Региональный этап Всероссийской олимпиады по информатике 2014

Задача 1 «POBEDA-2014»

Основная идея решения данной задачи основана на следующем факте: для того, чтобы нарисовать единичный квадрат, требуется либо по одному треугольнику типа 1 и 2, либо по одному треугольнику типа 3 и 4. Таким образом, из a_1 треугольников типа 1 и a_2 треугольников типа 2 можно составить не более $\min\{a_1, a_2\}$ квадратов, а из a_3 треугольников типа 3 и a_4 треугольников типа 4 – $\min\{a_3, a_4\}$ квадратов.

Теперь, чтобы решить поставленную задачу, требуется определить, какой максимальный квадрат можно составить из $K = \min\{a_1, a_2\} + \min\{a_3, a_4\}$ единичных квадратов. Для этого можно либо перебирать все квадраты натуральных чисел (1, 4, 9, 16, …) до тех пор, пока не встретится число, большее K, либо извлечь из K квадратный корень и округлить результат до ближайшего целого вниз. Стоит учитывать малую точность операции взятия квадратного корня из большого числа в некоторых языках программирования, поэтому необходимо результат проверить обратным возведением в квадрат или использовать тип с расширенной точностью.

В первом случае сложность алгоритма будет равна $O(K^{1/2})$, а во втором – O(1).

Задача 2 «Список школ»

При решении данной задачи необходимо сначала выделить номера школ из каждой записи с названием школы и сформировать массив, содержащий эти номера. Затем, используя полученный массив, определить, количество школ и номера школ, которые встречаются в нем не более пяти раз.

При выделении номера школы из ее названия *s* следует последовательно перемещаться по строке *s* слева направо, пока не встретится цифра. Это можно сделать с помощью следующего фрагмента программы:

```
i = 0
while s[i] not in '0123456789':
   i += 1
start = i
while i < len(s) and s[i] in '0123456789':
   i +=1
finish = i
```

После выполнения этого фрагмента программы подстрока строки *s* от символа с номером start до символа с номером finish (не включительно) и есть искомый номер школы. Следует выделить этот номер и сохранить его в массиве *sch*.

Поскольку номер школы может быть очень большим, то не следует переводить его в числовой тип данных, а сохранить как строку. Полученный массив номеров школ нужно отсортировать в лексикографическом (алфавитном) порядке. Теперь, чтобы получить искомый ответ, осталось подсчитать, сколько раз встречается в массиве каждая школа, и сформировать список школ, встречающихся не более 5 раз. Это можно сделать, например, с использованием следующего фрагмента программы:

```
count = 1
result = 0
answer = []
for i in range(1, len(sch)):
    if sch[i] == sch[i – 1]:
        count += 1
    else:
        if count <= 5:
            result += 1
            answer.append(sch[i-1])
        count = 0
if count <= 5: # не забываем последнюю школу
    result += 1
    answer.append(sch[-1])
```

Задача 3 «Межрегиональная олимпиада»

Разберем вначале частичное решение, основанное на предположении, что все задачи оцениваются одинаково. Представим каждую задачу отрезком на оси времени. Тогда задача сводится к классической: из заданного множества отрезков на прямой выбрать наибольшее количество отрезков, не имеющих общих точек.

Для решения этой задачи можно использовать следующий алгоритм. Вначале выберем из всех отрезков тот, который заканчивается левее всех. Из отрезков, которые начинаются правее него, опять выберем отрезок, заканчивающийся левее всех и т.д.

Не сложно определить, что реализация этого алгоритма имеет асимптотическую сложность $O(N^2)$. Более эффективное решение можно получить, отсортировав все начала и концы отрезков и пройдя слева направо по отсортированному массиву. Такое решение с учетом использования быстрой сортировки имеет уже асимптотическую сложность $O(N \log N)$.

Для решения исходной задачи в общем случае, когда каждая задача на межрегиональной олимпиаде оценивается своим количеством баллов, можно использовать различные подходы. Некоторые из них рассмотрены ниже в представленных вариантах решений.

Первый вариант решения**.** Он основан на использовании метода динамического программирования и заключается в следующем. Отсортируем все задачи, предложенные на межрегиональной олимпиаде, по *времени окончания* их решения. Пусть $a[i]$ – максимальное количество баллов, которые можно набрать, решая только задачи из числа "первых" i задач. Рассмотрим i-ю задачу. Пусть j – это номер последней задачи, которая заканчивается не позже, чем выдается i-я задача (то есть $s_j + t_j \le s_i$, *а* $s_{j+1} + t_{j+1} > s_j$). Тогда

$$a[i] = \max(a[i-1], a[j] + c[i]).$$

Из этого следует, что можно либо решить задачу с номером i и какой-то набор задач с номерами не больше j, либо не решать задачу с номером i. Ответом на поставленный вопрос будет число $a[n]$.

