Задача A. Зарядка для кота

В этой задаче было необходимо выполнить симуляцию движения кота.

Каждый раз, когда светящаяся точка перемещалась, необходимо было посчитать квадрат расстояния от текущего положения кота до светящейся точки, а затем — если условия для прыжка соблюдены — изменить положение кота («совершить прыжок» в терминах задачи). Кроме количества прыжков требовалось ещё подсчитывать, сколько раз кот «поймал» светящуюся точку, дотянувшись до нее лапой. Ответ — максимальная по абсолютному значению разность между этими двумя величинами, которая достигалась в процессе игры.

Это означает, что вычислять разность между количеством прыжков и количеством «пойманных» светящихся точек было необходимо после каждого перемещения светящейся точки и фиксировать как максимум по абсолютному значению, так и собственно значение.

Формально можно было вычислять эту разность только в случае изменения количества прыжков или количества «пойманных» точек, но практически вычисление этой разности после каждого перемещения светящейся точки приводит к упрощению кода и уменьшению количества ошибок.

Подзадачи предполагали некоторое упрощение вычислений.

В первой подзадаче предполагалось, что для прыжка существует только ограничение сверху; а светящуюся точку кот ловит только в том случае, когда её координаты совпадают с координатами кота.

Во второй подзадаче кот также ловит светящуюся точку лишь тогда, когда её координаты совпадают с координатами кота, однако для прыжка появляется ограничение снизу.

В третьей подзадаче кот отказывается от прыжка только в том случае, когда светящаяся точка слишком далеко; в остальных случаях он либо ловит точку, либо прыгает к ней.

Наконец, четвёртая подзадача представляет собой общий случай.

Ограничения во всех подзадачах позволяли обойтись 32-разрядными целыми типами (в тех языках, для которых это может быть критично).

Задача B. Прогнозы

В этой задаче можно было действовать разными способами, но при этом нужно было, во-первых, определить, какие именно игры ещё не состоялись, и, во-вторых вычислить количество уже набранных командами очков.

Чтобы понять, какие игры ещё не состоялись, можно было, например, установить фиксированные обозначения играющих пар команд, перенумеровать их и сопоставить с теми, которые приведены во входных данных. Может быть удобно использовать лексикографический порядок: AB, AC, AD, BC, BD, CD.

Набранные очки также несложно подсчитать, анализируя входные данные.

После этого наиболее простым и надёжным подходом является решение методом полного перебора вариантов. В случае, когда не сыграна одна игра, таких вариантов будет три (победа первой команды в паре, победа второй команды в паре, ничья). Если же игр осталось две, то вариантов будет девять (по три независимых варианта для каждой пары).

Будем выбирать очередной вариант в качестве реализуемого и выполнять подсчёт итоговых очков. Затем останется только определить максимум из подсчитанных очков и выяснить, совпадает ли с этим максимумом количество очков, набранное командой A.

Подзадачи позволяли решить задачу частично, не перебирая все возможные варианты.

Задача C. CAT

Будем рассматривать строку символов с конца и будем подсчитывать количество букв T, количество комбинаций AT, и, как итог, количество комбинаций CAT.

Для букв T заведём счётчик $cntT$, который будет увеличиваться на единицу каждый раз, когда при движении справа налево будем встречать очередную букву T.

Комбинации AT подсчитываются более хитрым образом. Как понятно, буква A может быть образовать комбинацию только с теми T, которые находятся после неё. Поэтому, каждый раз, когда при движении справа налево будем встречать букву A, будем добавлять к счётчику AT $cntAT$ количество накопленных к этому моменту букв T (текущее значение счётчика $cntT$).

Наконец, когда будем при движении справа налево встречать букву C, она будет образовывать комбинациями со всеми AT, которые встретились ранее (т.е. правее): к счётчику $cntCAT$ будем добавлять текущее значение счётчика $cntAT$.

Таким образом, когда вся строка будет просмотрена, счётчик $cntCAT$ будет содержать ответ. Асимптотика такого решения — $O(|s|)$, поскольку каждый символ строки будет просмотрен единожды.

Подзадачи позволяли набрать частичное количество баллов.

В первой подзадаче можно было перебрать все возможные варианты любым способом.

Вторая позадача предполагала работу со строкой спецфического вида. В силу её структуры понятно, что любая из имеющихся в ней букв C может образовать комбинацию с любой буквой A и любой буквой T . Поэтому было достаточно подсчитать количество каждой из этих букв, а затем перемножить полученные значения.

Третья подзадача также предполагала работу со строкой специфического вида. Поскольку в такой строке присутствует только одна буква A, достаточно подсчитать количество букв C слева от этой буквы и количество букв T справа от этой буквы. Полученные значения нужно будет перемножить.

Четвёртая подзадача предполагала работу со строкой произвольного вида, однако длина её не превосходила 100 символов. Это означает, что можно использовать алгоритм с асимптотикой $O(|s|^3)$, а именно: для каждой буквы C переберём все буквы A, которые расположены правее неё, а для каждой такой буквы A переберём все буквы T, которые расположены правее неё. Три вложенных цикла позволят получить ответ для строки небольшой длины.

Пятая подзадача, предполагающая работу со строкой произвольного вида большой длины, предполагала полное решение задачи. Таких способов можно построить несколько, один из них описан в начале этого разбора.

Задача D. Камышовый кот

Эта задача — достаточно типичный пример задачи на применение техники двух указателей.

В случае, когда в последовательности моментов времени, в которые кот ловит мышей, могут быть повторяющиеся значения, проведём предварительную подготовку. Для каждой последовательности одинаковых значений времени оставим единственное такое значение и сопоставим ему суммарное количество мышей, которых кот поймает в этот момент. Таким образом, можно перейти от задачи с неубывающей последовательностью моментов времени к задаче с возрастающей последовательностью.

