Оглавление

Аннотация

Работа посвящена разработке задач для школьных олимпиад по спортивному программированию. В ходе работы были изучены теоретические материалы, после чего были составлены несколько задач для перечневых олимпиад по информатике.

Репозиторий с подготовленными задачами доступен по ссылке.

Введение

Школьные олимпиады по информатике зародились в конце XX века. На таких олимпиадах участникам предлагают решить несколько задач за отведенное время (обычно от 3-х до 5-ти часов). Успешное участие в олимпиадах дает школьникам льготы при поступлении вплоть до поступления в ВУЗ без вступительных испытаний. Специфика олимпиад по информатике состоит в том, что решения участников проверяются автоматически с помощью тестов, которые заранее подготовили жюри олимпиады.

Задачи проекта

1. Изучить доступные методы подготовки олимпиадных задач
2. Изложить используемую методику и формат задач
3. Подготовить несколько задач для олимпиад
4. Описать процесс подготовки задач
5. Предоставить результаты соответствующих олимпиад, а также результаты решения задач участниками

Формат задачи по информатике

Олимпиады по информатике проводятся на специальных интернет-платформах, доступ к которым выдается участникам на время тура. За отведенное время участники могут отправить решения задач и сразу же в автоматическом режиме получать ответ от системы: количество баллов за данную посылку. Чем лучше решение участника, тем больше баллов он получит.

Задача разработчика олимпиады заключается в том, чтобы подготовить задачи в формате, требуемом той или иной тестирующей системой.

Участник сдает в систему код на одном из языков программирования. После чего код участника исполняется на тестах, которые заранее подготовило жюри. Некоторые из этих тестов известны участнику, остальные – нет. Если программа выдает правильный ответ на всех тестах жюри, она получает полный балл. Иначе, участник может получить частичный балл.

Как правило, в каждой задаче должны быть следующие составляющие:

1. Условие
2. Примеры входных и выходных данных
3. Набор тестов, скрытых от участника

Условие

Условие должно быть понятным. После прочтения участник должен сформировать формальную математическую модель задачи. Если этого сделать не получается, либо наоборот можно сделать несколькими способами, условие задачи считается плохим и требует уточнений. Также в условии должен быть описан протокол действий программы участника. Программа должна считывать входные данные из теста, после чего выводить выходные данные. Если программа на выходе вывела правильные данные, то она прошла тесты. Иначе, не прошла.

Примеры входных и выходных данных

Чтобы участнику было проще понять, какой формат входных и выходных данных, жюри предоставляет несколько соответствующих примеров.

Набор тестов, скрытых от участника

После отправки решение участника запускается на тестах, состав которых от участника скрыт. Чем больше тестов пройдет решение, тем больше баллов оно получит.

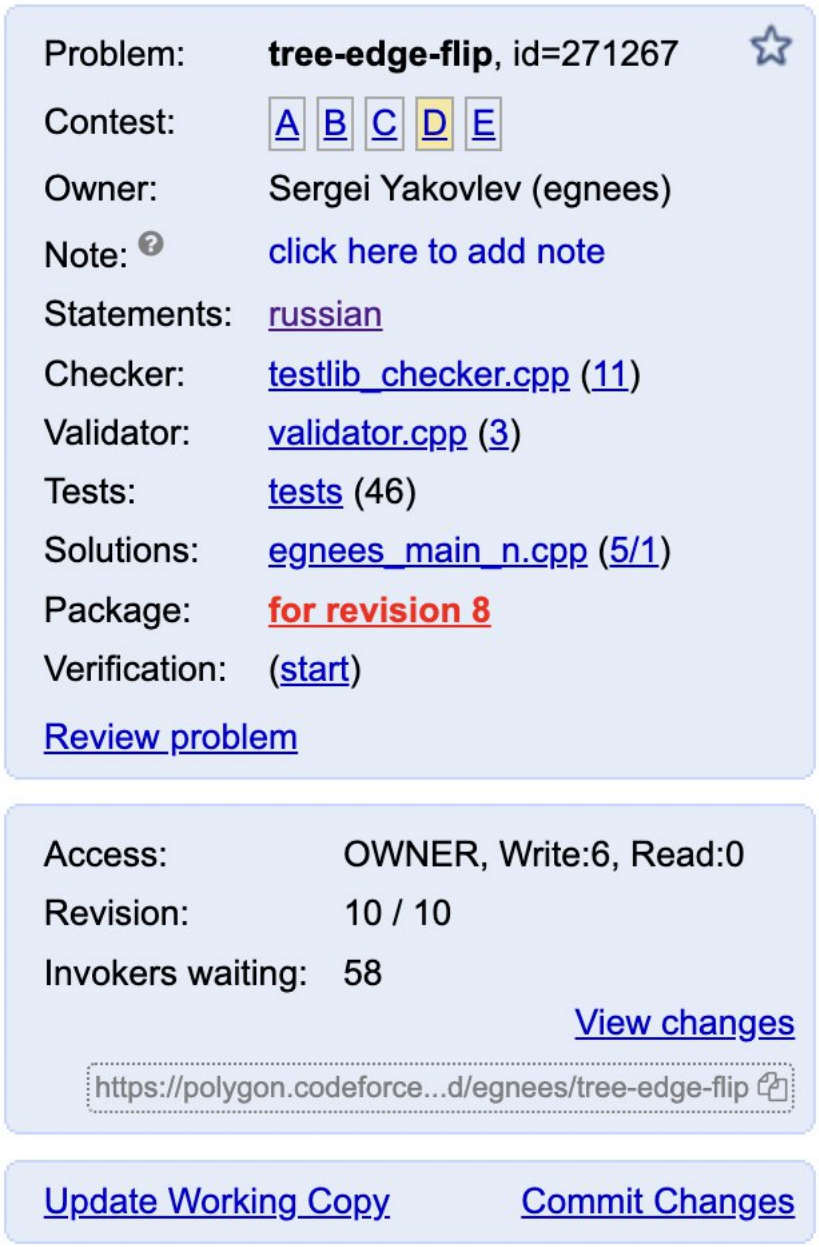
Задача жюри сделать качественные тесты. Например, если тестов слишком мало, то неправильное решение может пройти все тесты. А если тестов слишком много, то решения будут тестироваться слишком долго, что будет дестабилизировать работу тестирующей системы. Поэтому от составителей задач требуется найти золотую середину.

