Открытая олимпиада школьников по программированию 2017

Разбор задач

Задача Cheesecake. «Кукушки»

Автор задачи: Максим Ахмедов, МГУ
Разработчик задачи: Роман Горбунов, МГУ

Формальная постановка

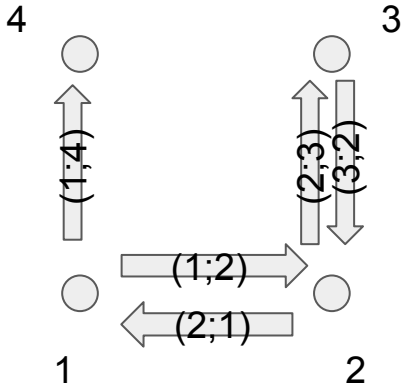
- Дано первоначальное расположение яиц на дереве
- Требуется отвечать на запросы трёх типов:
  - Можно ли теоретически добавить данное яйцо в дерево?
  - Можно ли действительно добавить данное яйцо в дерево? Если да, то добавить его.
  - Сказать количество различных пар (яйцо, гнездо), которые описывают возможное корректное добавление.

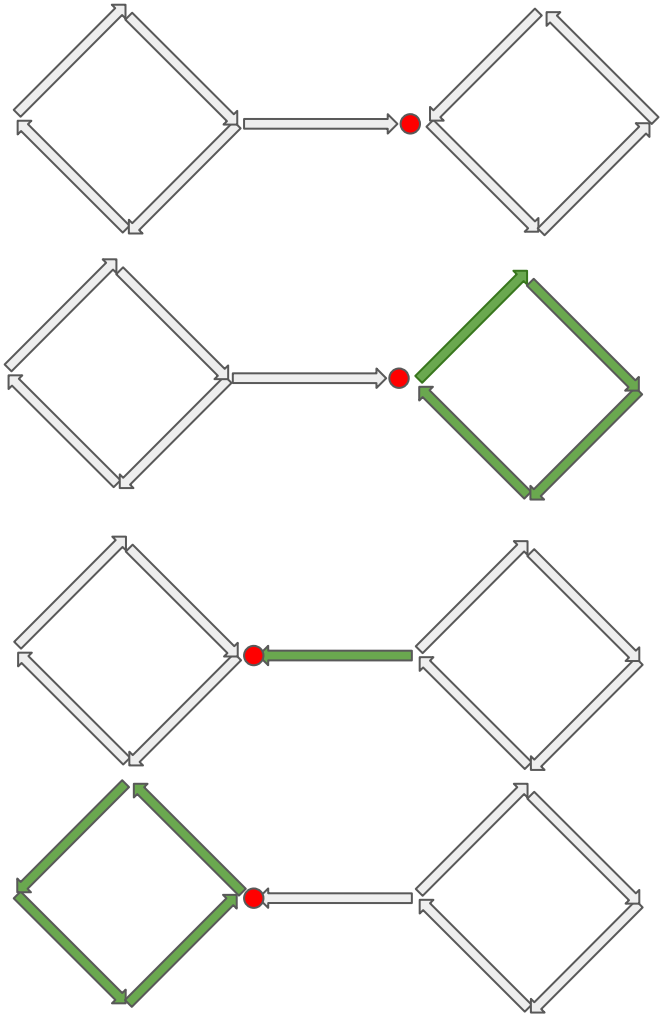
Решение на 13 баллов

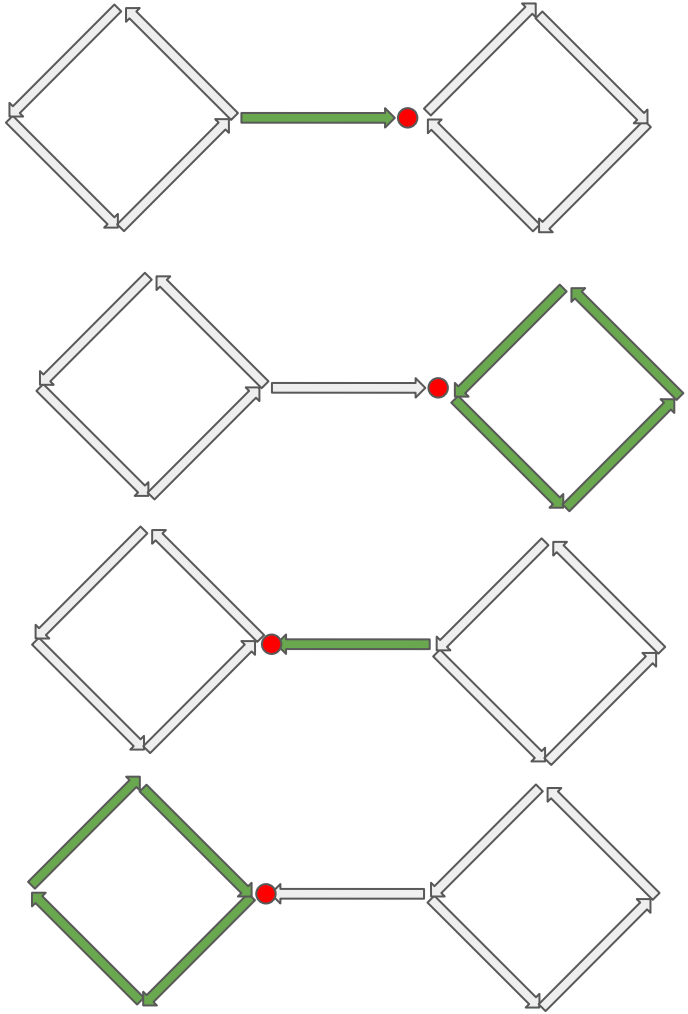
- Можно для каждого гнезда хранить яйцо, которое лежит в нём, и моделировать процесс
- Для корректного моделирования требовалось заметить, что одну вершину мы можем посетить 2 раза
- Будем моделировать $2n$ итераций процесса, если он не завершится за это время, то он не завершится никогда
- Полученное решение работает за $O(T_1 * n) = O(n^2)$

Решение на 13 баллов

- Например: пусть у нас есть яйца (1, 2) (2, 3) (2, 3) и мы хотим добавить яйцо (1, 4)
- Тогда (1, 4) поместим в гнездо 1
- Яйцо (1, 2) - в 2
- (2, 3) и (2, 3) поменяются местами и опять сдвинут (1, 2) в гнездо 1
- В итоге яйцо (1, 2) и (1, 4) мы рассмотрим 2 раза!







Ключевые идеи

- Сопоставим каждому из яиц ребром в графе
- Рассмотрим компоненту связности: каждому ребру в компоненте соответствует своя вершина, поэтому рёбер не больше, чем вершин
- Значит количество рёбер в компоненте размера *n* либо *n*, либо *n-1* (меньше в связной компоненте быть не может)

Ключевые идеи

- Назовём компоненту из *n - 1* ребер компонентой первого типа, а и из n рёбер -- второго типа
- Яйцо может быть добавлено, если соответствующее ему ребро соединяет компоненту первого типа с любой другой компонентой, либо соединяет две вершины компоненты первого типа

