ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГАОУ ВО НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Прикладная математика и информатика»

Отчет о программном проекте на тему:

Разработка задач для олимпиад школьников

Выполнил студент:

группы БПМИ221, 2 курса Дацковский Алексей Сергеевич

Принял руководитель проекта:

Густокашин Михаил Сергеевич
Директор Центра студенческих олимпиад
Факультет компьютерных наук НИУ ВШЭ

Москва 2024

Содержание

Аннотация

В данной работе описывается процесс составления и разработки задач по программированию для школьных олимпиад, проводимых Высшей школой экономики. Целью работы — вклад в школьное олимпиадное движение: составление задач для олимпиад и помощь другим составителям в подготовке задач. В результате был подробно описан процесс подготовки задачи в системе Polygon. Также было подготовлено 2 задачи, которые были использованы в школьных олимпиадах.

Ключевые слова

Олимпиады, информатика, спортивное программирование, задачи, разработка задач, Polygon, тестирование задач

1 Введение

При решении задач по программированию от участников требуется проявить знания в области алгоритмов, структур данных и математики, показать умение реализовывать решения задач в виде кода. Данные навыки важны для будущего обучения в институте в области компьютерных наук и для работы в IT-сфере. Для стимулирования развития вышеописанных навыков проводятся школьные олимпиады по информатике, в рамках которых участники соревнуются в решении задач, подготовленных жюри. По результатам олимпиад лучшие участники получают привилегии при поступлении в ВУЗы, а также становятся востребованными сотрудниками на рынке труда.

Для проведения честного соревнования, на котором участники находятся в равных условиях, для каждого тура разрабатывается набор уникальных задач. Особую важность имеет новизна задач: задачи должны отличаться от задач, которые школьники могли решать при подготовке к олимпиаде: задачи школьных олимпиад прошлых лет, задачи из архивов (Codeforces [7], Timus, AtCoder, SortMe, Algocode).

При решении задачи по информатике участнику необходимо написать код, который будет корректно работать в рамках заданных ограничений. Для оценки правильности решения используется тестирование на заранее подготовленном наборе тестов. Количество баллов, которое участник получает за своё решение, зависит от пройденных тестов и также устанавливается жюри. Для корректной оценки решений участников важно разработать тесты, которые покрывают обширный набор случаев.

При разработке задач используется система Polygon, которая предоставляет инструменты для составления условия задачи, генерации тестов, тестирования решений. Также с её помощью удобно собирать задачи в контест (набор задач для олимпиады) и экспортировать тур в тестирующую систему для непосредственного проведения олимпиады. Система была создана Михаилом Мирзаяновым в 2009 году и имеет свои недостатки, но при этом является самым популярным способом подготовки задач по олимпиадному программированию во всём мире. Также есть локальные сервисы у некоторых площадок для соревнований (CodeChef Campus).

2 Обзор литературы

За время существования задач по спортивному программированию было создано множество инструкций и документаций. Для того чтобы научиться работать с системой Polygon

я использовал справку `polygon.rtf` [4], которую предоставляет сама система. Для знакомства с системой также оказался полезным блог [1] от координатора платформы Codeforces.

Для написания вспомогательных программ используется библиотека `testlib.h` [5], к которой доступны примеры [8], блоги с подробным описанием [6], а также множество обсуждения на форуме Codeforces [7].

Для составления условий в соответствии со стандартами олимпиады «Высшая проба» я ориентировался на условия олимпиад прошлых лет [2]. Также, для качественной подготовки задач я следовал указаниям для разработки раундов на Codeforces: файлу «Polygon Advices» [3]

3 Формат задачи по информатике

3.1 Классические задачи

При решении стандартной задачи по программированию целью участника является написание программы, которая корректно решает задачу, поставленную жюри. Жюри предоставляет участнику текстовое условие задачи. Обычно оно состоит из следующих разделов:

- «Математическое» условие задачи (формальная постановка или легенда). В этой части неформально описывается, какие данные есть у участника, и что от него ожидается в качестве решения;
- Формат входных данных. В этом блоке подробно объясняется, в какой строке ввода какая информация содержится. Также в этом разделе описываются ограничения, накладываемые жюри на ввод. Например, максимальная длина массива или свойства графа, который даётся участнику;
- Формат выходных данных. От программы участника требуется выводить ответ в стандартизированном формате, который описывается в этом блоке;
- Примеры. Для большего понимания условия задачи, а также форматов ввода и вывода, жюри предоставляет участнику примеры ввода, а также один из корректных ответов на данном вводе. Обычно примеры даются в виде таблицы с колонками «Входные данные» и «Выходные данные»;
- Ограничения времени работы программы участника и используемой памяти;

- Примечания. В этом блоке могут содержаться пояснения к примерам, приведённым выше: как был получен ответ на данном вводе. Также здесь объясняются термины, используемые в условии, или оставляются любые другие пояснения по условию;
- Система оценки. Если решение участника может получить частичный балл, если оно работает правильно только на некоторых тестах (система «IOI»), то в данном разделе описывается, как участник может получить неполный балл за задачу. Например, участник может получить частичные баллы, если его решение корректно работает при меньших ограничениях, чем изначально в задаче.

После прочтения условия участник может написать программу, которая решает поставленную задачу. Допустимые языки программирования варьируются в зависимости от олимпиады (например, на Московской студенческой олимпиаде разрешено писать решения только на языках «`Python`», «`C++`» и «`Java`»). Самым распространённым языком в сообществе спортивного программирования является `C++`, благодаря его скорости и возможностям. Из-за разницы в скорости работы языков жюри часто не гарантирует возможность решения задачи на более медленных языках.

Программе участника доступен только ввод (в формате, описанном жюри), а также поток для вывода ответа. У программы отсутствует возможность делать запросы в Интернет, создавать дополнительные файлы или процессы. Таким образом, в решении задачи может участвовать только код, написанный участником.

Большинство задач по спортивному программированию являются «оптимизационными», то есть участник должен написать оптимальное решение задачи, а не любое. Оптимальность решения определяется по времени работы программы при заданных жюри ограничениях на входные данные. Именно поэтому ограничения на время работы программы и на ввод являются одной из самых важных частей условия задачи.

После решения задачи участник отправляет своё решение как файл в тестирующую систему. Там решение (в зависимости от языка) компилируется и запускается на подготовленных жюри тестах. Правильность ответа участника определяется специальной программой жюри — чекером. После решение участника либо зачитывается или не засчитывается как верное (если задача), либо выставляется балл (обычно от 0 до 100). При этом, участнику недоступны тексты тестов жюри, а также вердикты работы его решения. Так, тесты жюри не дают участнику дополнительную информацию о том, в чём заключается ошибка в его решении.

3.2 Интерактивные задачи

В этом типе задач программа участника не считывает данные жюри, а «общается» с программой жюри — «интерактором». Обычно в таких задачах жюри загадывает объект, а участник может делать запросы определённого вида, чтобы получать информацию об этом объекте. Например, жюри загадывает массив известной длины, и разрешает участнику ограниченное количество раз сделать запрос, чтобы получить информацию об этом массиве. После, участник должен вывести массив, который по его мнению загадало жюри.

В таких задачах участникам нужно оптимизировать не время выполнения программы (хотя оно тоже ограничено), а количество запросов, которое делает программа.

3.3 Задачи с открытыми тестами

Участнику помимо условия даются также тесты жюри. Таком образом, время работы программы участника ограничено только продолжительностью тура. Участник должен написать решение, которое не является слишком сложным в реализации, но, тем не менее, выполняется за отведённое время.

4 Разработка задачи в системе Polygon

Для подготовки задач составители обычно используют платформу Polygon. Она позволяет оптимизировать все этапы создания задачи, а также совместно работать над задачами и контестами.

Разработка задачи состоит из нескольких этапов.

4.1 Условие задачи

Содержимое и разделы условия были описаны ранее в разделе 3.1. Обычно условие составляется с помощью `LaTeX` (для удобства вёрстки и написания формул), а после из него генерируется файл в формате `.pdf` для последующей печати или в формате `.html`, для использования на онлайн соревнованиях.