Подготовка задач

Раньше олимпиады готовились вручную, без вспомогательных средств. Так было по появления системы polygon.codeforces.com, которая на данный момент считается стандартным способом подготовки задач.

После регистрации сразу же появляется возможность создать задачу: достаточно ввести имя для задачи на английском языке.

После чего автору становится доступен богатый интерфейс для редактирования задачи.



1. Statement – условие задачи
2. Checker – программа для проверки вывода программы участника на соответствие формату выходных данных. В случае несоответствия, checker должен осведомить участника
3. Validator – программа, благодаря которой жюри могут проверить себя. В частности, соответствие тестов заявленному формату входных данных
4. Tests – вкладка, на которой находятся сгенерированные тесты
5. Solutions – вкладка с авторскими решениями

Statement

Математическая формулировка задачи звучит следующим образом:

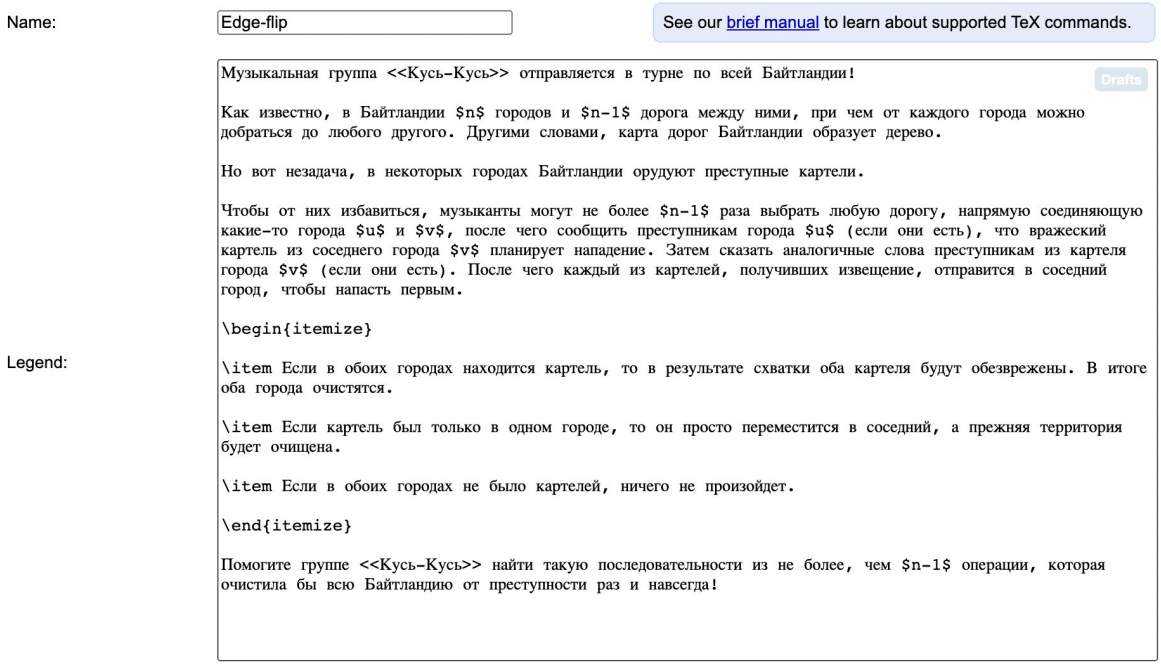
Дано дерево из n вершин. Каждая вершина покрашена в один из двух цветов (красный и синий). В синий покрашено k вершин. За одну операцию можно выбрать произвольное ребро в дереве и поменять цвета вершин, соединенных ребром, на противоположные. Требуется проверить, можно покрасить все дерево в красный цвет, сделав не более n-1 операции. Если можно, вывести необходимые операции.

Данную задачу можно решать несколькими способами. От выбранного способа будет зависеть время исполнения решения. Чем быстрее работает решение участника, тем более оптимальный способ он выбрал, тем больше баллов он должен получить.

