Разбор задачи «Автобусы»

Прежде всего, определим, в каких случаях требуется конечное число автобусов. Построим ориентированный граф, вершинами которого будут города, а ребрами – рейсы автобусов. Чтобы автобусы не скапливались в городе, количество прибывающих и отъезжающих автобусов должно совпадать, поэтому в полученном графе количество входящих ребер должно совпадать с количеством исходящих ребер. Если это свойство не выполняется, значит автобусов бесконечное число.

С этого момента будет решать задачу в предположении, что количество автобусов конечно. Сначала разберем случай, когда отсутствуют рейсы, которые захватывают полночь. Изначально будем считать, что во всех городах общее количество автобусов равно нулю. Проследим за движением автобусов в течение первого дня. В тот момент, когда приходит время отправления автобуса по расписанию, мы отправляем любой из автобусов, который находится в городе. Если на данный момент в городе автобусы отсутствуют, добавляем новый автобус и увеличиваем ответ на единицу. Можно заметить, что количества автобусов утром и вечером совпадают, так как в течение дня из города уезжает столько же автобусов, сколько в него возвращается. Поэтому дополнительные автобусы для обеспечения движения по расписанию в последующие дни не потребуются.

Рассмотрим второй случай. Будем считать, что каждый из рейсов, ровно в полночь делает остановку в виртуальном N+1 городе. Тогда рейс, для которого время отправления превышает время прибытия, распадается на две части HH:MM – 24:00 и 00:00 – HH:MM. В результате такого преобразования задача сводится к предыдущему случаю. Другой подход заключается в том, что можно изначально на каждом рейсе, который захватывает полночь, запустить по автобусу и после этого свести задачу к предыдущему случаю.

Существует несколько способов реализовывать перечисленные выше идеи. При решении задачи можно отдельно не разбирать случай бесконечного количества автобусов и рейсы, которые захватывают полночь. Однако при этом появляется возможность допустить ошибку, которую в дальнейшем будет трудно найти и исправить.

Правильное решение задачи требует $O(N + M)$ памяти и работает за время $O(N + M \log M)$. Заведем специальный массив, в котором будем хранить текущее количество автобусов в городе. Каждый рейс преобразуем в два события: отправление и прибытие автобуса. Отсортируем все события по времени их возникновения. Если два события возникают одновременно, сначала будем рассматривать прибытие автобуса, а только потом отправление. Такая модификация позволить автобусу отправиться сразу же после прибытия. Будем просматривать события одно за другим. Если автобус прибывает в город, увеличиваем количество автобусов в этом городе на единицу, если отправляется – уменьшаем на единицу. Если при отправлении, в городе отсутствуют автобусы, увеличиваем ответ на единицу. Понятно, что такой алгоритм будет работать за линейное время, если не учитывать сортировку данных. Это решение набирает 100 баллов.

Можно непосредственно имитировать передвижение автобусов в течение дня, рассматривая дискретные моменты времени с интервалом в одну минуту. Тогда в каждый момент времени автобус либо находится в движении, либо простаивает. Если в указанный момент времени должен отправиться один из автобусов по расписанию, находим автобус, который простаиваем, либо заводим новый автобус. Такое решение на некоторых тестах не укладывается в ограничение по времени и набирает 70 баллов.

Еще одно решение основано на «склеивании» рейсов и выделении циклов в графе. Если есть два рейса из города A в город B и из города B в город C заменяем их одним рейсом из города A в город C, пересчитываем при этом продолжительность рейса. В общем случае через город B может проходить несколько рейсов. В этом случае чтобы общее количество автобусов было минимальным необходимо рейсы разбить на пары так, чтобы минимизировать суммарное время простоя. Продолжаем склейку до тех пор, пока все рейсы не превратятся в петли (циклы). Следует отметить, что в некоторых тестах правильное решение может в одном городе содержать несколько петель и склейка таких маршрутов приводит к ухудшению результата. Это идею можно довести до правильного решения, но трудоемкость этого перехода по сложности такая же, как правильное решение задачи. Решение, которое при склеивании перебирает все пары рейсов, проходящих через заданный город на некоторых тестах не укладывается в ограничения по времени и набирает 70 баллов.

