Разработка задач в системе Polygon

Александр Горбунов (Gornak40)

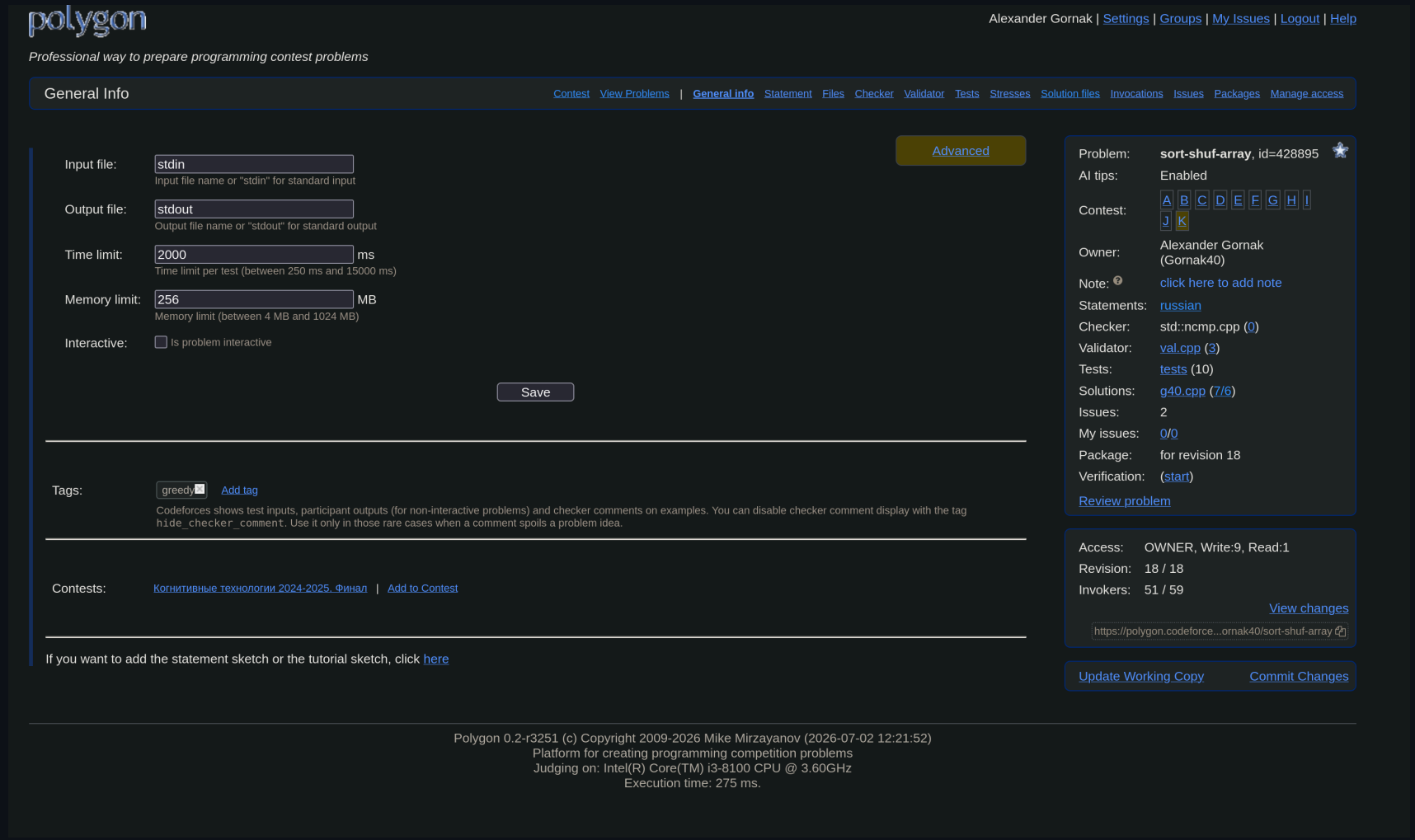
Обо мне

- Регулярно преподаю на московских сборах и в ЛКШ
- Координирую крупную перечневую олимпиаду
- Готовил и координировал раунды на Codeforces
- Работал админом в Т-Поколении
- Написал тулинг для админов github.com/Gornak40/algolymp

