Задача A. Выгодная Покупка

Для решения данной задачи необходимо для начала вычислить стоимость каждого телефона с учетом скидки: для i-го телефона она равна $a_i = c_i - d_i$. Теперь в массиве a необходимо найти первый и второй минимум.

Для нахождения первого и второго минимума будем поддерживать две переменные: $first$ — индекс первого минимума и $second$ — индекс второго минимума. Пойдем по массиву слева направо и будем пересчитывать индексы первого и второго минимума. Если $a_i \leqslant a_{first}$, скажем, что $second := first$ и $first := i$. Если же $a_i \leqslant a_{second}$, скажем, что $second := i$.

Асимптотика решения: $\mathcal{O}(n)$.

Пример решения на языке C++:

```
#include <cstdio>

using namespace std;

using ll = long long;

const int N = 100100;

ll c[N], d[N];

ll calc(int i) {
    return c[i] - d[i];
}

int main() {
    int n;
    scanf("%d", &n);

    for (int i = 0; i < n; ++i) {
        scanf("%lld%lld", &c[i], &d[i]);
    }

    int min1 = -1, min2 = -1;
    for (int i = 0; i < n; ++i) {
        if (min1 == -1 || calc(i) <= calc(min1)) {
            min2 = min1;
            min1 = i;
        } else if (min2 == -1 || calc(i) <= calc(min2)) {
            min2 = i;
        }
    }

    printf("%d %d\n", min1 + 1, min2 + 1);

    return 0;
}
```

Задача B. Карантин

Переберем дни, в которые Костя будет решать домашние задания по математике и английскому (дни a_M и a_E). После того, как мы их зафиксировали, мы знаем необходимую разность: $d = a_E - a_M$.

Теперь нужно проверить, есть ли в массиве элементы $a_M + 2d$ и $a_M + 3d$. Это можно сделать при помощи бинарного поиска, либо структур данных `set` или `unordered_set`.

Асимптотика решения: $\mathcal{O}(n^2 \log n)$ или $\mathcal{O}(n^2)$.

Пример решения на языке C++:

```
#include <cstdio>
#include <algorithm>

using namespace std;

const int N = 1100;

int a[N];

int main() {
    int n;
    scanf("%d", &n);

    for (int i = 0; i < n; ++i) {
        scanf("%d", &a[i]);
    }

    int ans = 0;
    for (int i = 0; i < n; ++i) {
        for (int j = i + 1; j < n; ++j) {
            int delta = a[j] - a[i];
            int x = a[i] - delta;
            int y = a[j] + delta;

            if (binary_search(a, a + n, x) && binary_search(a, a + n, y)) {
                ++ans;
            }
        }
    }

    printf("%d\n", ans);

    return 0;
}
```

Задача C. Красивая Ленточка

Заметим несколько интересных фактов.

- Если каждый цвет встречается четное количество раз, ленточка сама по себе является красивой, поэтому ответ на задачу равен 1.
- Если есть хотя бы один цвет, который встречается нечетное количество раз, то разрезать ленточку требуемым образом невозможно, то есть ответ равен -1.

Заметив данные факты, задача решается просто. Нужно для каждого цвета посчитать, сколько раз он встречается. Это можно сделать при помощи `map` или `unordered_map`.

Асимптотика решения: $\mathcal{O}(n \log n)$ или $\mathcal{O}(n)$.

Пример решения на языке C++:

```
#include <cstdio>
#include <map>

using namespace std;

int main() {
    int n;
    scanf("%d", &n);

    map<int, int> cnt;
    for (int i = 0; i < n; ++i) {
        int x;
        scanf("%d", &x);
        ++cnt[x];
    }

    for (auto& [x, c] : cnt) {
        if (c % 2 != 0) {
            printf("-1\n");
            return;
        }
    }

    printf("1\n");
    return 0;
}
```

Задача D. Апгрейд

Для начала отсортируем роутеры по неубыванию a_i. Теперь воспользуемся динамическим программированием. Пусть dp_i — максимальная сумма скоростей передачи данных роутеров, которые можно выбрать среди первых i роутеров.

Пусть $dp_0 = 0$. Теперь научимся пересчитывать dp_i. Для начала сделаем это за $\mathcal{O}(n^2)$. Для начала рассмотрим вариант, когда мы не покупаем i-й роутер: тогда надо сделать переход из dp_{i-1}. Теперь пусть мы возьмем i-й роутер, тогда сделаем переход из всех $j < i$, таких что $a_i - a_j \leqslant k$ и сделаем переход при помощи $dp_j + b_i$.

