берляндии

Можно думать, что в задаче дан двудольный граф. Мальчики и девочки это вершины графа. Если мальчик и девочка готовы танцевать вместе, то между ними проводится ребро. В этом графе надо выбрать два ребра, которые не пересекаются по вершинам.

Пусть $deg(x)$ — количество ребер входящих в вершину x.

Переберем первое ребро —(a, b). Оно заблокирует $deg(a) + deg(b) - 1$ других ребер (все соседние с вершиной a, с вершиной b, но ребро (a, b) запретится два раза. Все незапрещеные ребра не пересекаются с (a, b) по вершинам. Значит можно прибавить $k - deg(a) - deg(b) + 1$ к ответу.

```cpp
#include <bits/stdc++.h>

using namespace std;
using ll = long long;
using ld = long double;

void solve() {
  int A, B, k;
  cin >> A >> B >> k;
  vector<int> a(A), b(B);
  vector<pair<int, int>> edges(k);
  for (auto &[x, y] : edges) {
    cin >> x;
  }
  for (auto &[x, y] : edges) {
    cin >> y;
  }
  for (auto &[x, y] : edges) {
    x--;
    y--;
    a[x]++;
    b[y]++;
  }
  ll ans = 0;
  for (auto &[x, y] : edges) {
    ans += k - a[x] - b[y] + 1;
  }
  cout << ans / 2 << "\n";
}

int main() {
  int t;
  cin >> t;
  while (t--) {
    solve();
  }
}
```

E Очистка телефона

Пусть мы удалили x приложений у которых $b_i = 1$ и y приложений у которых $b_i = 2$. Очевидно, что среди всех приложений с $b_i = 1$, надо было взять x максимальных по памяти (так мы очистим больше всего памяти).

Разобьем все приложения на два массива, у которых $b_i = 1$ и $b_i = 2$ и отсортируем их. Тогда из каждого массива нужно взять какой-то префикс.

Переберем какой префикс мы берем из первого массива. Для него мы можем однозначно найти второй префикс (набираем приложения, пока сумма не превысит m). Если мы теперь увеличим первый префикс, взяв новое приложение, то можно не брать какие-то приложения во втором массиве. Значит при увеличении первого префикса второй может только уменьшиться.

Для решения задачи можно применить метод двух указателей.

```cpp
#include <bits/stdc++.h>

using namespace std;
using ll = long long;
using ld = long double;

void solve() {
  int n, m;
  cin >> n >> m;
  vector<int> a, b;
  vector<int> v(n);
  for (int &e : v) {
    cin >> e;
  }
  for (int &e : v) {
    int x;
    cin >> x;
    if (x == 1) {
      a.push_back(e);
    } else {
      b.push_back(e);
    }
  }
  sort(a.rbegin(), a.rend());
  sort(b.rbegin(), b.rend());
  ll curSumA = 0;
  int r = (int)b.size();
  ll curSumB = accumulate(b.begin(), b.end(), 0ll);
  int ans = INT_MAX;
  for (int l = 0; l <= a.size(); l++) {
    while (r > 0 && curSumA + curSumB - b[r - 1] >= m) {
      r--;
      curSumB -= b[r];
    }
    if (curSumB + curSumA >= m) {
      ans = min(ans, 2 * r + l);
    }
    if (l != a.size()) {
      curSumA += a[l];
    }
  }
```

```
  cout << (ans == INT_MAX ? -1 : ans) << "\n";
}

int main() {
  int t;
  cin >> t;
  while (t--) {
    solve();
  }
}
```

F Рекламное агентство

Очевидно, что Маша будет заключать договора только с блогерами, у которых больше всего подписчиков. Можно отсортировать всех блогеров и жадно выбирать префикс.

Пусть x — минимальное количество подписчиков у нанятого блогера. Тогда мы обязаны нанять всех блогеров у которых больше подписчиков. Пусть m — количество блогеров у которых больше, чем x подписчиков, $cnt[x]$ — количество блогеров, у которых ровно x подписчиков. Тогда мы должны выбрать $k - m$ блогеров из $cnt[x]$. Число способов сделать это равно биномиальному коэфициенту из $cnt[x]$ по $k - m$.

Вы могли посчитать его с помощью поиска обратного элемента по модулю. Тогда можно было посчитать факториалы и использовать равенство $\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$.

