**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

специальность 1-40 05 01 «Информационные системы и технологии в проектировании и производстве»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по дисциплине «Конструирование программ и языки программирования»

на тему: **«Задачи квалификационного раунда чемпионата Беларуси и Балтии по программированию среди студентов ВУЗов»**

Исполнитель: студент группы ИТП-21 Коркуц С.И.

Руководитель: преподаватель

Кравченко О.А.

Дата проверки: \_\_\_\_\_\_\_\_\_\_\_\_\_

Дата допуска к защите: \_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты \_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка работы: \_\_\_\_\_\_\_\_\_\_\_\_\_

Подписи членов комиссии по защите курсового проекта \_\_\_\_\_\_\_\_\_\_\_\_\_

г. Гомель 2019

СОДЕРЖАНИЕ

Введение…………………………………………………………………………….3

1 Алгоритмы обработки структур данных..………………………………………7

1.1 Структуры данных………………………………………………………….7

1.2 Численные задачи…………………………………………………………10

1.3 Комбинаторные задачи …………………………………………………...11

1.4 Задачи с графами ………………………………………………………….13

1.5 Вычислительная геометрия……………………………………………….14

1.6 Множества и строки……………………………………………………….15

2 Олимпиадные задачи, их верификация и решения…………………………………………………………………………….16

2.1 Задача *L. PPI*..……………………………....……………………………...16

2.2 Задача *K. Roads*....……………………….…………………………………17

2.3 Задача *J. Puzzle*..……………………………………………………………18

2.4 Задача *G. Median*…………………………………………………………...20

2.5 Задача *I. Progression*………………………………………………………..21

2.6 Задача *H. Polygon*...………………………………………………………...23

2.7 Задача *F. Equation*...………………………….…………………………….24

2.8 Задача *E. Egypt Approximation*……………………………………………25

2.9 Задача *D. Disconnect*...……………………………………………………..27

2.10 Задача *B. Chess Queen*……………………………………………………29

2.11 Структура программного комплекса……………………………………31

Заключение………………………………………………………………………...33

Список литературы………………………………………………………………..34

Приложение А Листинг исполняемого файла...………………………………...35

Приложение Б Листинг модуля графического меню...………………………....38

Приложение В Листинг модуля решений задач...………………………..……..44

Приложение Г Листинг модуля для работы с текстовым файлом.……...……..50

Приложение Д Листинг текстового файла с условиями задач……..…..……....51

ВВЕДЕНИЕ

Спортивное программирование – это самый перспективный интеллектуальный вид спорта, который можно назвать шахматами будущего. Уже сейчас им увлекаются лучшие умы планеты, и число участников растет год от года. Рост популярности олимпиадного программирования положительно влияет на другие сферы жизнедеятельности человека. Навыки быстрого решения сложнейших задач помогают сегодняшним студентам в будущем эффективно справляться с реальными проблемами человечества.

Соревнования по программированию имеют довольно долгую историю. Международные студенческие олимпиады по программированию (*ICPC*) для студентов университетов проводились уже в 1970-х годах, а первая Международная олимпиада по информатике для учащихся старших классов состоялась в 1989 году. Ныне оба мероприятия стали регулярными и собирают множество участников со всего мира.

В наши дни олимпиадное программирование популярно как никогда. Важную роль в его распространении сыграл Интернет. Существует активное сетевое сообщество увлеченных этим движением программистов, каждую неделю проводятся различные соревнования. Одновременно растет и трудность заданий. Технические приемы, которыми еще несколько лет назад владели лишь лучшие из лучших, стали стандартными инструментами, известными многим.

Своими корнями олимпиадное программирование уходит в научное исследование алгоритмов. Но если ученый приводит доказательство работоспособности своего алгоритма, то олимпиадный программист реализует алгоритм и подает его на вход системы оценивания результатов. Эта система прогоняет алгоритм через различные тесты, и если все они проходят, то решение принимается. Это важный элемент олимпиадного программирования, который позволяет автоматически получить убедительные аргументы в пользу правильности алгоритма. Вообще, олимпиадное программирование стало отличным способом изучения алгоритмов, т.к. от участника требуется спроектировать алгоритм, который действительно работает, а не ограничиваться наброском идей, может, правильных, а может, и нет.

У олимпиадного программирования есть еще одно достоинство – конкурсные задачи заставляют думать. В формулировках задач не бывает никакого жульничества, в отличие от многих курсов по алгоритмам, где вам предлагают решить красивую задачу, но только последнее предложение звучит, к примеру, так: «Подсказка: для решения задачи модифицируйте алгоритм Дейкстры». Ну, а дальше думать особенно нечего, поскольку поход к решению уже известен. В олимпиадном программировании так не бывает. У вас имеется полный комплект инструментов, а уж какими из них воспользоваться, решайте сами.

Решение олимпиадных задач развивает навыки программирования и отладки. Обычно решение засчитывается, только если пройдены все тесты, поэтому программист, стремящийся к успеху, должен реализовывать алгоритм без ошибок. Это умение высоко ценится в программной инженерии, так что неудивительно, что ИТ-компании проявляют интерес к имеющим опыт олимпиадного программирования.

Олимпиадное программирование состоит из двух частей – проектирования алгоритмов и реализации алгоритмов.

Проектирование алгоритмов.

По сути своей, олимпиадное программирование – это придумывание эффективных алгоритмов решения корректно поставленных вычислительных задач. Для проектирования алгоритмов необходимы навыки в решении задач и знание математики.

Важную роль в олимпиадном программировании играет математика.

На самом деле четких границ между проектированием алгоритмов и математикой не существует.

Реализация алгоритмов.

В олимпиадном программировании решение задачи оценивается путем проверки реализованного алгоритма на ряде тестов. Поэтому придумать алгоритм недостаточно, его еще нужно корректно реализовать, для чего требуется умение программировать.

Олимпиадное программирование сильно отличается от традиционной

программной инженерии: программы короткие (несколько сотен строк –

уже редкость), писать их нужно быстро, а сопровождение после соревнования не требуется.

В настоящее время на соревнованиях по программированию популярнее всего языки *C++, Python и Java.* Например, среди 3000 лучших участников *Google Code Jam* 2017 79% писали на *C++*, 16% – на *Python* и 8% – на *Java*. Многие считают *C++* самым подходящим выбором для олимпиадного программиста. К его достоинствам можно отнести очень высокую эффективность и тот факт, что в стандартной библиотеке много разнообразных структур данных и алгоритмов.

Соревнования по программированию.

Международная олимпиада по информатике (*IOI*) – ежегодное соревнование для старшеклассников. От каждой страны допускается команда из четырех человек. Обычно набирается около 300 участников из 80 стран.

*IOI* проводится в течение двух дней. В каждый день участникам предлагается решить три трудные задачи, на решение отводится пять часов.

Задачи разбиты на подзадачи, за каждую из которых начисляются баллы.

Хотя участники являются членами команды, соревнуются они самостоятельно.

Участники *IOI* отбираются на национальных олимпиадах. *IOI* предшествует множество региональных соревнований, например: Балтийская олимпиада по информатике (*BOI*), Центрально-Европейская олимпиада по информатике (*CEOI*) и Азиатско-Тихоокеанская олимпиада по информатике (*APOI*).

*ICPC* (Международная студенческая олимпиада по программированию) проводится ежегодно для студентов университетов. В каждой команде три участника; в отличие от *IOI*, студенты работают вместе, и каждой команде выделяется только один компьютер.

*ICPC* включает несколько этапов, лучшие команды приглашаются на финальный этап мирового первенства. Хотя в соревновании принимают участие тысячи студентов, количество участников финала ограничено, поэтому даже сам выход в финал считается большим достижением.

На соревновании *ICPC* у команды есть пять часов на решение примерно десяти алгоритмических задач. Решение засчитывается, только если программа прошла все тесты и показала свою эффективность. В ходе соревнования участники могут видеть результаты соперников, но в начале пятого часа табло замораживается, и результаты последних прогонов не видны.

Чтобы научиться олимпиадному программированию, нужно напряженно работать. Но способов приобрести практический опыт много, одни лучше, другие хуже.

Решая задачи, нужно иметь в виду, что не так важно количество решенных задач, как их качество. Возникает соблазн выбирать красивые и легкие задачи, пропуская те, что кажутся трудными и требующими кропотливого труда. Но чтобы отточить свои навыки, нужно отдавать предпочтение именно задачам второго типа.

Важно и то, что большинство олимпиадных задач решается с помощью простого и короткого алгоритма, самое трудное – придумать этот алгоритм. Смысл олимпиадного программирования – не в заучивании сложных и малопонятных алгоритмов, а в том, чтобы научиться решать трудные задачи, обходясь простыми средствами, а умение быстро реализовывать задачи даст вам огромное преимущество.

ГЛАВА 1 АЛГОРИТМЫ ОБРАБОТКИ СТРУКТУР ДАННЫХ

1.1 Структуры данных

Структура данных - способ представления информации, посредством которого совокупность отдельно взятых элементов образует нечто единое, обусловленное их взаимосвязью друг с другом. Скомпонованные по каким-либо правилам и логически связанные межу собой, данные могут весьма эффективно обрабатываться, так как общая для них структура предоставляет набор возможностей управления ими – одно из того за счет чего достигаются высокие результаты в решениях тех или иных задач.

Но не каждый объект представляем в произвольной форме, а возможно и вовсе для него имеется лишь один единственный метод интерпретации, следовательно, несомненным плюсом для программиста будет знание всех существующих структур данных. Таким образом, часто приходиться делать выбор между различными методами хранения информации, и от такого выбора зависит работоспособность продукта.

Говоря о не вычислительной технике, можно показать ни один случай, где у информации видна явная структура. Наглядным примером служат книги самого разного содержания. Они разбиты на страницы, параграфы и главы, имеют, как правило, оглавление, то есть интерфейс пользования ими. В широком смысле, структурой обладает всякое живое существо, без нее органика навряд-ли смогла бы существовать.

Массивы

Массив – это структура данных с фиксированным и упорядоченным набором однотипных элементов (компонентов). Доступ к какому-либо из элементов массива осуществляется по имени и номеру (индексу) этого элемента. Количество индексов определяет размерность массива. Так, например, чаще всего встречаются одномерные (вектора) и двумерные (матрицы) массивы. Первые имеют один индекс, вторые – два.

Пусть одномерный массив называется *A*, тогда для получения доступа к его *i*-ому элементу потребуется указать название массива и номер требуемого элемента: *A[i]*. Когда *A* – матрица, то она представляема в виде таблицы, доступ к элементам которой осуществляется по имени массива, а также номерам строки и столбца, на пересечении которых расположен элемент: *A[i, j]*, где *i* – номер строки, *j* – номер столбца.

В разных языках программирования работа с массивами может в чем-то различаться, но основные принципы, как правило, везде одни. В языке *Pascal,* обращение к одномерному и двумерному массиву происходит точно так, как это показано выше, а, например, в *C++* двумерный массив следует указывать так: *A[i][j]*. Элементы массива нумеруются поочередно. На то, с какого значения начинается нумерация, влияет язык программирования. Чаще всего этим значением является 0 или 1.

Массивы, описанного типа называются статическими, но существуют также массивы по определенным признакам отличные от них: динамические и гетерогенные. Динамичность первых характеризуется непостоянностью размера, т.е. по мере выполнения программы размер динамического массива может изменяться. Такая функция делает работу с данными более гибкой, но при этом приходится жертвовать быстродействием, да и сам процесс усложняется.

Обязательный критерий статического массива, как было сказано, это однородность данных, единовременно хранящихся в нем. Когда же данное условие не выполняется, то массив является гетерогенным. Его использование обусловлено недостатками, которые имеются в предыдущем виде, но оно оправданно во многих случаях.

Таким образом, даже если Вы определились со структурой, и в качестве нее выбрали массив, то этого все же недостаточно. Ведь массив это только общее обозначение, род для некоторого числа возможных реализаций. Поэтому необходимо определиться с конкретным способом представления, с наиболее подходящим массивом.

Графы

Раздел дискретной математики, занимающийся изучением графов, называется теорией графов. В теории графов подробно рассматриваются известные понятия, свойства, способы представления и области применения этих математических объектов. Нас же интересует, лишь те ее аспекты, которые важны в программировании.

Граф – совокупность точек, соединенных линиями. Точки называются вершинами (узлами), а линии – ребрами (дугами).

Различают два основных вида графов: ориентированные и неориентированные. В первых ребра являются направленными, т.е. существует только одно доступное направление между двумя связными вершинами, например из вершины 1 можно пройти в вершину 2, но не наоборот. В неориентированном связном графе из каждой вершины можно пройти в каждую и обратно. Частный случай двух этих видов – смешанный граф. Он характерен наличием как ориентированных, так и неориентированных ребер.

Степень входа вершины – количество входящих в нее ребер, степень выхода – количество исходящих ребер.

Ребра графа необязательно должны быть прямыми, а вершины обозначаться именно цифрами. К тому же встречаются такие графы, ребрам которых поставлено в соответствие конкретное значение, они именуются взвешенными графами, а это значение – весом ребра. Когда у ребра оба конца совпадают, т.е. ребро выходит из вершины *F* и входит в нее, то такое ребро называется петлей.

Графы широко используются в структурах, созданных человеком, например в компьютерных и транспортных сетях, *web-*технологиях. Специальные способы представления позволяют использовать граф в информатике (в вычислительных машинах). Самые известные из них: «Матрица смежности», «Матрица инцидентности», «Список смежности», «Список рёбер». Два первых, как понятно из названия, для репрезентации графа используют матрицу, а два последних – список.

Деревья

Дерево как математический объект это абстракция из соименных единиц, встречающихся в природе. Схожесть структуры естественных деревьев с графами определенного вида говорит о допущении установления аналогии между ними. А именно со связанными и вместе с этим ациклическими (не имеющими циклов) графами. Последние по своему строению действительно напоминают деревья, но в чем то и имеются различия, например, принято изображать математические деревья с корнем расположенным вверху, т.е. все ветви «растут» сверху вниз. Известно же, что в природе это совсем не так.