Система Polygon позволяет писать и компилировать условия в формате `LaTeX` прямо в браузере. Также поддерживается несколько языков условия для международных соревнований, а в недавних обновлениях была добавлена интеграция с ChatGPT, который может автоматически перевести условие на другой язык или предложить исправления ошибок в тексте.

4.2 Набор тестов

#	Content	Size	Desc	Example 3	Group	Points 100
1	2 3 4 1 3 2 2 2 1 1 1	27		Y	0	
2	2 2 3 1 2 1 2 1 1	22		Y	0	
3	2 4 4 1 3 1 4 1 1 2 4	27		Y	0	
4	5 5 1 1 1	12			1	5
5	100 100 1 1 57	17			1	5
6	100000 100000 1 99179 1	26			1	5
7	100000 100000 1 100000 100000	32			1	5
8	gen n=50 q=100 mode=random				2	5
9	gen n=100 q=100 mode=heavy_in				2	5
10	gen n=200 q=200 mode=maxdist noise_in noise				2	5
11	gen n=200 q=200 mode=maxdist noise				2	5

Рисунок 4.1: Фрагмент тестов для задачи

Тесты являются самой важной частью задачи, и на их разработку уходит основная часть времени подготовки задачи. Общепринятым способом составления тестов считается создание программ — «генераторов», которые принимают на вход желаемые параметры теста и выводят подходящий тест. Для написания генераторов используется библиотека `testlib.h` [5], которая позволяет зафиксировать случайность при создании тестов. Также она содержит много полезных функций, которые может использовать генератор.

Polygon предоставляет удобный интерфейс для управления тестами: распределение тестов на группы, управление баллами за тесты, добавление тестов как примеры в условие, предпросмотр сгенерированных тестов и ответов.

Пример набора тестов в задаче приведён на рисунке 4.1

4.3 Проверка корректности решения

File: check.cpp

```cpp
#include "testlib.h"
#include <string>

using namespace std;

pattern pnum("0|-?[1-9][0-9]*");

bool isNumeric(const string &p) {
    return pnum.matches(p);
}

int main(int argc, char *argv[]) {
    setName("compare two signed huge integers");
    registerTestlibCmd(argc, argv);

    string ja = ans.readWord();
    string pa = ouf.readWord();

    if (!isNumeric(ja))
        quitf(_fail, "%s is not a valid integer", compress(ja).c_str());

    if (!ans.seekEof())
        quitf(_fail, "expected exactly one token in the answer file");

    if (!isNumeric(pa))
        quitf(_pe, "%s is not a valid integer", compress(pa).c_str());

    if (ja != pa)
        quitf(_wa, "expected '%s', found '%s'", compress(ja).c_str(), compress(pa).c_str());

    quitf(_ok, "answer is '%s'", compress(ja).c_str());
}
```

Рисунок 4.2: Пример чекера

Для проверки корректности решения разработчик задачи пишет специальную программу — чекер. Она обычно пишется на языке `C++` с использованием библиотеки `testlib.h` [5], но иногда используется и язык Python.

У чекера есть доступ к вводу, ответу жюри и ответу участника на тесте, и он должен вернуть вердикт о правильности решения участника. Иногда это сравнение ответов составителей и участника на совпадение, но также это может быть и более сложный алгоритм, если у поставленной задачи может быть несколько корректных ответов на заданном вводе.

Система Polygon позволяет написать тесты для чекера, чтобы проверить корректность его вердиктов на наборе входных данных и возможных ответов участника. Это помогает избежать некорректных вердиктов во время олимпиады.

Пример чекера, который проверяет совпадение ответа участника с ответом жюри, если на вывод подаётся большое число, приведён на рисунке 4.2

4.4 Проверка корректности тестов

Для дополнительной проверки корректности тестов разработчики часто пишут специальные программы — валидаторы. Аналогично чекеру, это программа на `C++` с использованием `testlib.h`, используемая для проверки корректности теста.

Программа проверяет, что для теста выполняются ограничения из раздела «Формат входных данных», а также дополнительные ограничения, если в задаче есть группы с дополнительными ограничениями.

