

A. Azat and bookshelf

Ответ равен разности объемов двух прямоугольных параллелепипедов:

$$W \cdot H \cdot L - (W - 2D)(H - 2D)(L - D).$$

Асимптотика: $O(1)$.

```
1 #include <iostream>
2 using namespace std;
3 int main() {
4     long long d, w, h, l, volume_ext, volume_int;
5     cin >> d >> w >> h >> l;
6     volume_ext = w * h * l;
7     volume_int = (w - 2 * d) * (h - 2 * d) * (l - d);
8     cout << volume_ext - volume_int << endl;
9     return 0;
10 }
```

B. Bekarys and khet

Длина пути луча в каждой клетка равна 1. Значит, ответ — количество клеток, через которые пройдет луч. Для этого можно построить простой автомат, определяющий (x_k, y_k) — текущее положение, v_k — направление на входе в клетку.

Направление меняется следующим образом, если в клетке (x_k, y_k) находится зеркало вида '/' и вида '\'

$$v_{k+1} = \begin{cases} UP, & v_k = RIGHT \\ DOWN, & v_k = LEFT \\ RIGHT, & v_k = UP \\ LEFT, & v_k = DOWN \end{cases} \quad v_{k+1} = \begin{cases} DOWN, & v_k = RIGHT \\ UP, & v_k = LEFT \\ LEFT, & v_k = UP \\ RIGHT, & v_k = DOWN \end{cases}$$

Положение меняется в зависимости от направления естественным образом.

Асимптотика: $O(M \cdot N)$ (так как каждую клетку луч пересечет не более двух раз).

```

1 #include <iostream>
2 using namespace std;
3 int main() {
4     long long n, m;
5     char field[101][101];
6     cin >> n >> m;
7     for (int i = 0; i < n; i++)
8         cin >> field[i];
9
10    int dx[] = {0, 0, -1, 1};
11    int dy[] = {1, -1, 0, 0};
12    enum dir{RIGHT, LEFT, UP, DOWN};
13    int forward_slash[] = {UP, DOWN, RIGHT, LEFT};
14    int backward_slash[] = {DOWN, UP, LEFT, RIGHT};
15    int x = 0, y = 0, v = RIGHT;
16    int answer = 0;
17    while (x >= 0 && x < n && y >= 0 && y < m) {
18        if (field[x][y] == '/')
19            v = forward_slash[v];
20        if (field[x][y] == '\\')
21            v = backward_slash[v];
22        x += dx[v];
23        y += dy[v];
24        answer++;
25    }
26
27    cout << answer << endl;
28    return 0;
29 }
```

C. Computer vision

Обозначим: $dp[m][r]$ — количество m -значных чисел, которые соответствуют первым m символам шаблона и дают остаток r при делении на 11.

В случае, если символ $a[m]$ в шаблоне является цифрой, то количество $dp[m][r]$ определяется как количество соответствующих $(m - 1)$ -значных чисел:

$$dp[m][r_k] = dp[m - 1][k],$$

где $r_k = 10k + a[m] \pmod{11}$, для всех $k = 0, \dots, 10$.

В случае, если символ $a[m]$ является звездочкой, то вместо неё можно подставить любую цифру. Для фиксированного k от 0 до 9, значение $10 * k + r \pmod{11}$ дает все возможные остатки, кроме $10 - k$ (так как никакая цифра не может давать остаток $r = 10$ при делении на 11):

$$dp[m][r] = \sum_{k+r \neq 10, k=0}^{10} dp[m - 1][k] = dp[m][r] = \sum_{k=0}^{10} dp[m - 1][k] - dp[m - 1][10 - r],$$

Асимптотика: $O(N)$.

```

1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 int dp[100001][11];
6 int main() {
7     string s;
8     cin >> s;
9     long long mod = 1000000007LL;
10    int n = s.size();
11    dp[0][0] = 1;
12    for (int m = 1; m <= n; m++) {
13        if (s[m - 1] != '*') {
14            for (int k = 0; k < 11; k++) {
15                int r = (10 * k + s[m - 1] - '0') % 11;
16                dp[m][r] = dp[m - 1][k];
17            }
18        } else {
19            long long s = 0;
20            for (int k = 0; k < 11; k++)
21                s = (s + dp[m - 1][k]) % mod;
22            for (int r = 0; r < 11; r++) {
23                dp[m][r] = (s + mod - dp[m - 1][10 - r]) % mod;
24            }
25        }
26    }
27    cout << dp[n][0] << endl;
28    return 0;
29 }
```

D. DPK rover

При $M = 1$, ответ очевидно будет $D/2$. Пусть M — простое число. Обозначим x — расстояние, после которого первые $(M - 1)$ марсоходов отдадут топливо M -му марсоходу. На обратную дорогу они должны оставить себе столько же топлива, сколько потратили. Значит, каждый из них отдаст M -му марсоходу топлива, на котором можно проехать расстояние $(D - 2x)$. Итого: $(M - 1)(D - 2x)$.