Либо же можно использовать равенство $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$ и посчитать с помощью динамического программирования. Этот метод больше известен как треугольник Паскаля.

```cpp
#include <bits/stdc++.h>

using namespace std;
using ll = long long;
using ld = long double;

int mod = 1e9 + 7;

int fast_pow(int a, int p) {
  int res = 1;
  while (p) {
    if (p % 2 == 0) {
      a = a * 1ll * a % mod;
      p /= 2;
    } else {
      res = res * 1ll * a % mod;
      p--;
    }
  }
  return res;
}

int fact(int n) {
  int res = 1;
  for (int i = 1; i <= n; i++) {
    res = res * 1ll * i % mod;
  }
  return res;
}
```

```
int C(int n, int k) {
  return fact(n) * 1ll * fast_pow(fact(k), mod - 2) % mod * 1ll * fast_pow(fact(n - k), mod - 2) % mod;
}

void solve() {
  int n, k;
  cin >> n >> k;
  vector<int> cnt(n + 1);
  for (int i = 0; i < n; i++) {
    int x;
    cin >> x;
    cnt[x]++;
  }
  for (int i = n; i >= 0; i--) {
    if (cnt[i] >= k) {
      cout << C(cnt[i], k) << "\n";
      return;
    } else {
      k -= cnt[i];
    }
  }
  cout << 1;
}

int main() {
  int t;
  cin >> t;
  while (t--) {
    solve();
  }
}
```

G Необычная матрица

Понятно, что порядок операций не влияет на конечный результат нет смысла применять одну и ту же операцию больше одного раза (по свойства операции `xor`). Построим последовательность операций, которая будет приводить матрицу a к матрице b (если ответ существует). Давайте переберём: будем ли мы применять операцию «горизонтальный `xor`».

Теперь по первому элементу первой строки ($a_{1,j}$) мы можем понятно, необходимо ли применять операцию «вертикальный `xor`» (если $a_{1,j} \neq b_{1,j}$). Применим все необходимые операции «вертикальный `xor`». Осталось понятно, необходимо ли для i ($2 \leq i \leq n$) применять операцию «горизонтальный `xor`». Посмотрим на каждый элемент первого столбца ($a_{i,1}$) по нему можно понять, необходимо ли применять операцию «горизонтальный `xor`» (если $a_{i,1} \neq b_{i,1}$).

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

using pii = pair<int, int>;

bool check(vector<vector<int>> a, vector<vector<int>> const &b) {
  int n = (int) a.size();
  for (int j = 0; j < n; j++) {
    if (a[0][j] != b[0][j]) {
      for (int i = 0; i < n; i++) {
        a[i][j] ^= 1;
      }
    }
  }
  for (int i = 0; i < n; i++) {
    int need_xor = (a[i][0] ^ b[i][0]);
    for (int j = 1; j < n; j++) {
      if (need_xor != (a[i][j] ^ b[i][j])) {
        return false;
      }
    }
  }
  return true;
}
```

```
void solve() {
  int n;
  cin >> n;
  vector<vector<int>> a(n, vector<int>(n));
  vector<vector<int>> b(n, vector<int>(n));
  for (int i = 0; i < n; i++) {
    string s;
    cin >> s;
    for (int j = 0; j < n; j++) {
      a[i][j] = s[j] - '0';
    }
  }
  for (int i = 0; i < n; i++) {
    string s;
    cin >> s;
    for (int j = 0; j < n; j++) {
      b[i][j] = s[j] - '0';
    }
  }

  for (int times = 0; times < 2; times++) {
    if (check(a, b)) {
      cout << "YES\n";
      return;
    }
    for (int j = 0; j < n; j++) {
      a[0][j] ^= 1;
    }
  }
  cout << "NO\n";
}
```

```
int main() {
  int test;
  cin >> test;
  while (test-- > 0) {
    solve();
  }
  return 0;
}
```

H Странная красота

Давайте для каждого числа x посчитаем сколько раз он встречается в массиве a. Обозначим это число за cnt_x.

Воспользуемся методом динамического программирования. Пусть $dp(x)$ равно максимальному количество чисел не больших x, таких что для каждой пары из них выполнено одно из условий выше. Более формально, если $dp(x) = k$, тогда существуют числа $b_1, b_2, \ldots, b_k$ ($b_i \leq x$) из массива a, такие что для всех $i \neq j$ ($1 \leq i, j \leq k$) выполнено одно из условий выше.

Тогда для подсчёта $dp(x)$ можно воспользоваться следующей формулой:

$$dp(x) = cnt(x) + \max_{y=1, x \bmod y = 0}^{x-1} dp(y)$$

.

Заметим, что для подсчёта $dp(x)$ необходимо пройтись по списку делителей числа x. Для этого воспользуемся решетом Эратосфена.

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

const int N = (int) 2e5 + 100;

int dp[N];
int cnt[N];

void solve() {
  int n;
  cin >> n;
  fill(dp, dp + N, 0);
  fill(cnt, cnt + N, 0);
  for (int i = 0; i < n; i++) {
    int x;
    cin >> x;
    cnt[x]++;
  }
  for (int i = 1; i < N; i++) {
    dp[i] += cnt[i];
    for (int j = 2 * i; j < N; j += i) {
      dp[j] = max(dp[j], dp[i]);
    }
  }
  cout << (n - *max_element(dp, dp + N)) << endl;
}

int main() {
  int test;
  cin >> test;
  while (test-- > 0) {
    solve();
  }
  return 0;
}
```