Решение на 27 баллов

- Для ответа на запрос третьего типа поймём для каждой вершины тип компоненты, в которой она находится
- Далее переберём все пары вершин
- Пользуемся критерием, когда ребро можно добавить, сформулированным ранее
- Получаем решение для таких запросов за $O(T_1 \cdot n + T_2 \cdot n^2) = O(T_1 \cdot n + n^2) = O(n^2)$

Решение на 39 баллов

- Храним граф и поддерживаем две величины: количество вершин компонент первого типа (*cnt1*) и количество вершин компонент второго типа (*cnt2*)
- Нетрудно понять, что ответом будет являться число

$$2 \cdot cnt_1 \cdot cnt_2 + cnt_1 \cdot (cnt_1 - 1)$$

- При перестроении графа пересчитываем *cnt1* и *cnt2*, учитывая, какие вершины поменяли тип своей компоненты
- Получаем решение за $O(T_1 \cdot n + T_2 \cdot n + T_3 \cdot n) = O(n^2)$

Решение на 51 балл

- Определим для каждой вершины тип компоненты, в которой она находится.
- Теперь мы за O(1) можем определить по ребру, какие можно ли его провести
- Суммарное время работы:
  $O(T_1 \cdot 1) = O(n)$

Решение на 69 баллов

- Определим для каждой вершины тип компоненты, в которой она находится
- Также определим количество вершин, находящихся в компонентах первого (*cnt1*) и второго типа (*cnt2*) и используем ту же формулу:

$$2 \cdot cnt_1 \cdot cnt_2 + cnt_1 \cdot (cnt_1 - 1)$$

- Получили решение за $O(T_1 \cdot 1 + T_2 \cdot n) = O(n)$

Решение на 100 баллов

- Чтобы поддерживать для каждой вершины тип её компоненты, а также количество вершин первого и второго типов воспользуемся системой непересекающихся множеств
- Дополнительно ассоциируем с каждым множеством количество рёбер в нём
- При добавлении ребра внутри компоненты первого типа увеличиваем количество рёбер внутри множества, при соединении двух разных компонент суммируем их количества рёбер плюс 1