Решение, которое правильно определяет случай бесконечного количества автобусов и проходит небольшие ручные тесты набирает 20 баллов. Решение, которое не учитывает рейсы,

захватывающие полночь набирает 40 баллов.

На последок стоит отметить, что решение задачи достаточно сложно отлаживать. Основная проблема в том, что не всегда возможно с первого взгляда найти правильный ответ к задаче. Это может казаться парадоксальным, но в некоторых случаях требуется автобусов больше, чем количество рейсов:

Входные данные	Выходные данные
1 2 1 10:00 2 18:00 2 17:00 1 11:00	3

В общем случае количество автобусов не превышает удвоенного количества рейсов, так как автобус не может простаивать более суток.

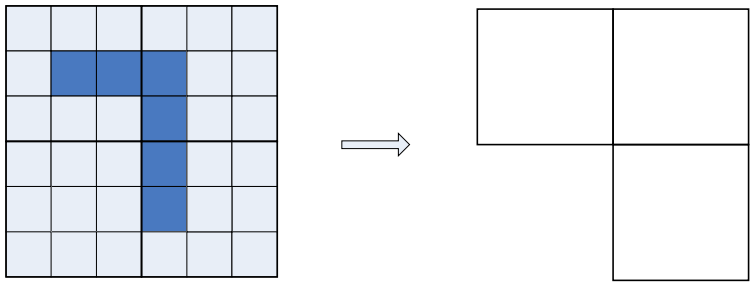
Разбор задачи «Полимино»

Пусть H_1, W_1 —размеры минимального охватывающего прямоугольника нового полимино, N — количество клеток в новом полимино, а H_2, W_2, M — размеры минимального охватывающего прямоугольника старого полимино и количество клеток в нем, соответственно.

Для того, чтобы найти число способов вырезать новое полимино из исходного, увеличенного в K раз, можно перебрать все возможные сдвиги одного относительно другого и проверить что фигуры, приложенные друг к другу таким образом совпадают. Под сдвигом одного полимино относительно другого будем понимать сдвиг фиксированной клетки одного полимино относительно фиксированной клетки другого. Однако, это неэффективное решение с асимптотикой $O(K^2MN)$, которое не укладывается в ограничение по времени.

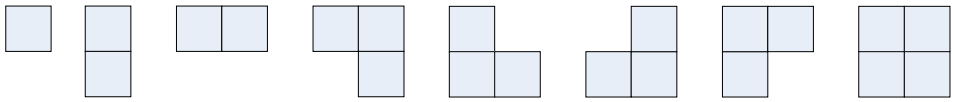
Рассмотрим K^2 возможных сдвигов нового полимино относительно одного из увеличенных квадратов исходного. Для каждого сдвига рассмотрим минимальную конфигурацию клеток, такую, что после увеличения в ней будет встречаться нужное полимино с текущим сдвигом. Назовем такую конфигурацию шаблоном.

Пример: полимино, сдвинутое относительно «клетки» 3 × 3 и шаблон для этого сдвига.

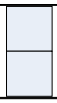


Теперь необходимо лишь проверить, сколько раз встречается этот шаблон в исходном полимино. Количество клеток в шаблоне есть $O(N / K)$, построить его можно за $O(N)$, найти число вхождений шаблона в исходное полимино можно за $O(NM / K)$. Поэтому асимптотика этого решения $O(NMK + K^2N)$

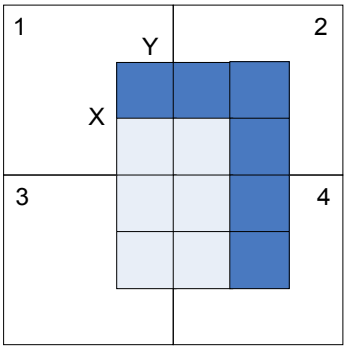
Однако при больших значениях K и это решение не укладывается в ограничение по времени. Нетрудно заметить, что при $K \geq \max(W_1, H_1)$ шаблон для любого сдвига содержит не более 4 клеток. То есть возможны следующие варианты шаблонов:



Для каждого из этих шаблонов за $O(M)$ можно подсчитать количество вхождений его в исходное полимино. Осталось лишь для каждой такой конфигурации найти количество сдвигов, при которых она станет шаблоном для нового полимино. Для первых трех типов шаблонов несложно выписать формулы:

Шаблон:			
Количество сдвигов:	$(K - H_1 + 1)(K - W_1 + 1)$	$(K - W_1 + 1)(H_1 - 1)$	$(K - H_1 + 1)(W_1 - 1)$

Теперь необходимо подсчитать число сдвигов, при которых остальные конфигурации клеток будут шаблонами. Для этого рассмотрим все возможные сдвиги (X, Y) прямоугольника, содержащего полимино, относительно «клетки» размера $K \times K$, так как показано на рисунке ниже.



Число всех возможных сдвигов есть $O(W_1H_1)$. Для каждого такого сдвига можно получить соответствующий шаблон. Для этого необходимо для каждого из четырех прямоугольников определить, есть ли в нем клетки полимино. Это можно сделать за $O(1)$, если предварительно для каждой клетки прямоугольника, содержащего полимино, определить есть ли клетки, находящиеся левее и выше данной, правее и выше, левее и ниже, правее и ниже данной (за $O(W_1H_1)$ для всех клеток прямоугольника). Решение, которое для каждого сдвига строит шаблон за $O(N)$ также укладывается в ограничение по времени.

Пусть A_i – количество сдвигов полимино относительно квадрата $K \times K$, таких, что i-я конфигурация клеток будет шаблоном для каждого из этих сдвигов. Пусть B_i — количество вхождений i-ого шаблона в исходное полимино. Тогда ответом будет сумма $A_i \times B_i$, для i от 1 до 8.

Таким образом, при $K < \max(H_1, W_1)$ асимптотика решения $O(NMK + K^2N)$,

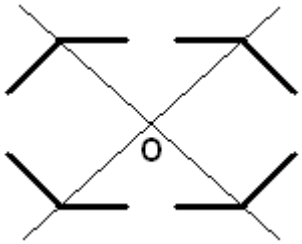
при $K \geq \max(H_1, W_1)$ — $O(W_1H_1 + M)$.

Разбор задачи «Треугольная реформа»

Участникам была предложена задача разрезать многоугольник на минимальное количество треугольников, разрезая только вдоль внутренних диагоналей.

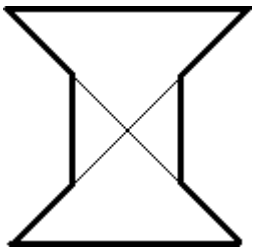
Для начала упомянем, что разрезая вдоль произвольных прямых линий, а не только вдоль диагоналей, невозможно уменьшить количество частей. Ограничение на разрезы было предложено лишь для устранения проблем, связанных с точностью вычислений.

Теперь исследуем, имеет ли смысл разрезать многоугольник вдоль двух пересекающихся внутренних диагоналей:



Предположим, что вдоль одной из диагоналей разрез уже сделан. Тогда мы имеем два отдельных многоугольника, для каждого из которых требуется решить поставленную задачу. Но ни в одном из них точка *O* не является вершиной, а значит, по приведенному выше утверждению, можно решить для них задачу, не проводя разрезы через точку *O*. Значит всегда можно ликвидировать такие пары пересекающихся разрезов.

Впрочем, это не значит, что такие пары разрезов строго увеличивают количество треугольников; вот один из контрпримеров:



Итак, каждая проведенная внутренняя диагональ делит наш многоугольник на два отдельных независимых многоугольника.

Теперь приведем следующие факты из геометрии.

Факт 1. В любом (простом) многоугольнике с более чем тремя вершинами найдется внутренняя диагональ.

Факт 2. Любой *N*-угольник можно разрезать на *N*-2 треугольника.

Построим на базе этих фактов следующий алгоритм.

Алгоритм разделения *N*-угольника на *N*-2 треугольника.

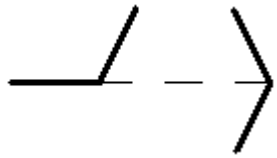
При *N*=3 треугольник сам является единственной искомой областью.