Polygon и `testlib.h`

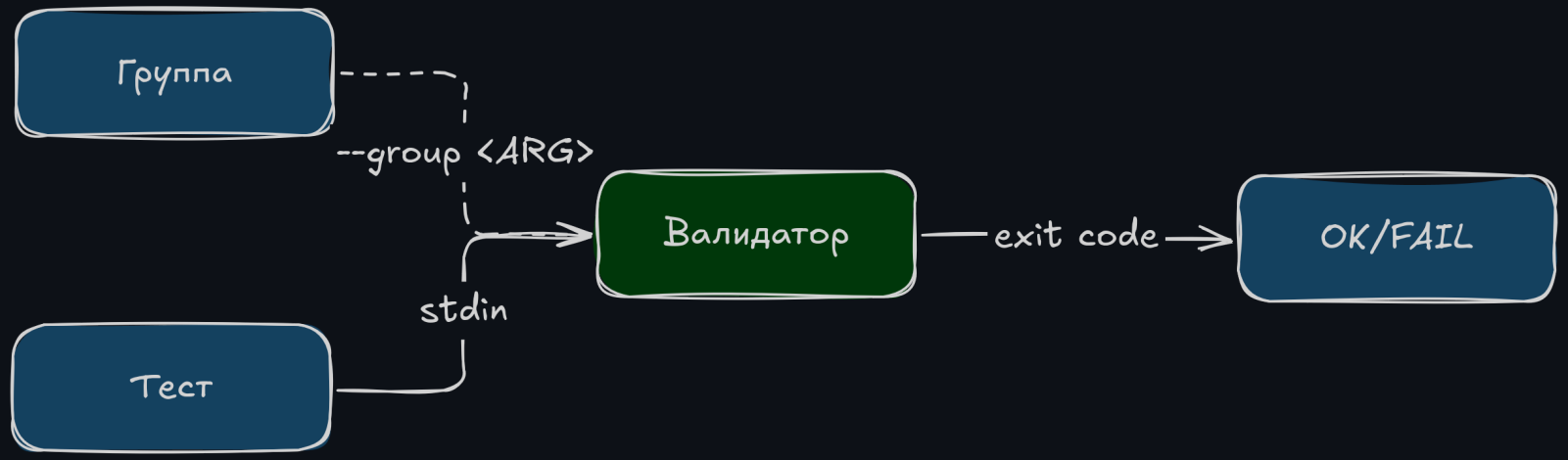
- *Professional way to prepare programming contest problems*
- Бесплатный веб-сервис polygon.codeforces.com
- Разработан MikeMirzayanov в инфраструктуре codeforces.com
- Упрощает и стандартизирует подготовку задач
- Построен вокруг open-source либы github.com/MikeMirzayanov/testlib
- Используется авторами всех серьёзных российских соревнований
- Интегрируется с ejudge, PCMS, Яндекс контест и т.д.
- Предоставляет API codeforces.github.io/polygon-misc/API
- *Позволяет совместно разрабатывать задачи

Интерфейс Polygon



Валидатор на `testlib.h`

Программа, которая проверяет входные данные



Валидатор на `testlib.h`

```cpp
// Числа `n` и `k` на первой строке, массив из `n` чисел от 1 до 10 на второй
#include "testlib.h"
using namespace std;

int main(int argc, char* argv[]) {
    registerValidation(argc, argv);
    int n = inf.readInt(1, 200'000, "n");
    inf.readSpace();
    int k = inf.readInt(1, 1'000'000'000, "k");
    inf.readEoln();
    inf.readInts(n, 1, 10, "a");
    inf.readEoln();
    inf.readEof();
}
```

```
./val
./val < 01.txt
```