Решение на 100 баллов

- Мы в любой момент времени знаем всю нужную информацию для того, чтобы отвечать как на запросы первого и второго типов, так и на запрос третьего типа
- Получаем решение за:
  $O(T_1 \cdot \alpha + T_2 \cdot \alpha + T_3 \cdot 1) = O(n \cdot dsu)$

Задача Halva. «Глеб и два числа»

Автор задачи: Максим Ахмедов, МГУ
Разработчик задачи: Глеб Побегайло, МГУ

Формальная постановка

- Даны два целых числа *l*, *r*.
- Требуется подобрать такие целые *a*, *b*, что *l* <= *a* <= *b* <= *r*,
  и расстояние Хэмминга между числами *a* и *b* максимально.

Решение на 19 баллов

- Ограничения позволяют перебрать все возможные пары чисел a, b удовлетворяющие l <= a <= b <= r,
- Выбираем пару с максимальным расстоянием Хэмминга.
- Полученное решение работает за $O((r - l)^2 * \lg(r))$

Решение на 40 баллов

- Заметим, что выгодно рассматривать пары соседних чисел
- Будем рассматривать только пары соседних чисел *a* и *b*.
- Перебираем, считаем расстояние Хэмминга, выбираем самую лучшую пару.
- Полученное решение работает за $O((r - l) * \lg(r))$
- Существует также решение использующее идею динамического программирования.

Ключевая идея

- Пусть k - номер самого старшего разряда (при нумерации разрядов с единицы) в котором числа l и r различаются. Утверждается, что число k и будет максимальным возможным расстоянием Хэмминга.
- Доказательство: можно рассмотреть пару чисел $(x, x + 1)$, с наименьшим возможным x, в которой числа отличаются в k-ом разряде. Получили пару отличающуюся в k разрядах, больше нельзя.
- Итого получаем решение за $O(\lg(r))$

Решение на 72-100 баллов

- В зависимости от способа хранения числа (в числовом или строковом типе) получаем 72-100 баллов.

Задача Strudel. «Эффективное планирование»

Автор задачи: жюри олимпиады
Разработчик задачи: Владислав Макеев, МГУ

Формальная постановка

- Дан массив: 1, 2, 3, …., *m*
- Поступают запросы удалить элементы стоящие на позициях *l*, *l* + 2, *l* + 4, …., *r*
- Оставшиеся элементы сдвинуть в начало, чтобы не оставалось свободных мест
- Для каждого запроса вывести самый левый и самый правый удаленный элемент в изначальной нумерации.

Решение на 10 или 19 баллов

- Моделируем требуемое поведение.
- Сложность решения – $O(nm)$
- В зависимости от аккуратно реализации и выбора языка программирования можно получить 10 или 19 баллов

Решение на 32 балла

- Удалять элемент из массива можно быстрее, чем за линейное время
- Воспользуемся для этой операции деревом отрезков или декартовым деревом
- Получим сложность $O(m \log m)$

Решение на 64 балла

- Пусть мы знаем позицию элемента после *i*-го запроса
- Несложно пересчитать его позицию до *i*-го запроса
- Обрабатывая *i*-й запрос, "проматываем" его границы от текущего состояния массива до первоначального
- Возвращая удалённые очередным запросом элементы в массив, пересчитываем, как меняются индексы границ *i*-го запроса
- Получаем решение за $O(n^2)$
- Альтернативный вариант: бинпоиск по ответу, то же самое в прямом направлении и $O(n^2 \log n)$

Решение на 100 баллов

- Воспользуемся неявным деревом отрезков или декартовым деревом.
- Необходимо поддерживать следующие операции:
  1. Удалить каждый второй неудаленный на отрезке
  2. Протолкнуть модификатор от операции типа 1.
  3. Сказать, сколько неудаленных элементов на отрезке.
- Будем хранить log *m* последних операций вида 1 в каждой вершине дерева отрезков/декартова дерева.
- 2 и 3 делаются стандартным образом.
- Сложность $O(n \log^2 m)$

Решение на 100 баллов v.2

- Будем хранить неудаленные элементы как объединение непересекающихся арифметических прогрессий
- Например: [1, 5, 9], [10, 11, 12, 13, 14], [16, 18], [23]
- Будем изменять все арифметические прогрессии, которые затрагивает данный запрос

