Разбор задачи «Problem Order»

Самая простая задача. Можно просто взять проблемсет и посчитать вручную, выведя ответ. Можно написать код, который считывает строки входного файла и сравнивает первую букву названия каждой задачи с предыдущей.

Разбор задачи «Interactor»

Достаточно просто прочитать условие задачи и реализовать через условные переходы то, что описано в условии:

```
ans = interact;
if (exitcode != 0)
 {
  if ( (interact==4) || (interact==0) ) ans = 3;
 }

if (interact <= 1) ans = checker;

if (interact>=6) ans = interact-6;
```

Разбор задачи «Signals in the Space»

Заметим, что две последовательности можно получить друг из друга перестановкой тогда и только тогда, когда они состоят из одного и того же набора элементов. Подсчитаем сигналы каждого типа, причём сигналы, отправленные станцией, учитываются с плюсом, а сигналы, принятые на Земле, учитываются с минусом, то есть итоговые положительные суммы составлены из сигналов, которые были отправлены станцией, но не были приняты на Земле, и для получения ответа достаточно сложить эти суммы.

Разбор задачи «PalINTdromes»

Так как существенно палиндромическое число должно записываться в искомой системе счисления более, чем одной цифрой, то $b \leqslant n$. Если $b = n$, то в b-ичной системе счисления n записывается как 10 и палиндромом не является. Если $b = n-1$, то $n = b+1$ и n записывается как 11 — палиндром. То есть в задаче требуется вывести число $n - 1$.

Разбор задачи «Ugly Polyomino»

Для решения задачи для n достаточно для всех некрасивых $(n - 1)$-полимино добавить одну клетку и проверить, существует ли такое n-полимино среди уже сгенерированных.

Заметим, что в данной задаче $n \leqslant 7$.

Для $n = 5$ ответ равен 1, так как все способы добавления к квадрату 2×2 одной клетки, связной с ним по стороне, дают с точностью до поворотов и отражений одну и ту же фигуру.

Разобрав все возможные случаи добавления одной клетки к этой фигуре, получим, что для $n = 6$ ответ равен 8.

Выполнив подобный перебор как на компьютере, так и вручную, можно найти ответ и для $n = 7$, равный 25.

Разбор задачи «DHCP troubles»

В задаче требуется реализовать то, что написано в условии.

Все IP-адреса будем хранить как целые беззнаковые 32-битные числа: адресу `a.b.c.d` соответствует число $addr = a \cdot 2^{24} + b \cdot 2^{16} + c \cdot 2^8 + d$. Отметим, что тип `int` переполнится, так что надо использовать `unsigned int` или `long long`. Представим в виде беззнаковых 32-битных чисел и маску ($mask = (2^M - 1) \cdot 2^{32-M}$), а также адрес подсети (побитовое «и» заданного адреса и числа $mask$) и широковещательный адрес подсети (побитовое «или» адреса подсети и числа $2^{32-M} - 1$).

Заметим, что адрес принадлежит подсети тогда и только тогда, когда в результате применения побитового «и» к адресу и маске подсети получается адрес подсети.

Изначально количество доступных адресов в подсети с маской M равно $2^{32-M} - 2$. Далее мы проверяем адреса в логе на принадлежность подсети и в случае, если адрес соответствует маске и встречается впервые, уменьшаем количество доступных адресов. Если же адрес совпадает с широковещательным адресом или адресом подсети, прерываем обработку адресов и выводим -1.

Разбор задачи «Array Test»

Отметим, что в данной задаче k мало; попробуем этим воспользоваться.

Заметим, что длина искомого подмассива, в случае ее существования, не превышает k (ибо у любого подмассива C массива A длины большей, чем k, есть как минимум k префиксов, которые меньше C и являются подмассивами A); следовательно, достаточно рассматривать все подмассивы массива A длины не более k.

Такое соображение приводит нас к решению за $O(nk^2\ log\ (nk))$: выпишем все подмассивы длины не более k, отсортируем их, удалим дубликаты и выведем k-ый. Однако такое решение не проходило ограничения по времени.

По времени не проходил и вариант решения, в котором все подмассивы добавлялись в сет (std::set в C++ либо TreeSet в Java), из которого потом доставалось k минимальных подмассивов; в силу того, что каждый элемент добавляется в сет за время $O(text * k)$, то такое решение работало за время $O(nk^2log(nk))$, что не лучше, чем предыдущее решение. Однако в силу того, что нам нужно лишь k минимальных подмассивов, то каждый раз, когда при добавлении очередного подмассива в сете оказывается $k+1$ подмассив, мы удаляем максимальный из них. Тогда по добавлении всех подмассивов, в сете останутся лишь k минимальных подмассивов, и нужно лишь вывести максимальный из них (или обнаружить, что различных подмассивов меньше, чем k). Это улучшает асимптотику до $nk^2log\ k$, но это все еще недостаточно быстро.