На рисунке 4.3 приведён пример валидатора. В задаче есть 3 группы, пронумерованные от 0 до 2 (за 0 группу обычно обозначают примеры из условия). Чекер проверяет, что на ввод в первой строке подаётся 3 числа, ограничения на которые накладывает номер группы. Далее должно следовать q строк, содержащие по 2 числа в каждой: первое число должно лежать в диапазоне $[1; n]$, а второе — в $[1; m]$.

File: valid.cpp

```cpp
#include "testlib.h"

using namespace std;

const int MAXGROUP = 3;
int N_LIMITS[] = {100'000, 100'000, 200, 100'000};
int Q_LIMITS[] = {100'000, 1, 200, 100'000};


int main(int argc, char *argv[]) {
    registerValidation(argc, argv);

    int group = (!validator.group().empty() ? stoi(validator.group()) : 0);
    ensuref(0 <= group && group <= MAXGROUP, "Incorrect group number");

    int n = inf.readInt(1, N_LIMITS[group], "n");
    inf.readSpace();
    int m = inf.readInt(1, N_LIMITS[group], "m");
    inf.readSpace();
    int q = inf.readInt(1, Q_LIMITS[group], "q");
    inf.readEoln();
    for (int i = 1; i <= q; i++) {
        int x = inf.readInt(1, n, format("x_%d", i));
        inf.readSpace();
        int y = inf.readInt(1, m, format("x_%d", i));
        inf.readEoln();
    }
    inf.readEof();
}
```

Рисунок 4.3: Пример валидатора

4.5 Решения

У задачи должно быть одно ровно авторское решение, которое считается всегда верным. Результат работы этой программы на тесте будет авторским ответом, с которым работает чекер.

Также в процессе разработки и тестирования задачи могут писаться альтернативные решения: другие правильные решения, намеренно неправильные, или решения с ошибками, которые написали соавторы. Они могут использоваться для проверки качества тестов и создания разбалловки в задаче.

Система Polygon позволяет упорядоченно хранить решения, указывать ожидаемые вердикты, а также запускать решения на тестах (такой запуск называется «invocation»). Это помогает легко проверять качество тестов после изменений, а также быстро узнавать вердикт решения, не экспортируя задачу в тестирующую систему.

5 Подготовленные задачи

5.1 «Произведение без квадратов»

Процесс создания задачи

Изначально я решил сделать задачу, в решении которой используется достаточно сложный алгоритм. Для этой цели я выбрал алгоритм «3D-Mo». Данный алгоритм позволяет обрабатывать на массиве запросы 2 типов: узнать значение функции от подотрезка массива и изменить значение элемента массива. Такое решение работает за $O(n^{5/3})$.

Для того чтобы решение с 3D-Mo было оптимальным решением придуманной задачи, нужно было выбрать достаточно сложную функцию, которую нельзя поддерживать в дереве отрезков (иначе решение с деревом отрезков работало бы за $O(n \log n)$). Я решил выбрать функцию, связанную с разложением числа на делители. По разложению двух чисел на делители проанализировать разложение их произведения можно, только явно пройдясь по всем делителям. Значит, операцию $merge$ в дереве отрезков невозможно сделать оптимально. А для алгоритма «3D-Mo» необходима только возможность добавлять или удалять из ответа одно число за раз, поэтому разложение текущего подотрезка можно быстро обновлять (при выбранных мною ограничениях в результате такой операции изменится не более 15 делителей).

Итого, в качестве функции на отрезке я выбрал такую функцию:

«Любое натуральное число x может быть представлено единственным образом в виде $x = a^2 \cdot b$, где b не делится ни на один квадрат натурального числа, большего 1. Тогда, введём функцию $f(x) = b$, где b — значение из такого разложения.»

Далее, в процессе предложения данной задачи на олимпиаду, координатор предложил изменить задачу: убрать запросы изменения, но выполнять запросы на дереве (связном графе без циклов). Таким образом, в задаче участнику нужно обрабатывать запросы 2 типов: «посчитать f от произведения чисел в вершинах на единственном простом пути между вершинами x и y в дереве» и «изменить значение в вершине». В итоге основная идея задачи сохраняется, но вместо алгоритма «3D-Mo» используется алгоритм «Mo на дереве». Асимптотика авторского решения такой задачи будет $O(n^{3/2})$.

