

**РАЗБОР ЗАДАЧ**  
**региональной олимпиады студентов вузов Санкт-Петербурга**  
**по информатике и программированию 2009**

**Задача А. Auxiliary Question of the Universe**

В данной задаче требовалось путем вставки дополнительных символов в исходное некорректное математическое выражение преобразовать его в корректное выражение. При этом, как исходное, так и полученное выражения должны содержать только цифры, круглые скобки и знак «+».

Эта задача может быть решена различными способами. Опишем один из них.

Припишем в начало выражения цифру 1, а затем будем заменять каждый символ исходного выражения на такую последовательность символов, чтобы уже преобразованная часть выражения была корректной: Для этого цифры будем преобразовывать в последовательность из символа «+» и этой цифры, а все остальные символы («+», «(», «)») — в комбинацию «+( 1 )».

Таким образом, после преобразования всего исходного выражения, получим корректное итоговое выражение. При этом длина выражения в худшем случае увеличится в три раза плюс один символ, что при длине исходного выражения не превышающей 1000 символов составит не более 3001 символа.

Рассмотрим применение этого алгоритма на примере входного файла:

Обработанные символы	Полученное выражение
	0
1	0 +1
1+	0 +1 +(0)
1+0	0 +1 +(0)+ 0
1+0+	0 +1 +(0)+ 0 +(0)
1+0+1	0 +1 +(0)+ 0 +(0) +1
1+0+1)	0 +1 +(0)+ 0 +(0) +1 +(0)

Приведенное решение имеет сложность по памяти  $O(1)$  и по времени —  $O(L)$ , где  $L$  — количество символов в исходном выражении.

## Задача В. Bureaucracy

В данной задаче требовалось определить множество законов, действующих после того, как некоторые из ранее принятых законов были отменены. При этом, «отменяющие» законы, в свою очередь, тоже могут быть отменены, что может приводить отмененные законы обратно в действующее состояние.

В соответствии с условием задачи, законы перечислены в порядке их принятия, и закон может отменить только один из предшествующих законов. Таким образом, закон может быть отменен только действующим законом с большим номером.

Рассмотрим закон, принятый последним. Он является действующим. Если он отменяет один из предыдущих законов, отметим его как отмененный. Продолжим рассмотрение законов в порядке, противоположном порядку их принятия. При этом если рассматриваемый закон не помечен как отмененный, запомним, что он является действующим, и, при необходимости, пометим как отмененный отменяемый им закон.

После рассмотрения всех законов, получим, что все законы, не помеченные как отмененные, являются действующими.

Рассмотрим работу данного алгоритма на примере (рис. а). Вначале рассматривается закон № 5, который отменяет закон № 3 (рис. б). Далее рассматриваются закон № 4, отменяющий закон № 2 (рис. в). Законы № 3 и № 2 не рассматриваются, как уже отмененные. Рассмотрение закона № 1 не приводит к изменению набора законов. Таким образом, действующими являются законы №№ 1, 4, 5.

1. declare	1. declare	1. declare
2. cancel 1	2. cancel 1	2.
3. declare	3.	3.
4. cancel 2	4. cancel 2	4. cancel 2
5. cancel 3	5. cancel 3	5. cancel 3

**а                      б                      в**  
**Пример работы алгоритма**

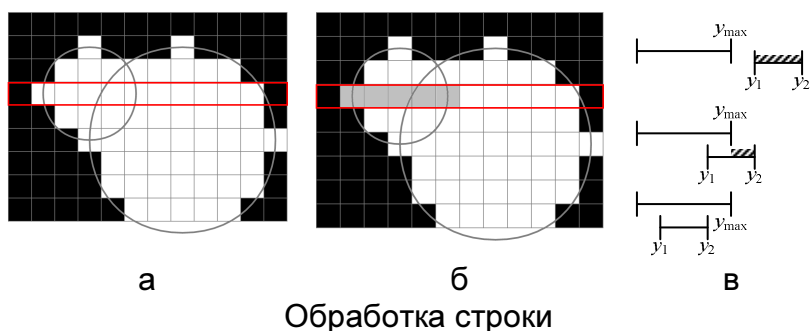
Приведенное решение имеет сложность по времени и по памяти  $O(n)$ , где  $n$  — общее количество законов.