Чтобы сделать следующий шаг, заметим следующее. Пусть B_1 и B_2 - два подмассива размера k массива A, и пусть $B_1 \leqslant B_2$. Тогда в сет не нужно добавлять подмассивы - "начала"B_2 (действительно, пусть C - подмассив, состоящий из нескольких первых элементов массива B_2. Тогда либо C — это начало и подмассива B_1 тоже, и тогда нет смысла добавлять его лишний раз, либо по свойствам лексикографического сравнения все k "начал" B_1 окажутся меньше, чем C, и C также бессмысленно добавлять в сет). Тогда найдем за $O(nk)$ минимальный лексикографически подмассив длины k и добавим его и все его "начала" в сет. Также обязательно рассмотрим подмассив D, состоящий из последних $k-1$ элементов, и добавим в сет все его подмассивы, коих всего $k(k-1)/2$. После этого выберем k-ый элемент в получившемся сете. Такое решение работает за $O(nk + k * (k(k-1)/2 + k)log(k(k-1)/2 + k)) = O(nk + k^3log\ k)$, что уверенно (с пятикратным запасов) укладывалось в ограничения по времени.

Разбор задачи «Favorite Points»

Сначала найдем величину H - количество таких пар пар точек (именно пар пар, а не пар) $((A,B),(C,D))$ (пары (P,Q) и (Q,P) считаем одинаковыми для любых двух точек (P,Q); однако пары $((A,B),(C,D))$ и $((C,D),(A,B))$ будем считать различными) таких, что прямые AB и CD - две различные параллельные прямые. Для этого создадим массив lst, в который для каждой пары различных любимых точек P,Q добавим тот из векторов $\vec{P}Q$ либо $\vec{Q}P$, у которого координата y положительна либо равна нулю при положительной координате x. Отсортируем все эти вектора по полярному углу и разделим получившийся lst на подмассивы сонаправленных векторов; присвоим H сумму квадратов длин этих подмассивов.

Что же учтено в H? В H учтены все требуемые пары ровно по два раза, а также пары $((A,B),(C,D))$, состоящие из четырех лежащих на одной прямой точек. Чтобы учесть этот момент, найдем для каждой пары точек (A,B) количество пар точек (X,Y), т.ч. A,B,X,Y лежат на одной прямой. Для этого рассмотрим $n-1$ пару точек (A,Z) с фиксированной точкой A и произвольной точкой Z и соответствующие парам векторы. Отсортируем эти векторы по полярному углу и разобьем на подмассивы сонаправленных. Пусть пара (A,B) попала в подмассив длины r; тогда на одной с A и B прямой находится $r+1$ точка, считая A и B. Тогда количество пар (X,Y) есть $(r+1) * r/2$; именно такую величину нужно вычесть для каждой пары (A,B) (но обязательно проследить, что для каждой пары величина вычитается лишь один раз!). Все вышеописанные действия выполняем за $O(n^2log\ n)$.

Зачем же нам нужна такая величина H? Заметим, что H есть количество трапеции плюс удвоенное количество параллелограммов; следовательно, имея H и найдя количество параллелограммов, мы за $O(1)$ найдем количество трапеций.

Перейдем к оставшимся случаям. Будем использовать следующие эквивалентные заданным в условии определения:

- параллелограмм есть четырехугольник, середины диагоналей которого совпадают;
- ромб есть четырехугольник, середины диагоналей которого совпадают, а сами диагонали перпендикулярны;
- квадрат есть четырехугольник, середины диагоналей которого совпадают, а сами диагонали перпендикулярны и равны по длине.

Выпишем список $mids$ различных середин всех возможных отрезков; для каждой точки $mids[i]$ найдем список всех отрезков, серединой которых $mids[i]$ является. Каждый параллелограмм/ромб/квадрат задается двумя непараллельными отрезками с совпадающими серединами; поэтому будем решать задачу для каждого списка отрезков с фиксированной серединой в отдельности.

Итак, пусть p - некоторая точка, а $segments$ - список отрезков с серединой в т. p. Отсортируем $segments$ по полярному углу (из двух векторов, соединяющих концы отрезка, выберем, как и в случае трапеций, вектор, смотрящий "вверх"). Тогда:

- чтобы найти количество параллелограммов, нужно разбить вектора на группы сонаправленных, найти их длины l_j, $j = 1, 2, \ldots$, и вычесть все $l_j(l_j - 1)/2$ из $l(l - 1)/2$, где l - размер $segments$;
- чтобы найти количество ромбов, нужно рассмотреть группы сонаправленных, найти с помощью метода двух указателей пары групп с перпендикулярными векторами, найти в каждой такой паре произведения размеров групп и просуммировать их;
- количество же квадратов ищется так же, как и количество ромбов, только нужно рассмотреть не группы сонаправленных, а группы одинаковых (при равенстве полярных углов сортируем отрезки по длине) отрезков; эти группы состоят по факту из одного отрезка.