Поскольку дерево это по своей сути граф, у него с последним многие определения совпадают, либо интуитивно схожи. Так корневой узел (вершина 6) в структуре дерева – это единственная вершина (узел), характерная отсутствием предков, т.е. такая, что на нее не ссылается ни какая другая вершина, а из самого корневого узла можно дойти до любой из имеющихся вершин дерева, что следует из свойства связности данной структуры.

Узлы, не ссылающиеся ни на какие другие узлы, иначе говоря, ни имеющие потомков называются листьями, либо терминальными узлами. Элементы, расположенные между корневым узлом и листьями – промежуточные узлы. Каждый узел дерева имеет только одного предка, или если он корневой, то не имеет ни одного.

Поддерево – часть дерева, включающая некоторый корневой узел и все его узлы-потомки. Так, например, одно из поддеревьев включает корень и элементы.

С деревом можно выполнять многие операции, например, находить элементы, удалять элементы и поддеревья, вставлять поддеревья, находить корневые узлы для некоторых вершин и др. Одной из важнейших операций является обход дерева. Выделяются несколько методов обхода. Наиболее популярные из них: симметричный, прямой и обратный обход. При прямом обходе узлы-предки посещаются прежде своих потомков, а в обратном обходе, соответственно, обратная ситуация. В симметричном обходе поочередно просматриваются поддеревья главного дерева.

Представление данных в рассмотренной структуре выгодно в случае наличия у информации явной иерархии. Например, работа с данными о биологических родах и видах, служебных должностях, географических объектах и т.п. требует иерархически выраженной структуры, такой как математические деревья.

1.2 Численные задачи

Решение систем линейных уравнений

Метод Гаусса — классический метод решения системы линейных уравнений. Он включает в себя 2 этапа:

* Последовательное исключение
* Обратная подстановка

Исключения Гаусса основаны на идее последовательного исключения переменных по одной до тех пор, пока не останется только одно уравнение с одной переменной в левой части. Затем это уравнение решается относительно единственной переменной. Таким образом, систему уравнений приводят к треугольной (ступенчатой) форме. Для этого среди элементов первого столбца матрицы выбирают ненулевой (а чаще максимальный) элемент и перемещают его на крайнее верхнее положение перестановкой строк. Затем нормируют все уравнения, разделив его на коэффициент , где *i*– номер столбца.

Затем вычитают получившуюся после перестановки первую строку из остальных строк.

Получают новую систему уравнений, в которой заменены соответствующие коэффициенты.

После того, как указанные преобразования были совершены, первую строку и первый столбец мысленно вычёркивают и продолжают указанный процесс для всех последующих уравнений пока не останется уравнение с одной неизвестной.

Обратная подстановка предполагает подстановку полученного на предыдущем шаге значения переменной в предыдущие уравнения.

Эта процедура повторяется для всех оставшихся решений.

Разложение числа на множители и проверка чисел на простоту

Всякое число можно разложить на простые множители. При этом получается одно и то же разложение, если не учитывать порядка записи множителей.

Последовательность действий, которые выполняют при разложении числа на простые множители в математике

* Проверить, не является ли предложенное число простым
* Если нет, то подобрать делитель из простых чисел, начиная с наименьшего
* Повторять, пока частное не окажется простым числом

Для определения простоты числа можно использовать алгоритм «Решето Эрастофена».

* Задать начальное значение делителя = 2
* Проверить, делится ли число на делитель. Если да, записать делитель в список множителей и разделить число на делитель
* Повторить предыдущий шаг пока выполняется условие кратности
* Перейти к следующему делителю (в простейшем случае увеличить делитель на 1)
* Вычисления закончить когда частное от деления станет равным 1

1.3 Комбинаторные задачи

Сортировка

Сортировка пузырьком

Будем идти по массиву слева направо. Если текущий элемент больше следующего, меняем их местами. Делаем так, пока массив не будет отсортирован. Заметим, что после первой итерации самый большой элемент будет находиться в конце массива, на правильном месте. После двух итераций на правильном месте будут стоять два наибольших элемента, и так далее. Очевидно, не более чем после *n* итераций массив будет отсортирован. Таким образом, асимптотика в худшем и среднем случае – *O()*, в лучшем случае – *O(n).*

Быстрая сортировка

Выберем некоторый опорный элемент. После этого перекинем все элементы, меньшие его, налево, а большие – направо. Рекурсивно вызовемся от каждой из частей. В итоге получим отсортированный массив, так как каждый элемент меньше опорного стоял раньше каждого большего опорного. Асимптотика: *O()* в среднем и лучшем случае, *O()*. Наихудшая оценка достигается при неудачном выборе опорного элемента. Моя реализация этого алгоритма совершенно стандартна, идем одновременно слева и справа, находим пару элементов, таких, что левый элемент больше опорного, а правый меньше, и меняем их местами. Помимо чистой быстрой сортировки, участвовала в сравнении и сортировка, переходящая при малом количестве элементов на сортировку вставками. Константа подобрана тестированием, а сортировка вставками — наилучшая сортировка, подходящая для этой задачи (хотя не стоит из-за этого думать, что она самая быстрая из квадратичных).

Поиск

Последовательный поиск

Последовательный поиск предполагает последовательный просмотр всех записей множества, организованного как массив.

Индексно-последовательный поиск

Для индексно-последовательного поиска в дополнение к отсортированной таблице заводится вспомогательная таблица, называемая индексной.

Каждый элемент индексной таблицы состоит из ключа и указателя на запись в основной таблице, соответствующей этому ключу. Элементы в индексной таблице, как элементы в основной таблице, должны быть отсортированы по этому ключу.

Если индекс имеет размер, составляющий 1/8 от размера основной таблицы, то каждая восьмая запись основной таблицы будет представлена в индексной таблице.

Достоинство алгоритма индексно-последовательного поиска заключается в том, что сокращается время поиска, так как последовательный поиск первоначально ведется в индексной таблице, имеющей меньший размер, чем основная таблица. Когда найден правильный индекс, второй последовательный поиск выполняется по небольшой части записей основной таблицы.

Генерирование перестановок

Рассмотрим задачу получения всех перестановок чисел 1…*N* (то есть последовательности длины *N*), где каждое из чисел входит ровно по 1 разу.

Существует множество вариантов порядка получения перестановок. Однако наиболее часто решается задача генерации перестановок в лексикографическом порядке (см. пример выше). При этом все перестановки сортируются сначала по первому числу, затем по второму и т.д. в порядке возрастания. Таким образом, первой будет перестановка 1 2 … *N*, а последней — *N N-1* … 1.

Рассмотрим алгоритм решения задачи. Дана исходная последовательность чисел. Для получения каждой следующей перестановки необходимо выполнить следующие шаги

* Необходимо просмотреть текущую перестановку справа налево и при этом следить за тем, чтобы каждый следующий элемент перестановки (элемент с большим номером) был не более чем предыдущий (элемент с меньшим номером). Как только данное соотношение будет нарушено необходимо остановиться и отметить текущее число (позиция 1)
* Снова просмотреть пройденный путь справа налево пока не дойдем до первого числа, которое больше чем отмеченное на предыдущем шаге
* Поменять местами два полученных элемента
* Теперь в части массива, которая размещена справа от позиции 1 надо отсортировать все числа в порядке возрастания. Поскольку до этого они все были уже записаны в порядке убывания необходимо эту часть подпоследовательность просто перевернуть

1.4 Задачи с графами

Обход графа в ширину

Под обходом понимается последовательное посещение (обработка) вершин графа в определённом порядке. Одним из двух часто использующихся способов обхода является обход в ширину, или *BFS* (англ. *breadth-first search*, поиск в ширину). Его иногда также называют волновым, по аналогии с распространяющейся волной.

Суть *BFS* достаточно проста. Обход начинается с посещения определённой вершины (для обхода всего графа часто выбирается произвольная вершина). Затем алгоритм посещает соседей этой вершины. За ними - соседей соседей, и так далее.

Более формально, пусть *d**[**i**]* - расстояние от начальной вершины до вершины с номером *i* (длина кратчайшего пути в рёбрах). *BFS* посещает вершины в порядке возрастания *d**[**i**]*: от наименее до наиболее отдалённых.

Сам алгоритм достаточно тривиален. Поддерживается очередь из вершин для посещения. При посещении очередной вершины в очередь добавляются все её соседи, которые ещё не были посещены и ещё не находятся в очереди. Для проверки, была ли вершина уже посещена, используется массив меток. Изначально *v**i**s**i**t**e**d**[**i**]**=**f**a**l**s**e* для всех *i* кроме начальной вершины. При добавлении вершины *i* в очередь *v**i**s**i**t**e**d**[**i**]* присваивается *t**r**u**e*.

Обход графа в глубину

Поиском в глубину (*DFS – depth first search*) называется один из методов обхода графа *G = (V, E)*, суть которого состоит в том, чтобы идти “вглубь” пока это возможно. В процессе поиска в глубину вершинам графа присваиваются номера, а ребра помечаются. Обход вершин графа происходит согласно принципу: если из текущей вершины есть ребра, ведущие в непройденные вершины, то идем туда, иначе возвращаемся назад.

Поиск в глубину начинается с выбора начальной вершины *v* графа *G*, которая сразу же помечается как пройденная. Потом для каждой не помеченной вершины, смежной с *v*, рекурсивно вызывается поиск в глубину. Когда все вершины, достижимые из *v*, будут помечены, поиск заканчивается.

Если на некотором (не начальном) шаге обхода поиск закончился, но некоторые вершины остаются не помеченными (такой случай возможен в случае ориентированного или несвязного графа), то выбирается произвольная из них и поиск повторяется. Процесс поиска продолжается до тех пор, пока все вершины графа *G* не будут помечены.

1.5 Вычислительная геометрия

Проверка принадлежности точки многоугольнику

Как известно, простой многоугольник — это фигура, состоящая из не пересекающихся отрезков («сторон»), соединённых попарно с образованием замкнутого пути. По заданному простому многоугольнику и точке требуется определить, лежит ли эта точка внутри или на границе этого многоугольника или вне его.

Задача сводится к поиску площадей. Считаю площадь треугольника, образующегося тремя последовательными точками многоугольника. Далее считаю площади треугольников которые заданная точка образует с каждой парой точек из этой пары. Если площадь первого треугольника равна сумме площадей этих, то точка находится в треугольнике, а следовательно и в многоугольнике. Если нет перехожу к следующей тройке точек. Если точка не принадлежит ни одному треугольнику, то точка находится вне многоугольника.

Вычисление площади многоугольника

Заданы координаты n последовательных вершин многоугольника. Определить его площадь.

Для того, чтобы вычислить его площадь, воспользуемся формулой:

, где

Нахождения угла между векторами

Формула нахождения угла между векторами

1.6 Множества и строки

Поиск максимальной общей подпоследовательности

Последовательность представляет собой упорядоченный набор элементов. Строка — это частный случай последовательности, дальнейшие примеры будут для простоты рассматривать именно строки, но без изменений их можно использовать и для произвольного текста или чего-нибудь последовательного еще.

Пусть имеется последовательность *x*, состоящая из элементов и последовательность *y*, состоящая из элементов . *Z* — подпоследовательность *x* в том случае, если существует строго возрастающий набор индексов элементов *x*, из которых получается *z*.

Общей подпоследовательностью для *x* и *y* считаем такую последовательность *z*, которая является одновременно подпоследовательностью *x* и подпоследовательностью *y*.  
Максимальная общая подпоследовательность — это общая подпоследовательность с максимальной длинной. Далее по тексту будем использовать сокращение *LCS*.

В качестве примера, пусть *x=HABRAHABR, y=HARBOUR*, в этом случае *LCS(x, y)=HARBR*. Можно уже переходить непосредственно к алгоритму вычисления *LCS*, но, хорошо бы понять, для чего нам может это может понадобиться.

Алгоритм Нидлмана—Вунша(*Needleman-Wunsch*).

Весь подход сводится к поэтапному заполнению матрицы, где строки представляют собой элементы *x*, а колонки элементы *y*. При заполнении действуют два правила, вытекающие из уже сделанных наблюдений:

* Если элемент равен то в ячейке *(i,j)* записывается значение ячейки *(i-1,j-1)* с добавлением единицы
* Если элемент не равен то в ячейку *(i,j)* записывается максимум из значений *(i-1,j)* и *(i,j-1)*.

ГЛАВА 2 ОЛИМПИАДНЫЕ ЗАДАЧИ, ИХ ВЕРИФИКАЦИЯ И РЕШЕНИЯ

2.1 Задача *L — PPI*

Условие

Задано разрешение экрана в пикселях и физический размер его диагонали в дюймах. Требуется вычислить разрешающую способность матрицы в точках на дюйм (англ. *ppi — pixels per inch*).

Можно считать, что пиксель имеет форму квадрата.

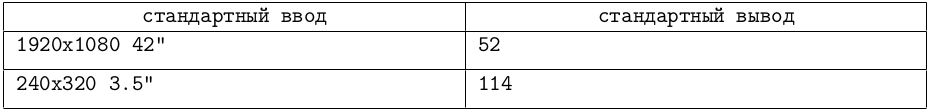
Формат входных данных

Разрешение записано как произведение *w×h* (), числа *w* и *h* целые, разделителем является строчная латинская буква *x*). Длина диагонали *d* записана в виде вещественного числа с не более чем одним знаком после точки (), после числа идёт символ дюйма в виде двойной кавычки (").

Формат выходных данных

Выведите ответ, округлённый до ближайшего целого (в случае неоднозначности можно округлять в любую сторону).