### Задача С. Circles on a Screen

В данной задаче было необходимо определить суммарную площадь кругов, нарисованных на растровом экране. При этом части кругов могут накладываться друг на друга. При этом в области наложения нескольких кругов все пиксели являются закрашенными и учитываются один раз.

Отметим, что для любого пикселя можно определить, закрашен ли он за время  $O(n)$ , где  $n$  — количество кругов. Данный подход приводит к решению за  $O(WHn)$ , где  $W$  и  $H$  — ширина и высота экрана в пикселях соответственно, что не укладывается в ограничение по времени.

Рассмотрим строку пикселей с координатой, равной  $y_c$  (рис а.) Каждый круг либо не закрашивает ни одного пикселя из этой колонки, либо закрашивает на ней непрерывный отрезок  $(y - \lfloor \sqrt{r^2 - (x - x_c)^2} \rfloor, y + \lfloor \sqrt{r^2 - (x - x_c)^2} \rfloor)$ , где  $x, y, r$  — координаты центра и радиус круга (рис. б).



Обработка строки

Закрашиваемые отрезки могут перекрываться (рис. в), поэтому требуется найти их объединение. Рассмотрим отрезки в порядке возрастания координат их начала. При этом будем поддерживать  $y_{\max}$  — максимум из координат концов уже рассмотренных отрезков. Рассмотрим отрезок  $(y_1, y_2)$ .

- $y_{\max} < y_1$  (верхний случай). Новый отрезок не пересекается с уже рассмотренными, при этом, количество закрашиваемых пикселей равно  $y_2 - y_1 + 1$ .
- $y_1 \leq y_{\max} < y_2$  (средний случай). Закрашиваются новые пиксели в количестве  $y_2 - y_{\max}$ .
- $y_2 \leq y_{\max}$  (нижний случай). Рассматриваемый отрезок уже целиком закрашен.

Обновим  $y_{\max}$ :  $y_{\max} = \max(y_{\max}, y_2)$ .

Рассмотрев все отрезки, получим количество пикселей, закрашенных в колонке с координатой  $x_c$ . При этом, построение отрезков их объединение потребует  $O(n)$  времени, а сортировка  $O(n \log n)$ . Таким образом, рассмотрение одного столбца требует  $O(n \log n)$  времени.

Для решения задачи в целом — рассмотрим по очереди все столбцы экрана, просуммировав количество пикселей, закрашенных на них. Это решение имеет сложность по  $O(Wn \log n)$  по времени и  $O(n)$  по памяти, что укладывается в ограничения задачи.

### Задача D. Dragon's Question

В данной задаче требовалось найти натуральное число ровно из  $n$  цифр, делящееся на заданное число  $d$ , либо определить, что это невозможно.

Обозначим количество цифр в числе  $d$  через  $q$ . Так как все натуральные числа меньше  $d$  не делятся на  $d$ , при  $n < q$  требуемого числа не существует. Если  $n = q$ , то само  $d$  удовлетворяет условию. Если же  $n > q$ , то к  $d$  можно приписать  $n - q$  нулей, получив число требуемой длины.

Приведенное решение имеет сложность  $O(n)$  по времени и  $O(1)$  по памяти.

## Задача Е. Enigmatic Device

В данной задаче требовалось реализовать операции с отрезками списка чисел:

- возведение каждого числа на отрезке в квадрат;
- получение суммы чисел на отрезке.

В случае «наивной» реализации — то есть, непосредственного выполнения всех действий, сложность решения по времени составит  $O(mn)$ , где  $m$  — количество операций,  $n$  — количество чисел в списке. Такое время работы, при ограничениях, заданных в условии задачи, не укладывается в лимит времени. Таким образом, требуется более сложное решение.