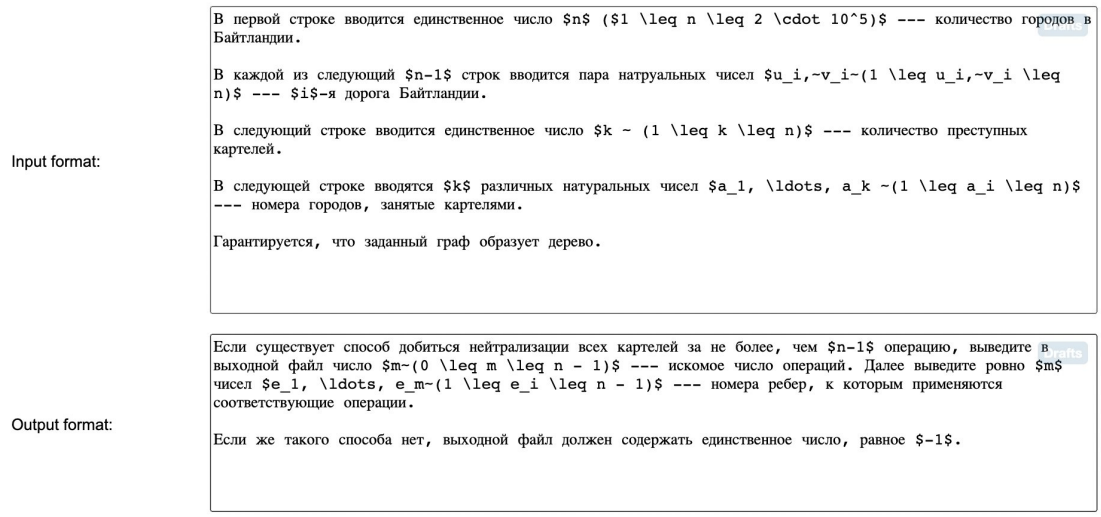
Возможные варианты решения подробно описаны в разделе Solutions. Ниже приведена таблица, в которой для каждого решения записана его асимптотика и максимальное n, для которого оно отработает за не более, чем 2 секунды. Также приведено количество баллов, которое получит участник, если его решение будет правильно работать на данной группе тестов.

Решение	Асимптотика	Максимальное n	Баллы
Перебор всех решений с учетом порядка	O(n! * n^2)	8	15
Перебор всех решений без учета порядка	O(2^n * n)	20	15
Сообщить о отсутствии решения при нечетном k	O(n)	-	10
Каждый раз находить пару синих вершин и красить путь между ними	O(nk)	2000	30
Использовать DFS	O(n)	-	30

В условиях задач принято использовать систему компьютерной верстки *LaTeX*. Она позволяет оформлять математические конструкции в привычном для нас виде.



В поле Name вписывается название задачи.
В поле Legend пишется непосредственно условие задачи. Условие следует делать небольшим, но достаточно подробным.



Далее идут поля Input format и Output format. В этих полях описывается соответственно формат входных и выходных данных.

Scoring:

Баллы за каждую подзадачу начисляются только в случае, если все тесты для этой подзадачи и необходимых подзадач успешно пройдены.

```
\begin{center}
\renewcommand{\arraystretch}{1.3}
\begin{tabular}{|c|c|c|c|c|}
\hline
\textbf{Подзадача} &
\textbf{Баллы} &
\textbf{Ограничения} &
\parbox{3cm}{\textbf{\centering\\Необходимые\\подзадачи\\\vspace{2mm}}} &
\parbox{3cm}{\textbf{\centering\\Информация\\о проверке\\\vspace{2mm}}}
\\ \hline
0 & 0 & Тесты из условия & --- & полная \\
\hline
1 & 15 & $n \le 8$ & --- & первая ошибка \\
\hline
2 & 15 & $n \leq 20$ & $1$ & первая ошибка \\
\hline
3 & 10 & $k$ нечетное & --- & первая ошибка \\
\hline
4 & 30 & $n \leq 2000$ & 1, 2 & первая ошибка \\
\hline
5 & 30 & Нет дополнительных ограничений & 1~---4 & первая ошибка \\
\hline
\end{tabular}
\end{center}
```

Далее идет секция Scoring, в которой описывается разбалловка. Например, если участник сумел реализовать правильное, но медленное решение, ему предусмотрена часть баллов.

Tutorial:

Давайте немного переформулируем условие. Скажем, что если на обоих концах дороги нет картелей, то применение операции к данной дороге <<создает>> картели на обоих ее концах. Несложно заметить, что такая формулировка эквивалентна исходной. Тогда задачу можно переформулировать так: дано дерево, в котором каждая вершина покрашена либо в белый, либо в черный цвет. За одну операцию можно выбрать ребро и поменять цвет у вершин на его концах.

В такой формулировке становится очевидно, что порядок операций неважен, а также то, что применять операцию к одному и тому же ребру бесполезно. Отсюда следует, что если решение существует, то состоит из не более чем $n-1$ операции. Эта идея приводит к решению с асимптотикой $O(2^n \cdot n)$, которое заключается в том, чтобы перебрать все возможные варианты (которых $O(2^n)$, так как каждое ребро либо было использовано один раз, либо не было использовано вовсе). Это решение проходит первые две подгруппы и набирает 30 баллов.

Заметим, что каждая операция не меняет четность количества черных вершин. Поскольку необходимо получить 0 черных вершин (а ноль "--- это четное число), то и изначальное количество вершин должно быть четным. Иначе, ответ -1. Эта идея позволяет решить подгруппу, где k нечетное.

Оказывается, что если количество черных вершин четно, то ответ всегда существует. Для его построения посмотрим на произвольный лист дерева. Если он белый, то просто удалим его из дерева. Иначе применим операцию к единственному ребру, инцидентному данному листу. После чего лист станет белым. Удалим его. Таким образом, задача свелась к меньшей. В конце концов от дерева останется только корень, который обязательно будет покрашен в белый цвет. Асимптотика решения $O(n)$.

Последняя секция – Tutorial. В ней описывается решение задачи. Участник не видит решения, однако после конца олимпиады жюри может выложить все решения в открытый доступ.

После сборки условие задачи выглядит следующим образом:

Edge-flip

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Музыкальная группа «Кусь-Кусь» отправляется в турне по всей Байтландии!

Как известно, в Байтландии n городов и $n-1$ дорога между ними, при чем от каждого города можно добраться до любого другого. Другими словами, карта дорог Байтландии образует дерево.

Но вот незадача, в некоторых городах Байтландии орудуют преступные картели.