Условие задачи

Условие задачи в том виде, в котором оно было предоставлено участникам (за исключением примеров и примечаний), приведено на рисунке 5.1.

Произведение без квадратов

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	3 секунды
Ограничение по памяти:	256 мегабайт

Любое натуральное число x может быть представлено единственным образом в виде $x = a^2 \cdot b$, где b не делится ни на один квадрат натурального числа, большего 1. Тогда, введём функцию $f(x) = b$, где b — значение из такого разложения.

Дано дерево (связный неориентированный граф без циклов), на рёбрах которого записаны числа. Вам нужно ответить на q запросов: узнать f от произведения чисел на рёбрах простого пути от вершины v_i до вершины u_i. Так как ответ может быть очень большим, выведите его по модулю $10^9 + 7$.

Если начало и конец пути совпадают, то f от произведения пустого множества чисел будем считать равным 1.

Формат входных данных

В первой строке находится два целых числа n и q ($2 \leqslant n \leqslant 5 \cdot 10^4$, $1 \leqslant q \leqslant 5 \cdot 10^4$) — количество вершин в дереве и количество запросов.

В следующих $n - 1$ строках находится по 3 числа a_i, b_i, c_i ($1 \leqslant a_i, b_i \leqslant n$, $1 \leqslant c_i \leqslant 10^6$) — концы ребра и число, записанное на данном ребре. Гарантируется, что данные рёбра образуют дерево.

В следующих q строках содержится по 2 числа v_i, u_i ($1 \leqslant v_i, u_i \leqslant n$) — концы пути в данном запросе.

Формат выходных данных

Для каждого запроса выведите 1 число — значение f от произведения чисел на рёбрах на пути, по модулю $10^9 + 7$.

Система оценки

Решения, корректно работающие для $n \leqslant 1000, q \leqslant 1000, a_i \leqslant 1000$ получат не менее 15 баллов.

Решения, корректно работающие для $n \leqslant 1000, a_i \leqslant 1000$ получат не менее 30 баллов (включая предыдущие 15 баллов).

Решения, корректно работающие для $n \leqslant 1000$ получат не менее 50 баллов (включая предыдущие 30 баллов).

Решения, корректно работающие для случаев, когда дерево является бамбуком (то есть степень каждой вершины не превосходит 2), получат не менее 20 баллов.

Рисунок 5.1: Условие задачи «Произведение без квадратов»

Разработка тестов

Для задач, где эталонное решение и неоптимальное отличаются по времени работы достаточно слабо, особенно важно сделать качественные тесты.

Для отсечения решений с деревом отрезков я использую в тестах много чисел с большими простыми делителями. Так множество делителей произведения растёт, и объединение в дереве отрезков работает долго (так как оно должно перебрать все делители). Также, для того, чтобы длины путей в запросах были достаточно большими, я генерирую деревья с большими диаметрами (самым длинным расстоянием между двумя вершинами в дереве).

В качестве системы оценки я решил сделать в задаче 20 тестов (без учёта примера), за прохождение каждого участник получает по 5 баллов. Также я сделал несколько подгрупп для оценки менее оптимальных решений.

Для генерации тестов я написал генератор `gen`, который принимает на вход параметры теста:

- `n` — количество вершин в дереве;
- `q` — количество запросов;