У M -го марсохода бак свободен ровно настолько, сколько он уже проехал. Чтобы всё полученное топливо поместилось в его в бак необходимо выполнение:

$$x \geq (M - 1)(D - 2x).$$

Схема движения обратно соответствует и прямой схеме (если произвести все движения в обратном порядке). То есть полученного топлива должно хватать для движения обратно:

$$x \leq (M - 1)(D - 2x).$$

Таким образом, до точки разворота группа проедет:

$$x = \frac{M - 1}{2M - 1} D.$$

Покажем, что в случае с составным числом $M = A \cdot B$ всегда выгодно делиться на группы. Два деления (на группы размером B и далее на группы размером A) выгоднее, чем одно (на группу размера AB). Для этого проверим неравенство:

$$\begin{aligned} \frac{A - 1}{2A - 1} D + \frac{B - 1}{2B - 1} D &\geq \frac{AB - 1}{2AB - 1} D \\ \frac{(4AB - 1)(B - 1)(A - 1)}{(2AB - 1)(2B - 1)(2A - 1)} &\geq 0 \end{aligned}$$

Это верно с учетом $A > 1$ и $B > 1$.

Для решения задачи в общем виде достаточно получить каноническое разложение числа $M = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k}$ на простые множители. Сделать это можно стандартным алгоритмом разложения на простые множители, проверяя делители до \sqrt{M} . Ответ равен:

$$L = D \left(\frac{1}{2} + \sum_{i=1}^k \alpha_i \frac{p_i - 1}{2p_i - 1} \right).$$

Асимптотика: $O(\sqrt{M})$.

Решение за $O(M)$ не проходило ограничения по времени.

```
1 #include <iostream>
2 #include <iomanip>
3 using namespace std;
4 int main() {
5     long long d, m;
6     long double answer = 0.5;
7     cin >> d >> m;
8     for (long long p = 2; p * p <= m; p++) {
9         while (m % p == 0) {
10             answer += (p - 1.0) / (2.0 * p - 1.0);
11             m /= p;
12         }
13     }
14     if (m != 1) {
15         answer += (m - 1.0) / (2.0 * m - 1.0);
16     }
17     cout << fixed << setprecision(10) << answer * d << endl;
18     return 0;
19 }
```

E. Excellent idea

Необходимо было вывести k -й символ текста, где k — это число в этом же тексте. Число было во всех тестах на последнем месте. В некоторых тестах числа были больше 10. Во всех тестах числа не превышали длину теста.

Асимптотика: $O(N)$, где N — длина теста.

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4 int main() {
5     string s;
6     getline(cin, s);
7     int index = 0, n = s.size();
8     for (int i = 0; i < n; i++)
9         if (s[i] >= '0' && s[i] <= '9')
10             index = 10 * index + s[i] - '0';
11     cout << s[index - 1] << endl;
12     return 0;
13 }
```

F. Forced reduction

Решение 1. Дерево представим в виде списка ребер ориентированного графа (от родителя к детям). Для быстрого выделения ребер с одинаковыми буквами у каждой вершины будем хранить сразу 26 списков (по одному списку для каждой буквы).

Запустим обход в ширину из корня. При этом для каждой вершины, которая изымается из очереди, будет объединять всех её детей с одинаковыми ребрами. Например, у вершины v есть дети a_1, a_2, \dots, a_k , в которые ведут ребра с буквой a . В отличие от классического добавления в очередь всех вершин a_i , в очередь добавим только a_1 . При этом в список детей вершины a_1 добавим детей из a_2, a_3, \dots, a_k . То же самое исполним для списков b_i, c_i, \dots, z_i .

Каждый ребенок (точнее ребро) будет перенесено не более одного раза. Поэтому асимптотика аккумуляровано не превосходит $O(N)$. Поиск ребер с одинаковыми буквами требует $O(1)$ действий, так как у каждой вершины есть отдельный список для каждой буквы. Остается алгоритм обхода в ширину: $O(N)$ действий.

Асимптотика: $O(N)$.

Решение 2 (которое, увы, проходило на олимпиаде). Достаточно промоделировать требуемый процесс при обходе дерева в глубину или в ширину, начиная с корня. При этом поддерживать ответ в какой-либо структуре с элементами в виде строк (без повторов), например `set <string>`.

Такое решение в худшем будет работать на дереве типа «бамбук» (у каждого родителя ровно один ребенок). Длина k -й строки будет равна k . Поиск будет производиться за $O(\log k)$ в худшем, с добавлением за $O(k)$ в худшем. Общая сложность обработки $\sum_{k=1}^N k = O(N^2)$. Но оптимизации и кэширование значительно ускоряют процесс копирования строк.

Асимптотика: $O(N^2)$.