Чтобы от них избавиться, музыканты могут не более $n-1$ раза выбрать любую дорогу, напрямую соединяющую какие-то города u и v, после чего сообщить преступникам города u (если они есть), что вражеский картель из соседнего города v планирует нападение. Затем сказать аналогичные слова преступникам из картеля города v (если они есть). После чего каждый из картелей, получивших извещение, отправится в соседний город, чтобы напасть первым.

- Если в обоих городах находится картель, то в результате схватки оба картеля будут обезврежены. В итоге оба города очистятся.
- Если картель был только в одном городе, то он просто переместится в соседний, а прежняя территория будет очищена.
- Если в обоих городах не было картелей, ничего не произойдет.

Помогите группе «Кусь-Кусь» найти такую последовательности из не более, чем $n-1$ операции, которая очистила бы всю Байтландию от преступности раз и навсегда!

Формат входных данных

В первой строке вводится единственное число n ($1 \leqslant n \leqslant 2 \cdot 10^5$) — количество городов в Байтландии.

В каждой из следующий $n-1$ строк вводится пара натруальных чисел u_i, v_i ($1 \leqslant u_i$, $v_i \leqslant n$) — i-я дорога Байтландии.

В следующий строке вводится единственное число k ($1 \leqslant k \leqslant n$) — количество преступных картелей.

В следующей строке вводятся k различных натуральных чисел $a_1, \ldots, a_k$ ($1 \leqslant a_i \leqslant n$) — номера городов, занятые картелями.

Гарантируется, что заданный граф образует дерево.

Формат выходных данных

Если существует способ добиться нейтрализации всех картелей за не более, чем $n-1$ операцию, выведите в выходной файл число m ($0 \leqslant m \leqslant n-1$) — искомое число операций. Далее выведите ровно m чисел $e_1, \ldots, e_m$ ($1 \leqslant e_i \leqslant n-1$) — номера ребер, к которым применяются соответствующие операции.

Если же такого способа нет, выходной файл должен содержать единственное число, равное -1.

Система оценки

Баллы за каждую подзадачу начисляются только в случае, если все тесты для этой подзадачи и необходимых подзадач успешно пройдены.

Подзадача	Баллы	Ограничения	Необходимые подзадачи	Информация о проверке
0	0	Тесты из условия	—	полная
1	15	$n \leqslant 8$	—	первая ошибка
2	15	$n \leqslant 20$	1	первая ошибка
3	10	k нечетное	—	первая ошибка
4	30	$n \leqslant 2000$	1, 2	первая ошибка
5	30	Нет дополнительных ограничений	1 —4	первая ошибка

Примеры

стандартный ввод	стандартный вывод
3 1 2 2 3 2 1 2	1 1
3 1 2 1 3 1 1	-1
5 1 2 1 3 2 4 2 5 4 1 2 4 5	3 3 4 1

А разбор выглядит так:

Edge-flip

Давайте немного переформулируем условие. Скажем, что если на обоих концах дороги нет картелей, то применение операции к данной дороге «создает» картели на обоих ее концах. Несложно заметить, что такая формулировка эквивалентна исходной. Тогда задачу можно переформулировать так: дано дерево, в котором каждая вершина покрашена либо в белый, либо в черный цвет. За одну операцию можно выбрать ребро и поменять цвет у вершин на его концах.

В такой формулировке становится очевидно, что порядок операций неважен, а также то, что применять операцию к одному и тому же ребру бесполезно. Отсюда следует, что если решение существует, то состоит из не более чем $n-1$ операции. Эта идея приводит к решению с асимптотикой $O(2^n \cdot n)$, которое заключается в том, чтобы перебрать все возможные варианты (которых $O(2^n)$, так как каждое ребро либо было использовано один раз, либо не было использовано вовсе). Это решение проходит первые две подгруппы и набирает 30 баллов.

Заметим, что каждая операция не меняет четность количества черных вершин. Поскольку необходимо получить 0 черных вершин (а ноль — это четное число), то и изначальное количество вершин должно быть четным. Иначе, ответ -1. Эта идея позволяет решить подгруппу, где k нечетное.

Оказывается, что если количество черных вершин четно, то ответ всегда существует. Для его построения посмотрим на произвольный лист дерева. Если он белый, то просто удалим его из дерева. Иначе применим операцию к единственному ребру, инцидентному данному листу. После чего лист станет белым. Удалим его. Таким образом, задача свелась к меньшей. В конце концов от дерева останется только корень, который обязательно будет покрашен в белый цвет. Асимптотика решения $O(n)$.

Checker

Теперь надо написать код чекера. Чекер – это программа, считывающая ответ участника на очередной тест. В зависимости от ответа программы участника, чекер выдает один из нескольких вердиктов. В зависимости от вердикта, участник может узнать, если что-то пошло не так. Например, вердикт чекера *PE* означает, что вывод программы участника не соответствует формату выходных данных.

Чекер использует библиотеку *testlib.h*, которая предоставляет удобный способ взаимодействия разработчика и участника. Документация по библиотеки доступна по ссылке.

Код чекера выглядит следующим образом

```cpp
#include <bits/stdc++.h>
#include "testlib.h"

using namespace std;

int n, k;

vector<int> cols;
vector<pair<int, int>> edges;

const int MAXN = 200'000;

inline void readAndCheckAnswer(InStream& in) {
        int m = in.readInt(-1, n - 1, "m");
        if (k % 2 == 1) {
                if (m == -1) {
                        return;
                }
                in.quitf(_wa, "there is no answer");
        } else if (m == -1) {
                in.quitf(_wa, "there is answer");
        }
        for (int i = 0; i < m; ++i) {
                int j = in.readInt(1, n - 1, "e_i");
                cols[edges[j-1].first] ^= 1;
                cols[edges[j-1].second] ^= 1;
        }
        for (int i = 1; i <= n; ++i) {
                if (cols[i]) {
                        in.quitf(_wa, "some vertex is black");
                }
        }
}

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

    n = inf.readInt(1, MAXN, "n");
    for (int i = 0; i < n - 1; ++i) {
        int u, v;
        u = inf.readInt(1, n, "v");
        v = inf.readInt(1, n, "u");
        edges.emplace_back(u, v);
    }

    k = inf.readInt(1, n, "k");
    cols.resize(n + 1, 0);
    for (int i = 0; i < k; ++i) {
        int v = inf.readInt(1, n, "a_i");
        cols[v] = 1;
    }

    readAndCheckAnswer(ouf);

    quitf(_ok, "its ok");
}
```

В данном случае чекер проверяет, что ответ, выданный програмой участника, удовлетворяет условию задачи.