Для быстрого подсчета суммы на отрезке обычно применяются деревья отрезков [1], которые позволяют производить изменение одиночного элемента и вычислять сумму на отрезке за время  $O(\log n)$ , требуя при этом  $O(n)$  памяти, где  $n$  — количество элементов в дереве отрезков.



а б  
Дерево отрезков (а) и нахождение суммы на отрезке (б)

При помощи техники «проталкивания» информации, можно реализовать дерево отрезков, с изменением элементов на отрезке по простому закону, например, установка равных значений всех элементов или увеличения всех элементов на заданное число. Однако, для операции «возвести все числа в квадрат» данная техника на первый взгляд не применима.

В процессе решения задачи было необходимо заметить то, что для любого  $a$ , последовательность чисел  $a \bmod 2010$ ,  $a^2 \bmod 2010$ ,  $(a^2)^2 \bmod 2010$ , ... периодична, с периодом 10 и предпериодом 2 (при этом для некоторых чисел, например 1, минимальный предпериод и период могут быть короче). Таким образом, при последовательном возведении каждого числа в квадрат по модулю 2010 может быть получено не более 12 различных чисел.

Это наблюдение позволяет решить данную задачу с применением техники «проталкивания» информации. При этом для каждого узла дерева  $t$  будем хранить следующие величины:

- $sum_{i_i}$  — суммы на отрезке для чисел ( $i = 0$ ), их квадратов ( $i = 1$ ), четвертых степеней ( $i = 2$ ) и так далее, всего 12 сумм;
- $q_t$  — информацию о том, сколько раз узел возводился в квадрат.

При обработке дерева сохраняется инвариант: сумма на отрезке, соответствующей вершине  $t$  равна  $sum_{i_{q_t}}$ .

Возведение в квадрат чисел на отрезке производится на основе техники «сверху вниз». Если текущий узел целиком входит в рассматриваемый отрезок, то для него  $q_t$  увеличивается на единицу. Если же узел только частично пересекается с рассматриваемым отрезком, то вначале производится «проталкивание»  $q_t$ , после чего осуществляется рекурсивный вызов для поддеревьев  $2t$  и  $2t+1$ . При этом после обработки поддеревьев требуется восстановить инвариант дерева. Для этого необходимо пересчитать  $sum_{i_i}$  для всех  $i$  по формуле  $sum_{i_i} = sum_{2t, i+q_{2t}} + sum_{2t+1, i+q_{2t+1}}$ , при этом используется то, что после «проталкивания»  $q_t=0$ .

Сумма чисел на отрезке считается обычным образом. Если текущий узел целиком входит в рассматриваемый отрезок, то в соответствии с инвариантом дерева, сумма для соответствующего отрезка равна  $sum_{i_{q_t}}$ . Если узел только частично пересекается с рассматриваемым отрезком, то снова требуется «протолкнуть»  $q_t$ , и осуществить рекурсивные вызовы. Отметим, что в этом случае после «проталкивания» также требуется восстанавливать инвариант дерева.

Приведенное решение имеет сложность  $O(m p \log n)$  по времени и  $O(p n)$  по памяти, где  $p$  — суммарная длина периода и предпериода (в данной задаче — 12).