Пример входа, выхода на рисунке 1

Рисунок 1 — пример входа, выхода задачи *L - PPI*

Алгоритм решения

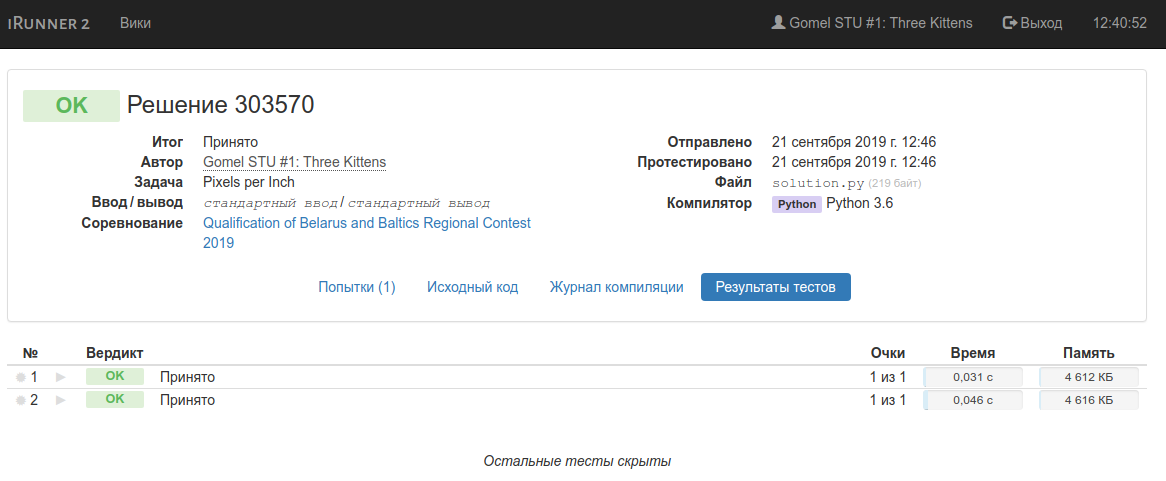
Задача весьма тривиальна и сводится к нахождению формулы. Что бы её найти необходимо знать как найти размер диагонали в пикселях и разделить на количество дюймов диагонали.

Формула для решения задачи

Верификация

Написанная программа прошла тесты на платформе *iRunner*.

Результаты теста на рисунке 2.

Рисунок 2 - результат прохождения теста задачи *L - PPI*

2.2 Задача *K. Roads*

Условие

В Байтландии *n* городов. Некоторые пары городов соединены между собой двусторонними дорогами. Сеть дорог позволяет из любого города проехать в любой другой. К сожалению, из-за недостатка финансирования дороги год за годом приходят в упадок и выводятся из эксплуатации.

Известен порядок, в котором будут закрываться все дороги. Определите, в какой момент при закрытии очередной дороги жители Байтландии потеряют возможность проехать из какого-либо одного города в другой город.

Формат входных данных

В первой строке через пробел записаны числа *n* и *m* () — число городов и число дорог соответственно.

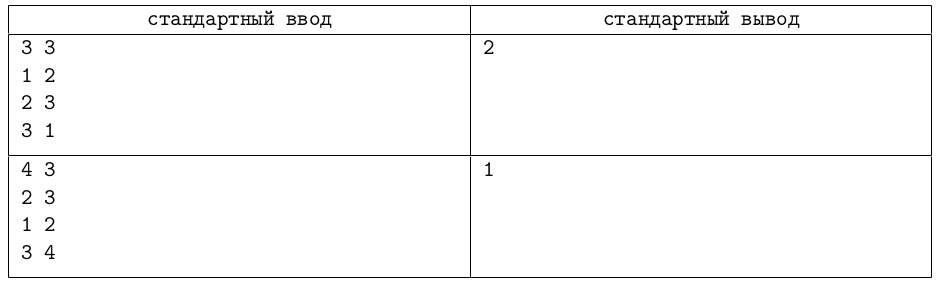
В последующих *m* строках заданы все дороги Байтландии в хронологическом порядке их закрытия. Дорога описывается парой чисел *u* и *v* (). Между парой городов может быть

построено несколько дорог.

Формат выходных данных

Выведите одно число — номер дороги, после закрытия которой дорожная сеть утратит связность. Дороги нумеруются начиная с единицы в соответствии с порядком следования на входе.

Пример входа, выхода на рисунке 3

Рисунок 3 - пример входа, выхода задачи *K - Roads*

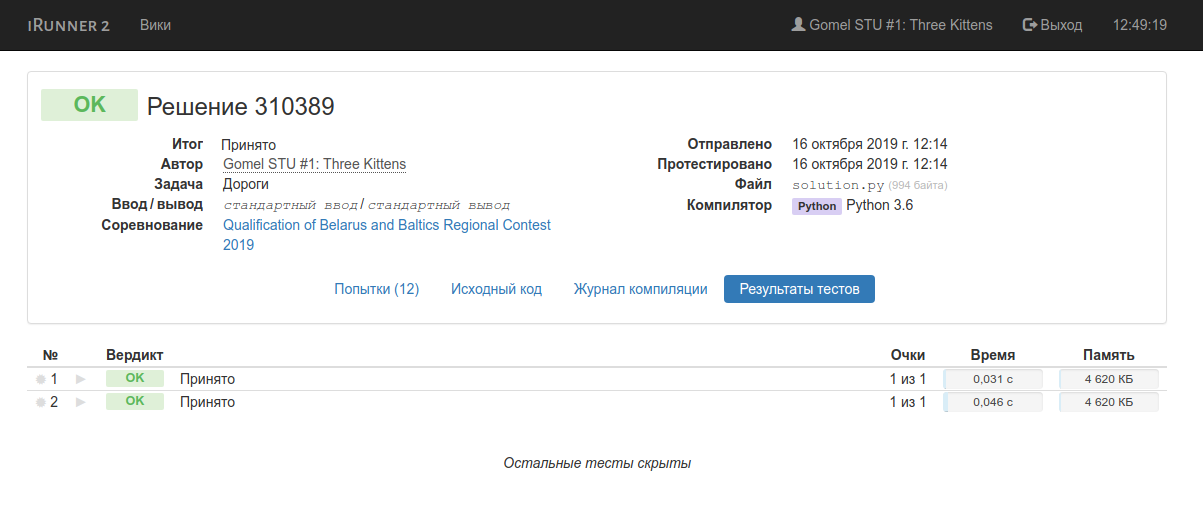
Алгоритм решения

Алгоритм основан на обходе графа в глубину. После каждого удаления дороги нужно проверить соединён ли какой-либо город со всеми городами, если да продолжаем цикл, иначе возвращаем номер дороги, которую удалили.

Верификация

Написанная программа прошла тесты на платформе *iRunner*.

Результаты теста на рисунке 4.

Рисунок 4 - результат прохождения теста задачи *K - Roads*

2.3 Задача *J. Puzzle*

Условие

Костя расположил в форме треугольника *3k − 3* шарика, каждая сторона треугольника состоит ровно из *k* шариков. На одной стороне треугольника находятся шарики с номерами 1, 2, . . . , *k*, на второй — шарики с номерами *k, k* + *1*, . . . , *2k − 1*, на третьей — шарики с номерами *2k − 1, 2k*, . . . , *3k − 3, 1*.

Косте нужно написать на шариках по одному целому числу из диапазона от 1 до 10 9 так, чтобы сумма чисел на шариках на каждой стороне треугольника была равна n и максимальное использованное число было как можно меньше. При этом Костя может использовать одинаковые числа сколько угодно раз. Если нельзя написать числа на шариках таким образом, то подскажите ему об этом.

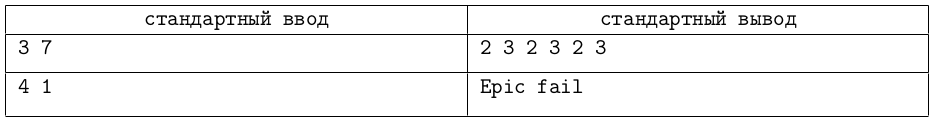
Формат входных данных

В единственной строке через пробел записаны два целых числа *k* и *n* ().

Формат выходных данных

Если подходящего способа написать числа на шариках нет, выведите *Epic fail*. Иначе выведите *3k − 3* целых числа, которые Костя может написать на шариках, в порядке их нумерации. Если наборов чисел, удовлетворяющих условию задачи, несколько, выведите любой из них.

Пример входа, выхода на рисунке 5

Рисунок 5 - пример входа, выхода задачи *J - Puzzle*

Алгоритм решения

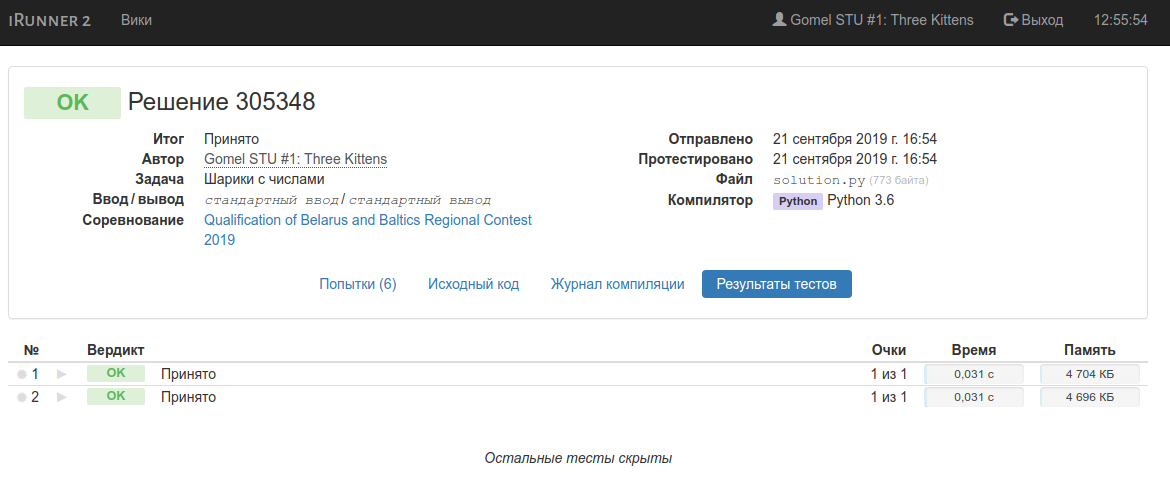
Для решения задачи необходимо проверить пару условий:

* Если *n < k* или *n <= 0* или *k <= 0*, то решения нет
* Если *k* - это делитель *n*, то просто выводим частное от деления *n* на *k*, умноженное на количество шариков
* Иначе нужно к остатку от деления *n* на *k* кружочкам, находящимся не на краю нужно добавить единицу

Верификация

Написанная программа прошла тесты на платформе *iRunner.*

Результаты теста на рисунке 6.

Рисунок 6 - результат прохождения теста задачи *J - Puzzle*

2.4 Задача *G. Median*

Условие

Медиана в математической статистике — число, характеризующее выборку (например, набор чисел). Медиану можно найти, упорядочив элементы выборки по не убыванию и взяв средний элемент.

Например, выборка (11, 9, 3, 5, 5) после упорядочивания превращается в (3, 5, 5, 9, 11), и её медианой является число 5. Если в выборке чётное число элементов, в качестве медианы будем использовать полусумму двух соседних значений (то есть медиану набора (1, 3, 5, 7) считаем равной 4).

На вход поступают запросы двух видов: на добавление одного числа в выборку и на вычисление медианы. Нужно научиться эффективно их обрабатывать.

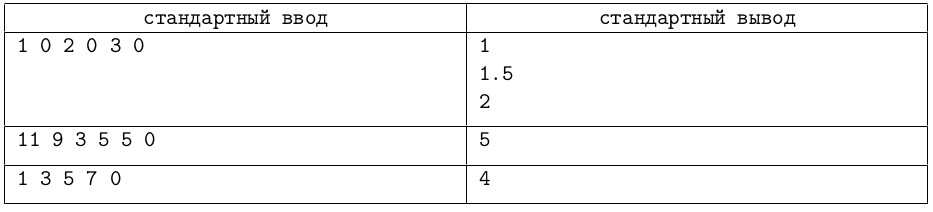
Формат входных данных

Целые числа () записаны в одну строку. Если число положительное, то его нужно добавить в выборку. Если число равно нулю, то нужно вычислить медиану выборки, которая была накоплена к данному моменту. Общее количество чисел на входе не менее 1 и не превосходит 300 000. Гарантируется, что первое число не равно нулю.

Формат выходных данных

Для каждого числа 0 на входе выведите в отдельной строке полученное значение медианы с абсолютной погрешностью не более 0,1.

Пример входа, выхода на рисунке 7

Рисунок 7 - пример входа, выхода задачи *G - Median*

Алгоритм решения

Циклом проходим по числам, если не равно нулю, то просто добавляем в массив чисел, иначе сортируем массив ненулевых чисел и находим число посередине, если длинна массива четная то находим среднее арифметическое двух медианных чисел и выводим его, иначе просто выводим медиану и продолжаем цикл.

Верификация

Написанная программа прошла тесты на платформе *iRunner*.

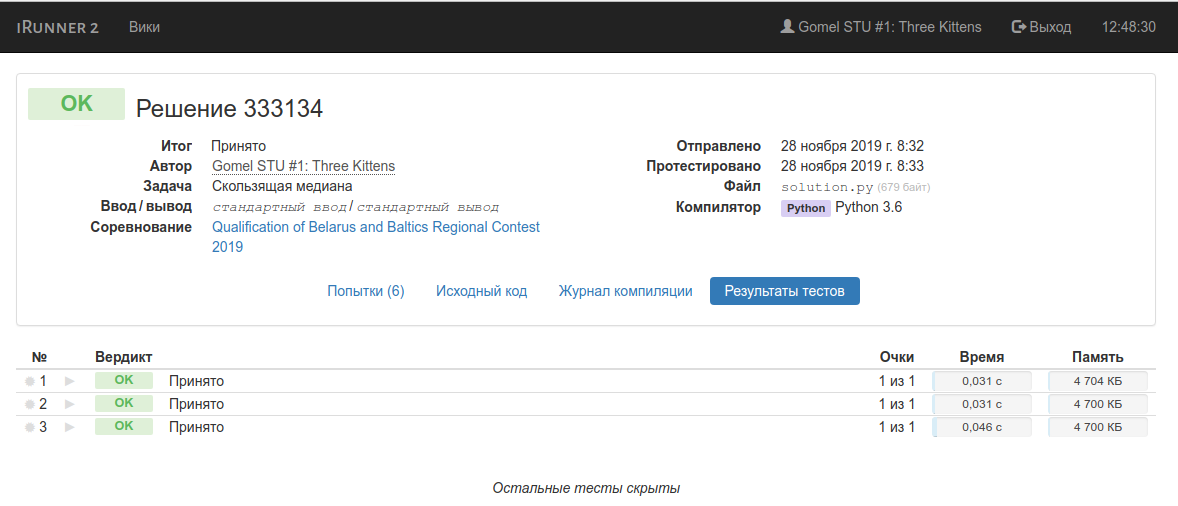
Результаты теста на рисунке 8.