При *N*>3 найдем произвольную внутреннюю диагональ и разрежем многоугольник вдоль нее. Пусть одна из частей оказалась *M*-угольником. Тогда другая часть будет иметь *N*-*M*+2 вершин (убедитесь в этом!). Запустим наш алгоритм рекурсивно в обеих частях. В одной из них мы получим *M*-2 треугольника, а в другой — *N*-*M* треугольников. Итого имеем *N*-2 треугольника, что нам и требовалось.

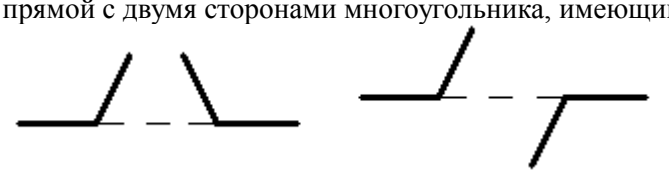
Однако результат работы этого алгоритма может оказаться неоптимальным. Тонкое место в рассуждении – это утверждение о количестве вершин в первой и второй частях. Улучшение может иметь место, если количество вершин в одной или в обеих частей окажется меньше. Это произойдет, если три или четыре вершины, идущие подряд в новом многоугольнике, окажутся на одной прямой. Тогда одну или две из них можно будет перестать рассматривать как вершину.

Это приводит нас к следующим важным определениям.

Определение Диагональю первого типа будем называть внутреннюю диагональ, лежащую на одной прямой ровно с одной стороной многоугольника, имеющей с ней общий конец.



Определение Диагональю второго типа будем называть внутреннюю диагональ, лежащую на одной прямой с двумя сторонами многоугольника, имеющими с ней общий конец.



Определение Диагонали обоих типов будем называть особенными диагоналями.

Таким образом, если в предыдущем алгоритме в какой-то момент выбрать не обычную внутреннюю диагональ, а особенную, то суммарное количество вершин в новых двух многоугольниках уменьшится на номер типа этой диагонали. А значит, на это же число уменьшится и количество треугольников в разбиении.

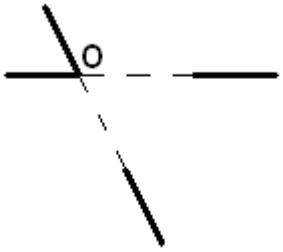
Теперь задача оптимизации видна: в процессе работы алгоритма требуется использовать множество диагоналей с максимальной суммой номеров типов. К сожалению, использовать абсолютно все особенные диагонали получается не всегда: например, две пересекающиеся особенные диагонали использовать вместе невыгодно.

Исследуем подробно все случаи взаимного расположения двух особенных диагоналей.

- Если диагонали пересекаются (строго пересекаются, а не касаются), то их не следует использовать вместе.
- Если диагонали не имеют общих точек, то их можно использовать вместе.

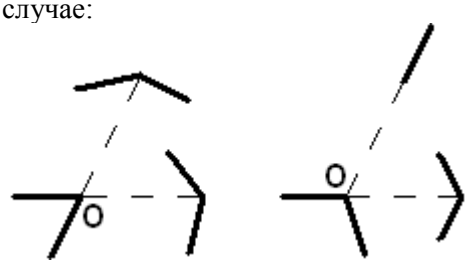
Особенная внимательность требуется в случае, когда две особенные диагонали выходят из одной вершины. В этом случае имеет значение их тип.

- Если это две диагонали второго типа, то их использовать вместе не следует:

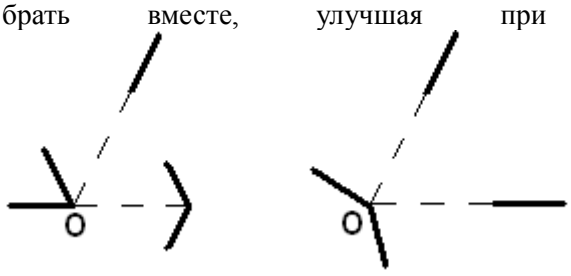


(после разрезания вдоль одной из них точка *O* перестает быть вершиной в новых многоугольниках)