Понятно, что автору задачи необходимо удостовериться, что чекер работает нужным образом. Для этого в системе polygon.codefoces.com существуют тесты для чекера. Они позволяет программисту проверить, что чекер выдает правильные вердикты во всех возможных случаях. Тесты для чекера могут выглядеть, например, так:

Checker tests

Test count: 11

Tests

Run tests Add test

#	Input	Output	Answer	Total size	Expected verdict	Checker verdict	Checker comment	Actions Delete Copy	
1	7 2 1 3 2 4 2 5 2 ...	6 1 1 2 3 4 5	-1	69	WRONG_ANSWER	WRONG_ANSWER	wrong answer there is no answer	Delete Edit Copy	
2	2 2 1 2 2 1	0	1 1	24	WRONG_ANSWER	WRONG_ANSWER	wrong answer some vertex is black	Delete Edit Copy	
3	2 2 1 2 2 1	1 1	1 1	24	OK	OK	ok its ok	Delete Edit Copy	
4	2 2 1 2 2 1	3 1 1 1	1 1	31	WRONG_ANSWER	WRONG_ANSWER	wrong answer Integer parameter [name=m] equals to 3, violates the range [-1, 1]	Delete Edit Copy	
5	6 2 1 3 1 4 3 5 4 ...	-1	-1	40	OK	OK	ok its ok	Delete Edit Copy	
6	1 1 1	-1	-1	15	OK	OK	ok its ok	Delete Edit Copy	
7	1 1 1	0	-1	12	WRONG_ANSWER	WRONG_ANSWER	wrong answer there is no answer	Delete Edit Copy	
8	17 2 1 3 2 4 2 5 4 ...	12 9 7 10 11 12 2 1...	12 9 7 10 11 12 2 1...	207	OK	OK	ok its ok	Delete Edit Copy	
9	20 2 1 3 1 4 1 5 4 ...	8 5 6 11 15 16 17 ...	8 16 19 17 15 5 18...	211	OK	OK	ok its ok	Delete Edit Copy	
	3 1 2								

Как видно, вердикт чекера совпал с ожидаемым вердиктом во всех придуманных мною случаях.

Validator

Следующий шаг – написание валидатора. Валидатор – это программа, которая проверяет тесты жюри на соответсвие заявленному формату входных данных. Валидатор – это не обязательная часть, однако он позволяет жюри проверить самих себя.

Валидатор для данной задачи выглядит следующим образом:

```cpp
#include "testlib.h"
#include <vector>
#include <set>

using namespace std;

const int GROUP_CNT = 5;

int N_MAX[1 + GROUP_CNT] = {200'000, 8, 20, 200'000, 2000, 200'000};

int n, k;
const int MAXN = 200'005;
vector<int> g[MAXN];
int used[MAXN];
int is_tree;
void dfs(int v, int p = -1) {
        used[v] = 1;
        for (int u : g[v]) if (u != p) {
                if (used[u]) {
                        is_tree = 0;
                } else {
                        dfs(u, v);
                }
        }
}

int main(int argc, char* argv[]) {
        registerValidation(argc, argv);
        int group = atoi(validator.group().c_str());
        n = inf.readInt(1, N_MAX[group], "n");
        inf.readEoln();
        for (int i = 0; i < n - 1; ++i) {
                int u, v;
                u = inf.readInt(1, n, "u");
                inf.readSpace();
                v = inf.readInt(1, n, "v");
                inf.readEoln();
                g[v].push_back(u);
                g[u].push_back(v);
        }
        k = inf.readInt(1, n, "k");
        inf.readEoln();
        set<int> s;
        for (int i = 0; i < k; ++i) {
                s.insert(inf.readInt(1, n, "a_i"));
                if (i != k - 1) {
                        inf.readSpace();
                } else {
                        inf.readEoln();
                }
        }
        ensuref((int) s.size() == k, "not all kartels and distinct");
        is_tree = 1;
        dfs(1);
        for (int v = 1; v <= n; ++v) {
                if (!used[v]) {
                        is_tree = 0;
                }
        }
        ensuref((bool) is_tree, "graph is not a tree");
        inf.readEof();
        return 0;
```

Как видно, он проверяет, что граф, который дан в тесте, действительно является деревом. Также валидатор проверяет, что каждый символ теста действительно соответствует заявленному формату входных данных (что в файле нет лишних пробелов и т.п).

Для валидатора также можно сгенерировать тесты. Как видно, реальные вердикты совпали с ожидаемыми.