Рисунок 8 - результат прохождения теста задачи *G - Median*

2.5 Задача *I. Progression*

Условие

Задан массив *A* длины *n.* Требуется изменить его элементы так, чтобы они образовали арифметическую прогрессию. Каждый элемент массива можно изменить не более одного раза: или увеличить на единицу, или уменьшить на единицу. Какое наименьшее число элементов нужно модифицировать?

Формат входных данных

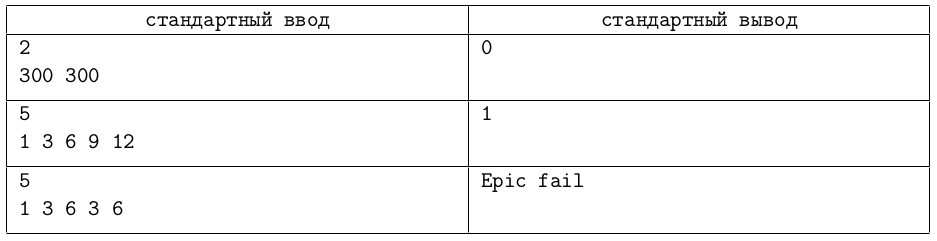
В первой строке задано целое число *n* ().

Во второй строке через пробел записано *n* целых чисел ().

Формат выходных данных

Выведите одно число — минимально возможное число операций. Если получить арифметическую прогрессию указанным способом невозможно, выведите *Epic fail*.

Пример входа, выхода на рисунке 9

Рисунок 9 - пример входа, выхода задачи *I - Progression*

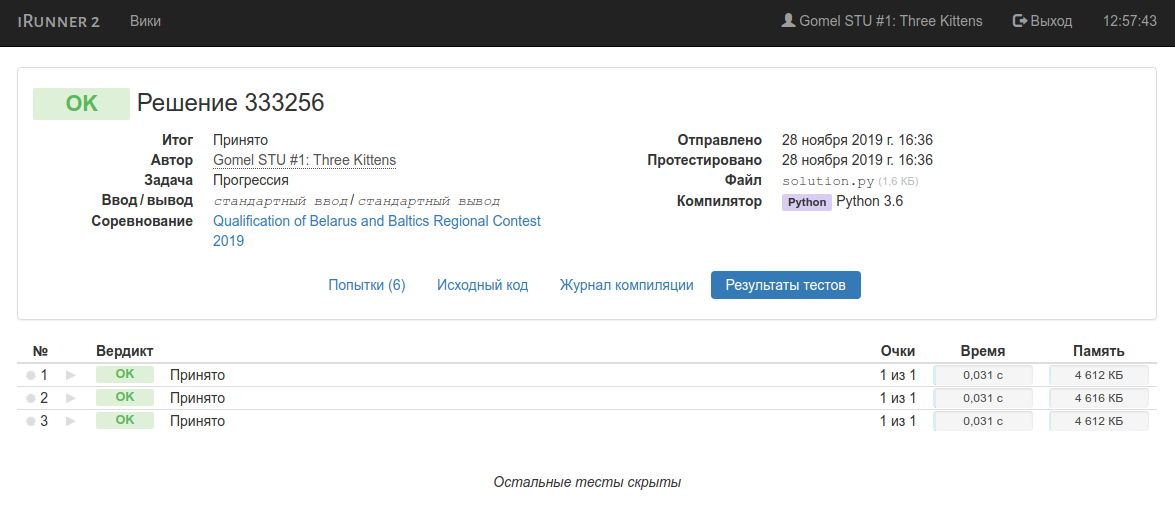
Алгоритм решения

Для начала нужно найти среднюю разность между всеми числами. Потом рассмотреть 3 ситуации, когда первый элемент - 1, когда первый элемент без изменения и когда первый элемент + 1. Проходим по списку и изменяем значения так, что бы разность между двумя соседними числами была равна средней разнице, и если мы изменяем какой-нибудь элемент, то добавляем к общему числу измененных элементов число на которое меняем. В заключении выводим минимальное число измененных элементов среди трёх случаев.

Верификация

Написанная программа прошла тесты на платформе *iRunner*.

Результаты теста на рисунке 10.

Рисунок 10 - результат прохождения теста задачи *I - Progression*

2.6 Задача *H. Polygon*

Условие

Вершины правильного тридцатиугольника пронумерованы числами от 1 до 30 против часовой стрелки. Выберите *k* вершин таким образом, чтобы центр масс этих точек совпадал с центром тридцатиугольника.

Формат входных данных

В единственной строке входных данных записано одно целое число *k* ().

Формат выходных данных

Если подходящего набора вершин не существует, выведите строку *Epic fail*. Иначе выведите строку из 30 символов. *i*-й символ должен быть равен ‘1’, если *i*-я вершина была выбрана, или ‘0’ в противном случае. Если подходящих наборов вершин несколько, выведите любой из них.

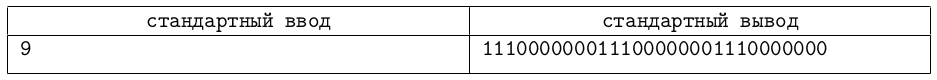
Пример входа, выхода на рисунке 11

Рисунок 11 - пример входа, выхода задачи *H - Polygon*

Алгоритм решения

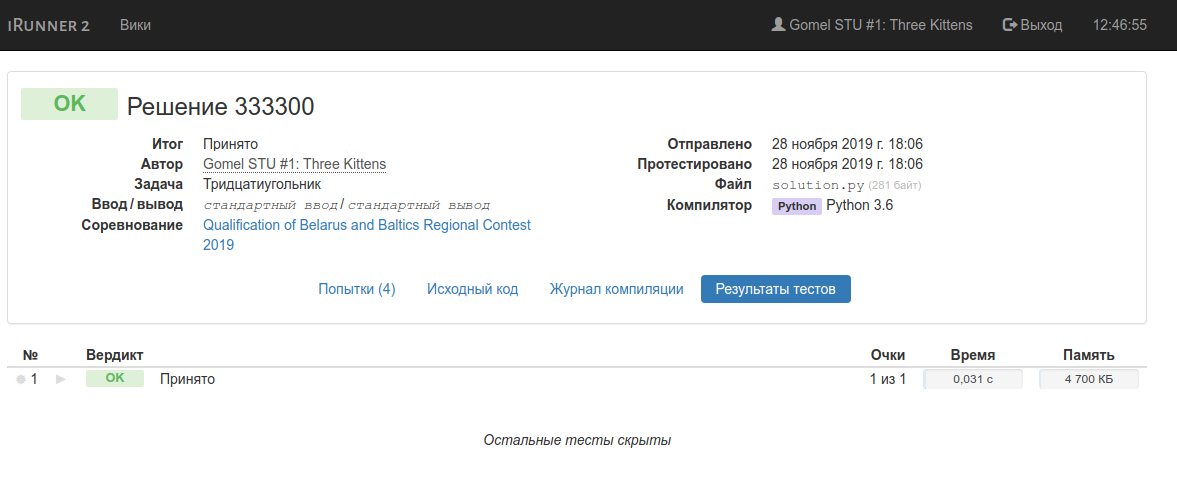
На самом деле в этой задаче можно обойтись без формулы данной в условии. Достаточно найти пару закономерностей.

Например, если число равно единице то решения нет. Потом достаточно проверить делится ли исходное число на первые три простых числа (2, 3, 5) . Если да то получаем решение. Ещё одна проверки, делится ли 30 на исходное число, опять же если делится получаем решение, иначе решения нет.

Верификация

Написанная программа прошла тесты на платформе *iRunner*.

Результаты теста на рисунке 12.

Рисунок 12 - результат прохождения теста задачи *H - Polygon*

2.7 Задача *F. Equation*

Условие

Дано *n* положительных целых чисел

Найдите число решений уравнения в положительных целых числах .

Так как искомое число решений может быть слишком большим, найдите его остаток от деления на 998 244 353.

Формат входных данных

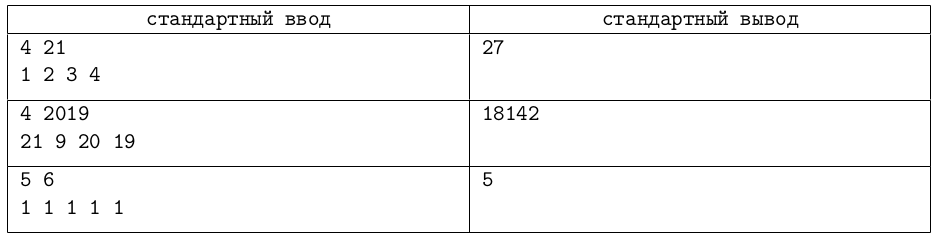
В первой строке через пробел записаны два целых числа *n* и *m* ()

Во второй строке записаны n целых положительных чисел

Формат выходных данных

Выведите остаток от деления искомого числа решений уравнения на 998 244 353.

Пример входа, выхода на рисунке 13

Рисунок 13 - пример входа, выхода задачи *F - Equation*

Алгоритм решения

Положим = + 1. Тогда получим уравнение + + + = 21, где все - натуральные. Запишем это уравнение в виде (1+1+...+1) + (1+1+...+1) + (1+1+...+1) + (1+1+...+1) = 21, где в первых скобках стоит сумма y1 единиц, во вторых - сумма единиц и т.д., а все группы слагаемых разделены тремя знаками "+". Тогда каждому решению полученного уравнения взаимно однозначно соответствует способ выбора 3-х знаков "+" из 20-ти в записи 1+1+...+1=21 с последующей группировкой слагаемых между выбранными знаками. Следовательно, всего должно быть решений .

Верификация

Написанная программа прошла тесты на платформе *iRunner*.

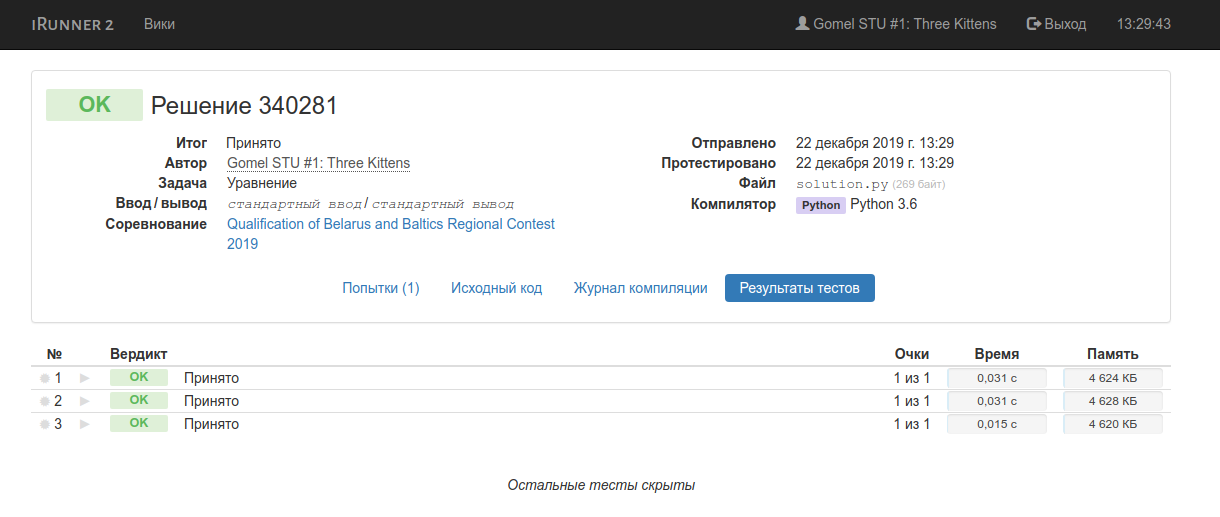
Результаты теста на рисунке 14.

Рисунок 14 - результат прохождения теста задачи *F - Equation*

2.8 Задача *E. Egypt Approximation*

Условие

Египетские математики в 1800 году до н.э. записывали рациональные числа от 0 до 1 как сумму 1 нескольких обратных различным целым. Например, число они могли записать как или , но не как

Вам необходимо приблизить дробь суммой нескольких обратных различным целым. Ваше приближение должно удовлетворять неравенству .

В своём приближении вы можете использовать не более 1000 слагаемых, все используемые числа в разложении не должны превосходить . Гарантируется, что разложение, удовлетворяющее данным требованиям, всегда существует.

Формат входных данных

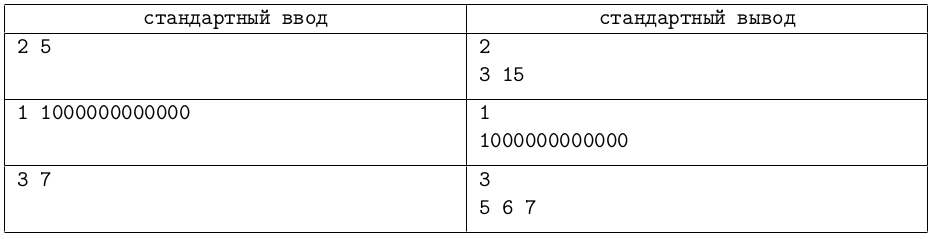
Единственная строка содержит два разделённых пробелом целых числа *a* и b (). Гарантируется, что дробь является несократимой.

Формат выходных данных

В первой строке выведите целое число *k* () — количество слагаемых. Во второй строке выведите *k* попарно различных целых чисел от 1 до — знаменатели слагаемых, дающих в сумме приближение числа .

Если число можно приблизить несколькими различными способами, выведите любой из них.