```
1 #include <iostream>
2 #include <vector>
3 #include <queue>
4 using namespace std;
5
6 vector < vector < vector <int> > > son;
7 int main() {
8     int n;
9     cin >> n;
10    son.resize(n + 1);
11    for (int from = 1; from <= n; from++)
12        son[from].resize(26);
13    for (int to = 2; to <= n; to++) {
14        int from;
15        char ch;
16        cin >> from >> ch;
17        son[from][ch - 'a'].push_back(to);
18    }
19
20    queue <int> q;
21    q.push(1);
22    int answer = 0;
23    while (!q.empty()) {
24        int current = q.front();
25        q.pop();
26        for (char ch = 0; ch < 26; ch++) {
27            if (son[current][ch].size() > 0) {
28                int dst = son[current][ch][0];
29                for (size_t i = 1; i < son[current][ch].size(); i++) {
30                    int src = son[current][ch][i];
31                    for (char c = 0; c < 26; c++) {
32                        for (size_t j = 0; j < son[src][c].size(); j++)
33                            son[dst][c].push_back(son[src][c][j]);
34                    }
35                }
36                q.push(dst);
37                answer++;
38            }
39        }
40    }
41
42    cout << answer << endl;
43    return 0;
44 }
```

G. Giant chandelier

Центр тяжести люстры:

$$\vec{m} = \frac{1}{n} \sum_{i=1}^n \vec{v}_i.$$

Чтоб понять насколько отклоняется люстра вдоль центра тяжести, найдем длины проекций каждого вектора на вектор \vec{m} через скалярное произведение:

$$p_i = (\vec{v}_i, \vec{e}),$$

где $\vec{e} = \frac{\vec{m}}{|\vec{m}|}$.

Ответ на задачу: $\max p_i - \min p_i$.

Асимптотика: $O(N)$.

```
1 #include <iostream>
2 #include <iomanip>
3 #include <cmath>
4 using namespace std;
5
6 struct Point {
7     double x, y, z;
8 };
9 double dot(const Point &a, const Point &b) {
10     return a.x * b.x + a.y * b.y + a.z * b.z;
11 }
12
13 Point lamp[100000];
14 int main() {
15     int n;
16     cin >> n;
17     Point m = {0, 0, 0};
18     for (int i = 0; i < n; i++) {
19         cin >> lamp[i].x >> lamp[i].y >> lamp[i].z;
20         m.x += lamp[i].x; m.y += lamp[i].y; m.z += lamp[i].z;
21     }
22     double length = sqrt(dot(m, m));
23     m.x /= length; m.y /= length; m.z /= length;
24     double p_min = 0.0, p_max = 0.0;
25     for (int i = 0; i < n; i++) {
26         double d = dot(lamp[i], m);
27         if (p_min > d)
28             p_min = d;
29         if (p_max < d)
30             p_max = d;
31     }
32     cout << fixed << setprecision(10) << p_max - p_min << endl;
33     return 0;
34 }
```

Н. Hyper numbers

Заметим, что всегда верно $(n, x) = (n, n - x)$. По определению гипер-чисел, не может быть несколько взаимно простых чисел. Значит, $x = n - x$. Тогда $n = 2x$ и $(n, x) = x$. Опять же по определению эти числа должны быть взаимно просты, то есть $x = 1$, $n = 2$. Таким образом, достаточно проверить, входит ли число 2 в указанный диапазон.

Асимптотика: $O(1)$.

```
1 #include <iostream>
2 using namespace std;
3 int main() {
4     long long L, R, answer;
5     cin >> L >> R;
6     answer = (L == 2);
7     cout << answer << endl;
8     return 0;
9 }
```