Script:

```
<#--n, q, a_i <= 1000 -->
gen n=1000 q=1000 c=1000 tree_mode=bamboo num_mode=primes > $
gen n=100 q=100 c=100 tree_mode=random num_mode=random > $
gen n=1000 q=1000 c=1000 tree_mode=diam_add_to_diam diam_len=600 num_mode=biggest > $
<#--n, a_i <= 1000 -->
gen n=1000 q=3e4 c=1000 tree_mode=bamboo num_mode=max_divs num_cnt=1000 > $
gen n=1000 q=2e4 c=1000 tree_mode=diam_add_to_any diam_len=300 num_mode=top_primes num_cnt
gen n=1000 q=5e4 c=1000 tree_mode=random num_mode=biggest > $
<#--n <= 1000 -->
gen n=1000 q=3e4 c=1e6 tree_mode=bamboo num_mode=max_divs num_cnt=1e4 > $
gen n=1000 q=2e4 c=1e6 tree_mode=diam_add_to_any diam_len=500 num_mode=top_primes num_cnt=
gen n=1000 q=4e4 c=1e6 tree_mode=diam_add_to_diam diam_len=900 num_mode=biggest > $
gen n=1000 q=5e4 c=1e6 tree_mode=cross cnt=9 num_mode=primes > $
<#--no constraints -->
gen n=2e4 q=1e4 c=1e6 tree_mode=bamboo num_mode=max_divs num_cnt=1e3 > $
gen n=2e4 q=1e4 c=1e6 tree_mode=diam_add_to_any diam_len=1e4 num_mode=top_primes num_cnt=1
gen n=19999 q=2e4 c=1e6 tree_mode=cross cnt=6 num_mode=primes > $
gen n=3e4 q=2e4 c=1e6 tree_mode=diam_add_to_diam diam_len=25e3 num_mode=biggest > $
gen n=4e4, q=4e4, c=1e6, tree_mode=random num_mode=top_primes num_cnt=100 > $
gen n=4e4, q=4e4, c=1e6, tree_mode=random num_mode=max_divs num_cnt=1000 > $
gen n=4e4, q=4e4, c=1e6, tree_mode=diam_add_to_any diam_len=2e4 num_mode=random > $
gen n=39747, q=4e4, c=1e6, tree_mode=cross cnt=7 num_mode=max_divs num_cnt=100 > $
gen n=5e4, q=5e4, c=1e6, tree_mode=diam_add_to_diam diam_len=3e4 num_mode=same num=32768 >
gen n=49999, q=5e4, c=1e6, tree_mode=cross cnt=3 num_mode=max_divs num_cnt=1 > $
```

Рисунок 5.2: Тесты для задачи «Произведение без квадратов» (без примера)

- `c` — ограничение на числа в дереве (используется для генерации тестов для подгрупп);
- `tree_mode` — режим генерации дерева. Либо полностью случайное (значение *random*), либо генерация длинного бамбука, а после добавление рёбер к нему по алгоритму;
- `num_mode` — режим генерации чисел в вершинах: `primes` (только простые), `top_primes` (самые большие простые числа в диапазоне), `biggest` (самые большие числа), `max_divs` (числа с максимальным количеством делителей);
- `num_cnt` — дополнительный параметр для генератора чисел на дереве. Используется для режимов `top_primes` и `biggest` и обозначает, из скольки самых больших чисел генератор выбирает числа для вершин.

Итоговый скрипт генерации тестов для задачи показан на рисунке 5.2

Валидатор и чекер

Так как ответ в задаче единственен, я использовал встроенный чекер `ncmp.cpp` [9], который сравнивает последовательность чисел на совпадение.

Валидатор для задачи должен был проверять корректность расположения чисел во входном файле, а также то, что введённые рёбра образуют дерево. Последнюю проверку я делал с помощью стандартного алгоритма `dfs`: запускал обход в глубину из вершины 0, а после проверял, что все вершины посещены.

Также, во 2 группе гарантируется, что граф является бамбуком. Для верификации этого в валидаторе я проверяю, что максимальная степень вершины не превышает 2.

Статистика решений

Задача была использована на втором отборочном туре олимпиады «Высшая проба» по информатике и являлась последней из трёх задач.

Всего в отборе участвовало 1223 ученика из 11 класса. Среди них 251 человек отправили решения по задаче, из которых 68 человек смогли набрать больше 0 баллов: 8 участников получили полный балл за задачу, 14 — от 80 до 95, 22 — от 30 до 75, а от 5 до 25 баллов набрали 24 участника.

Сложность задачи оказалась сбалансированной. Среди участников с ненулевым баллом за задачу больше половины участников набрали хотя бы 50 баллов, а значит участники воспользовались возможностью получить большое количество баллов за достаточно простое решение при $n \leq 5000$.