Пример входа, выхода на рисунке 15

Рисунок 15 - пример входа, выхода задачи *E - Egypt Approximation*

Алгоритм решения

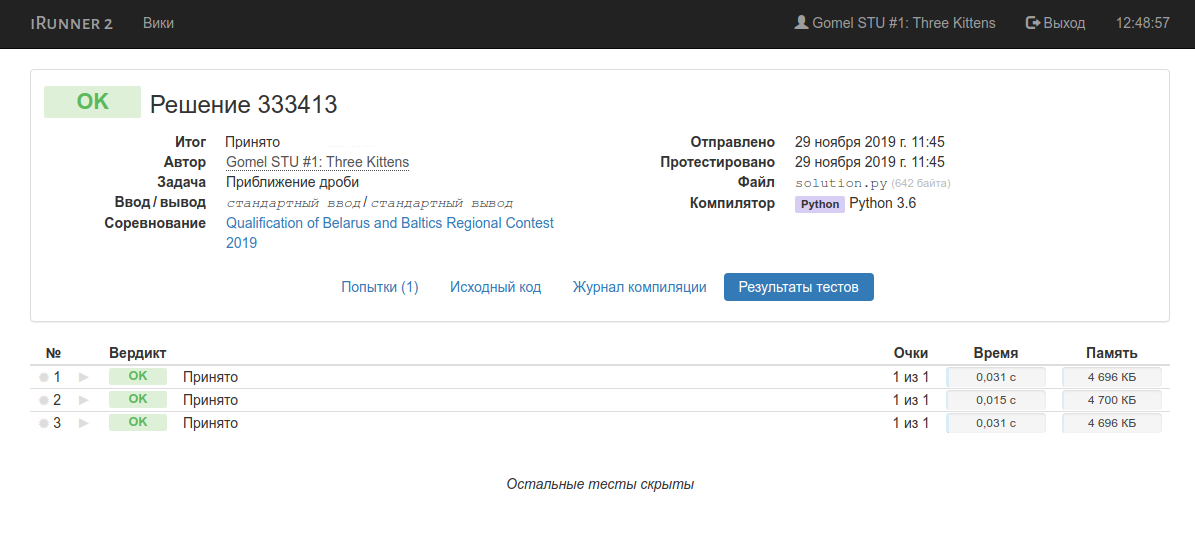
Алгоритм этой задачи достаточно не стандартный:

* Первое приближение получаем по формуле , где *round* — это округление до большего выражения
* Проверяем, равно ли это приближение искомому числу, если да то выводим в ответ одно это приближение
* Иначе добавляем это приближение в массив приближений
* И запускаем цикл *while*, в котором мы генерируем новые приближения из суммы старых, суммируем все приближения и проверяем больше ли это приближение исходного числа, если да, то выводим длинну массива приближений и знаменатели чисел в массиве приближений

Верификация

Написанная программа прошла тесты на платформе *iRunner*.

Результаты теста на рисунке 16.

Рисунок 16 - результат прохождения теста задачи *E - Egypt Approximation*

2.9 Задача *D. Disconnect*

Условие

Ориентированный граф *G = (V, E)* называется сильно связным, если из любой его вершины *a ∈ V* можно попасть в любую вершину *b ∈ V* , следуя вдоль рёбер графа из множества *E*.

Будем называть множество *E“* развязывающим множеством графа *G*, если *E“ ⊆ E* и граф *G“ = (V, E \ E“ )* не является сильно связным. Иными словами, если после удаления всех рёбер из множества *E“* граф *G* больше не сильно связен, то множество *E“* является для него развязывающим.

По заданному графу *G* найдите его развязывающее множество наименьшей мощности либо определите, что такого множества нет.

Формат входных данных

В первой строке записано целое число *T* () — количество тестовых примеров. Каждый тестовый пример описывается *m + 1* строкой. В первой строке через пробел записаны два целых числа *n и m* () - количество вершин и количество рёбер графа соответственно. Далее в каждой из *m* строк находится по два целых числа () - соответственно начало и конец *i*-го ориентированного ребра.

Рёбра нумеруются от 1 до *m* в том порядке, в котором они поступают на вход. Гарантируется, что в каждом тестовом примере никакая упорядоченная пара () не появляется дважды. Также гарантируется, что в каждом тесте находится не более чем 5 графов с *n* > 50, не более чем 20 графов с *n* > 20 и не более чем 100 графов с *n* > 10.

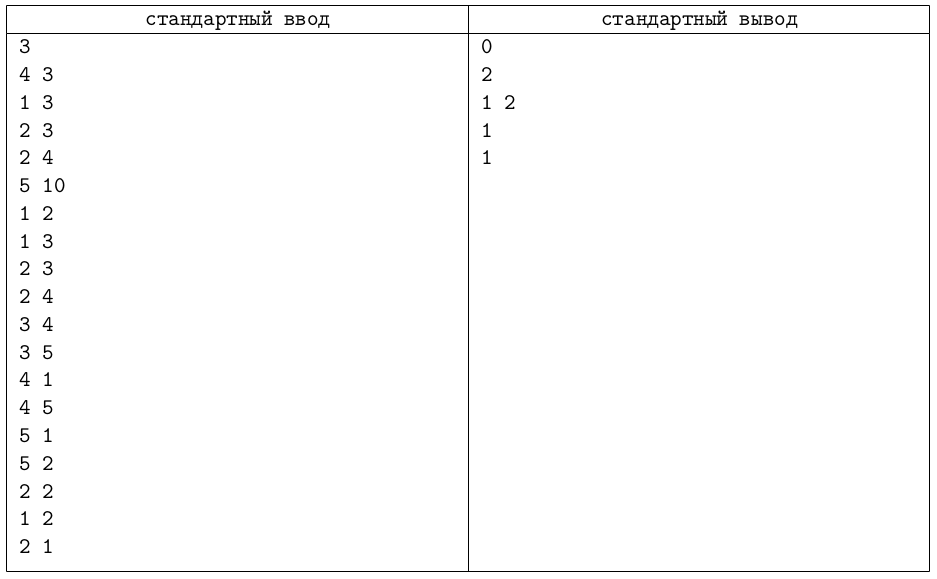
Формат выходных данных

Для каждого тестового примера выведите ответ на задачу.

Если развязывающего множества для графа не существует, выведите в строке число -1. В противном случае выведите в первой строке целое число *k* () - наименьшая возможная мощность развязывающего множества. Если *k* > 0, то во второй строке выведите через пробел *k* номеров рёбер, входящих в искомое множество. Номера рёбер должны следовать по возрастанию.

Если есть несколько ответов, выведите любой из них.

Пример входа, выхода на рисунке 17

Рисунок 17 - пример входа, выхода задачи *D - Disconnect*

Алгоритм решения

Решение задачи не такое сложное как её условие.

Опять же в этой задаче нам помогает обход графа в глубину.

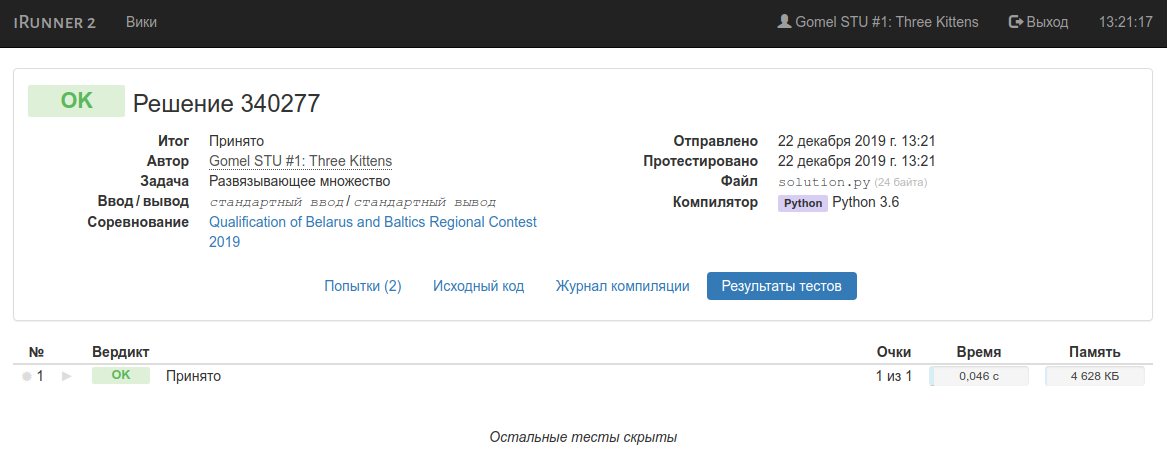
Алгоритм решения

* Сперва разбиваем весь ввод на части, которые будем исследовать и решать
* Потом проходим по каждой части и проверяем после удаления какого ребра наш граф теряет связность
* Выводим ответ на каждую подзадачу

Верификация

Написанная программа прошла тесты на платформе *iRunner*.

Результаты теста на рисунке 18.

Рисунок 18 - результат прохождения теста задачи *D - Disconnect*

2.10 Задача *B. Chess Queen*

Условие

Ферзь — самая сильная шахматная фигура. Ферзь может перемещаться на любое число свободных полей в любом направлении по прямой (по горизонтали, по вертикали и по диагонали), совмещая в себе возможности ладьи и слона.

Найдите на шахматной доске позицию, из которой ферзь атакует ровно *k* полей.

Формат входных данных

В единственной строке записано одно целое число *k* ().

Формат выходных данных

Если подходящей позиции на шахматной доске нет, то выведите *Epic fail*. Иначе выведите позицию ферзя, находясь в которой, он атакует *k* других позиций шахматной доски. Сначала выведите обозначение вертикали — символ от a до h, затем номер горизонтали — число от 1 до 8.

Если решений несколько, выведите любое.

Пример входа, выхода на рисунке 19

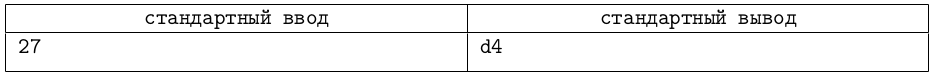


Рисунок 19 - пример входа, выхода задачи *B - Chess Queen*

Алгоритм решения

На самом деле задача очень простая.

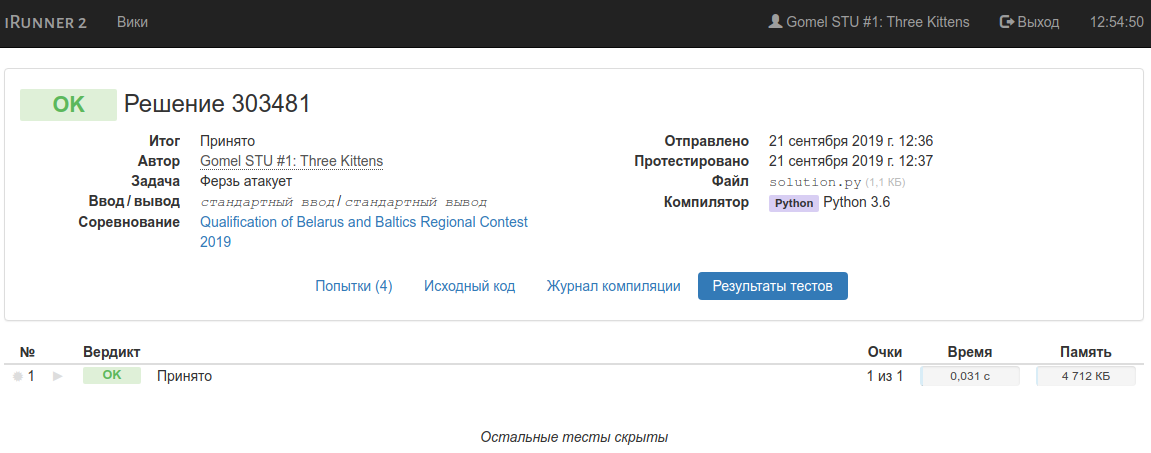
Есть всего 4 проверки и соответственно 4 возможных варианта.

Если *k* = 21 выводим *a1*, *k* = 23 выводим *b2*, *k* = 25 выводим *c3*, *k* = 27 выводим *d4*. Если ни одно из условий не выполнилось, выводим *Epic fail.*

Верификация

Написанная программа прошла тесты на платформе *iRunner*.

Результаты теста на рисунке 20.

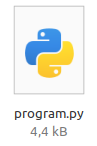
Рисунок 20 - результат прохождения теста задачи *B - Chess Queen*

2.11 Структура программного комплекса

Программный комплекс содержит несколько компонентов.

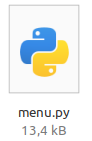
Первый компонент - непосредственно сама исполняемая программа (Рисунок 21).

В нём как бы происходит «сборка» всей программы.

Рисунок 21 - файл исполняемой программы

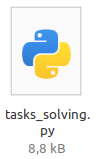
Второй компонент - модуль, содержащий графическое меню (Рисунок 22).

Здесь содержится класс, описывающий пользовательский интерфейс.

Рисунок 22 - файл исполняемой программы

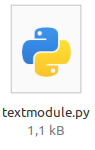
Третий компонент - модуль, содержащий решения задач (Рисунок 23).

Здесь в виде функций содержатся решения олимпиадных задач.

Рисунок 23 - файл исполняемой программы

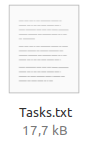
Четвертый компонент - модуль, для взаимодействия с текстовым файлом (Рисунок 24).

Здесь содержится метод, который берет данные из текстового файла и возвращает их в программный модуль.

