

Как дебагать

Интерактивки: 0) сбрасывайте буфер ввода - endl 1) не дебагайте руками интерактивки, пишите интерактор и функцию ask 2) есть вердикт ожидания ответа 3) не верьте вердиктам 4) интеракторы бывают адаптивными или обычными Двойной запуск: 1) работают так же как интерактивки 2) "В задачах с двойным запуском время работы и память измеряются отдельно в каждом из запусков. памятка участника Открытые тесты: 1) выучите freopen("input.in"r stdin) и freopen("output.out"w stdout); 2) научить распаковывать zip архив Задачи с грейдером: 1) оценка за тест или набор тестов вычисляется по формуле и всё

Этот файл для региона, в будущем сделаю его и для всероса, добавлю интерактивки, двойной запуск и открытые тесты. WL Лёха

Общая информация

Основные выводы из памятки участника регионального этапа:

- у вас всего 50 попыток на одну задачу, 200 суммарно по всем задачам. Раньше было ограничение 150 попыток суммарно;
- берется лучшее решение (склейки нету, если первое решение заходит на одну подгруппу, а второе на другую, то заифайте и объедините решения);
- из системы нельзя выгружать посылки, поэтому сохраняйте все решения (например, название файла - номер посылки в системе);
- в задачах возможна потестовая оценка.

Не заходит по PE (presentation error)

PE на реге 21 века. Что за кринж? Да, на регионе может быть PE, но это не значит, что если вы выведете в неправильном формате, то у вас будет PE, может и WA.

В памятке участника как пример возникновения PE: "Программа должна вывести числа в одной строке через пробел, а вывела их в разных строках или наоборот". Современные чекеры не будут ругаться на такие случаи, но все же выводите ответ, как просит задача.

Бывает два вида чекеров (проверяющих ответ участника программ):

- **Стандартный** (проверяет, равен ли ответ участника ответу жюри). В таком случае PE можно получить только если выводить совсем не то, например строку вместо числа.
- **Кастомный** (когда верных ответов несколько и нельзя просто сверять с ответом жюри). В таком случае PE совсем нельзя получить, всегда будете получать WA.

В целом нужно воспринимать PE как WA, скорее всего оно не имеет смысла.

Не заходит по WA (wrong answer)

- Забыли убрать отладочный вывод.
- Когда используете double, может вывести nan или inf (возможно выведет, как PE).
- Переполнение int.
- Undefined behavior (поведение зависит от компилятора, бывает, например, при выходе за границы стат. массива).
- Неверная идея или баг в коде (ну тут мои полномочия всё).

Не заходит по TL (time limit)

На полигоне (g++ 14) одна секунда - это $2e9$ ifov (потому что прагма O2 автоматически включена на CF), $2e8$ операций деления и $5e8$ операций остатка. На моем компьютере (g++ 17) одна секунда - это $7e8$ операций сложения, $3e8$ операций деления, $5e8$ ifov, $4e8$ остатка. И появляется резонный вопрос, чему равна одна секунда в c++? Невозможно понять из-за константы операций, однако я думаю безопасно считать, что это примерно от $1e8$ до $3e8$ действий, вы сможете сделать больше если захотите.

Однако создатели задач региона и всероса очень редко планируют, чтобы вы их запикивали (либо дают за это мало баллов). Поэтому:

Ограничения по n	Авторская асимптотика	Вы можете упихать (я в вас верю >-<)
до 10	$O(n \cdot n!)$	$O(n^2 \cdot n!)$ или $O(n^8)$
до 15-18	$O(3^n)$	$O(3^n)$
до 20	$O(2^n)$ или $O(2^n \cdot n)$	$O(2^n \cdot n^2)$
до 40-50 (MITM)	$O(2^{\frac{n}{2}})$ или $O(2^{\frac{n}{2}} \cdot n)$	То же самое, но с большой константой
до 1000	$O(n^2)$ или $O(n^2 \cdot \log n)$	$O(n^2 \cdot \log^2 n)$
до $1e5$	$O(n)$, $O(n \cdot \log(n))$, $O(n \cdot \log^2 n)$ или $O(n \cdot \sqrt{n})$	$O(\cdot n \cdot \log^2 n)$ или $O(\cdot n \cdot \sqrt{n})$, где $C=10$
до $3e5$	То же самое, но без $O(n \cdot \log^2 n)$	То же самое, только C меньше $C=3$
до $1e6$	$O(n)$, $O(n \cdot \log n)$	$O(C \cdot n \cdot \log n)$, $C=5$

Не забывайте, что у вас может появиться бесконечный цикл. Тут ускорения не помогут :)

“Программа ждет ввода данных, хотя входной поток уже закончился” - памятка участника. Не думаю, что это у вас будет)

Ускорение потоков ввода и вывода

Решения были запущены на одном и том же коде, вводящем $1e5$ чисел и выводящем их, с добавлением `(ios_base::sync_with_stdio(0), cin.tie(0) или cout.tie(0))`

Видно, что `cout.tie(0)` бесполезно. ВЫУЧИТЕ другие два ускорения, ведь они в 3 раза улучшают время, и не каждая среда может подсказать.

Правильно расставляйте типы переменных, т. к. `int` занимает меньше времени на создание.

Сделать статические массивы вместо векторов. При этом в местах, где `vector` всё же используется как стек, перед использованием сделать `v.reserve(MAXN)`.