Валидатор на `testlib.h`

```cpp
// Дерево из `n` вершин в виде списка рёбер
#include "testlib.h"
#include <numeric>
using namespace std;

int main(int argc, char* argv[]) {
    registerValidation(argc, argv);
    int n = inf.readInt(1, 200'000, "n");
    inf.readEoln();
    vector<int> lnk(n); // CHM
    iota(lnk.begin(), lnk.end(), 0);
    function<int(int)> getp =
        [&lnk,&getp](int v) { return v == lnk[v] ? v : lnk[v] = getp(lnk[v]); };
    for (int i = 1; i < n; ++i) {
        int v = inf.readInt(1, n, "v");
        inf.readSpace();
        int u = inf.readInt(1, n, "u");
        inf.readEoln();
        int vv = getp(v - 1), uu = getp(u - 1);
        ensuref(vv != uu, "graph contains cycle %d <-> %d", v, u);
        lnk[uu] = vv;
    }
    inf.readEof();
}
```

Валидатор на `testlib.h`

```cpp
// Массив из `n` элементов от 1 до 1'000'000'000
#include "testlib.h"
using namespace std;

const int MAXN[] = {200'000, 100, 2000, 2000, 200'000};

int main(int argc, char* argv[]) {
    registerValidation(argc, argv);
    int gr = getGroup()[0] - '0';
    int n = inf.readInt(1, MAXN[gr], "n");
    inf.readEoln();
    auto arr = inf.readInts(n, 1, 1'000'000'000, "a");
    inf.readEoln();
    if (gr == 3) ensuref(is_sorted(arr.begin(), arr.end()), "array is not sorted");
    inf.readEof();
}
```

```
./val --group 1
```

Валидатор на `testlib.h`

- Подключите заголовочный файл `testlib.h`
- В начале `main` вызовите `registerValidation(argc, argv);`
- Не забывайте про пробелы, переводы строк и EOF
- Указывайте имена переменных из условия во всех `read` функциях
- Используйте `ensuref` для проверки условий
- Пишите понятные сообщения в `ensuref`
- Используйте `readInts` вместо цикла
- Напишите много тестов на валидатор
- Если в валидатор нужно вставить решение, меняйте задачу?

Генератор на `testlib.h`

Программа, которая генерирует входные данные

Генератор на `testlib.h`

```cpp
// Строка длины `n` из символов от 'a' до `max-c`
#include "testlib.h"
using namespace std;

int main(int argc, char* argv[]) {
    registerGen(argc, argv, 1);
    int n = opt<int>("n", 200'000);
    auto max_c = opt<string>("max-c", "z");
    auto s = rnd.next(string("[a-") + max_c + "]{" + to_string(n) + "}");
    println(n);
    println(s);
}
```

```
./gen
./gen -n 100
./gen -n 2000 -max-c b
```

Генератор на `testlib.h`

```cpp
// Массив длины `n` из чисел от 1 до 10 и число `k` от 1 до `max-k`
#include "testlib.h"
using namespace std;

const int MAXA = 10; // Что-то можно не параметризировать

int main(int argc, char* argv[]) {
    registerGen(argc, argv, 1);
    int n = opt<int>("n", 200'000);
    int max_k = opt<int>("max-k", 1'000'000'000);
    int k = rnd.next(1, max_k);
    bool razn = opt<bool>("razn"); // -razn
    bool srt = opt<bool>("srt"); // -srt
    vector<int> arr(n);
    if (razn) arr = rnd.distinct(n, 1, MAXA);
    else for (int i = 0; i < n; ++i)
        arr[i] = rnd.next(1, MAXA);
    if (srt) sort(arr.begin(), arr.end());
    println(n, k);
    println(arr);
}
```

Генератор на `testlib.h`

```cpp
// Массив предков дерева из `n` вершин с корнем в 1
#include "testlib.h"
#include <numeric>
using namespace std;

int main(int argc, char* argv[]) {
    registerGen(argc, argv, 1);
    int n = opt<int>("n", 200'000);
    int w = opt<int>("w");
    vector<int> prr(n);
    for (int v = 1; v < n; ++v)
        prr[v] = rnd.wnext(v, w);
    vector<int> perm(n);
    iota(perm.begin(), perm.end(), 0);
    shuffle(perm.begin() + 1, perm.end());
    vector<int> par(n - 1);
    for (int v = 1; v < n; ++v)
        par[perm[v] - 1] = perm[prr[v]] + 1;
    println(n);
    println(par);
}
```