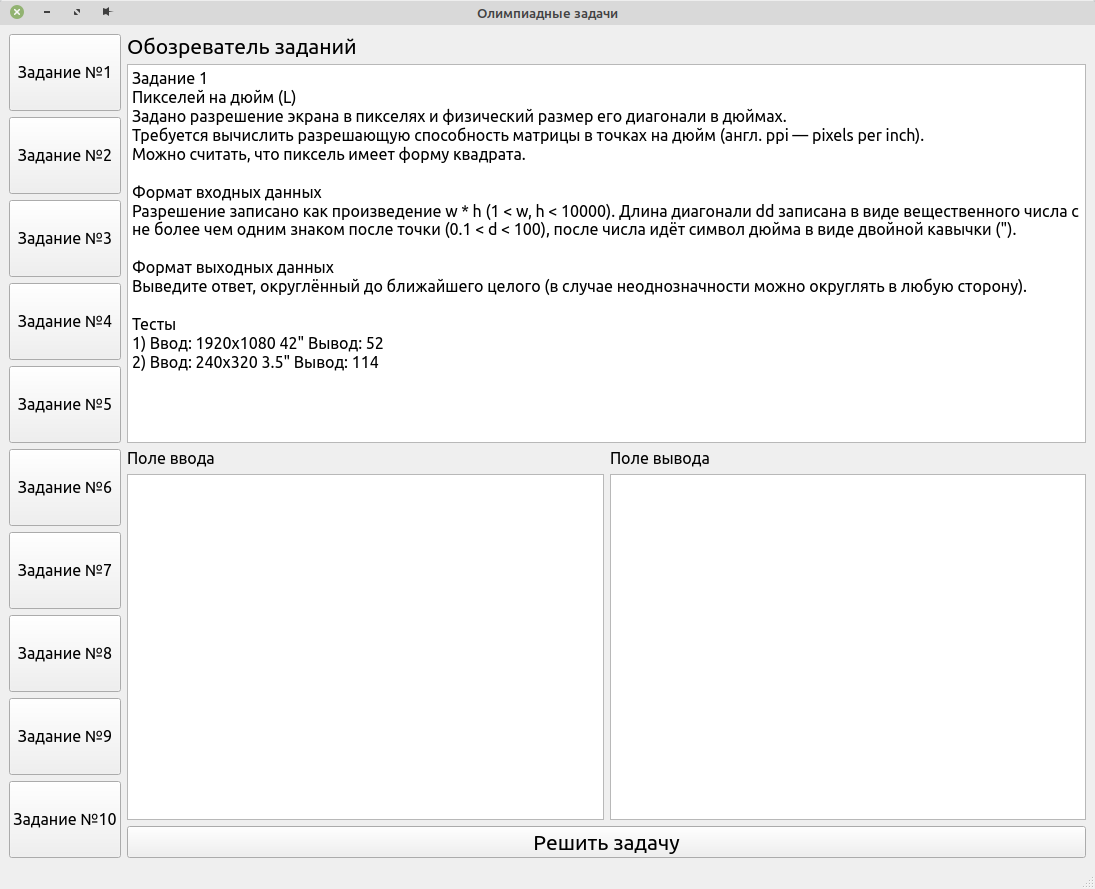
Рисунок 24 - файл исполняемой программы

Пятый компонент - непосредственно сам текстовый файл (Рисунок 25).

Здесь содержатся условия всех задач.

Рисунок 25 - файл исполняемой программы

Графическое меню для взаимодействия с пользователем на рисунке 26

Рисунок 26 - графическое меню для взаимодействия с пользователем

ЗАКЛЮЧЕНИЕ

Результаты выполнения работы.

Изучены и использованы алгоритмы, которые применяют для решения олимпиадных задач. Получены практические навыки работы с графическим интерфейсом, а также навыки реализации взаимодействия модулей программного комплекса, написанного на языке *python*.

Исследованы методы взаимодействия различных алгоритмов и структур данных. Исследованы зависимости между используемым алгоритмом для решения той или иной задачи и эффективностью её выполнения. Подобраны максимально эффективные алгоритмы.

Программа может применяться для тестирования решений олимпиадных заданий, то есть для их верификации.

Возможности модификации заключаются в следующем:

* Добавление новых задач для тестирования
* Создание веб сервиса из разработанного приложения
* Улучшение структуры программного комплекса
* Добавление возможности пользователя проверять свои собственные решения

Полнота решений задач полностью соответствует заданию курсовой работы. Все задачи решены и отлажены. Также разработан графический интерфейс программы.

Результаты выполнения программного комплекса проверены тестовой системой iRunner, которая применяется для тестирования задач на всех этапах международной олимпиады по программированию среди студентов *ICPC*. Все тесты пройдены на 100%.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Андреев А. Е. Дискретная математика: прикладные задачи и сложность алгоритмов : учебник и практикум для академического бакалавриата / А. Е. Андреев, А. А. Болотов, К. В. Коляда, А. Б. Фролов. — 2-е изд., испр. и доп. — Москва : Издательство Юрайт, 2019. — 317 с.
2. Аверина, Т. А. Численные методы. Верификация алгоритмов решения систем со случайной структурой : учебное пособие для вузов / Т. А. Аверина. — Москва : Издательство Юрайт, 2019. — 179 с.
3. Игошин, В.И. Теория алгоритмов: Учебное пособие / В.И. Игошин. - М.: ИНФРА-М, 2013. - 318 c.
4. Канцедал, С.А. Алгоритмизация и программирование : Учебное пособие / С.А. Канцедал. - М.: ИД ФОРУМ, НИЦ ИНФРА-М, 2013. - 352 c.
5. Крупский, В.Н. Математическая логика и теория алгоритмов: Учебное пособие для студентов учреждений высшего проф. образования / В.Н. Крупский, В.Е. Плиско. - М.: ИЦ Академия, 2013. - 416 c.
6. Семакин, И.Г. Основы алгоритмизации и программирования. Практикум: Учебное пос. для студ. учреждений сред. проф. образования / И.Г. Семакин, А.П. Шестаков. - М.: ИЦ Академия, 2013. - 144 c.
7. Судоплатов, С. В. Математика: математическая логика и теория алгоритмов : учебник и практикум для среднего профессионального образования / С. В. Судоплатов, Е. В. Овчинникова. — 5-е изд., стер. — Москва : Издательство Юрайт, 2019. — 255 с.
8. Семакин, И.Г. Основы алгоритмизации и программирования: Учебник для студ. учреждений сред. проф. образования / И.Г. Семакин, А.П. Шестаков. - М.: ИЦ Академия, 2013. - 304 c.
9. Трофимов, В. В. Алгоритмизация и программирование : учебник для академического бакалавриата / В. В. Трофимов, Т. А. Павловская ; под редакцией В. В. Трофимова. — Москва : Издательство Юрайт, 2019. — 137 с.
10. Черняк А. А. Методы оптимизации: теория и алгоритмы : учебное пособие для академического бакалавриата / А. А. Черняк, Ж. А. Черняк, Ю. М. Метельский, С. А. Богданович. — 2-е изд., испр. и доп. — Москва : Издательство Юрайт, 2019. — 357 с.

Приложение А Листинг исполняемого файла

import sys

from menu import \*

import textmodule

import tasks\_solving

from PyQt5 import QtCore, QtGui, QtWidgets

class MyWin(QtWidgets.QMainWindow):

def \_\_init\_\_(self, parent = None):

QtWidgets.QWidget.\_\_init\_\_(self, parent)

self.ui = Ui\_MainWindow()

self.ui.setupUi(self)

# Кнопки

self.ui.task1\_button.clicked.connect(self.task1\_button\_clicked)

self.ui.task2\_button.clicked.connect(self.task2\_button\_clicked)

self.ui.task3\_button.clicked.connect(self.task3\_button\_clicked)

self.ui.task4\_button.clicked.connect(self.task4\_button\_clicked)

self.ui.task5\_button.clicked.connect(self.task5\_button\_clicked)

self.ui.task6\_button.clicked.connect(self.task6\_button\_clicked)

self.ui.task7\_button.clicked.connect(self.task7\_button\_clicked)

self.ui.task8\_button.clicked.connect(self.task8\_button\_clicked)

self.ui.task9\_button.clicked.connect(self.task9\_button\_clicked)

self.ui.task10\_button.clicked.connect(self.task10\_button\_clicked)

#Решение задания

self.ui.solve\_button.clicked.connect(self.solve\_task)

# Функции при нажатии на кнопки

def task1\_button\_clicked(self):

self.ui.tasks\_data\_text.clear()

for item in textmodule.find\_task\_data(1).values():

self.ui.tasks\_data\_text.appendPlainText(item)

def task2\_button\_clicked(self):

self.ui.tasks\_data\_text.clear()

for item in textmodule.find\_task\_data(2).values():

self.ui.tasks\_data\_text.appendPlainText(item)

def task3\_button\_clicked(self):

self.ui.tasks\_data\_text.clear()

for item in textmodule.find\_task\_data(3).values():

self.ui.tasks\_data\_text.appendPlainText(item)

def task4\_button\_clicked(self):

self.ui.tasks\_data\_text.clear()

for item in textmodule.find\_task\_data(4).values():

self.ui.tasks\_data\_text.appendPlainText(item)

def task5\_button\_clicked(self):

self.ui.tasks\_data\_text.clear()

for item in textmodule.find\_task\_data(5).values():

self.ui.tasks\_data\_text.appendPlainText(item)

def task6\_button\_clicked(self):

self.ui.tasks\_data\_text.clear()

for item in textmodule.find\_task\_data(6).values():

self.ui.tasks\_data\_text.appendPlainText(item)

def task7\_button\_clicked(self):

self.ui.tasks\_data\_text.clear()

for item in textmodule.find\_task\_data(7).values():

self.ui.tasks\_data\_text.appendPlainText(item)

def task8\_button\_clicked(self):

self.ui.tasks\_data\_text.clear()

for item in textmodule.find\_task\_data(8).values():

self.ui.tasks\_data\_text.appendPlainText(item)

def task9\_button\_clicked(self):

self.ui.tasks\_data\_text.clear()

for item in textmodule.find\_task\_data(9).values():

self.ui.tasks\_data\_text.appendPlainText(item)

def task10\_button\_clicked(self):

self.ui.tasks\_data\_text.clear()

for item in textmodule.find\_task\_data(10).values():

self.ui.tasks\_data\_text.appendPlainText(item)

#Функция для решения задания

def solve\_task(self):

tasks = {"Задание 1": tasks\_solving.task1\_ppi, "Задание 2": tasks\_solving.task2\_roads,

"Задание 3": tasks\_solving.task3\_puzzle, "Задание 4": tasks\_solving.task4\_median,

"Задание 5": tasks\_solving.task5\_progression, "Задание 6": tasks\_solving.task6\_polygon,

"Задание 7": None, "Задание 8": tasks\_solving.task8\_egypt\_approx,

"Задание 9": tasks\_solving.task9\_disconnect, "Задание 10": tasks\_solving.task10\_chess\_queen}

try:

inp = self.ui.input\_text.toPlainText()

"""Вызывают функцию из словаря по имени исполняемого задания"""

self.ui.output\_text.setPlainText(str(tasks[self.ui.tasks\_data\_text.toPlainText().split('\n')[0]](inp)))

except:

self.ui.statusbar.showMessage("Неверный ввод")

def main():

app = QtWidgets.QApplication(sys.argv)

myapp = MyWin()

myapp.show()

sys.exit(app.exec\_())

if \_\_name\_\_ == "\_\_main\_\_":

main()

Приложение Б Листинг модуля графического меню

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui\_MainWindow(object):

def setupUi(self, MainWindow):

MainWindow.setObjectName("MainWindow")

MainWindow.setWindowModality(QtCore.Qt.NonModal)

MainWindow.setEnabled(True)

MainWindow.resize(1095, 864)

MainWindow.setMouseTracking(False)

icon = QtGui.QIcon()

icon.addPixmap(QtGui.QPixmap("../../../../../Загрузки/delicious\_logo\_icon\_124323.ico"), QtGui.QIcon.Normal, QtGui.QIcon.Off)

MainWindow.setWindowIcon(icon)

self.centralwidget = QtWidgets.QWidget(MainWindow)

self.centralwidget.setObjectName("centralwidget")

self.horizontalLayout = QtWidgets.QHBoxLayout(self.centralwidget)

self.horizontalLayout.setObjectName("horizontalLayout")

self.menu = QtWidgets.QVBoxLayout()

self.menu.setSpacing(6)

self.menu.setObjectName("menu")

self.task1\_button = QtWidgets.QPushButton(self.centralwidget)

sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Expanding)

sizePolicy.setHorizontalStretch(0)

sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.task1\_button.sizePolicy().hasHeightForWidth())

self.task1\_button.setSizePolicy(sizePolicy)

font = QtGui.QFont()

font.setPointSize(12)

self.task1\_button.setFont(font)

self.task1\_button.setContextMenuPolicy(QtCore.Qt.DefaultContextMenu)

self.task1\_button.setObjectName("task1\_button")

self.menu.addWidget(self.task1\_button)

self.task2\_button = QtWidgets.QPushButton(self.centralwidget)

sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Expanding)

sizePolicy.setHorizontalStretch(0)

sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.task2\_button.sizePolicy().hasHeightForWidth())

self.task2\_button.setSizePolicy(sizePolicy)

font = QtGui.QFont()

font.setPointSize(12)

self.task2\_button.setFont(font)

self.task2\_button.setContextMenuPolicy(QtCore.Qt.DefaultContextMenu)

self.task2\_button.setObjectName("task2\_button")

self.menu.addWidget(self.task2\_button)

self.task3\_button = QtWidgets.QPushButton(self.centralwidget)

sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Expanding)

sizePolicy.setHorizontalStretch(0)

sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.task3\_button.sizePolicy().hasHeightForWidth())

self.task3\_button.setSizePolicy(sizePolicy)

font = QtGui.QFont()

font.setPointSize(12)

self.task3\_button.setFont(font)

self.task3\_button.setContextMenuPolicy(QtCore.Qt.DefaultContextMenu)

self.task3\_button.setObjectName("task3\_button")

self.menu.addWidget(self.task3\_button)

self.task4\_button = QtWidgets.QPushButton(self.centralwidget)

sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Expanding)

sizePolicy.setHorizontalStretch(0)

sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.task4\_button.sizePolicy().hasHeightForWidth())

self.task4\_button.setSizePolicy(sizePolicy)

font = QtGui.QFont()

font.setPointSize(12)

self.task4\_button.setFont(font)

self.task4\_button.setContextMenuPolicy(QtCore.Qt.DefaultContextMenu)

self.task4\_button.setObjectName("task4\_button")

self.menu.addWidget(self.task4\_button)