Осталось пояснить, как находить число j. Это можно делать несколькими способами:

- перебором всех чисел от 1 до i (решение с асимптотической сложностью $O(N^2)$);
- бинарным поиском на отрезке от 1 до i (сложность $O(N \log N)$);
- используя метод "двух указателей", то есть, запоминая предыдущее j и осуществляя поиск каждого следующего j на отрезке от предыдущего j, поскольку новое значение будет больше или равно предыдущему (таким образом, для вычисления всех $a[i]$ каждого кандидата каждый j мы проверим лишь единожды и сложность такого решения будет равна $O(N \log N)$ с учетом сортировки).

Восстановление ответа. Кроме поиска наибольшей суммы баллов, которую наберет Артур на олимпиаде, требуется также вывести количество и перечень задач, решение которых позволит ему набрать такую сумму. Это можно сделать одним из стандартных способов восстановления ответа в динамическом программировании. Например, будем кроме массива a хранить дополнительный массив $prev$, в котором укажем, какой выбор был сделан на каждом шаге. Если $a[i] = c[i] + a[j]$, то в $prev[j]$ запишем j. Это будет означать, что решается задача с номером i, а предыдущая задача имеет номер не больше j. В противном случае необходимо записать в $prev[i]$ число –1. Теперь, пройдя с конца по массиву $prev$, можно восстановить список решаемых задач.

Второй вариант решения. При реализации этого варианта вначале отсортируем вместе все начала и концы отрезков, сохраняя для каждой точки ее тип (начало или конец) и номер отрезка, к которому она относится. Будем рассматривать отсортированные события слева направо. В переменной $best$ будем хранить лучший результат, который можно получить, используя только задачи, время выполнения которых уже закончилось к текущему моменту («закончившиеся» задачи). Если очередная точка – это начало отрезка с номером i, то запишем в $b[i]$ текущее значение переменной $best$. Таким образом, $b[i]$ –

это наилучший результат, который можно получить до начала решения i-й задачи. Если очередная точка – это конец i-го отрезка, то необходимо обновить значение переменной *best* следующим образом: если решается i-я задача, то $best = b[i] + c[i]$, в противном случае значение переменной *best* не изменяется. Из двух вариантов нужно выбрать тот, в котором значение переменной *best* наибольшее, то есть $best = \max\{best, b[i] + c[i]\}$.

Задача 4 «Дом Мэра»

Если рассматривать небольшие ограничения, то решение задачи может быть следующим. Построим граф, соответствующий городу без застройки, при этом вершинам графа будут соответствовать перекрестки, а ребрам – дороги. Затем удалим в графе все ребра, попавшие в застроенные кварталы (сложность такой процедуры равна O(n * площадь города)). В полученном графе начнем поиск в ширину из точки (0, 0), чтобы определить расстояния до всех возможных расположений будущего дома Мэра, и выберем из них самое близкое. Восстанавливая путь при реализации поиска в ширину стандартным способом, можно найти точки поворота пути.

Если площадь города велика, а кварталов застройки не много, то можно воспользоваться *сжатием координат*. Для этого при построении графа будем использовать только те горизонтали и вертикали, на которых расположены мэрия, будущие дома или границы кварталов застройки.

В случае максимальных ограничений решение исходной задачи может быть следующим. Будем решать задачу отдельно для каждого возможного расположения будущего дома Мэра. Пусть рассматриваемый дом имеет координаты (x_i, y_i). Будем считать, что $x_i \geq 0$, $y_i \geq 0$. Легко понять, что нужно изменить в решении, чтобы рассмотреть и случаи отрицательных координат.

Рассмотрим пути, выходящие из мэрии на Север. Возможны два варианта пути:

1) едем на Север, затем поворачиваем направо (на Восток), и затем налево (опять на Север); при этом любой отрезок пути может иметь длину, равную 0;
2) едем на Север, пересекая горизонтальную улицу, на которой будет расположен дом Мэра, затем поворачиваем направо (на Восток), и затем еще раз направо (обратно на Юг).

Сравнивая эти два пути можно сказать, что первый путь всегда короче второго.

