Разбор задач

A. Квадратик Рубика

Данная задача могла быть решена, например, с применением алгоритма поиска в ширину на графе (такой математический объект). При таком подходе можно было найти не просто, как собрать «Квадратик «Рубика», но и найти кратчайший способ сделать это.

Вершинами графа при такой реализации были бы состояния квадратика между элементарными действиями (действиями поворота), а рёбрами сами действия (их восемь по условию задачи).

Уникальный идентификатор вершины можно было получить построением уникальных строк, однозначно определяющим состояние головоломки. Например, самый левый квадрат на рисунке мог быть закодирован строкой «123894765», центральный – «312894765» и правый «712394865», то есть выписыванием символов из матрицы слева направо и сверху вниз.

Переходы из одного состояния в другие возможные, при реализации на графе алгоритма поиска в ширину, можно получать перебором всех возможных действий над текущим состоянием (их восемь – элементарные действия). Для каждого состояния можно дополнительно запоминать, с помощью какого последнего элементарного действия пришли в это состояние. Дважды в одно и то же состояние не входить.

Если описанные действия начать из состояния собранного «Квадратика «Рубика», то после окончания работы алгоритма в ассоциативном массиве (где могли храниться все посещённые состояния) будет и позиция, данная в задаче (тестах).

Останется только по сохранённым для каждой позиции элементарным переходам восстановить всю последовательность действий.

Можно было эту задачу решить и с помощью алгоритма поиска в глубину, для этого в силу простоты головоломки можно было предположить, что минимальное количество ходов, которые потребуются для сборки невелико, гораздо меньше 100. Например, для начала задаться глубиной поиска в 12 ходов, оценить, сколько времени уходит на поиск решения для разобранных случайным образом головоломок. Если время приемлемо, и всегда найден способ сборки, отправить решение на сервер. Получив ошибку – увеличить глубину поиска. Получив вердикт превышения времени исполнения подумать над оптимизациями.

B. Знаки на матрице

Эта задача в одном из своих наиболее простых вариантов решения потребует применения полиномиальных хешей. Хеш – это некоторая свёртка начальных данных, которая получается применением некоторой хеш-функции над начальными данными. Применение полиномиального хеша позволит эффективно, используя метод

динамического программирования, вычислить хеш для каждого прямоугольника матрицы или строки.

После получения хеша для каждого пикселя матрицы и получения одного хеша для знака можно с применением двух вложенных символов пройти по всему массиву поиска (матрица) и сравнивая хеши, отвергать те позиции, в которых хеши не совпали и только для других (где совпали) производить сравнение попиксельно.

Если хеш-функция будет выбрана «удачно», то хеши при несовпадении изображений будут совпадать редко, это уменьшит количество вычислений во много раз. Таким образом, решение данной задачи выполнялось за время порядка O(HW).

Кроме построения одного хеша, можно было обойтись несколькими более простыми, например, получить хеш для каждой подстроки длины w, тогда время работы оценивалось бы как O(HWh).

C. Сортировка по алфавиту

Во всех современных языках программирования имеется очень высокопроизводительная функция сортировки. Такие функции по умолчанию умеют сортировать строки в порядке «обычного» алфавита. Для решения задачи можно было подменить функцию сравнения, которую используют функции сортировки.

Можно было вместо подмены функции сравнения произвести во всех словах замену букв, произвести сортировку, и поменять буквы обратно.

D. Сортировка пузырьком

В данном разборе сложно придумать что рассказывать, здесь нет сложных алгоритмов и изощрённых структур данных, математических изысков и исследовательской работы. Эта задача на реализацию.

Дадим рекомендации, которым можно следовать при аналогичных задачах. Во-первых, прочитать условие задачи, выяснить, что нужно сделать, во-вторых, проверить правильно ли понимаете условие задачи, провести все действия над входными данными и получить тот же результат, что и в примере, если нет, вчитываться в условие задачи.

Если имеется подтверждение того, что вы понимаете задачу правильно (это может быть и не так, вы не понимаете условие, но думаете, что понимаете) и она на реализацию, то просто пункт за пунктом реализовать всё, что указано в задании (на этом этапе не следует заниматься оптимизацией).

После реализации, проверить решение на всех доступных входных данных, в том числе созданных самостоятельно, если программа выдаёт всегда ожидаемый вариант, то пора отправлять решение судьям на сервер. Если программа выдала что-то, что вы не ожидали, проверьте, где вы ошиблись, эта ошибка может быть как в решении, так и в тесте.

В данной задаче было уместным вынести вывод (печать) каждого этапа в отдельную функцию, тогда основной код стал бы легко читаем и реализуем.

E. Дата и время в хронологическом порядке

В этой задаче формат данных даты и время имеет одну особенность (не зря его часто применяют). Дело в том, что сравнение дат и сравнение строк, представляющих такую дату в указанном формате, приводит, с точки зрения упорядочивания, к одному и тому же результату. Таким образом, для упорядочивания дат не требовалось их распознавать и переводить в какой-то особый формат хранения.

Однако в задаче были строчки, которые не подходили под условие обрабатываемых дат, такие строчки следовало отсеять. Формат настолько жёстко задан, что достаточно сравнить длину строки с длиной правильно отформатированной, проверить на каких знакоместах стоят какие типы знаков, пробел только в одном месте, дефисы и двоеточия тоже на конкретных позициях в строке, на всех остальных должны быть цифры. И после этих проверок проверить может ли быть такое время и дата, подстрока представляющая год должны быть больше строки «1811» и меньше «2021», подстрока месяца больше «00» и меньше «13» и т.д.

Оставив все «нужные» строки, осталось произвести их сортировку (сортировку строк) и вывести результат.

Альтернативным вариантом может быть использование средств из стандартной библиотеки некоторых языков. Например для проверки формата даты можно воспользоваться регулярным выражением:

```
^\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}$
```

А для проверки корректности даты воспользоваться стандартными средствами работы с датами, предварительно убедившись, что они поддерживают данный интервал годов.

F. Новое умножение

В этой простой задаче можно вывести значение выражения, использовав его в том виде, как оно приведено в условии. Также можно раскрыть скобки и после всех преобразований получить:

```
A x B = A * B
```

E. Число из остатков

Так как 999983 и 1000003 взаимно простые числа, то любое целое число N на диапазоне [0, 999983 x 1000003) взаимно-однозначно соответствует кортежу (N mod 999983, N mod 1000003).

Самый простой способ, который не пройдёт по времени при указанных ограничениях с применением имеющегося сервера, это перебор всех целых чисел из диапазона [0, 999983 x 1000003) и проверкой получаемых остатков A = N mod 999983

и B = N mod 1000003 с данными на входе. Такой алгоритм потребует 999985999949 проверок (слишком много).

Однако если к A раз за разом прибавлять 999983, то получаемые числа всегда при делении на 999983 будут давать остаток равный A. Прибавлять меньшие величины нет необходимости, так как получаемые числа уже не будут давать при делении на 999983 остаток равный A, так что их можно благополучно пропустить.

Таких прибавлений (по 999983) потребуется не более 1000003, каждое получаемое число нужно будет сравнить по модулю с B, с таким объёмом вычислений современные компьютеры и проверяющий сервер легко справятся за отведённое время.

Для изучения менее вычислительно ёмких способов получения ответов на поставленную задачу рекомендуем ознакомиться с темами: Китайская теорема об остатках, нахождение обратных элементов в кольце по модулю, сравнение по модулю и ознакомиться с другими вопросами модульной арифметики.