Теперь зафиксируем самый первый (самый левый с точки зрения положения в последовательности) момент времени в этой последовательности и будем продвигаться вправо вдоль последовательности до тех пор, пока суммарное количество пойманных мышей впервые не станет большим или равным требуемого значения k.

Когда это произойдёт, зафиксируем правый (только что найденный) момент времени, вычислим разность между этими моментами и будем сохранять её как наилучшее из найденных пока значений.

После этого начнем двигать вправо левую границу найденной последовательности — теперь уже до тех пор, пока суммарное количество мышей, пойманных в моменты времени этой последовательности, будет оставаться большим или равным значению k. Чтобы вычислить это значение, при движении левой границы вправо потребуется каждый раз вычитать из накопленной ранее суммы значение, соответствующие моменту времени, который мы перестаём учитывать.

Когда накопленная сумма впервые станет меньше k, следует остановить движение левой границы и начать перемещать правую границу — пока вновь не наберётся k или более мышей.

Обратите внимание, что прекратить это попеременное движение сразу по совпадении правой границы с последним элементом последовательности нельзя: это может дать не самый оптимальный ответ. После остановки движения правой границы необходимо ещё «до упора» двигать левую границу, и лишь когда накопленная сумма станет меньше k, можно быть уверенными, что правильный ответ найден.

Асимптотика такого решения составляет $O(n)$, поскольку каждый элемент последовательности будет обработан не более, чем два раза (один раз добавлен в сумму и один раз вычтен).

Подзадачи предполагали возможность набрать часть баллов за менее оптимальное решение.

В подзадачах с первой по четвёртую гарантируется, что последовательность моментов времени является строго возрастающей, что избавляет нас от необходимости проводить предварительную обработку.

В первой подзадаче количество моментов времени не превосходит 1000, при этом в каждый момент времени кот ловит ровно одну мышь. Это позволяет при решении задачи синхронно менять положение левой и правой границ, поскольку «расстояние» между границами всегда будет составлять k элементов. Впрочем, учитывая ограничение на количество моментов, можно просто перебрать левую границу и, каждый раз независимо отсчитывая от неё k элементов, определить правую границу. Задача сведётся к определению минимальной разности между соответствующими моментами времени.

Вторая подзадача отличается от первой тем, что в каждый момент кот ловит количество мышей, которое может отличаться от единицы. Таким образом, при переборе левой границы уже нельзя будет надеяться отсчитать фиксированное количество элементов, потребуется каждый раз подсчитывать сумму пойманных мышей и использовать её как критерий определения правой границы.

Асимптотика решения, перебирающего левую границу и каждый раз независимо от предыдущих вычислений определяющего правую границу, составляет $O(n^2)$.

Третья подзадача позволяет, подобно первой, сосредоточиться на поиске минимальной разности между моментами времени, отстоящими друг от друга ровно на k позиций. Однако ограничения на n предполагают, что должна быть использована описанная выше техника двух указателей (иногда такая версия метода двух указателей, когда «расстояние» между ними остаётся постоянным, именуется методом скользящего окна).

В четвёртой подзадаче единственным «упрощением» по сравнению с полной версией задачи является уменьшение максимального количества пойманных в один момент мышей. Результат вычислений при этом будет помещаться в 32-разрядный тип данных (для типизированных языков) (и это справедливо для всех предыдущих подзадач).

В пятой подзадаче уже необходимо учитывать то, что ряд значений требует для своего хранения 64-разрядного типа данных.

Задача E. Фонари

Эта задача предполагает аккуратную симуляцию процесса замены лампочек.

Полное решение задачи предполагает использование сортирующих структур данных (очереди с приоритетами или упорядоченного множества / словаря). Одна из таких структур призвана хранить данные о фонарях, упорядоченные по времени окончания работы лампочек, которые установлены в настоящий момент в эти фонари. Вторая же структура сохраняет данные о фонарях, в которых в настоящий момент находятся перегоревшие лампочки; эта структура упорядочивает данные о фонарях по их номерам.

В ходе симуляции данные о фонарях перемещаются между этими структурами данных: сначала из структуры, упорядоченной по времени окончания работы лампочек, извлекается m или более элементов (более m элементов может быть в том случае, если в день, когда перегорает m-я лампочка, перегорает не только эта лампочка). Затем эти данные помещаются во вторую структуру, упорядоченную по номерам фонарей.

Далее структура, упорядоченная по номерам фонарей, опустошается, начиная с меньшего номера фонаря: в этот фонарь устанавливается первая неиспользованная лампочка со склада, после чего данные об этом фонаре следует вновь поместить в первую структуру, упорядоченную по времени окончания работы лампочек.

Разумеется, для корректной работы перед началом установки лампочек необходимо проверить, что на складе имеется достаточное их количество. Если это не так, то можно останавливать процесс симуляции и фиксировать день, в который работники парка могут отправиться в отпуск.

Вычисление значений, интересующих кота Бенедикта, производится при перемещении данных о фонарях между описанными структурами данных, и является достаточно несложной и скорее технической задачей.

Подзадачи с первой по третью позволяют так или иначе избежать использования сортирующих

структур данных. Так, в первой подзадаче одновременно будут существовать не более чем два времени окончания работы лампочек. Во второй подзадаче упрощение состоит в том, что все вновь установленные лампочки имеют одинаковое время работы (и это несколько упрощает обращение с группами лампочек с разным временем окончания работы). Наконец, в третьей подзадаче (как и в двух предыдущих) количество фонарей не превышает 1000, что позволяет вместо сортирующей структуры поддерживать массивы, к которым применяется сортировка.

Последняя подзадача традиционно предполагает решение с учётом максимально возможных ограничений.