Далее выясним, как далеко мы сможем проехать из точки (0, 0) на Север (см. рис. 1). Для этого переберем все кварталы застройки и пересечем их лучом, выходящим из начала координат на Север. Пусть мы можем беспрепятственно проехать до точки $(0, R)$. Если рассматриваемый дом находится на этом отрезке, то задача решена.

Аналогично рассмотрим луч, выходящий из точки (x_i, y_i) на Юг, и найдем на нем точку (x_i, S) с минимальной положительной ординатой S, такую что из этой точки мы сможем беспрепятственно доехать до дома Мэра. Если $S > R$, то пути первого типа нет. В противном случае, попытаемся найти минимальное значение t на отрезке $[S, R]$, такое, что можно проехать из точки $(0, t)$ в точку (x_i, t). Для этого рассмотрим все кварталы, которые пересекаются с полосой $0 \le x \le x_i$. Рассмотрим их проекции на ось Oy, найдем объединение этих проекций (открытых интервалов), и найдем точку Y на оси Oy с минимальной координатой, не меньшей S, не покрытую объединением интервалов. Для этого можно отсортировать вместе начала и концы интервалов и пройти по этим точкам, считая *баланс*. Если Y окажется не больше R, то искомый путь имеет поворот направо в точке $(0, Y)$ и поворот налево в точке (x_i, Y). В противном случае, пути первого вида нет, и нужно перейти к поиску пути второго вида.

Если $R \le y_i$, то пути второго вида также нет. Иначе рассмотрим луч, выходящий из точки (x_i, y_i) на Север (см. рис. 2), и найдем на нем самую далекую точку S, до которой можно беспрепятственно добраться из точки (x_i, y_i). Рассмотрим отрезок $[x_i, \min\{R, S\}]$ и найдем на нем минимальную точку Y, не покрытую объединением проекций интервалов, рассмотренным выше. Если такая точка Y существует, то из всех путей, выходящих из точки (0, 0) на Север, кратчайший путь имеет повороты в точках $(0, Y)$ и (x_i, Y). В противном случае таких путей нет.

Заметим, что если $Y = y_i$, то второй поворот не нужен.

Данный алгоритм имеет асимптотическую сложность $O(kn \log n)$.

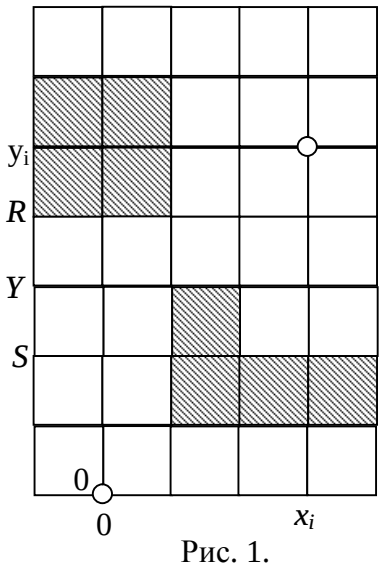

Рис. 1.

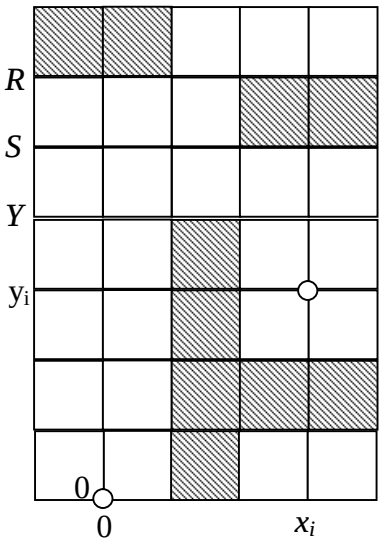

Рис. 2.

Задача 5 «Светофоры»

Поскольку решения с целочисленным ответом является частичными и также оцениваются, то начнем рассмотрение методов решения исходной задачи именно с таких решений. Для этого сформулируем следующую вспомогательную задачу: «*Электрокар движется со скоростью* v. *Может ли он проехать оба светофора на зеленый свет?*».