Validator tests

Test count: 3

Tests

Run tests Add test

#	Input	Size	Expected verdict	Validator verdict	Validator comment	Actions Delete Edit Copy	
1	3 1 2 2 3 2 1 2	21	VALID	VALID		Delete Edit Copy	☐
2	3 1 2 2 3 2 1 1	21	INVALID	INVALID	Validator 'validator.exe' returns exit code 3 [FAIL not all kartels and distinct]	Delete Edit Copy	☐
3	4 1 2 1 3 2 3 2 ...	26	INVALID	INVALID	Validator 'validator.exe' returns exit code 3 [FAIL graph is not a tree]	Delete Edit Copy	☐

Tests

Следующий шаг – это подготовка тестов. Для этого есть несколько способов. Первый из них – это сгенерировать файлы с тестами локально на компьютере, после чего загрузить их в систему polygon. Второй способ – воспользоваться встроенными в polygon генераторами тестов. Для этого генератор нужно написать, используя библиотеку *testlib.h*. После чего генератор можно использовать в скрипте для создания тестов.

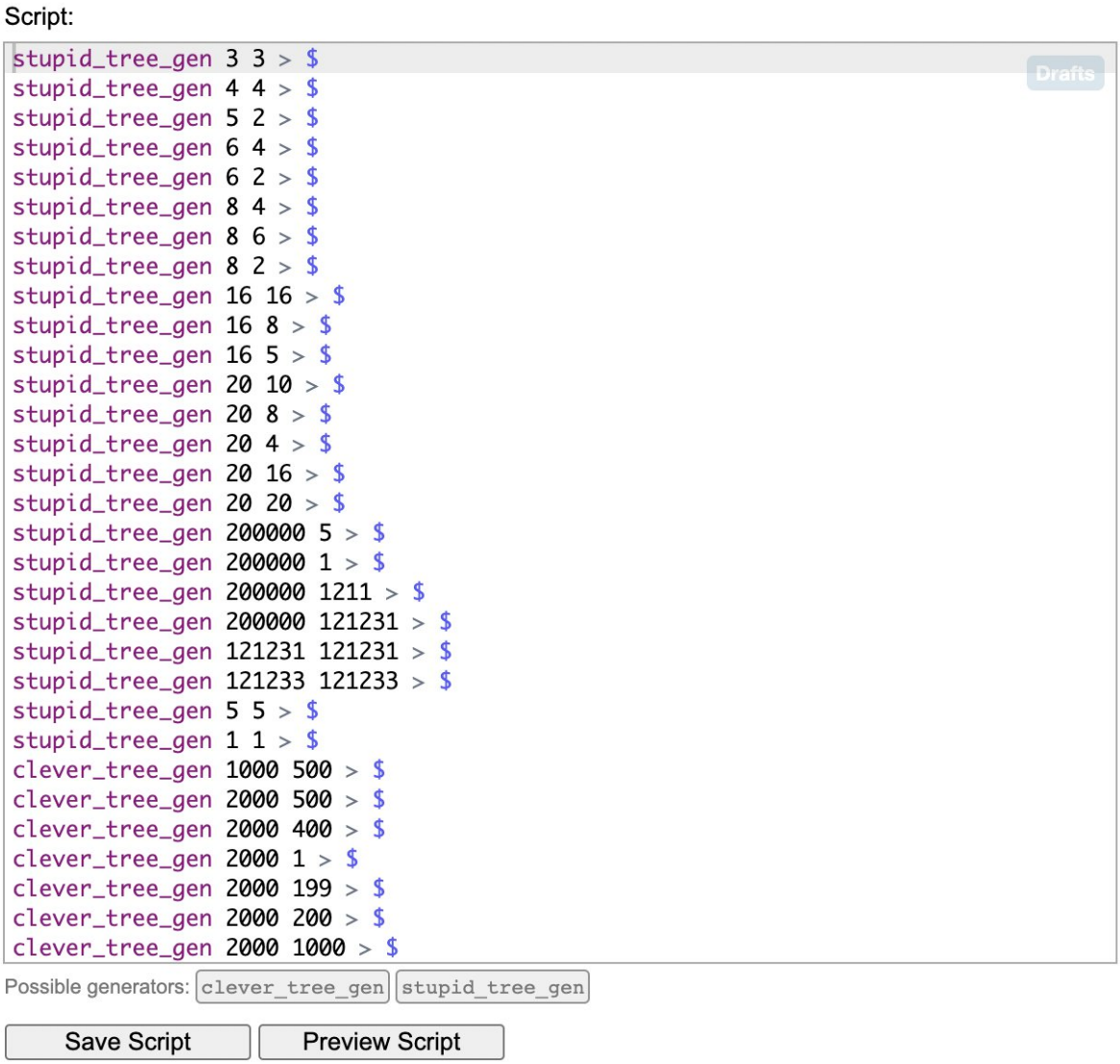
Скрипт может выглядеть так. Как видно, система polygon предлагает на выбор один из нескольких написанных генераторов. В этой задаче было полезно сделать два генератора – один для простых тестов, где не важна производительность программы – это *stupid_tree_gen*. В тестах, которые созданы этим генератором, высота деревьев имеет порядок O(N^{1/3}), что не есть хорошо. Если бы я использовал только этот генератор, то тесты были бы слабые, и неправильные решения могли бы получать 100 баллов.

Чтобы этого избежать, был предусмотреть второй генератор *clever_tree_gen.* Коды генераторов доступны в репозитории проекта.

Для каждой группы тесты были сгенерированы отдельно.

Ниже приведена в таблица, в которой для каждой группы обозначено количество тестов в данной группе и их параметры, а также группы, а которых зависит данная.