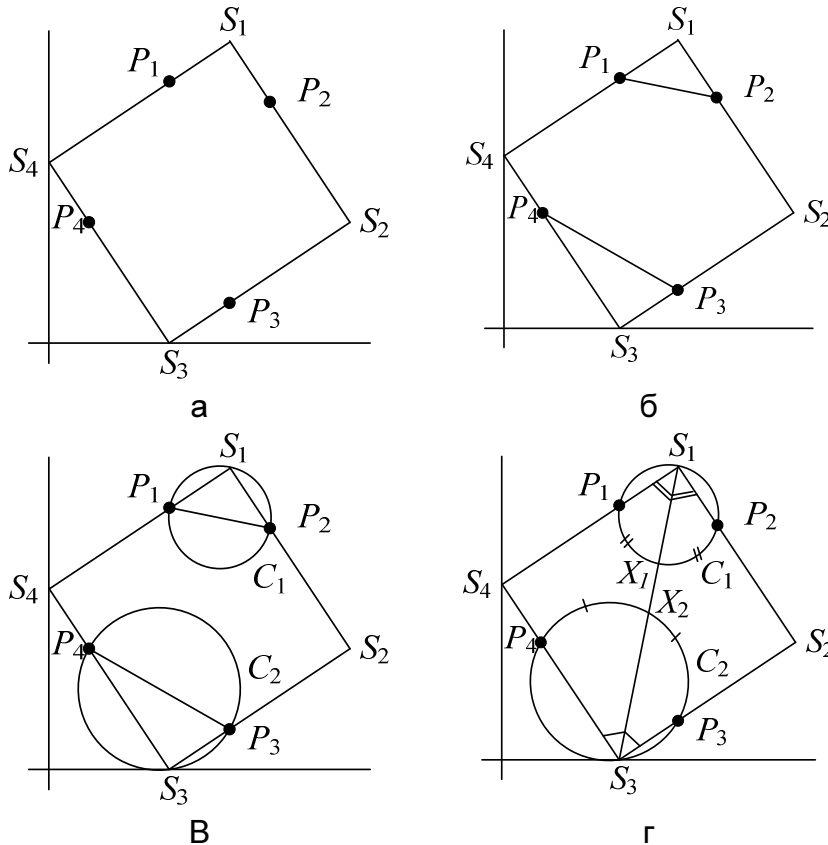
### Задача F. Four Points

В данной задаче было необходимо по четырем точкам на плоскости построить такой квадрат, чтобы на каждой его стороне лежало не менее одной точки.

У данной задачи есть два существенно разных решения: точное математическое и численное.

Вначале рассмотрим точное решение.

Пусть четыре точки  $P_1, P_2, P_3, P_4$  лежат на сторонах квадрата  $S_1, S_2, S_3, S_4$  (рис. а).



Построение квадрата по четырем точкам

Выберем из них две пары точек  $(P_1, P_2)$  и  $(P_3, P_4)$  так, чтобы точки из пары лежали на соседних сторонах и соединим отрезками точки в парах (рис. б). Построим на полученных отрезках  $P_1P_2$  и  $P_3P_4$ , как на диаметрах окружности  $C_1$  и  $C_2$  соответственно (рис. в). Отметим, что, так как углы  $P_1S_1P_2$  и  $P_3S_3P_4$  прямые, то точки  $S_1$  и  $S_3$  лежат на соответствующих окружностях. Проведем диагональ квадрата  $S_1S_3$ , она пересечет окружности  $C_1$  и  $C_2$  в точках  $X_1$  и  $X_2$  соответственно (рис. г). Отметим, что, так как диагональ квадрата является биссектрисой его углов, то  $\angle P_1S_1X_2 = \angle X_1S_1P_2$  и  $\angle P_3S_3X_2 = \angle X_2S_3P_4$ . Соответственно, равны пары стягиваемых этими углами дуг  $P_1X_1$  и  $X_1P_2$ , а также  $P_3X_2$  и  $X_2P_4$ . Таким образом,  $X_1$  и  $X_2$  являются серединами дуг  $P_1P_2$  и  $P_3P_4$ .

Это приводит к следующему решению задачи:

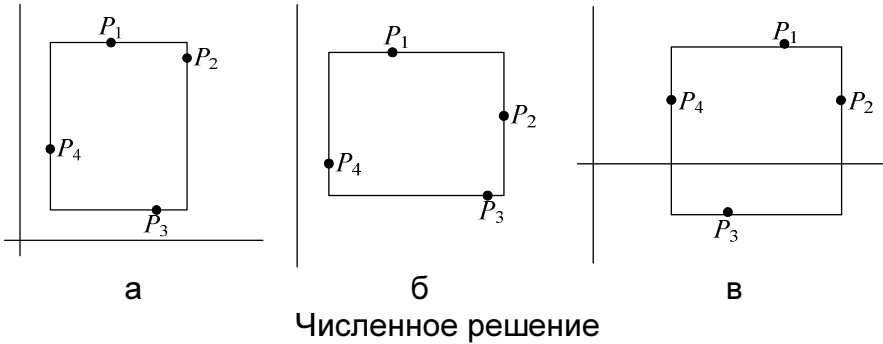
1. Выбрать пары точек  $(P_1, P_2)$  и  $(P_3, P_4)$ , при этом необходимо, перебрать все восемь возможных вариантов.
2. Построить на отрезках  $P_1P_2$  и  $P_3P_4$  как на диаметрах окружности  $C_1$  и  $C_2$ .
3. Найти точки  $X_1$  и  $X_2$  центры дуг  $P_1P_2$  и  $P_3P_4$ , при этом из двух возможных вариантов выбирается более близкий к противоположной паре точек.
4. Провести через точки  $X_1$  и  $X_2$  прямую, и найти точки  $S_1$  и  $S_2$  как точки пересечения этой прямой с окружностями  $C_1$  и  $C_2$ . При этом из двух возможных вариантов выбирается более удаленный от противоположной пары точек.
5. Построить точки  $S_2$  и  $S_3$  как пересечения пар прямых  $S_1P_2, S_3P_3$  и  $S_1P_2, S_3P_4$  соответственно.
6. Проверить, что точки  $P_1, P_2, P_3, P_4$  лежат на сторонах квадрата.

7. Если ни один из рассмотренных восьми случаев не дает правильного расположения точек, то квадрат построить невозможно.

Точное решение имеет сложность  $O(1)$  как по времени, так и по памяти.

Перейдем к рассмотрению численного решения.

Построим на точках  $P_1, P_2, P_3, P_4$  прямоугольник со сторонами, параллельными осям координат (рис. а). Пусть этот прямоугольник не является квадратом, и для определенности, у него высота больше ширины ( $h - w < 0$ ). Повернем точки на  $90^\circ$  градусов относительно начала координат и построим аналогичный прямоугольник (рис. б). За счет поворота у него высота будет меньше ширины ( $h - w > 0$ ). Так как поворот является непрерывным преобразованием, а высота и ширина прямоугольника непрерывно зависит от координат точек, то по теореме Больцано-Коши [2] существует такой угол поворота  $\alpha$  ( $0^\circ < \alpha < 90^\circ$ ), что  $h - w = 0$ , то есть построенный прямоугольник является квадратом.



Необходимый угол поворота  $\alpha$  можно найти двоичным поиском. Заметим, что, как и в случае точного решения, некоторые точки могут лежать внутри построенного квадрата. Тогда, требуется выбрать другой начальный угол поворота и повторить процедуру.

Отметим, что ни смотря на то, что численное решение формально имеет Точное решение имеет сложность  $O(1)$  и по времени, и по памяти, в случае неудачного выбора исходных углов поворота, решение может быть не найдено за отведенное время.