5.2 «Неравенство набора»

Процесс создания задачи

Изначально я долго думал над такой конструкцией: «Дано число, разложить его на множители так, чтобы попарная разность модулей чисел была минимальной». Я не мог придумать для этой задачи оптимальное решение, хотя мне казалось, что оно есть.

В итоге я решил, что данная задача подходит на олимпиаду с открытыми тестами, так как в ней нужно оптимизировать перебор. В итоге она вошла в «Московскую олимпиаду школьников» по информатике для 10-11 классов.

Написание решения

Сначала я написал перебор всех разбиений на делители с помощью рекурсивной функции `gen`, которая принимает несколько параметров:

- `vector <pair <int, int>> fact` — разложение оставшегося числа на множители
- `vector <int> in` — уже набранный префикс разложения на делители
- `int left` — оставшееся число, которое нужно разложить
- `int min_val` — минимальное число, которое можно добавить в разложение

Во избежание использования одного и того же разложения несколько раз, рекурсивная функция генерирует только возрастающие разложения (с помощью параметра `min_val`).

Сама рекурсивная функция с помощью массива `in` для каждого простого числа перебирает, в какой степени оно будет входить в следующее число в разложении. Если получившееся число не меньше `min_val`, то мы нашли разрешённое дополнение текущего разложения. Пересчитываем массив `fact` для оставшегося числа, и рекурсивно вызываемся от новых параметров.

Рекурсия останавливается, когда мы разложили число на делители (то есть `left` $= 1$). В таком случае нам нужно только посчитать сумму модулей попарных разностей и обновить глобальный ответ.

Далее я стал применять отсечения. Во-первых, мы можем поддерживать сумму попарных разностей модулей для текущего набора. При этом, при добавлении числа в набор, попарная разность модулей только увеличивается. Значит, мы можем досрочно выходить из рекурсии, если ответ для текущего набора уже больше минимального найденного ответа.

Также, для изначального разложения числа из ввода на множители, я применил линейное Решето Эратосфена, чтобы вычислить минимальные делители всех чисел до $\sqrt{maxC}$, а также сделать массив простых чисел до $\sqrt{maxC}$. С помощью этого я смог раскладывать на делители, проверяя только делимость на простые числа, а также, если в какой-то момент текущее число разложения становилось меньше $\sqrt{maxC}$, то я мог легко найти разложение с помощью насчитанного массива минимальных делителей.

Последней оптимизацией стал перебор всех разложений размера 2 перед запуском рекурсии. Это являлось достаточно неплохой аппроксимацией ответа, что позволяло рекурсивной функции заходить в меньшее число неоптимальных веток.

Разработка тестов

Так как данная задача предназначалась на олимпиаду МОШ, это была задача с открытыми тестами. Мне нужно было подготовить маленький тест стоимостью 30 баллов и большой тест стоимостью 70 баллов. Результат работы на маленьком тесте участники могли узнать за время олимпиады, а результат большого им не известен до публикации итоговых результатов.

Для генерации маленького теста я разбил его на 6 диапазонов, и в каждом диапазоне взял числа с максимальным количеством делителей (так как именно на таких числах дольше всего работает перебор). Итого, в маленьком тесте присутствовало 3000 чисел до 10^{10}, и за каждый правильный ответ участники получали по 0.01 балла. Моё изначально решение (без оптимизаций) работает на этом тесте 3 секунды, а оптимальное решение работает 1 секунду.

Стоит заметить, что в оптимальном решении заметную часть времени работы занимает Решето Эратосфена, которое выполняется 1 раз в начале решения, и не зависит от количества запросов.

Для большого теста я уже не мог перебрать все числа, чтобы выбрать оптимальные, поэтому я генерировал числа случайным образом: 70% чисел в тесте занимают числа с большим количеством маленьких делителей, а остальные числа — просто случайные из диапазона, чтобы проверить корректность на тестах с большими делителями. Моё оптимальное решение локально работает на этом тесте 7 минут 26 секунд (зависит от многих факторов, но даже на более медленных компьютерах не должно превышать 20 минут). Неоптимизированное решение за примерно 5 часов работы смогло посчитать только ответы для первых 2000 запросов из 7000. Значит, количество ответов, которое сможет посчитать участник за время тура, правда зависит от оптимальности его перебора.