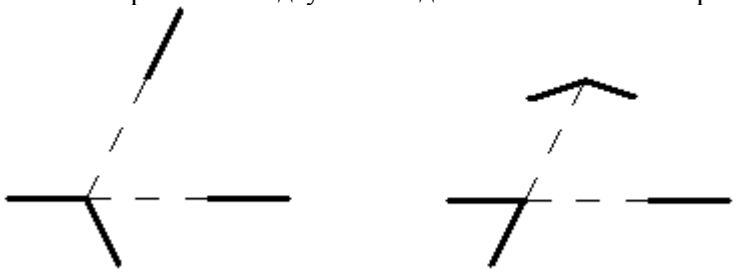
- Для двух диагоналей первого типа есть четыре различных случая. В первых двух случаях две диагонали нельзя брать вместе по той же причине, что и в предыдущем случае:



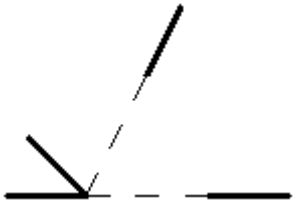
Еще в двух случаях точка *O* является вершиной в обеих частях и диагонали можно брать вместе, улучшая при этом ответ на задачу:



- Если эти две особенные диагонали разных типов, то возникают три различных случая. В первых двух диагонали выбирать вместе нельзя:



И лишь при следующем взаиморасположении диагоналей можно использовать их вместе:



Зная это, поступим следующим образом: построим граф, в котором вершины – это особенные диагонали, а ребра между вершинами проведены тогда, когда их можно использовать вместе. Кроме того, введем вес вершины, равный типу соответствующей особенной диагонали.

Здесь стоит оценить размер этого графа. Каждая особенная диагональ является продолжением некоторой, хотя бы одной, стороны. В то же время, продолжением каждой стороны может являться не более одной особенной диагонали, поскольку иначе появятся две внутренние диагонали многоугольника, лежащие на одной прямой. Отсюда ясно, что количество особенных диагоналей не превышает количества вершин многоугольника.

В построенном графе требуется найти множество вершин, попарно соединенных ребрами между собой, с наибольшим суммарным весом. Эта задача называется “Задачей о максимальной клике” и является NP-полной. Это наталкивает на мысль, что и наша задача не решается за полиномиальное время.

В то же время экспоненциальное решение задачи придумать несложно. Например, можно перебрать все возможные множества вершин в этом графе (их 2^m, где m – количество особенных диагоналей), и для каждого из них проверить, все ли вершины этого множества соединены между собой. Из всех подходящих множеств – выбрать множество с наибольшим весом, оно и будет искомым.

Получив требуемый набор особенных диагоналей, решение изначальной задачи доводится до конца следующим образом:

Разрежем многоугольник вдоль всех особенных диагоналей из найденного набора.

Многоугольник при этом разобьется на некоторое количество меньших многоугольников, в каждом из которых нет особенных диагоналей (иначе их можно было бы добавить в наш максимальный набор). Для каждого из этих многоугольников, следовательно, можно запустить первичный алгоритм, который правильно и оптимально разрежет его на треугольники.

Время работы полученного алгоритма составляет $O(2^N \cdot N^2)$.

Имеется также решение с этой асимптотикой, оформленное в виде динамического программирования.

Рассмотрим некоторый многоугольник, появившийся в процессе разрезания исходного многоугольника внутренними диагоналями. Множество вершин этого многоугольника – подмножество вершин изначального, причем порядок обхода этих вершин определяется однозначно.

Таким образом, имеется не более чем 2^N многоугольников, подлежащих исследованию, и их удобно идентифицировать масками из N битов.

Применим динамическое программирование, чтобы для каждого такого многоугольника подсчитать количество треугольников в его минимальной триангуляции, зная ответы для всех многоугольников с меньшим количеством вершин.

Для треугольника подсчет очевиден.

Для произвольного выпуклого многоугольника подсчет также не представляет сложности.

Теперь рассмотрим некоторый невыпуклый многоугольник. Выберем в нем минимальную по

номеру вершину, угол при которой превышает 180°.

В любой триангуляции из этой вершины должна выходить хотя бы одна диагональ. Проведем из нее все возможные внутренние диагонали. В каждом из случаев многоугольник разделится на два других многоугольника, обработанных ранее. Среди всех вариантов выберем вариант с минимальным суммарным количеством треугольников.