Решение на 100 баллов v.2

- Заметим, что каждый раз, когда мы задеваем какую-то прогрессию, её шаг увеличивается вдвое
- Если задеть прогрессию более, чем log m раз, она становится тривиальной
- Итого, если хранить эти прогрессии в декартовом дереве, получаем сложность $O(n \log m + n \log n)$

Задача Tiramisu. «Ваня и куртки»

Автор задачи: Михаил Тихомиров, МФТИ
Разработчик задачи: Олег Мингалёв, МАИ

Формальная постановка

- Даны температуры *N* последовательных дней
- Дано *M* курток, для каждой указаны ограничения на температуру
- Требуется для каждого дня выбрать такую куртку, что
  - Соблюдаются температурные ограничения
  - Никакая куртка не используется два дня подряд

Решение на 20 баллов

- Переберём все возможные варианты сопоставления дням курток и проверим на корректность
- Асимптотика решения $O(M^N N)$

Решение на 39 баллов

- Динамическое программирование
- *dp*[*i*][*j*] = существует ли решение для первых *i* дней, если в *i*-й день используется куртка *j*

Решение на 39 баллов

- Динамическое программирование
- *dp*[*i*][*j*] = существует ли решение для первых *i* дней, если в *i*-й день используется куртка *j*
- *dp*[*i*][*j*] =
  - `False`, если день *i* не подходит под температурные ограничения куртки *j*, иначе
  - `True`, если *i* = 0, иначе
  - `True`, если ∃ *k* != *j*: *dp*[*i-1*][*k*] = `True`
  - иначе `False`

Решение на 39 баллов

- *dp*[*i*][*j*] =
  - `False`, если день *i* не подходит под температурные ограничения куртки *j*, иначе
  - `True`, если *i* = 0, иначе
  - `True`, если ∃ *k* != *j*: *dp*[*i-1*][*k*] = `True`, иначе
  - `False`
- Сохраняем переходы для восстановления ответа
- Асимптотика решения $O(NM^2)$

Решение на 60 баллов

- Скажем, что куртка *j доступна* в день *i*, если она удовлетворяет температурным условиям
- Если в какой-то день нет доступной куртки, то решения нет

Решение на 60 баллов

- Скажем, что куртка j *доступна* в день i, если она удовлетворяет температурным условиям
- Если в какой-то день нет доступной куртки, то решения нет
- Рассмотрим куртки, доступные в день *i*. Заметим, что
  - Если в предыдущий день больше одной доступной куртки, то *dp*[*i*][*j*] = `True` для всех *j*, которые доступны в день *i*

Решение на 60 баллов

- Рассмотрим куртки, доступные в день *i*. Заметим, что
  - Если в предыдущий день больше одной доступной куртки, то *dp*[*i*][*j*] = `True` для всех *j*, которые доступны в день *i*
  - Если в предыдущий день всего одна доступная куртка *k*, то *dp*[*i*][*j*] = `True` для всех *j* != *k*, которые доступны в день *i*
- Асимптотика решения $O(NM)$

Решение на 100 баллов

- Покажем, что для каждого дня достаточно рассмотреть **не больше трёх** доступных курток
- Зафиксируем для каждого дня три доступные куртки, являющиеся *избранными*
- Рассмотрим оптимальный ответ: если в нем есть хотя бы одна куртка, не являющаяся *избранной*, то заменим ее на одну из *избранных*

- Это всегда возможно, так как у дня не больше двух соседей

Решение на 100 баллов

- Научимся для каждого дня искать не больше трёх доступных курток
- Массив событий
  - Для каждого дня i добавим тройку (t_i, 0, i)
  - Для каждой куртки j добавим (l_j, -1, j) и (r_j, +1, j)
  - Отсортируем

Решение на 100 баллов

- Массив событий
  - Для каждого дня i добавим тройку (t_i, 0, i)
  - Для каждой куртки j добавим (l_j, -1, j) и (r_j, +1, j)
  - Отсортируем
- Проход по массиву событий
  - (l_j, -1, j): добавим в множество курток куртку j
  - (r_j, -1, j): удалим из множества курток куртку j
  - (t_i, 0, i): сохраним до трёх курток из множества курток как доступные в день i