Генератор на `testlib.h`

- Подключите заголовочный файл `testlib.h`
- В начале `main` вызовите `registerGen(argc, argv, 1);`
- Используйте `opt<int>`, `opt<bool>`, `opt<string>` для аргументов
- Используйте значения аргументов по умолчанию
- Используйте `println` для вывода теста
- Всю рандомизацию пишите с `rnd` из `testlib.h`
- Не используйте `rand`, `random_shuffle`, `std::mt19937`, `std::shuffle`
- Ознакомьтесь с примерами в github.com/MikeMirzayanov/testlib
- Загрузите генератор в `Files -> Source Files` в Polygon

Скрипт генерации

- Знак `$` заменяется на номер теста инкрементально
- Можете указать сид рандома свободным аргументом
- Не делайте много одинаковых тестов с разными сидами
- Не используйте `FreeMaker`

```
gen -t 10000 -max-n 5 -sum-a 25 > $
gen -t 10000 -max-n 20 -sum-a 500 > $
gen -t 1000 -max-n 200 -sum-a 40000 > $
gen -t 100 -max-n 2000 -sum-a 4000000 > $
gen -t 10 -max-n 20000 -sum-a 400000000 > $
gen -t 1 -min-n 200000 > $
gen -t 1 -min-n 200000 -sum-a 10000000 > $
gen -t 1 -min-n 200000 -sum-a 400000 > $
gen_max > $
```

Разметка групп

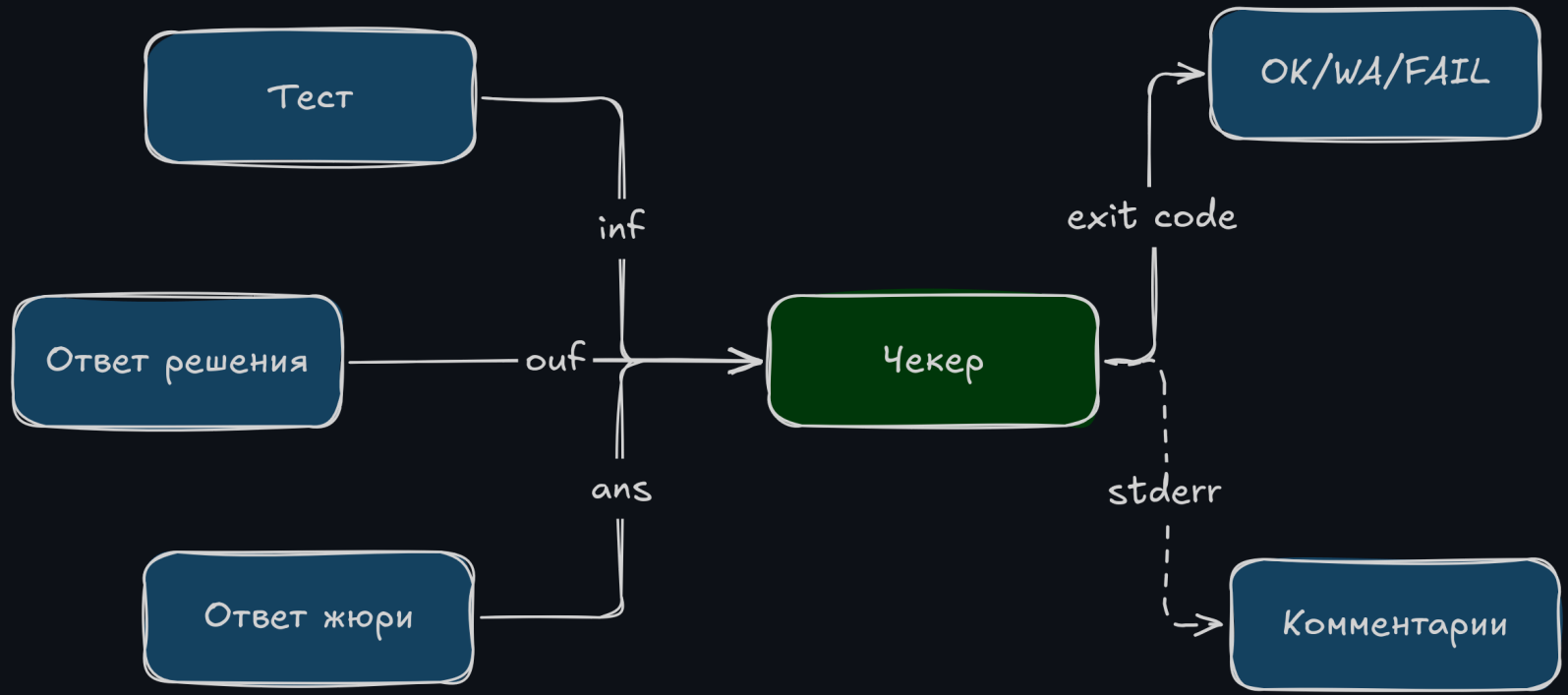
- Обращайте внимание на предупреждения Polygon
- `COMPLETE_GROUP` для групп, `EACH_TEST` для потестовой
- Называйте группы последовательно 0, 1, 2, ...
- Расставляйте полные зависимости, игнорируйте группу с семплами