Группа	Ограничения	Параметры генераторов	Количество	Зависимости
1	n <= 8	stupid_tree_gen, n <= 8	8	-
2	n <= 20	stupid_tree_gen, n <= 20	8	1
3	k нечетное	stupid_tree_gen, n <= 20000, k нечетное	8	-
4	n <= 2’000	clever_tree_gen, n <= 2000	8	1, 2
5	n <= 200’000	clever_tree_gen, n <= 200’000	11	1-4



Следующий шаг – это проставить разбалловку и разбить тесты на группы, в соответствии с условием задачи (и секцией scoring в частности). Разбиение тестов на группы в системе polygon.codeforces.com делается с помощью следующего интерфейса:

Groups points policy and dependencies

Name	Tests	Points	Points policy	Feedback policy	Dependencies	
0	3	0	EACH_TEST Change?	COMPLETE Change?	Add	
1	8	15	COMPLETE_GROUP Change?	ICPC Change?	Add	
2	8	15	COMPLETE_GROUP Change?	ICPC Change?	1 × Add	
3	8	10	COMPLETE_GROUP Change?	ICPC Change?	Add	
4	8	30	COMPLETE_GROUP Change?	ICPC Change?	1 × 2 × Add	
5	11	30	COMPLETE_GROUP Change?	ICPC Change?	1 × 2 × 3 × 4 × Add	

Здесь же можно задать зависимости между группами, как видно в условии.

Solutions

Рассмотрим группы и их решения более подробно.

1) Для решения первой группы участнику нужно заметить, что применять операцию к одному и тому же ребру не имеет смысла. То есть к каждому ребру операция либо применена один раз, либо не применена вообще. Таким образом можно рассмотреть все возможные перестановки из n-1 элемента, и проверить каждый префикс каждой перестановки. Такое решение работает довольно долго, а написать его совсем не сложно. Поэтому такое решение получает 15 баллов.
2) Для решения второй группы участнику необходимо заметить, что порядок операций тоже ни на что не влияет. Это значит, что достаточно рассмотреть все подмножества {1, …, n-1}. Таких подмножеств всего 2^(n-1). Каждое из них можно рассмотреть за O(n). Таким образом получается асимптотика O(2^n*n), что намного быстрее предыдущего решения, однако все еще недостаточно быстро. Такое решение писать не сложнее, чем предыдущее. Поэтому в сумме с предыдущим получает 30 баллов, так как сложнее идейно
3) В данной группе k нечетное. Участник мог заметить, что каждая операция не меняет четности количества синих вершин (и красных тоже). Так как в итоге должно получиться 0 синих вершин, а 0 – число четное, то для нечетного k ответа не существует. Поняв это, участник мог легко получить +10 баллов.
4) В 4й группе n не больше 2000. Для решения этой группы участник мог заметить, что с помощью описанных операций можно одновременно поменять цвет любых двух различных вершин, применив операцию ко всем ребрам на пути между этими вершинами. Таким образом, если k четно (а иначе ответа не существует), можно было разбить все синие вершины на пары, и поочередно применить операции к путям между парными вершинами. Далее для каждого ребра нужно найти количество примененных к нему операций. Если это число нечетное, то нужно включить ребро в ответ. Простейшая реализация такого решения работает за O(n*k). Такое решение не очень просто придумать и не очень просто реализовать. За это данная группа стоит 30 баллов.
5) Для решения 5й группы есть несколько способов. Первый из них – это использовать продвинутые структуры данных для ускорения решения для 4й подгруппы. Второй вариант состоит в том, чтобы заметить следующее. Если посмотреть на какой-нибудь лист дерева, становится сразу понятно, надо ли

применять операцию к соответствующему ребру, или же нет. Действительно, если лист покрашен в синий, то операцию применить необходимо. Иначе, операцию нельзя применять. Таким образом однозначно определяется, нужно ли делать операцию с данным ребром. Если нужно, сделаем, перекрасим соответствующие вершины. В любом случае, после этого данный лист можно выкинуть и свести задачу к меньшей. Используя DFS или же очередь, данное решение работает за O(n) и получает 100 баллов.

Далее в раздел Solutions необходимо загрузить авторские решения. Желательно загрузить несколько правильных и несколько неправильных решений, чтобы удостовериться, что все они набирают ожидаемое количество баллов.

Я загрузил по одному решению для каждой подгруппы. Далее нужно проверить, что каждое решение набирает заявленное количество баллов. Не больше и не меньше.

Во вкладке *invocations* можно проверить, что все решения получают ожидаемые вердикты