self.task5\_button = QtWidgets.QPushButton(self.centralwidget)

sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Expanding)

sizePolicy.setHorizontalStretch(0)

sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.task5\_button.sizePolicy().hasHeightForWidth())

self.task5\_button.setSizePolicy(sizePolicy)

font = QtGui.QFont()

font.setPointSize(12)

self.task5\_button.setFont(font)

self.task5\_button.setContextMenuPolicy(QtCore.Qt.DefaultContextMenu)

self.task5\_button.setObjectName("task5\_button")

self.menu.addWidget(self.task5\_button)

self.task6\_button = QtWidgets.QPushButton(self.centralwidget)

sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Expanding)

sizePolicy.setHorizontalStretch(0)

sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.task6\_button.sizePolicy().hasHeightForWidth())

self.task6\_button.setSizePolicy(sizePolicy)

font = QtGui.QFont()

font.setPointSize(12)

self.task6\_button.setFont(font)

self.task6\_button.setContextMenuPolicy(QtCore.Qt.DefaultContextMenu)

self.task6\_button.setObjectName("task6\_button")

self.menu.addWidget(self.task6\_button)

self.task7\_button = QtWidgets.QPushButton(self.centralwidget)

sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Expanding)

sizePolicy.setHorizontalStretch(0)

sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.task7\_button.sizePolicy().hasHeightForWidth())

self.task7\_button.setSizePolicy(sizePolicy)

font = QtGui.QFont()

font.setPointSize(12)

self.task7\_button.setFont(font)

self.task7\_button.setContextMenuPolicy(QtCore.Qt.DefaultContextMenu)

self.task7\_button.setObjectName("task7\_button")

self.menu.addWidget(self.task7\_button)

self.task8\_button = QtWidgets.QPushButton(self.centralwidget)

sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Expanding)

sizePolicy.setHorizontalStretch(0)

sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.task8\_button.sizePolicy().hasHeightForWidth())

self.task8\_button.setSizePolicy(sizePolicy)

font = QtGui.QFont()

font.setPointSize(12)

self.task8\_button.setFont(font)

self.task8\_button.setContextMenuPolicy(QtCore.Qt.DefaultContextMenu)

self.task8\_button.setObjectName("task8\_button")

self.menu.addWidget(self.task8\_button)

self.task9\_button = QtWidgets.QPushButton(self.centralwidget)

sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Expanding)

sizePolicy.setHorizontalStretch(0)

sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.task9\_button.sizePolicy().hasHeightForWidth())

self.task9\_button.setSizePolicy(sizePolicy)

font = QtGui.QFont()

font.setPointSize(12)

self.task9\_button.setFont(font)

self.task9\_button.setContextMenuPolicy(QtCore.Qt.DefaultContextMenu)

self.task9\_button.setObjectName("task9\_button")

self.menu.addWidget(self.task9\_button)

self.task10\_button = QtWidgets.QPushButton(self.centralwidget)

sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Expanding)

sizePolicy.setHorizontalStretch(0)

sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.task10\_button.sizePolicy().hasHeightForWidth())

self.task10\_button.setSizePolicy(sizePolicy)

font = QtGui.QFont()

font.setPointSize(12)

self.task10\_button.setFont(font)

self.task10\_button.setContextMenuPolicy(QtCore.Qt.DefaultContextMenu)

self.task10\_button.setObjectName("task10\_button")

self.menu.addWidget(self.task10\_button)

self.horizontalLayout.addLayout(self.menu)

self.user\_interface = QtWidgets.QVBoxLayout()

self.user\_interface.setObjectName("user\_interface")

self.tasks = QtWidgets.QVBoxLayout()

self.tasks.setObjectName("tasks")

self.tasks\_data\_label = QtWidgets.QLabel(self.centralwidget)

font = QtGui.QFont()

font.setPointSize(16)

self.tasks\_data\_label.setFont(font)

self.tasks\_data\_label.setContextMenuPolicy(QtCore.Qt.DefaultContextMenu)

self.tasks\_data\_label.setObjectName("tasks\_data\_label")

self.tasks.addWidget(self.tasks\_data\_label)

self.tasks\_data\_text = QtWidgets.QPlainTextEdit(self.centralwidget)

sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Expanding)

sizePolicy.setHorizontalStretch(0)

sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.tasks\_data\_text.sizePolicy().hasHeightForWidth())

self.tasks\_data\_text.setSizePolicy(sizePolicy)

font = QtGui.QFont()

font.setPointSize(12)

font.setItalic(False)

self.tasks\_data\_text.setFont(font)

self.tasks\_data\_text.setReadOnly(True)

self.tasks\_data\_text.setObjectName("tasks\_data\_text")

self.tasks.addWidget(self.tasks\_data\_text)

self.user\_interface.addLayout(self.tasks)

self.solve\_field = QtWidgets.QVBoxLayout()

self.solve\_field.setObjectName("solve\_field")

self.inout = QtWidgets.QHBoxLayout()

self.inout.setObjectName("inout")

self.input = QtWidgets.QVBoxLayout()

self.input.setObjectName("input")

self.input\_label = QtWidgets.QLabel(self.centralwidget)

font = QtGui.QFont()

font.setPointSize(12)

font.setItalic(False)

self.input\_label.setFont(font)

self.input\_label.setObjectName("input\_label")

self.input.addWidget(self.input\_label)

self.input\_text = QtWidgets.QPlainTextEdit(self.centralwidget)

font = QtGui.QFont()

font.setPointSize(12)

self.input\_text.setFont(font)

self.input\_text.setObjectName("input\_text")

self.input.addWidget(self.input\_text)

self.inout.addLayout(self.input)

self.output = QtWidgets.QVBoxLayout()

self.output.setObjectName("output")

self.output\_label = QtWidgets.QLabel(self.centralwidget)

font = QtGui.QFont()

font.setPointSize(12)

self.output\_label.setFont(font)

self.output\_label.setObjectName("output\_label")

self.output.addWidget(self.output\_label)

self.output\_text = QtWidgets.QPlainTextEdit(self.centralwidget)

font = QtGui.QFont()

font.setPointSize(12)

self.output\_text.setFont(font)

self.output\_text.setReadOnly(True)

self.output\_text.setObjectName("output\_text")

self.output.addWidget(self.output\_text)

self.inout.addLayout(self.output)

self.solve\_field.addLayout(self.inout)

self.solve\_button = QtWidgets.QPushButton(self.centralwidget)

sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Fixed)

sizePolicy.setHorizontalStretch(0)

sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.solve\_button.sizePolicy().hasHeightForWidth())

self.solve\_button.setSizePolicy(sizePolicy)

font = QtGui.QFont()

font.setPointSize(16)

font.setBold(False)

font.setWeight(50)

self.solve\_button.setFont(font)

self.solve\_button.setObjectName("solve\_button")

self.solve\_field.addWidget(self.solve\_button)

self.user\_interface.addLayout(self.solve\_field)

self.horizontalLayout.addLayout(self.user\_interface)

MainWindow.setCentralWidget(self.centralwidget)

self.statusbar = QtWidgets.QStatusBar(MainWindow)

self.statusbar.setObjectName("statusbar")

MainWindow.setStatusBar(self.statusbar)

self.retranslateUi(MainWindow)

QtCore.QMetaObject.connectSlotsByName(MainWindow)

def retranslateUi(self, MainWindow):

\_translate = QtCore.QCoreApplication.translate

MainWindow.setWindowTitle(\_translate("MainWindow", "Олимпиадные задачи"))

self.task1\_button.setText(\_translate("MainWindow", "Задание №1"))

self.task2\_button.setText(\_translate("MainWindow", "Задание №2"))

self.task3\_button.setText(\_translate("MainWindow", "Задание №3"))

self.task4\_button.setText(\_translate("MainWindow", "Задание №4"))

self.task5\_button.setText(\_translate("MainWindow", "Задание №5"))

self.task6\_button.setText(\_translate("MainWindow", "Задание №6"))

self.task7\_button.setText(\_translate("MainWindow", "Задание №7"))

self.task8\_button.setText(\_translate("MainWindow", "Задание №8"))

self.task9\_button.setText(\_translate("MainWindow", "Задание №9"))

self.task10\_button.setText(\_translate("MainWindow", "Задание №10"))

self.tasks\_data\_label.setText(\_translate("MainWindow", "Обозреватель заданий"))

self.input\_label.setText(\_translate("MainWindow", "Поле ввода"))

self.output\_label.setText(\_translate("MainWindow", "Поле вывода"))

self.solve\_button.setText(\_translate("MainWindow", "Решить задачу"))

Приложение В Листинг модуля решений задач

import math

# First task Task L - PPI

def task1\_ppi(string):

raz, inch = string.split(' ')

raz1, raz2 = raz.split('x')

square = (int(raz1) \*\* 2 + int(raz2) \*\* 2) \*\* 0.5

return round(square / float(inch[:-1]))

# Second task Task K - Roads

def task2\_roads(inp):

t, r = (int(i) for i in inp.split('\n')[0].split())

connections = {key: [] for key in range(1, t + 1)}

order = []

for i in range(r):

t1, t2 = (int(town) for town in inp.split('\n')[i + 1].split())

connections[t1].append(t2)

connections[t2].append(t1)

order.append((t1, t2))

def connect\_func(town, conset): # Проверяет, соединён ли город со всеми городами

flag = True

for i in connections[town]:

if i not in conset:

flag = False

break

if flag:

return conset

else:

for i in connections[town]:

if i not in conset:

conset.add(i)

conset.union(connect\_func(i, conset))

return conset

road\_number = 1

for town1, town2 in order:

connections[town1].remove(town2)

connections[town2].remove(town1)

if len(connect\_func(1, set())) != t:

return road\_number

else:

road\_number += 1

# Third task Task J - Puzzle

def task3\_puzzle(inp):

k, n = [int(i) for i in inp.split()]

if n < k or n <= 0 or k <= 0:

return 'Epic fail'

if n % k == 0:

return '{} '.format(n // k) \* (3 \* k - 3)

lst = [n // k] \* (3 \* k - 3)

if n % k == 2:

lst[0] += 1

lst[k - 1] += 1

lst[2 \* k - 2] += 1

elif n % k == k - 1:

for i in range(len(lst)):

lst[i] += 1

lst[1] -= 1

lst[k] -= 1

try:

lst[2 \* k - 1] -= 1

except:

return 'Epic fail'

else:

for i in range(n % k):

lst[1 + i] += 1

lst[k + i] += 1

lst[2 \* k - 1 + i] += 1

lst = [str(item) for item in lst]

return ' '.join(lst)

# Fourth task Task G - Median

def task4\_median(inp):

numbers = [int(number) for number in inp.split()]

median\_numbers = list()

output = []

for number in numbers:

if number == 0:

median\_numbers.sort()

if len(median\_numbers) % 2 == 1:

output.append(median\_numbers[len(median\_numbers) // 2])

continue

else:

output.append((median\_numbers[len(median\_numbers) // 2] + median\_numbers[len(median\_numbers) // 2 - 1]) / 2.0)

continue

else:

median\_numbers.append(number)

return "\n".join([str(number) for number in output])

# 5th task Task I - Progression

def task5\_progression(inp):

array = [int(number) for number in inp.split()[1:]] # массив чисел

mean = 0 # средняя разность массива чисел

for i in range(len(array) - 1):

mean += array[i + 1] - array[i]

mean = round(float(mean) / (len(array) - 1))

for i in range(len(array) - 1):

if abs((array[i + 1] - array[i]) - mean) > 2:

return "Epic fail"

# Вторая часть кода

first\_array = [array[0] - 1] + array[1:]

second\_array = array

third\_array = [array[0] + 1] + array[1:]

changed\_numbers = [None, None, None]

changed\_numbers[0] = 1 # количество измененных чисел при первом значении = буфер - 1

changed\_numbers[1] = 0 # количество измененных чисел при первом значении = буфер

changed\_numbers[2] = 1 # количество измененных чисел при первом значении = буфер + 1

for index, arr in enumerate([first\_array, second\_array, third\_array]):

for i in range(len(arr) - 1):

if mean - (arr[i + 1] - arr[i]) != 0:

if abs(mean - (arr[i + 1] - arr[i])) != 1:

changed\_numbers[index] = 10000000

break

else:

arr[i + 1] += mean - (arr[i + 1] - arr[i])

changed\_numbers[index] += 1

return str(min(changed\_numbers))

# 6th task Task H - Polygon

def task6\_polygon(inp):

number = int(inp)

if number == 1:

return "Epic fail"

for i in [2, 3, 5]:

if number % i == 0:

return ("1" \* (number // i) + "0" \* ((30 - number) // i)) \* i

if 30 % number == 0:

return ("1" + "0" \* (30 // number - 1)) \* number

return "Epic fail"

# 8th task Task E - Egypt Approximation

def task8\_egypt\_approx(inp):

a, b = (float(number) for number in inp.split())

aprox\_array = list()

first\_approx = math.ceil(1 / (a / b))

if (a / b) <= (1 / first\_approx):

return "1\n" + str(first\_approx)

else:

aprox\_array.append(first\_approx)

while((a / b) > sum([(1 / number) for number in aprox\_array])):

number\_to\_append = math.ceil(1 / (a / b - sum([(1 / number) for number in aprox\_array])))

aprox\_array.append(number\_to\_append)

return f"{len(aprox\_array)}\n" + " ".join([str(number) for number in aprox\_array])

# 9th task Task D - Disconnect

def task9\_disconnect(inp):

def connect\_func(vertex, conset, connections):

''' Проверяет, соединена ли вершина со всеми остальными вершинами

vertex - вершина, которую проверяем

conset - множество пройденных вершин

connections - словарь, содержащий все соединения (Например {1:[2, 3],2:[3, 1]})'''

flag = True

for i in connections[vertex]:

if i not in conset:

flag = False

break

if flag:

return conset

else:

for i in connections[vertex]:

if i not in conset:

conset.add(i)

conset.union(connect\_func(i, conset, connections))

return conset

result = []