Заменить `map` и `set` на `unordered_map` и `unordered_set`. Не забудьте сделать `mp.reserve(MAXN)`, иначе будет долго работать!

Бывают ситуации, когда нужно ключом `unordered_map` сделать структуру. Для этого нужно структуру захешировать и записывать в `unordered_map` хэш.

Прагмы

Стандартный набор, который я использую во всех решениях:

```
#pragma GCC optimize("O3,unroll-loops")

// #pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
```

Заметим, что вторая строка закомментирована. Если не заходит без неё, можно попробовать её раскомментировать и заслать. Но по умолчанию отправляйте без неё.

Почитать, за что каждая из них отвечает: <https://codeforces.com/blog/entry/129283?locale=ru>.

Пытаться загнать прагмы только тогда, когда вам нечего больше делать!

По возможности избавляться от любых делений и остатков по модулю. Например, взятие остатка по модулю, когда пишешь хэш, работает дольше, чем `if` сумма больше модуля.

Путем проб и ошибок стало понятно, что переписывать решение в тернарные операторы для ускорения по времени бессмысленно.

Переход по столбцам а потом по строкам в массиве `(j, i)` работает дольше, чем по строкам, а потом по столбцам `(i, j, стандартный обход)`, из-за прыжков по памяти длиной `n`, вместо 1.

Не заходит по ML (Memory limit)

256 мб - это примерно $6e7$ `int` и в 2 раза меньше `long long` и `double`.

Правильно расставляйте типы переменных, `int` занимает в 2 раза меньше бит, чем `long long`.

Большая рекурсия, возможно стоит попытаться заменить на циклы или передавать меньше параметров.

“Ошибки при работе с указателями в C/C++ также могут диагностироваться, как ML” - памятка участника (Устарело, бывает в старых плюсах).

`Unordered_map`, `unordered_set` стоит заменить на `map`, `set`.

Добавить к `vector`, `unordered_map` или `unordered_set` после момента их создания `.reserve(MAXN)`.

Не заходит по RE (Runtime error)

RE может быть не видно на вашем компьютере, это называется UB (Undefined Behavior).

`#define _GLIBCXX_DEBUG` — отлавливает все RE в STL-структурах. Если получили RE на 1 тесте, а локально всё правильно, надо написать это в начало кода (до `includes`). Также стоит заметить, что эта дебаг может делать TL, поэтому используйте его только локально (`#ifdef LOCAL`).

Превышение лимита по памяти также может диагностироваться, как “Ошибка выполнения” - памятка участника. (Устарело, возможно только на informatics.)

- Выход за пределы массива.
- Деление на 0 (причем в `double` и `long double` не возникает RE, компилятор выведет `inf`).
- Разыменовывание указателя на `end` в векторах и сетях.
- Забыли убрать `assert`, с помощью него можно удобно дебагать и `if`ать тест.

Немного мудрости от ваших любимых тренеров Москвы

Набирайте частички, не в ОКах счастье:)

Не тратьте много времени на одной задаче, придумайте для себя стратегию, которая не зависит от сложности задач или аномальных задач (C2 на рюкзак 2025 года).

Я (Лёха) решал в порядке DABC на то кол-во баллов, что мог, тратил не больше 20 минут на дебаг или размышления о задаче (чтение задачи и написание кода сюда не входит). Если понимал, что 20 минут прошли, то я переходил к следующей задаче по циклу. А в последний час оценивал ситуацию и решал, что мне лучше делать, чтобы набрать как можно больше баллов.

Можно сначала решить D и C на халявные подгруппы, а потом уже решать A и B на большие баллы. Так не надо будет беспокоиться о том, что остался 0 по D.

Хороший рандом

```
mt19937 rng(73);

uniform_int_distribution<int> uid(0, int(1e9));
int rnd(int n) {    // случайное целое число от 0 до n - 1 включительно
    return uid(rng) % n; // если берете всегда по одному и тому же модулю, лучше uid
}

ld rndd() {    // случайное вещественное число от 0 до 1
    return ld(rand()) / RAND_MAX;
}
```

```
vector<int> v;  
shuffle(v.begin(), v.end(), rng); // перемешать все числа
```

Сводите задачи к более мелким и решайте их по отдельности.

Если вы пишете ДО, геому или любой другой алгоритм, создавайте удобную структуру или шаблон, который вы умеете по памяти безошибочно писать, чтобы не тратить много время на дебаг уже известного алгоритма.

Сppreference поможет, если забыли, как что-то работает. Попробуйте им пользоваться дома и на пробном туре.

Если идеи не приходят

Если в условии дан граф, иногда бывает полезно запускать DFS от случайной вершины и сделать shuffle списков смежности. Это иногда позволяет немного уменьшить время работы (т.к. жюри чаще всего делает тесты под запуск от корня 1).

То же самое можно сказать про любую задачу, где структуру входных данных можно сделать более случайной.

Бывает полезно взять очень много жадников и рандомных решений и вывести наилучший из ответов, которые они получили.

Можно группы с маленьким n ифать и запускать на них переборное решение, а на больших группах запускать идею из 5 пункта.

У тебя всё получится, мы верим в тебя =)

Спасибо Антону Ныйкину, Антону Барисову, Марку Семенову, Тимуру Лузгову и всем прочитавшим файл за помощь в его создании.