« Back to invocations — Rejudge

#	egnees_2_n.cpp	egnees_main_n.cpp	egnees_n2.cpp	egnees_n_pow_n.cpp	egnees_no_only.cpp
1 0	OK 0 / 0	OK 0 / 3	OK 0 / 5	OK 0 / 0	WA 0 / 0
2 0	OK 0 / 0	OK 0 / 3	OK 0 / 5	OK 0 / 0	OK 0 / 0
3 0	OK 0 / 0	OK 30 / 3	OK 0 / 5	OK 0 / 0	WA 0 / 0
4 1	OK 0 / 0	OK 0 / 3	OK 0 / 5	OK 0 / 0	OK 0 / 0
5 1	OK 0 / 0	OK 0 / 3	OK 0 / 5	OK 0 / 0	WA 0 / 0
6 1	OK 0 / 0	OK 0 / 3	OK 0 / 5	OK 0 / 0	WA 0 / 0
7 1	OK 0 / 0	OK 0 / 3	OK 15 / 5	OK 0 / 0	WA 0 / 0
8 1	OK 0 / 0	OK 0 / 3	OK 15 / 5	OK 0 / 0	WA 0 / 0
9 1	OK 0 / 0	OK 0 / 3	OK 0 / 5	OK 0 / 0	WA 0 / 0
10 1	OK 0 / 0	OK 0 / 3	OK 15 / 5	OK 0 / 0	WA 0 / 0
11 1	OK 0 / 0 / 15.00000	OK 0 / 3 / 15.00000	OK 0 / 5 / 15.00000	OK 0 / 0 / 15.00000	WA 15 / 0 / 15.00000
12 2	OK 30 / 0	OK 0 / 3	OK 0 / 5	TL 2000 / 0	WA 0 / 0
13 2	OK 15 / 0	OK 0 / 3	OK 0 / 5	TL 2000 / 0	WA 0 / 0
14 2	OK 0 / 0	OK 0 / 3	OK 0 / 5	TL 2000 / 0	OK 15 / 0
15 2	OK 31 / 0	OK 0 / 3	OK 0 / 5	TL 2000 / 0	WA 0 / 0
16 2	OK 15 / 0	OK 0 / 3	OK 0 / 5	TL 2000 / 0	WA 0 / 0
17 2	OK 0 / 0	OK 0 / 3	OK 0 / 5	OK 93 / 0	WA 0 / 0
18 2	OK 187 / 0	OK 0 / 3	OK 15 / 5	TL 2000 / 0	WA 0 / 0
19 2	OK 171 / 0 / 15.00000	OK 15 / 3 / 15.00000	OK 0 / 5 / 15.00000	TL 2000 / 0 / 15.00000	WA 0 / 0 / 15.00000
20 3	OK 109 / 3	OK 109 / 8	OK 109 / 10	RE 0 / 0	OK 93 / 0
21 3	OK 93 / 3	OK 108 / 8	OK 93 / 10	RE 0 / 0	OK 93 / 0
22 3	OK 93 / 3	OK 139 / 8	OK 109 / 10	RE 0 / 0	OK 93 / 0
23 3	OK 108 / 3	OK 124 / 8	OK 139 / 10	RE 0 / 0	OK 93 / 0
24 3	TL 2000 / 3	OK 77 / 6	OK 92 / 8	RE 0 / 0	OK 46 / 0
25 3	TL 2000 / 3	OK 78 / 6	OK 93 / 8	RE 0 / 0	OK 31 / 0
26 3	OK 0 / 0	OK 0 / 3	OK 0 / 5	OK 0 / 0	OK 15 / 0
27 3	OK 15 / 0 / 10.00000	OK 0 / 3 / 10.00000	OK 0 / 5 / 10.00000	OK 0 / 0 / 10.00000	OK 0 / 0 / 10.00000
28 4	WA 0 / 0	OK 0 / 3	OK 15 / 5	RE 0 / 0	WA 0 / 0
29 4	WA 109 / 0	OK 0 / 3	OK 15 / 5	RE 46 / 0	WA 0 / 0
30 4	WA 109 / 0	OK 0 / 3	OK 0 / 5	RE 0 / 0	WA 0 / 0
31 4	OK 109 / 0	OK 15 / 3	OK 15 / 5	RE 0 / 0	OK 0 / 0
32 4	OK 109 / 0	OK 0 / 3	OK 0 / 5	RE 0 / 0	OK 0 / 0
33 4	WA 109 / 0	OK 0 / 3	OK 0 / 5	RE 0 / 0	WA 0 / 0
34 4	WA 109 / 0	OK 0 / 3	OK 0 / 5	RE 0 / 0	WA 0 / 0

Verification

Следующий шаг – это верификация задачи. Полигон автоматически проверяет, что никаких противоречий не возникло, и задача готова к использованию в олимпиаде.

После чего задача может быть добавлена в контест, который уже будет добавлен в тестирующую систему.

Статистика решений

Данная задача была представлена в качестве 4й (из 5ти) задач на финале Innopolis Open по ИИ и робототехнике 2022/2023. Финал писали 19 человек, из которых четверо решили задачу на полный балл. Двое из них сначала решили задачу на частичный балл, после чего улучшая свое решение. 8 человек решили задачу на 10 баллов, разобрав случай с нечетным k. Все, кто делал посылки по задаче, набрали ненулевое количество баллов по ней, что является признаком понятного условия.

Подготовленные задачи

В ходе проекта было подготовлено 5 задач. Ниже приведена таблица, где для каждой задачи есть ссылка на условие из соответствующей олимпиады, ссылка на официальные результаты олимпиады, а также ссылка на пакет с задачей, собранный в системе polygon.codefoces.

	Edge-flip	Максимальное покрытие	Matryoshka Inc	Спорт — это спорт
Условие	Финал Innopolis Open по ИИ / 2022-2023	Innopolis Open по ИИ, второй отборочный тур	Innopolis Open по информатике, второй отборочный тур	Муниципальный этап ВсОШ в Липецке
Результаты	Финал Innopolis Open по ИИ / 2022-2023	Innopolis Open по ИИ, второй отборочный тур	Innopolis Open по информатике, второй отборочный тур	Муниципальный этап ВсОШ в Липецке
Пакет	Google Drive	Google Drive	Google Drive	Google Drive

Доступ к каждой из задач в системе предоставлю по возможности.

Литература и полезные ссылки

1. Checkers with testlib.h https://codeforces.com/blog/entry/18431
2. Валидаторы на testlib.h https://codeforces.com/blog/entry/18426
3. Statements TeX manual https://polygon.codeforces.com/docs/statements-tex-manual
4. Основы LaTeX http://tug.ctan.org/info/russian/basiclatex-ru/BasicLatex.pdf
5. Polygon.codefoces public API https://codeforces.com/blog/entry/45923
6. Polygon.codefoces API документация https://docs.google.com/document/d/1mb6CDWpbLQsi7F5UjAdwXdbCpyvSgWSXTJVHl52zZUQ/edit?ccid=8880487d88727f44ab2a911727d4d952

7. Архив муниципального этапа в Липецкой области https://ejudge.strategy48.ru/archive-municipality

8. Олимпиада Innopolis Open https://dovuz.innopolis.university/io-informatika/

9. Олимпиада Innopolis Open по ИИ и робототехнике https://ioai.innopolis.ru/