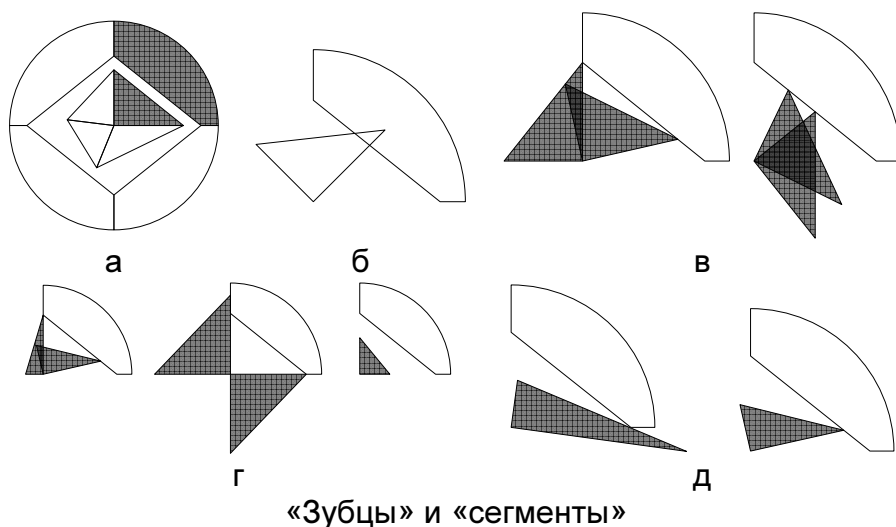
## Задача G. Grand Theft Auto Wheel

В данной задаче требовалось для каждого ключа из заданного набора определить, можно ли им открутить заданный болт. При этом формы ключей и головки болта были заданы звездчатыми многоугольниками.

В соответствии с условием задачи болт можно было открутить ключом, при выполнении двух условий:

- ключ помещается в головку болта;
- ключ не может сделать полный оборот внутри зафиксированного болта.

Рассмотрим один «зубец» болта и одну «сегмент» головки болта, как показано на рис. а. При этом «зубец» является треугольником, а «сегмент» — трапецией, у которой одна из сторон является дугой окружности.



Отметим, что при некоторых углах поворота ключа, часть «зубца» находится внутри «сегмента». Назовем такие углы запрещенными. В рассматриваемом примере зубец порождает два интервала запрещенных углов, как показано на рис. в. Однако, в отдельных случаях для пары «зубец»–«сегмент» может быть получен один или ноль интервалов запрещенных углов (рис. г). Так же следовало учесть, различные варианты касания (рис. д).

Рассмотрев все возможные пары «зубец»–«сегмент», получим множество запрещенных интервалов. Если оно пусто, то ключ, ни при каком угле поворота, не входит в зацепление с болтом. Если объединение множества запрещенных интервалов покрывает все возможные углы, то ключ невозможно вставить в болт. В противном случае, если оба приведенных условия не выполняются, то болт может быть отвернут ключом.

Для построения объединения множества интервалов на окружности можно их отсортировать по углу начала и далее действовать для углов по принципу, аналогичному принципу сметающей прямой.

Приведенное решение для каждого ключа имеет сложность по времени  $O(n m \log(n m))$ , где  $n$  — количество «сегментов» болта,  $m$  — количество зубцов ключа. Требуемое количество памяти —  $O(m+n)$ .

## Задача Н. Homo or Hetero?

В данной задаче требовалось определить, существуют ли в изменяющемся списке целых чисел после каждого изменения хотя бы два равных (гомогенное состояние) и хотя бы два различных числа (гетерогенное состояние).

Отметим тот факт, что порядок следования чисел несущественен для определения ответа. Таким образом, можно вместо списка рассматривать мультимножество, храня с каждым элементом количество его вхождений.

Отметим, что в гомогенном состоянии количество различных элементов меньше количества элементов в мультимножестве. А в гетерогенном состоянии — количество различных элементов больше одного. Таким образом, зная количество различных элементов после каждой операции и сравнивая его с единицей и общим количеством элементов, мы можем определить гомогенные и гетерогенные состояния списка.

Для поддержания мультимножества можно использовать хэш-таблицы или сбалансированные двоичные деревья поиска (например, АВЛ или красно-черные деревья). В первом случае амортизированное время обработки каждой операции будет  $O(1)$ , а во втором —  $O(\log n)$ , где  $n$  — количество операций.

Суммарное время обработки всех операций будет либо  $O(n)$  либо  $O(n \log n)$ , что удовлетворяет условиям задачи. Количество требуемой памяти в обоих случаях составляет  $O(n)$ .

## Задача I. Image Recognition

В данной задаче требовалось разработать алгоритм построения программы для робота, распознающего черно-белые изображения. При этом программа для робота должна была работать в худшем случае минимально возможное время.

Рассмотрим состояния, определяемые парой значений:

- клетка, в которой находится робот;
- множество изображений, которые требуется распознать.

Общее число таких состояний —  $WH2^n$ , где  $W, H$  — ширина и высота изображения,  $n$  — число изображений.