Теперь заметим, что можно сделать переход только из самого левого подходящего состояния, поддерживая указатель на первый подходящий элемент j (для понимания обратите внимание на реализацию). Нетрудно понять, что данный указатель всегда двигается только вправо, поэтому суммарно алгоритм будет работать за $\mathcal{O}(n)$.

Асимптотика решения: $\mathcal{O}(n \log n)$.

Пример решения на языке C++:

```
#include <cstdio>
#include <algorithm>

using namespace std;

using ll = long long;

const int N = 200100;
const int INF = (int)2e9;
```

```
class Router {
public:
    int a, b;
    int id;

    Router() = default;

    Router(int a, int b, int id) : a(a), b(b), id(id) {}

    bool operator<(const Router& other) const {
        return a < other.a;
    }
};

Router a[N];
ll dp[N];
int p[N];

int main() {
    int n, k;

    scanf("%d%d", &n, &k);
    for (int i = 1; i <= n; ++i) {
        scanf("%d%d", &a[i].a, &a[i].b);
        a[i].id = i;
    }

    a[0].a = -INF;
    sort(a + 1, a + n + 1);

    dp[0] = 0;
    int ptr = 0;
    for (int i = 1; i <= n; ++i) {
        dp[i] = dp[i - 1];
        p[i] = -1;

        while (ptr + 1 < i && a[i].a - a[ptr + 1].a >= k) {
            ++ptr;
        }

        if (dp[i] < dp[ptr] + a[i].b) {
            dp[i] = dp[ptr] + a[i].b;
            p[i] = ptr;
        }
    }

    printf("%lld\n", dp[n]);
    vector<int> res;
    int it = n;
    while (it > 0) {
        if (p[it] == -1) {
            --it;
```

```
            continue;
        }

        res.push_back(a[it].id);
        it = p[it];
    }

    reverse(res.begin(), res.end());
    printf("%d\n", (int)res.size());
    for (auto i : res) {
        printf("%d ", i);
    }
    printf("\n");
}
```

Задача E. Разрезание Строки

Для начала решим задачу за $\mathcal{O}(n^2)$. Воспользуемся динамическим программированием. Пусть dp_i — ответ для префикса строки длины i.

Пусть $dp_0 = 0$. Теперь пересчитаем dp_i. Переберем левую границу последнего отрезка от i до 1. Пусть левая граница равна j. Посчитаем, сколько есть различных символов на отрезке $[j, i]$. Пусть оно равно d. Тогда пересчитаем динамику при помощи $dp_{j-1} + d^2$.

Данное решение работает за $\mathcal{O}(n^2)$. Теперь заметим, что большинство переходов, которые мы сделали, бесполезны. Значение d может принимать значения от 1 до 26. Также, очевидно, что динамика слева направо неубывает. Рассмотрим отрезки, на которых значение d принимает значение $1, 2, 3, \ldots, 26$. Тогда нужно пересчитать динамику из левой границы каждого отрезка. Тогда данная динамика будет пересчитываться за $\mathcal{O}(n\Sigma)$.

Асимптотика решения: $\mathcal{O}(n\Sigma)$.

Пример решения на языке C++:

```
#include <cstdio>
#include <set>
#include <algorithm>

using namespace std;

using ll = long long;

const int N = 300100;

ll dp[N];
char s[N];
int pos[256];

int main() {
    int n;
    scanf("%d\n%s", &n, s);

    dp[0] = 0;
    set<pair<int, char>> setik;
    for (int i = 1; i <= n; ++i) {
        setik.erase(make_pair(pos[s[i - 1] - 'a'], s[i - 1]));
```

```
        pos[s[i - 1] - 'a'] = i - 1;
        setik.insert(make_pair(pos[s[i - 1] - 'a'], s[i - 1]));

        int cur = (int)setik.size();
        dp[i] = 1ll * cur * cur;

        for (auto& [j, c] : setik) {
            if (j == i - 1) {
                break;
            }
            --cur;
            dp[i] = min(dp[i], dp[j + 1] + 1ll * cur * cur);
        }
    }

    printf("%lld\n", dp[n]);

    return 0;
}
```