Общее время работы всех шагов - $O(n^2 log\ n)$.

Разбор задачи «Thorny Graph»

В условии задачи нас просят проверить, является ли данный ориентированный граф реберным кактусом.

Выделим компоненты сильной связности в графе. Поскольку ребро, проходящее между разными компонентами сильной связности, не может быть частью цикла, то его наличие не изменяет свойство графа быть кактусом, поэтому мы можем удалить все такие рёбра и проверить каждую компоненту сильной связности отдельно. Далее везде предполагаем, что граф является сильно связным.

Поймем, как выглядит сильно связный реберный кактус. Если в таком кактусе более одной вершины, то в кактусе найдется простой цикл; назовем его $Cycle1$. $Cycle1$ может быть всем графом; в противном случае, в $Cycle1$ входит вершина v, из которой выходит ребро (v, u) в вершину u, не лежащую в цикле $Cycle1$.

Так как граф сильно связен, то из u в v существует некоторый простой путь P. Тогда P не может содержать вершин цикла $Cycle1$, кроме v. Докажем это от противного. Действительно, пусть P содержит вершины $Cycle1$, отличные от v, и пусть w - первая из них на P. Тогда все ребра $Cycle1$, лежащие на пути от w до v, лежат на цикле $Cycle1$ и на цикле, проходящем от w до v по $Cycle1$, а от v до w - через u и далее по пути P, и мы получаем противоречие с определением реберного кактуса.

Следовательно, ребро (v, u) и путь P образуют цикл, пересекающийся с $Cycle1$ лишь по вершине v; назовем этот цикл $Cycle2$. Объединение двух циклов может быть всем графом; в противном случае

аналогичным образом доказываем, что существует цикл $Cycle3$, пересекающийся с объединением $Cycle1$ и $Cycle2$ лишь по одной вершине.

Повторяя такие рассуждения, можно показать, что сильно связный реберный кактус - это либо одна вершина, либо набор простых циклов, каждый из которых может пересекаться с другими циклами, с каждым циклом только по одной вершине. Более того, если мысленно построить граф, в котором вершинами будут являться циклы старого графа — вершины-циклы — и вершины, по которым эти циклы пересекаются — вершины-связующие вершины, а ребро между вершиной-циклом и вершиной-связующей вершиной провести, если связующая вершина принадлежит циклу, то, чтобы первоначальный граф был реберным кактусом, нужно, чтобы новый граф являлся деревом. Мы показали, что это - необходимое условие; однако очевидно, что граф описанного вида всегда является реберным кактусом.

Заметим, что в новом графе листом может быть только вершина-цикл, поскольку любая связующая вершина должна соединять минимум два цикла. Возьмём этот лист-цикл, он состоит хотя бы из двух вершин, причем только одна из них соединена с другими циклами, значит, у него есть вершина с одним входящим и одним исходящим ребрами — транзитная вершина.

Итак, мы показали, что в любом реберном кактусе существует транзитная вершина. Находим транзитную вершину v, входящую в неё $from$ и исходящую из неё to. Найдем любой путь из to в $from$, обозначим его за $path(to, from)$. Если из $from$ есть ребро в to, то у нас будет два разных цикла $from \to to \to path(to, from) \to$ и $from \to v \to to \to path(to, from) \to$, что противоречит определению.

Введем операцию стягивания: удаляем вершину v и проводим ребро из $from$ в to, если $from != to$. Ясно, что наша операция никак не изменила "кактусовость"графа: ребра $from \to v$ и $v \to to$ входят или не входят в любой цикл вместе.

Итак, получили следующий алгоритм проверки для каждой компоненты сильной связности: находим транзитную вершину, выводим "NO"если существует ребро $from \to to$, иначе стягиваем граф по вершине v. Если в итоге у нас осталась одна вершина, то данная компонента является реберным кактусом, иначе нет. Для эффективного нахождения транзитных вершин поддерживаем любое сбалансированное дерево поиска по ключу (входящая степень + исходящая степень) и извлекаем вершину с минимальным ключом. Итоговая асимптотика $O(m * log(n))$.