Будем осуществлять обход в ширину [3] по состояниям. Начнем с состояний, в которых множество изображений, которые требуется распознать, содержит ровно один элемент. В таких состояниях можно выдать команду «изображение распознано» и завершить программу. Для таких состояний время распознавания равно нулю.

Передвижение в соседнюю клетку требует одну единицу времени, при этом множество изображений, которые требуется распознать, не изменяется, а время распознавания нового состояния на единицу больше, чем время распознавания старого.

При выполнении ветвления, робот из одного состояния переходит сразу в два — по одному для каждого исхода сравнения. При этом множество изображений, которые требуются распознать в каждом из новых состояний, в общем случае, будет свое, так как в части изображений текущая клетка была черной, а в части — белой. Для того, что бы можно было осуществить ветвление требуется, чтобы оба новых состояния были уже достигнуты, при этом время распознавания состояния, в котором осуществлено ветвление рано максимуму из времен распознавания новых состояний.

Таким образом, ветвления можно «конструировать» путем рассмотрения возможных надмножеств множества изображений, которые требуется распознать. При этом вычисляется парное состояние, и если оно уже было достигнуто, то состояние, соответствующее надмножеству также является достижимым, а время его распознавания равно времени распознавания рассматриваемого состояния.

Программа робота в целом, может быть построена после достижения состояния (левый верхний угол, множество всех изображений).

Приведенное решение имеет сложность  $O(WH3^n)$  по времени и по памяти, при этом тройка в основании степени возникает по причине рассмотрения надмножеств, при обработке ветвлений.

### Задача J. Jealous Numbers

В данной задаче требовалось определить количество целых чисел из диапазона  $[a, b]$ , для которых простое число  $p$  доминирует над простым числом  $q$ . При этом, говорится что  $p$  доминирует над  $q$  для числа  $n$  если существует такое натуральное  $k$ , что  $n$  делится на  $p^k$  и не делится на  $q^k$ .

Для решения этой задачи подсчитаем  $F_{ij}$  — количество таких целых чисел  $n$  в диапазоне  $[a, b]$ , что  $n$  делится на  $p^i q^j$ , и не делится на  $p^{i+1} q^j$  и  $p^i q^{j+1}$ .

Пусть  $G_m$  — количество чисел в диапазоне  $[a, b]$  делящихся на заданное число  $m$ , тогда  $G_m = b / m - (a - 1) / m$ . При этом  $F_{ij} = G_{p^i q^j} - G_{p^{i+1} q^j} - G_{p^i q^{j+1}} + G_{p^{i+1} q^{j+1}}$ , так как вычитая количество чисел делящихся на  $p^{i+1} q^j$  и  $p^i q^{j+1}$  мы дважды вычли количество чисел делящихся на  $p^{i+1} q^{j+1}$ .

Ответом на задачу является  $\sum_{i=1}^{\log_2 b} \sum_{j=0}^{i-1} F_{ij}$ , при этом верхняя граница первой суммы взята из

соображения того, что  $b \leq p^{\log_2 b}$  и  $b \leq q^{\log_2 b}$ , то есть все неучтенные  $F_{ij}$  равны нулю.

Приведенное решение имеет сложность  $O((\log_2 b)^2)$  по времени и  $O(1)$  по памяти.

## Задача К. Kripke Model

В данной задаче требовалось верифицировать формулу темпоральной логики  $E(xU(AGy))$ , где  $x$  и  $y$  некоторые атомарные высказывания.

Переформулировав выражение  $AGy$  на русском языке мы получим условие «такие вершины, что каждый путь, начинающийся в одной из этих вершин проходит только по вершинам, для которых выполнено атомарное высказывание  $y$ ». Упростив это условие получим: «такие вершины, что из них достижимы только вершины, для которых выполнено атомарное высказывание  $y$ ». Назовем такие вершины вершинами первого типа.