Для решения этой вспомогательной задачи введем в рассмотрение два электрокара А и Б. Пусть электрокар А проезжает первый светофор в момент включения зеленого сигнала, а электрокар Б – на a минут позже, то есть, в момент включения следующего красного сигнала. Ко второму светофору оба эти электрокара подъедут с разницей в a минут. Если хотя бы один из них окажется у второго светофора, когда у него будет гореть зеленый свет, то задача может быть легко решена. Если же оба электрокара подъедут к светофору с красным светом, то и никакой другой электрокар не сможет проехать на зеленый свет, потому что зеленый свет горит a минут и интервал между электрокарами А и Б также составляет a минут. Понятно, что за это время зеленый свет на светофоре не зажигался, и любой электрокар, стартовавший позже А, но раньше Б, подъедет ко второму светофору, когда у него будет гореть красный свет. Осталось теперь научиться определять, какого цвета будет второй светофор, когда к нему подъедут электрокары А и Б.

Время, необходимое электрокару, чтобы проехать от одного светофора до другого, равно x / v. Период работы светофора равен $(a + b)$. Поэтому необходимо, чтобы выполнялись условия: $(x / v) \% (a + b) \le a$ (для первого электрокара) или $(x / v + a) \% (a + b) \le a$ (для второго электрокара). Эти условия также можно переписать, используя целочисленное деление. Например, для первого электрокара оно будет выглядеть следующим образом:

$$(x // v) \% (a + b) < a \text{ or } (x // v) \% (a + b) == a \text{ and } x \% v == 0$$

Теперь, когда понятно, как отвечать на вопрос, можно ли проехать оба светофора на зеленый свет, если двигаться со скоростью v, не трудно определить подходящую скорость среди целочисленных ответов. Для этого необходимо перебрать все целые скорости, начиная с 1000, до тех пор, пока не найдется правильный ответ.

В общем случае искомая скорость не обязательно является целочисленной. Для решения задачи в этом случае рассмотрим электрокары А и Б, скорости которых равны 1000 и которые проезжают первый светофор в момент включения и выключения у него зеленого света соответственно. Если один из них проезжает второй светофор на зеленый свет, то легко найти ответ, и задача решена. В противном случае, выгоднее всего будет уменьшить скорость электрокара Б так, чтобы он проехал второй светофор в момент

включения зеленого света. Для этого он должен увеличить время своего движения $t = x / 1000$ на величину, равную:

$$dt = (a + b) - (x / 1000 + a) \% (a + b).$$

Тогда его скорость будет равна $x / (t + dt)$.

Задача 6 «Кондиционеры»

Начнем решение этой задачи с рассмотрения частичного решения. Это решение основано на переборе и заключается в следующем. Для каждого класса перебираются все кондиционеры, и выбирается из них подходящий кондиционер с минимальной стоимостью. Легко видеть, что асимптотическая сложность такого решения равна $O(n \cdot m)$.

Решение на полный балл будет выглядеть следующим образом. Заметим, что если одна модель кондиционера дороже другой, а при этом объем обслуживаемого им помещения не больше, чем у первого, то этот кондиционер покупать не выгодно. С учетом этого отсортируем все кондиционеры по стоимости, а при равной стоимости – по мощности. Далее, последовательно рассматривая полученные в этом случае элементы упорядоченного массива, удалим в нем все модели кондиционеров, которые имеют не большую мощность при равной или большей стоимости. Это можно сделать за линейное время, используя тот же массив, или создать новый массив. После этого получается массив кондиционеров, в котором элементы расположены в порядке их возрастания как по стоимости, так и по мощности. Следовательно, для каждого класса нужно выбрать кондиционер из полученного массива с минимальной подходящей мощностью.

Теперь осталось упорядочить классы по требуемой мощности кондиционеров и пройтись по массиву классов и массиву кондиционеров «двумя указателями» - один указатель будет указывать на класс, другой на самый выгодный подходящий для него кондиционер. При увеличении мощности кондиционера для класса указатель, указывающий на самый выгодный кондиционер, будет только расти.

Такой алгоритм имеет сложность $O(n \cdot \log n + m \cdot \log m)$. Также можно осуществлять поиск кондиционера для каждого класса с использованием двоичного поиска. Сложность алгоритма в этом случае будет равна $O(m \cdot \log m + n \cdot \log m)$.

Задача 7 «Конфеты»

Для начала рассмотрим самый простой вариант решения данной задачи, когда $a = b = c = 1$. Не сложно увидеть, что в этом случае значения x, y и z должны быть по возможности равными, т.е. $x = y = z = n/3$. Если n не делится на 3, то какое-то из этих чисел надо округлить вверх, какое-то – вниз.

Для доказательства этого утверждения рассмотрим два случая: первый, когда $x = y = z = n/3$, и второй, когда $x = n/3 + t$, $y = n/3 – t$, $z = n/3$. В первом случае $xyz = (n^3/27)$, во втором $xyz = (n^3/27) – t^2n/3$, что меньше при любом ненулевом значении t.