Groups points policy and dependencies

Name	Tests	Points	Points policy	Feedback policy	Dependencies	
0	1	0	COMPLETE_GROUP Change?	COMPLETE Change?	Add	
1	8	11	COMPLETE_GROUP Change?	ICPC Change?	Add	
2	8	17	COMPLETE_GROUP Change?	ICPC Change?	1 Add	
3	8	25	COMPLETE_GROUP Change?	ICPC Change?	Add	
4	8	14	COMPLETE_GROUP Change?	ICPC Change?	1 2 Add	
5	8	33	COMPLETE_GROUP Change?	ICPC Change?	1 2 3 4 Add	

Чекер на `testlib.h`

Программа, которая проверяет решение



Чекер на `testlib.h`

```cpp
// Индекс максимума в массиве из `n` элементов
#include "testlib.h"
using namespace std;

inline void readAndCheck(InStream& in, int n, const vector<int>& arr) {
    int idx = in.readInt(1, n, "i");
    int mx = *max_element(arr.begin(), arr.end());
    in.ensuref(arr[idx - 1] == mx, "element with index %d is not max", idx);
}

int main(int argc, char* argv[]) {
    registerTestlibCmd(argc, argv);
    int n = inf.readInt();
    auto arr = inf.readInts(n);
    readAndCheck(ans, n, arr);
    readAndCheck(ouf, n, arr);
    quitf(_ok, "ok");
}
```

Чекер на `testlib.h`

```cpp
// Строка из символов 'a', 'b', 'c' длины `n`
#include "testlib.h"
using namespace std;

inline int readAndCheck(InStream& in, int n) {
    auto s = in.readToken("[a-c]{" + to_string(n) + "}", "s");
    int cnt[3]{};
    for (char c: s) ++cnt[c - 'a'];
    return *max_element(cnt, cnt + 3) - *min_element(cnt, cnt + 3);
}

int main(int argc, char* argv[]) {
    registerTestlibCmd(argc, argv);
    int n = inf.readInt();
    int jur = readAndCheck(ans, n);
    int par = readAndCheck(ouf, n);
    if (jur > par) quitf(_fail, "participant got better (%d < %d)", par, jur);
    if (jur < par) quitf(_wa, "jury got better (%d < %d)", jur, par);
    quitf(_ok, "answer is %d", jur);
}
```