Разбор задачи «Xor and segments »

Пусть $x \oplus y$, $x, y \in \{0, 1\}$ - это побитовое исключающее ИЛИ. Тогда в запросе второго типа с параметрами L, R требуется найти величину

$$ans = \bigoplus_{\substack{1 \leqslant i \leqslant j \leqslant n \\ L \leqslant j-i+1 \leqslant R}} \bigoplus_{k=i}^{j} a_k.$$

Чтобы упростить формулу, введем несколько уровней "частичных ксоров". А именно:

- Пусть $s1_0 = s1_{-1} = s1_{-2} = \ldots = 0$, а $s1_i = \bigoplus_{k=1}^{i} a_k$ для $i \in [1, n]$;
- Пусть $s2_0 = s2_{-1} = s2_{-2} = \ldots = 0$, а $s2_i = \bigoplus_{k=1}^{i} s1_k$ для $i \in [1, n]$;
- Пусть $s3_0 = s3_{-1} = s3_{-2} = \ldots = 0$, а $s3_i = \bigoplus_{k=1}^{i} s2_k$ для $i \in [1, n]$.

Тогда

$ans = \bigoplus_{\substack{1 \leqslant i \leqslant j \leqslant n \\ L \leqslant j-i+1 \leqslant R}} (s1_j \oplus s1_{i-1})) = \bigoplus_{L \leqslant len \leqslant R} \bigoplus_{1 \leqslant i \leqslant n-len+1} (s1_{i+len-1} \oplus s1_{i-1}) =$

$= \bigoplus_{L \leqslant len \leqslant R} (s2_n \oplus s2_{len-1} \oplus s2_{n-len}) = (s2_n * ((R-L+1) \bmod 2)) \oplus s3_{R-1} \oplus s3_{L-2} \oplus s3_{n-L} \oplus s3_{n-R-1} = ((s3_n \oplus s3_{n-1}) * ((R$

Поймем, как изменяются элементы массивов a, $s1$, $s2$, $s3$ при запросах первого типа с параметрами L, R. Рассмотрим сначала случай, когда $L = R$. Будем помечать единицей изменяющиеся элементы, а нулем - не изменяющиеся. Тогда изменения будут выглядеть следующим образом:

a	$000001000000000000000000\ldots$
$s1$	$000001111111111111111111\ldots$
$s2$	$000001010101010101010101\ldots$
$s3$	$000001100110011001100110\ldots$

Заметим, что изменение $s3$ происходит на некотором суффиксе массива, причем это изменение периодично и имеет период 4. Таким образом, если бы в первом запросе всегда было бы $L = R$, то мы бы умели решать задачу с помощью четырех деревьев отрезков, т.ч. в первом из них мы бы хранили $s3_1, s3_5, s3_9, \ldots$, во втором - $s3_2, s3_6, s3_{10}\ldots$ и т.д. . Тогда запрос первого типа есть четыре изменения в четырех деревбях отрезков, каждое из них - на некотором подотрезке; запросы же второго типа, как следует из выкладок выше, также могут быть обработаны за $O(1)$ запросов к тем же деревьям отрезков; время обработки каждого запроса было бы $O(log\ n)$.

Но что же делать, если в запросе первого типа $L < R$? Предположим, что $R = L + 3$, т.е. изменяемый в a подотрезок имеет длину 4. Выпишем, как изменяется $s3$ при изменении L-го, $L+1$-го, $L + 2$-го и $L + 3$-го и сложим изменения:

a	$000001111000000000000000\ldots$
L	$000001100110011001100110\ldots$
$L+1$	$000000110011001100110011\ldots$
$L+2$	$000000011001100110011001\ldots$
$L+3$	$000000001100110011001100\ldots$
$s3$	$000001000000000000000000\ldots$

Таким образом, изменение a на отрезке $[L, L + 3]$ есть изменение лишь $s3_L$ в $s3$! Отсюда следует, что изменение a на подотрезке $[L, L + 4k - 1]$, $k \in \mathbb{N}$ изменяет $s3$ на позициях $L, L + 4, L + 8, \ldots, L + 4k - 4$, что может быть обработано с помощью одного запроса к одному из вышеописанных деревьев отрезков:

a	$000001111111111110000000\ldots$
$s3$	$000001000100010000000000\ldots$

Чтобы обработать запрос первого типа с произвольными параметрами L, R, разобьем отрезок $[L, R]$ на отрезки $[L, M]$ и $[M+1, R]$, где $0 \leqslant R-M \leqslant 3$, $(M-L+1)\ mod\ 4 = 0$; первый из подотрезков обрабатываем за 1 запрос к деревьям отрезков, а второй, в силу его короткой длины - за не более чем 6 запросов к деревьям отрезков. Таким образом, мы научились выполнять запросы обоих типов за $O(\log n)$ на каждый запрос.