В общем случае можно предположить, что ax, by, cz близки к $n/3$. Чтобы убедиться в этом, сначала исследуем это утверждение для более простой задачи: $ax + by = n$.

Рассмотрим начальное значение $x = \left\lfloor \frac{n}{2a} \right\rfloor$ и $y = \left\lfloor \frac{n}{2b} \right\rfloor$, то есть, такие x и y, что ax и by близки к $n/2$. Заметим, что $ax > \frac{n}{2} - a$ и $by > \frac{n}{2} - b$. Поэтому $ax \times by = f_0 > (\frac{n}{2} - a) \times (\frac{n}{2} - b) = (\frac{n}{2})^2 - (a+b)(\frac{n}{2})$. Попробуем теперь взять значения x и y, отличающиеся от $n/2$ на t и исследуем, при каких значениях t результат может быть лучше. Получаем $ax \times by = f_t > (\frac{n}{2} - t) \times (\frac{n}{2} - t) = (\frac{n}{2})^2 - t^2$. Если $t^2 > (a+b)\frac{n}{2}$, то этот результат меньше, чем при первых значениях x и y.

Рассмотрим, при каких значениях t результат может быть лучше. Пусть $t = a \times dx = b \times dy$ и $a \geq b$. Тогда из предыдущего неравенства получается, что интересны $t^2 \leq (a+b)\frac{n}{2} \leq an$, т.е. $(a \times dx)^2 \leq an \Leftrightarrow a \times dx^2 \leq n$. Заметим, что $dx \leq b \leq a$. Получается, что $dx^3 \leq n$. Значит, лучший результат может быть только при $dx \leq \sqrt[3]{n}$.

Осталось теперь обобщить идею для трех переменных. Заметим, что для любых двух из трех неизвестных можно провести такое же рассуждение, то есть, ax и by не могут отличаться больше, чем на кубический корень из n, а также пара ax и cz и пара by и cz не могут сильно отличаться.

С учетом сказанного решение исходной задачи будет следующим.

1. Берем в качестве первого приближения $x = \left\lfloor \frac{n}{3a} \right\rfloor$, $y = \left\lfloor \frac{n}{3b} \right\rfloor$ и $z = \left\lfloor \frac{n}{3c} \right\rfloor$.

2. Переберем все перестановки a, b и c.

3. Переберем x в интервале от $\left\lfloor \frac{n}{3a} \right\rfloor - \sqrt[3]{n}$ до $\left\lfloor \frac{n}{3a} \right\rfloor + \sqrt[3]{n}$

4. Переберем y в интервале от $\left\lfloor \frac{n-ax}{2b} \right\rfloor - \sqrt[3]{n}$ до $\left\lfloor \frac{n-ax}{2b} \right\rfloor + \sqrt[3]{n}$

5. Получаем $z = \left\lfloor \frac{n-ax-by}{c} \right\rfloor$ и проверяем, стало ли произведение больше текущего.

Такое решение имеет асимптотическую сложность $O(n^{2/3})$. Заметим, что количество коробок конфет, определяемое произведением размеров коробки, может быть порядка

10^{27}, то есть, превышать максимальное значение целого 64-битного типа данных. Для того, чтобы избежать длинной арифметики при сравнении перебираемых вариантов, можно вместо неравенства вида $x_1 \times y_1 \times z_1 > x_2 \times y_2 \times z_2$ использовать неравенство $x_1 \times y_1 \times z_1 - x_2 \times y_2 \times z_2 > 0$. Если разность не превышает 2^{63}, то неравенство с переполнением будет определено верно. Также можно использовать вещественный тип с расширенной точностью.

Следует отметить, что частичное решение, основанное на переборе всевозможных значений x и y и вычислении z по формуле $z = \left\lfloor \frac{n - ax - by}{c} \right\rfloor$, имеет асимптотическую сложность O(n^2) и оценивается из 40 баллов.

Задача 8. «Волонтеры»

После формализации исходную задачу можно сформулировать в терминах теории графов следующим образом. Пусть заданы два дерева с общими листьями (рис. 1). Требуется найти количество пар вершин (u, v), где u принадлежит левому дереву, а v – правому, для которых существует путь из вершины u в вершину v, идущий всё время слева направо, как изображено на рис. 2.