Чекер на `testlib.h`

- Если возможно, используйте `ncmp.cpp`, `nyesno.cpp` или `wcmp.cpp`
- Проверьте, что ни один стандартный чекер вам не подходит
- Подключите заголовочный файл `testlib.h`
- В начале `main` вызовите `registerTestlibCmd(argc, argv);`
- Совсем не валидируйте чтения из `inf`
- Вызывайте `quitf` и `ensuref` от `InStream&` (`readAndCheck` паттерн)
- Пишите понятные сообщения в `quitf` и `ensuref`
- Строго валидируйте вывод участника, указывайте имена переменных
- Не читайте пробелы и переводы строк, они игнорируются
- Напишите много тестов на чекер

Решения

- Запускайте решения во вкладке `Invocations`
- Не вставляйте в авторское fast IO
- Напишите решения на подгруппы
- Напишите решение на Питоне
- Напишите несколько неправильных решений TL/WA/RE
- Попросите коллег решить задачу или напишите второе авторское
- Не ставьте слишком жёсткие ограничения TL/ML

#	brute.cpp	g40.cpp	g40_auto.cpp	g40_avx.cpp	g40_tl.cpp
1 0	OK 31 / 0	OK 0 / 16	OK 15 / 1	OK 0 / 1	OK 15 / 13
2 1	OK 15 / 0	OK 15 / 16	OK 15 / 1	OK 15 / 1	OK 15 / 12
3 1	OK 46 / 0	OK 31 / 16	OK 15 / 1	OK 15 / 1	OK 0 / 13
4 1	OK 15 / 0	OK 31 / 16	OK 15 / 1	OK 15 / 1	OK 0 / 13
5 1	OK 15 / 0	OK 31 / 16	OK 15 / 1	OK 0 / 1	OK 31 / 13
6 1	OK 15 / 0	OK 31 / 16	OK 15 / 1	OK 15 / 1	OK 15 / 13
7 1	OK 15 / 0	OK 31 / 16	OK 31 / 1	OK 15 / 1	OK 15 / 13
8 1	OK 15 / 0	OK 0 / 16	OK 15 / 1	OK 0 / 1	OK 0 / 13
9 1	OK 15 / 0 / 16.00000	OK 15 / 16 / 16.00000	OK 15 / 1 / 16.00000	OK 0 / 1 / 16.00000	OK 15 / 13 / 16.00000
10 2	OK 906 / 0	OK 15 / 16	OK 15 / 1	OK 15 / 1	OK 1953 / 13
11 2	OK 906 / 0	OK 46 / 16	OK 31 / 1	OK 15 / 1	OK 1921 / 13
12 2	OK 875 / 0	OK 46 / 16	OK 15 / 1	OK 15 / 1	TL 2000 / 13
13 2	OK 890 / 0 / 8.00000	OK 62 / 16 / 8.00000	OK 31 / 1 / 8.00000	OK 15 / 1 / 8.00000	OK 1984 / 13 / 8.00000
14 3	TL 2000 / 0	OK 765 / 16	TL 2000 / 1	TL 2000 / 1	OK 859 / 13
15 3	TL 2000 / 0	OK 765 / 16	TL 2000 / 1	TL 2000 / 1	OK 828 / 13
16 3	TL 2000 / 0	OK 859 / 16	TL 2000 / 1	TL 2000 / 1	OK 812 / 13
17 3	TL 2000 / 0	OK 796 / 16	TL 2000 / 1	TL 2000 / 1	OK 828 / 13
18 3	TL 2000 / 0	OK 843 / 16	TL 2000 / 1	TL 2000 / 1	OK 890 / 12
19 3	TL 2000 / 0	OK 906 / 16	TL 2000 / 1	TL 2000 / 1	OK 796 / 13
20 3	TL 2000 / 0 / 14.00000	OK 812 / 16 / 14.00000	TL 2000 / 1 / 14.00000	TL 2000 / 1 / 14.00000	OK 718 / 13 / 14.00000
21 4	TL 2000 / 0	OK 875 / 16	TL 2000 / 1	TL 2000 / 1	TL 2000 / 13
22 4	TL 2000 / 0	OK 828 / 16	TL 2000 / 1	TL 2000 / 1	TL 2000 / 13
23 4	TL 2000 / 0	OK 796 / 16	TL 2000 / 1	TL 2000 / 1	TL 2000 / 13
24 4	TL 2000 / 0	OK 703 / 16	TL 2000 / 1	TL 2000 / 1	TL 2000 / 13
25 4	TL 2000 / 0 / 10.00000	OK 796 / 16 / 10.00000	TL 2000 / 1 / 10.00000	TL 2000 / 1 / 10.00000	TL 2000 / 13 / 10.00000
26 5	TL 2000 / 0	OK 875 / 16	TL 2000 / 1	TL 2000 / 1	TL 2000 / 13
27 5	TL 2000 / 0	OK 843 / 16	TL 2000 / 1	TL 2000 / 1	TL 2000 / 13
28 5	TL 2000 / 0	OK 828 / 16	TL 2000 / 1	TL 2000 / 1	TL 2000 / 13
29 5	TL 2000 / 0	OK 859 / 16	TL 2000 / 1	TL 2000 / 1	TL 2000 / 13
30 5	TL 2000 / 0	OK 828 / 16	TL 2000 / 1	TL 2000 / 1	TL 2000 / 13
31 5	TL 2000 / 0	OK 843 / 16	TL 2000 / 1	TL 2000 / 1	TL 2000 / 13
32 5	TL 2000 / 0	OK 937 / 16	TL 2000 / 1	TL 2000 / 1	TL 2000 / 13
33 5	TL 2000 / 0	OK 906 / 16	TL 2000 / 1	TL 2000 / 1	TL 2000 / 12
34 5	TL 2000 / 0 / 18.00000	OK 812 / 16 / 18.00000	TL 2000 / 1 / 18.00000	TL 2000 / 1 / 18.00000	TL 2000 / 13 / 18.00000
35 6	TL 2000 / 0	OK 765 / 16	OK 1625 / 1	OK 1625 / 1	OK 1015 / 13
36 6	TL 2000 / 0	OK 781 / 16	OK 1500 / 1	OK 1593 / 1	OK 1140 / 13
37 6	TL 2000 / 0	OK 843 / 16	OK 1593 / 1	OK 1671 / 1	OK 1109 / 13
38 6	TL 2000 / 0	OK 843 / 16	OK 1656 / 1	OK 1437 / 1	OK 1000 / 12
39 6	TL 2000 / 0	OK 828 / 16	OK 1437 / 1	OK 1484 / 1	OK 1046 / 13