Система оценки

За правильный ответ на каждый тест и в маленьком тесте, и в большом, участник получает по 0.01 балла. При этом, для удобства участников, чекер (написанный как обычно с помощью testlib [5]) позволяет предоставить ответ только на несколько первых запросов. Для этого, программа считывает ответы участника по одному, пока не дойдёт до конца файла, и сравнивает с решением жюри. В конце количество правильных ответов делится на 100, и в результате получается итоговый балл.

Статистика решений

Всего в олимпиаде участвовало 524 школьника. Среди них полный балл по задаче получили 3 участника, 69 участников получили хотя бы 60 баллов, 293 участника получили хотя бы 30 баллов, а не 0 баллов получили 473 человека.

Можно сделать вывод, что баллы по задаче смогли получить почти все участники, а значит, она не была слишком сложной. При этом, большое количество баллов получили всего несколько участников. Сложность задачи оказалась оптимальной для данной олимпиады.

6 Заключение

По результатам проекта было подготовлено 2 задачи, которые были успешно использованы на школьных олимпиадах. Также был подробно описан формат задачи по программированию, а также процесс подготовки задачи в системе Polygon.

Ссылки на пакеты подготовленных задач:

- «Произведение без квадратов»: https://disk.yandex.ru/d/-aBvw9yS14GnVA
- «Неравенство набора»: https://disk.yandex.ru/d/fnWjjYqfJxrcSg

Список литературы

[1] Loc Tran Quang (darkkcyan). *[Tutorial] Polygon.Codeforces Tutorial — A Guide to Problem Preparation*. URL: https://codeforces.com/blog/entry/101072 (дата обр. 08.02.2024).

[2] Координаторы олимпиады «Высшая Проба». *Высшая проба по информатике. Задачи прошлых лет*. URL: https://olymp.hse.ru/mmo/tasks-it (дата обр. 08.02.2024).

[3] Координаторы платформы Codeforces. *Polygon Advices*. URL: https://docs.google.com/document/d/e/2PACX-1vRhazTXxSdj7JEIC7dp-nOWcUFiY8bXi9lLju-k6vVMKf4IiBmweJoOAMI-ZEZxatXF08I9wMOQpMqC/pub (дата обр. 08.02.2024).

[4] Михаил Мирзаянов. *Polygon End-User Documentation*. URL: https://polygon.codeforces.com/static/polygon.rtf (дата обр. 08.02.2024).

[5] Михаил Мирзаянов. *Библиотека testlib.h*. URL: https://github.com/MikeMirzayanov/testlib (дата обр. 08.02.2024).

[6] Михаил Мирзаянов. *Коротко о testlib.h*. URL: https://codeforces.com/testlib (дата обр. 08.02.2024).

[7] Михаил Мирзаянов. *Платформа Codeforces*. URL: https://codeforces.com (дата обр. 08.02.2024).

[8] Михаил Мирзаянов. *Примеры к testlib.h*. URL: https://github.com/MikeMirzayanov/testlib/tree/master (дата обр. 08.02.2024).

[9] Михаил Мирзаянов. *Чекер ncmp.cpp*. URL: https://github.com/MikeMirzayanov/testlib/blob/master/checkers/ncmp.cpp (дата обр. 08.02.2024).

Приложение

Используемые в работе термины и понятия:

- Тур, раунд — набор задач, которые участники решают одновременно в течение ограниченного времени. К ним относятся и этапы школьных олимпиад;
- Координатор — руководитель процесса составления задач, помогает авторам задач с разработкой и собирает несколько задач в тур;
- Система «IOI» — формат оценки решений участников, используемый на Международной олимпиаде по информатике. Решение участника может получить целое число баллов от 0 до 100 в зависимости от того, при каких условиях оно работает корректно;
- Чекер — программа для проверки корректности решения участника на определённом тесте;
- Валидатор — программа для проверки соответствия тестов жюри заявленным ограничениям.