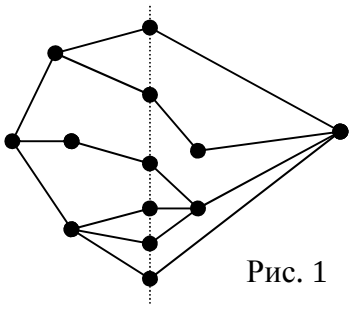

Рис. 1

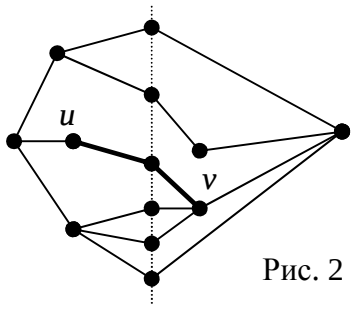

Рис. 2

Пронумеруем листья деревьев, и для каждой из остальных вершин определим, какие листья ей соответствуют (см. рис. 3). Заметим, что каждой вершине соответствуют несколько последовательных листьев, поэтому достаточно хранить только номера самого нижнего и самого верхнего листа. Это можно сделать, реализовав в каждом дереве из корня обход в глубину и отметив для каждой вершины номер самого первого потомка-листа и самого последнего потомка-листа. Суммарная асимптотическая сложность такой реализации будет равна O($n + m + k$).

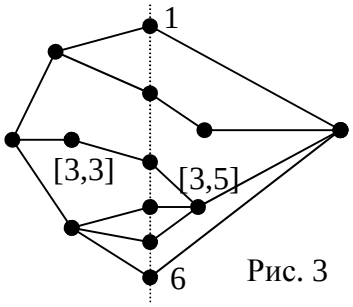

Рис. 3

Далее можно просто рассмотреть каждую пару вершин (u, v) и определить, имеют ли отнесенные к ним отрезки общее пересечение. Реализующий эти операции алгоритм будет иметь сложность $O(m \cdot k)$. Более эффективное решение можно построить следующим образом.

Пусть отрезки, отнесенные к вершинам левого дерева, будут синими, а отрезки, отнесенные к вершинам правого дерева, – красными. Отметим эти отрезки на одной координатной прямой. Подсчитаем для каждого синего отрезка, с каким количеством красных отрезков он пересекается. Это можно сделать следующим образом.

Будем двигаться по координатной прямой слева направо и подсчитывать отдельно количество начал красных отрезков и количество концов красных отрезков. Если рассмотреть конкретный синий отрезок AB, то можно установить, что с ним пересекаются все красные отрезки, кроме тех, которые заканчиваются левее A, и тех, которые начинаются правее B. Таким образом, дойдя до вершины A, мы определим количество концов красных отрезков, которые нам встретились, а дойдя до вершины B – количество начал красных отрезков, которые нам еще не встретились. Зная эти числа, можно легко вычислить искомое количество отрезков. Асимптотическая сложность такого решения будет равна $O((m + k) \cdot \log(m + k))$, включая сортировку концов отрезков.

Существует еще один способ реализации похожей идеи. Он заключается в следующем. Будем подсчитывать количество пар (u, v), таких, что из вершины u в вершину v нет пути. Для этого сначала подсчитаем для вершин *левого* дерева те же отрезки листьев, что и в предыдущем способе решения. С вершинами правого дерева поступим иначе. Для каждого листа w рассмотрим путь от него до корня правого поддерева (см. рис. 4) и подсчитаем отдельно количество вершин выше этого пути (назовем его $\mathrm{top}[w]$) и ниже этого пути (назовем его $\mathrm{bottom}[w]$).

Теперь рассмотрим вершину левого дерева. Научимся за $O(1)$ определять, со сколькими вершинами правого поддерева она *не* соединена. Для этого рассмотрим отрезок $[a, b]$, который ей соответствует. Ясно, что она не соединена с $(\mathrm{top}[a] + \mathrm{bottom}[b])$ вершинами правого поддерева. Осталось просуммировать данные выражения для всех вершин левого поддерева.

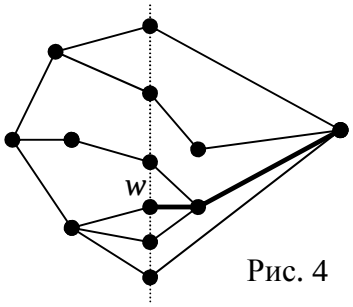

Рис. 4

Теперь, чтобы получить ответ на поставленную исходную задачу, достаточно вычесть полученное выше количество пар из общего количества пар $m \cdot k$.