Условие

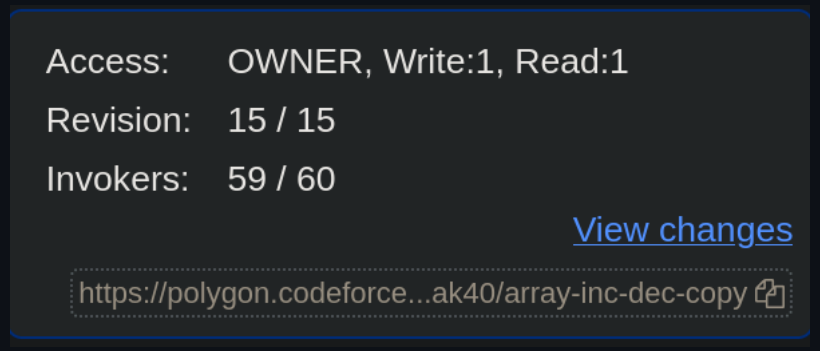
- Обращайте внимание на предупреждения Polygon
- Каждая переменная должна обозначать что-то одно
- Легенда должна хорошо ложиться на формальное условие
- Не бойтесь простых предложений и повторений слов
- В конце легенды повторите, что должен вывести участник
- Поясняйте и повторяйте все моменты, которые кажутся непонятными
- Формат входных/выходных данных это не часть легенды
- Пишите ограничения на входные данные `($1 \le N \le 2 \cdot 10^5$)`
- Выделения текста `\textbf`, `\textit`, `texttt` часто помогают
- Проверьте сборку в HTML и PDF, она не проверяется автоматически