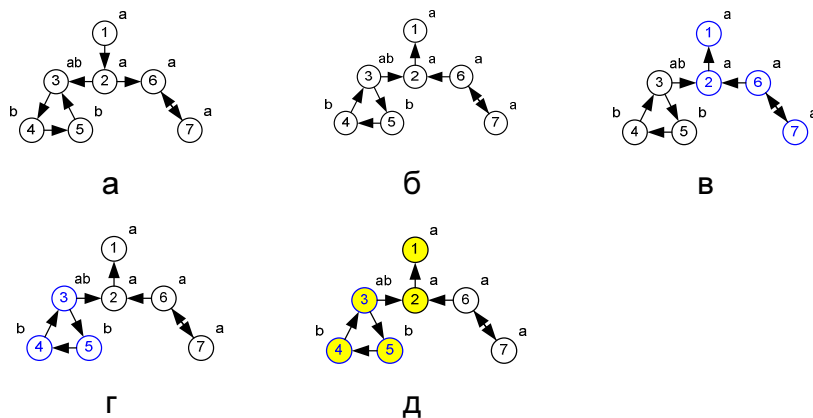
Построим множество вершин, являющихся дополнением к множеству вершин первого типа. Из каждой такой вершины достижима вершина, для которой не выполняется атомарное высказывание  $y$ . Таким образом, вершины, достижимые в обратном графе из вершин, для которых не выполняется атомарное высказывание  $y$ , являются искомым дополнением. (Обратным графом называют граф, содержащий все вершины исходного графа, и все дуги исходного графа, направленные в обратную сторону.) Таким образом, дополнение к вершинам первого типа можно найти обходом в глубину в обратном графе из вершин, для которых не выполняется атомарное высказывание  $y$ .

Теперь переформулируем выражение  $E(xU(AGy))$  на русском языке, используя определение вершин первого типа: «требуется найти такие вершины, что в них начинается как минимум один путь ( $E$ ), такой, что хотя бы для одна вершина этого пути является вершиной первого типа и во всех предыдущих вершинах этого пути выполняется атомарное высказывание  $x$  ( $U$ )». Упростив это условие, получим «требуется найти такие вершины, что из них достижимы вершины первого типа по таким путям, что для всех их промежуточных вершин выполняется атомарное высказывание  $x$ ». Назовем такие вершины вершинами второго типа.

Таким образом, вершина имеет второй тип тогда и только тогда, когда она достижима в обратном графе из какой-нибудь вершины первого типа по пути, для всех промежуточных вершин которого выполняется атомарное высказывание  $x$ . Эти вершины можно найти обходом в глубину в обратном графе из вершин первого типа, проходя только через вершины, для которых выполняется атомарное высказывание  $x$ .

Ответом на задачу является множество вершин второго типа. Таким образом, задача решается двумя последовательными обходами в глубину обратного графа.

Рассмотрим работу алгоритма на примере (рис. а). Так как оба обхода осуществляются в обратном графе, сразу обратим все ребра (рис. б). Осуществим поиск в глубину, начиная с вершин, не помеченных  $b$ : 1, 2, 6, 7 (рис. в.). Так как другие вершины из не достижимы из указанных, то  $\{1, 2, 6, 7\}$  является дополнением к множеству вершин первого типа:  $\{3, 4, 5\}$  (рис. г.). Осуществляя обход в глубину из вершин первого типа, проходя только вершины, помеченные символом  $a$ , получим множество вершин второго типа:  $\{1, 2, 3, 4, 5\}$ .



Пример работы алгоритма

Приведенное решение имеет сложность  $O(V+E)$  по времени и по памяти, где  $V$  — количество вершин графа, а  $E$  — количество его ребер.

## Список литературы

1. Дерево отрезков / [http://ru.wikipedia.org/wiki/Дерево\\_отрезков](http://ru.wikipedia.org/wiki/Дерево_отрезков)
2. Фихтенгольц Г. М. Курс дифференциального и интегрального исчисления. Том 1. М.: Физматлит, 2001, 616 с.
3. Кормен Т. Х., Лейзерсон Ч. И., Ривест Р. Л., Штайн К. Алгоритмы: построение и анализ. Первое издание, раздел 22.2. М.: МЦНМО. 2001. 960 с.