# Ввод

tests = []

pointer = 1

inp\_list = inp.split('\n')

for ind in range(int(inp\_list[0])): # формирование массивов тестов

current\_test = inp\_list[pointer:pointer + int(inp\_list[pointer].split()[1]) + 1]

pointer = pointer + int(inp\_list[pointer].split()[1]) + 1

tests.append(current\_test)

# Начало алгоритма

for test in tests:

vertex\_count, edges\_count = map(int, test[0].split())

connections = {k:list() for k in range(1, vertex\_count + 1)}

edges = test[1:]

for edge in edges:

e1, e2 = map(int, edge.split())

connections[e1].append(e2)

connections[e2].append(e1)

edges.append("1 1") # для холостого прогона по рёбрам если развязывающее множество заканчивается последним ребром

count = -1

for edge in edges:

e1, e2 = map(int, edge.split())

if (len(connect\_func(1, set(), connections)) != vertex\_count):

if count <= 0:

result.append(str(count))

else:

result.append(str(count) + "\n" + " ".join([str(i) for i in range(1, count + 1)]))

break

else:

connections[e1].remove(e2)

connections[e2].remove(e1)

count += 1

return "\n".join(result)

# 10th task Task B - Chess Queen

def task10\_chess\_queen(inp):

k = int(inp)

dic = {1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e', 6: 'f', 7: 'g', 8: 'h'}

matrix = [[(i + 1, j + 1) for i in range(8)] for j in range(8)]

for row in matrix:

for item in row:

atk = 14

xt = item[0]

yt = item[1]

while xt - 1 > 0 and yt - 1 > 0:

atk += 1

xt -= 1

yt -= 1

xt = item[0]

yt = item[1]

while xt - 1 > 0 and yt + 1 < 9:

atk += 1

xt -= 1

yt += 1

xt = item[0]

yt = item[1]

while xt + 1 < 9 and yt - 1 > 0:

atk += 1

xt += 1

yt -= 1

xt = item[0]

yt = item[1]

while xt + 1 < 9 and yt + 1 < 9:

atk += 1

xt += 1

yt += 1

if atk == k:

return '{}{}'.format(dic[item[0]], 8 - item[1])

return 'Epic fail'

Приложение Г Листинг модуля для работы с текстовым файлом

def find\_task\_data(number\_of\_task):

task = {"Задание": "Задание {}\n".format(number\_of\_task), "Формат входных данных": "", "Формат выходных данных": "", "Тесты": ""}

with open("/home/kor/university/course2/term1/kpiyap/course\_work/Tasks.txt") as file:

for line in file:

if line == "Задание {}\n".format(number\_of\_task):

break

for line in file:

if line == "\n":

break

else:

task["Задание"] += line

for line in file:

if line == "\n":

break

else:

task["Формат входных данных"] += line

for line in file:

if line == "\n":

break

else:

task["Формат выходных данных"] += line

for line in file:

if line == "\n":

break

else:

task["Тесты"] += line

return task

Приложение Д Листинг текстового файла с условиями задач

Задание 1

Пикселей на дюйм (L)

Задано разрешение экрана в пикселях и физический размер его диагонали в дюймах.

Требуется вычислить разрешающую способность матрицы в точках на дюйм (англ. ppi — pixels per inch).

Можно считать, что пиксель имеет форму квадрата.

Формат входных данных

Разрешение записано как произведение w \* h (1 < w, h < 10000). Длина диагонали dd записана в виде вещественного числа с не более чем одним знаком после точки (0.1 < d < 100), после числа идёт символ дюйма в виде двойной кавычки (").

Формат выходных данных

Выведите ответ, округлённый до ближайшего целого (в случае неоднозначности можно округлять в любую сторону).

Тесты

1) Ввод: 1920x1080 42" Вывод: 52

2) Ввод: 240x320 3.5" Вывод: 114

Задание 2

Дороги (K)

В Байтландии n городов. Некоторые пары городов соединены между собой двусторонними дорогами.

Сеть дорог позволяет из любого города проехать в любой другой.

К сожалению, из-за недостатка финансирования дороги год за годом приходят в упадок и выводятся из эксплуатации.

Известен порядок, в котором будут закрываться все дороги.

Определите, в какой момент при закрытии очередной дороги жители Байтландии потеряют возможность проехать из какого-либо одного города в другой город.

Формат входных данных

В первой строке через пробел записаны числа n и m (1 < n < 100000, 1 < m < 300000) — число городов и число дорог соответственно.

В последующих m строках заданы все дороги Байтландии в хронологическом порядке их закрытия.

Дорога описывается парой чисел u и v (1 < u, v < n1, u != v). Между парой городов может быть построено несколько дорог.

Формат выходных данных

Выведите одно число — номер дороги, после закрытия которой дорожная сеть утратит связность.

Дороги нумеруются начиная с единицы в соответствии с порядком следования на входе.

Тесты

1)

Ввод:

3 3

1 2

2 3

3 1

Вывод: 2

2)

Ввод:

4 3

2 3

1 2

3 4

Вывод: 1

Задание 3

Шарики с числами (J)

Костя расположил в форме треугольника 3k-3 шарика, каждая сторона треугольника состоит ровно из k шариков.

На одной стороне треугольника находятся шарики с номерами 1, 2, …, k, на второй — шарики с номерами k, k+1, …, 2k-1, на третьей — шарики с номерами 2k-1, 2k, …, 3k-3, 1.

Косте нужно написать на шариках по одному целому числу из диапазона от 1 до 10^9 так, чтобы сумма чисел на шариках на каждой стороне треугольника была равна n и максимальное использованное число было как можно меньше.

При этом Костя может использовать одинаковые числа сколько угодно раз. Если нельзя написать числа на шариках таким образом, то подскажите ему об этом.

Формат входных данных

В единственной строке через пробел записаны два целых числа k и n (2 < k < 100, 1 < n < 10^9)

Формат выходных данных

Если подходящего способа написать числа на шариках нет, выведите Epic fail.

Иначе выведите 3k-3 целых числа, которые Костя может написать на шариках, в порядке их нумерации.

Если наборов чисел, удовлетворяющих условию задачи, несколько, выведите любой из них.

Тесты

1)Ввод: 3 7 Вывод: 2 3 2 3 2 3

2)Ввод: 4 1 Вывод: Epic fail

Задание 4

Скользящая медиана (G)

Медиана в математической статистике — число, характеризующее выборку (например, набор чисел).

Медиану можно найти, упорядочив элементы выборки по неубыванию и взяв средний элемент.

Например, выборка (11,9,3,5,5) после упорядочивания превращается в (3,5,5,9,11), и её медианой является число 5.

Если в выборке чётное число элементов, в качестве медианы будем использовать полусумму двух соседних значений (то есть медиану набора (1,3,5,7) считаем равной 4).

На вход поступают запросы двух видов: на добавление одного числа в выборку и на вычисление медианы. Нужно научиться эффективно их обрабатывать.

Формат входных данных

Целые числа (от 0 до 10^18)записаны в одну строку.

Если число положительное, то его нужно добавить в выборку.

Если число равно нулю, то нужно вычислить медиану выборки, которая была накоплена к данному моменту.

Общее количество чисел на входе не менее 11 и не превосходит 300000. Гарантируется, что первое число не равно нулю.

Формат выходных данных

Для каждого числа 0 на входе выведите в отдельной строке полученное значение медианы с абсолютной погрешностью не более 0,1.

Тесты

1)Ввод: 1 0 2 0 3 0 Вывод: 1 1.5 2

2)Ввод: 11 9 3 5 5 0 Вывод: 5

3)Ввод: 1 3 5 7 0 Вывод: 4

Задание 5

Прогрессия (I)

Задан массив A длины n. Требуется изменить его элементы так, чтобы они образовали арифметическую прогрессию.

Каждый элемент массива можно изменить не более одного раза: или увеличить на единицу, или уменьшить на единицу.

Какое наименьшее число элементов нужно модифицировать?

Формат входных данных

В первой строке задано целое число n (1 < n < 10^5).

Во второй строке через пробел записано n целых чисел ai(1 < ai < 10^9)

Формат выходных данных

Выведите одно число — минимально возможное число операций.

Если получить арифметическую прогрессию указанным способом невозможно, выведите Epic fail.

Тесты

1)

Ввод:

2

300 300

Вывод: 0

2)

Ввод:

5

1 3 6 9 12

Вывод: 1

3)

Ввод:

5

1 3 6 3 6

Вывод: Epic Fail

Задание 6

Тридцатиугольник (H)

Вершины правильного тридцатиугольника пронумерованы числами от 1 до 30 против часовой стрелки.

Выберите k вершин таким образом, чтобы центр масс этих точек совпадал с центром тридцатиугольника.

Формат входных данных

В единственной строке входных данных записано одно целое число k(1 < k < 30)

Формат выходных данных

Если подходящего набора вершин не существует, выведите строку Epic fail.

Иначе выведите строку из 30 символов. i-й символ должен быть равен `1', если i-я вершина была выбрана, или `0' в противном случае.

Если подходящих наборов вершин несколько, выведите любой из них.

Тесты

1)Ввод: 9 Вывод: 111000000011100000001110000000

Задание 7

Уравнение (F)

Дано n положительных целых чисел a1, a2, ..., an.

Найдите число решений уравнения a1x1 + a2x2 + ... + anxn = m в положительных целых числах x1, x2, ..., xn.

Так как искомое число решений может быть слишком большим, найдите его остаток от деления на 998244353.

Формат входных данных

В первой строке через пробел записаны два целых числа n и m (1 < n < 100, 0 < m < 2^63 - 1).

Во второй строке записаны n целых положительных чисел a1, a2, ...,an(sum(ai) < 100).

Формат выходных данных

Выведите остаток от деления искомого числа решений уравнения на 998244353.

Тесты

1)

Ввод:

4 21

1 2 3 4

Вывод:

27

2)

Ввод:

4 2019

21 9 20 19

Вывод:

18142

3)

Ввод:

5 6

1 1 1 1 1

Вывод:

5

Задание 8

Приближение дроби (E)

Египетские математики в 1800 году до н.э. записывали рациональные числа от 0 до 1 как сумму нескольких обратных различным целым. Например число 2/5 они могли записать как 1/3 + 1/15 или 1/4 + 1/10 + 1/20, но не как 1/5 + 1/5.

Вам необходимо приблизить дробь a/b суммой нескольких обратных различным целым. Ваше приближение x = sum(1 / ni) должно удовлетворять неравенству (a / b) < x < ((a + 1) / b). В своём приближении вы можете использовать не более 1000 слагаемых, все используемые числа в разложении не должны превосходить 10^18. Гарантируется, что разложение, удовлетворяющее данным требованиям, всегда существует.

Формат входных данных

Единственная строка содержит два разделённых пробелом целых числа a и b (1 < a < b < 10^13).

Гарантируется, что дробь a/b является несократимой.

Формат выходных данных

В первой строке выведите целое число k (1 < k < 1000) - количество слагаемых.

Во второй строке выведите k попарно различных целых чисел от 1 до 10^18 - знаменатели слагаемых, дающих в сумме приближения числа a/b.

Если число a / b можно приблизить несколькими различными способами, выведите любой из них.

Тесты

1)

Ввод: 2 5

Вывод:

2

3 15

2)

Ввод: 1 1000000000000

Вывод:

1

1000000000000

3)

Ввод: 3 7

Вывод:

3

5 6 7

Задание 9

Развязывающее множество (D)

Ориентированный граф G = (V, E) называется сильно связным, если из любой его вершины a ∈ V можно попасть в любую вершину b ∈ V, следуя вдоль рёбер графа из множества E.

Будем называть множество E' развязывающим множеством графа G, если E' ⊆ E и граф G' = (V, E \ E') не является сильно связным.

Иными словами, если после удаления всех рёбер из множества E' граф G больше не сильно связен, то множество E является для него развязывающим.

По заданному графу G найдите его развязывающее множество наименьшей мощности либо определите, что такого множества нет.

Формат входных данных

В первой строке записано целое число T (1 < T < 1000) - количество тестовых примеров.

Каждый тестовый пример описывается m + 1 строкой. В первой строке через пробел записаны два целых числа n и m (1 < n < 100, 0 < m < n(n - 1)) - количество вершин и количество ребер графа соответственно. Далее в каждой из m строк находится по 2 целых числа ai, bi (1 < ai, bi < n, ai != bi) - соответственно начало и конец i-го ориентированного ребра.

Рёбра нумеруются от 1 до m в том порядке, в котором они поступают на вход. Гарантируется, что в каждом тестовом примере никакая упорядоченная пара (ai, bi) не появляется дважды. Также гарантируется, что в каждом тесте находится не более чем 5 графов с n > 50, не более чем 20 графов с n > 20 и не более чем 100 графов с n > 10.

Формат выходных данных

Для каждого тестового примера выведите ответ на задачу.

Если развязывающего множества для графа не существует, выведите в строке число -1.

В противном случае выведите в первой строке целое число k (0 < k < m) — наименьшая возможная мощность развязывающего множества. Если k > 0, то во второй строке выведите через пробел k номеров рёбер, входящих в искомое множество. Номера рёбер должны следовать по возрастанию.

Если есть несколько ответов, выведите любой из них.

Тесты

1)

Ввод:

3

4 3

1 3

2 3

2 4

5 10

1 2

1 3

2 3

2 4

3 4

3 5

4 1

4 5

5 1

5 2

2 2

1 2

2 1

Вывод:

0

5

1 2 3 4 5

1

1

Задание 10

Ферзь атакует (B)

Ферзь может перемещаться на любое число свободных полей в любом направлении по прямой (по горизонтали, по вертикали и по диагонали), совмещая в себе возможности ладьи и слона.

Найдите на шахматной доске позицию, из которой ферзь атакует ровно k полей.

Формат входных данных

В единственной строке записано одно целое число k (1 < k < 63)

Формат выходных данных

Если подходящей позиции на шахматной доске нет, то выведите Epic fail. Иначе выведите позицию ферзя, находясь в которой, он атакует k других позиций шахматной доски. Сначала выведите обозначение вертикали — символ от a до h, затем номер горизонтали — число от 1 до 8. Если решений несколько, выведите любое.

Тесты

1)Ввод: 27 Вывод: d4