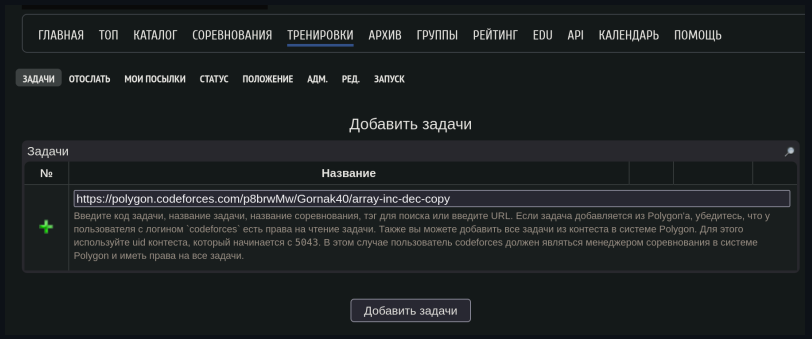
Общие советы

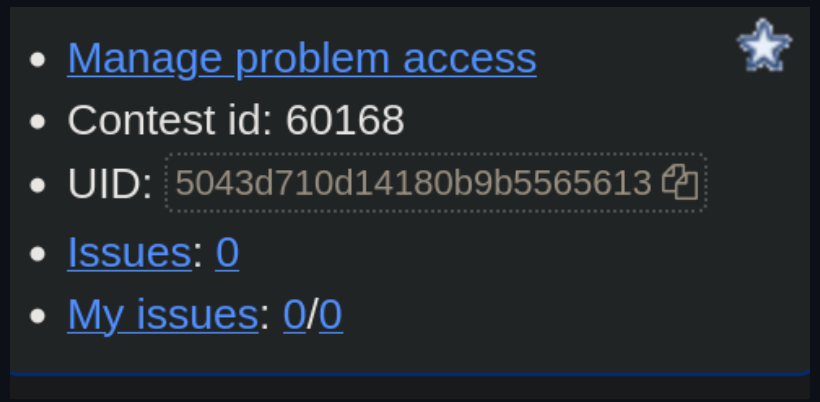
- Обращайте внимание на предупреждения Polygon
- Не бойтесь читать код `testlib.h`
- Исследуйте чужие задачи, скачайте архивы олимпиад
- Чаще коммитьте свои изменения
- Если что-то потерялось, используйте `Drafts`
- Запросите ревью у коллег, пусть открывают issues
- Собирайте пакет с верификацией
- При совместной разработке заранее разделите ответственность
- Не забывайте ставить галочку `Don't send email notification`
- Не используйте ИИ для генерации условия/генераторов и т.д.

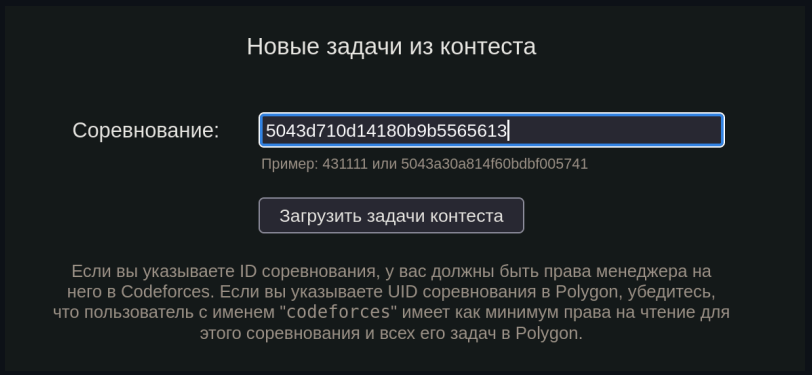
Использование задачи на Codeforces

- Пошарьте задачу на пользователя `codeforces`
- Автоматическое обновление при сборке пакета









Спасибо за внимание

- t.me/gornak40
- gornak40.org