Решение на 100 баллов

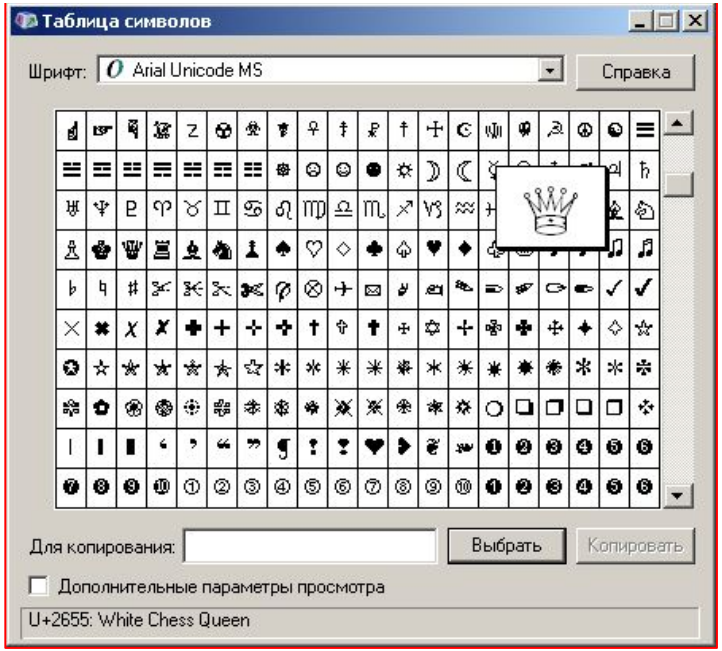
- Новая динамика: *dp*[*i*][*j*] = существует ли решение для первых *i* дней, если в *i*-ый день используется *j*-я доступная куртка
- *j* не превосходит 3
- Асимптотика решения $O((N+M)\log(N+M))$

Бонус

- Почему недостаточно двух курток?

```
4  3
1  3  2  1
1  3
2  3
3  3
```

Задача Eclair. «Открытая олимпиада по дизайну»



Автор и разработчик задачи: Максим Ахмедов, МГУ

Формальная постановка

- Дан набор из *n* слов, от которых известны только длины
- Слова должны следовать в лексикографическом порядке
- Требуется найти минимальный алфавит, при котором это возможно

Решение на 11 баллов

- Известно, что длины слов одинаковые и равны *l*
- Предположим, что алфавит состоит из *a* символов
- Тогда различных слов над таким алфавитом всего x^l.
- Найдём минимальное *x*, такой что $x^l >= n$. Нам нужен хотя бы такой алфавит, потому что иначе не хватит слов.
- Ровно такого алфавита хватит: берём первые *n* из существующих a^l слов в лексикографическом порядке.
- Ответ: $\lceil n^{1/l} \rceil$, сложность решения: любая

Решение на 18 баллов

- Заметим, что алфавита из n символов всегда хватает
- Переберём алфавит от 1 до n, предположим, что алфавит размера x
- Сгенерируем все слова всех длин <= 10 над алфавитом (их не более $10^5 + 9^5 + \ldots + 1^5 <= 10^6$)
- Выбираем в качестве каждого следующего слова лексикографически минимальное возможное (большее предыдущего) в списке слов длины l_i
- Сложность решения: $O(n \cdot n^l)$

Решение на 38 баллов

- Научимся за линейное время от суммарного размера всех слов проверять, что фиксированный алфавит подходит
- Генерируем каждое следующее слово как минимальное лексикографически, большее последнего слова в списке
- Если большего слова такой длины не существует, значит алфавита не хватает, пробуем с б**о**льшим алфавитом

Решение на 38 баллов

- Будем считать, что алфавит начинается с *0, 1, 2, ...*
- Если новое слово длиннее старого, дописываем буквы *0*
  x = 5, l_{i-1} = 7, l_i = 10: 0231444 -> 0231444**000**
- Если короче, укорачиваем до l_i и генерим следующее лексикографически слово такой же длины стандартным алгоритмом
  x = 5, l_{i-1} = *9*, l_i = *7:* 023144410 -> 023**2000**
- При фиксированном алфавите проверяем за $O(nl)$, итоговая сложность решения – $O(n^2l)$.

Решение на 58 баллов

- Заметим, что алфавита до какого-то критического момента не хватает, а после него всегда хватает
- Значит, минимальный алфавит, которого хватает, можно искать бинарным поиском
- Получаем решение за сложность $O(nl \log n)$

Решение на 79 или 100 баллов

- Надо избавиться от линейного времени в переходе от слова к слову, для этого надо как-то неявно хранить слово, чтобы уметь быстро сокращать/удлинять его и переходить от слова к следующему лексикографически
- Можно воспользоваться структурами данных для быстрых массовых операций: дерево отрезков позволяет присваивать на отрезке и находить первый справа элемент, меньший x - 1
- Обычное ДО – 79 баллов, неявное – 100
- Сложность: $O(n \log n \log l)$

Простая структура данных

- Будем хранить запись текущего слова в виде объединения отрезков из одинаковых букв, отрезки храним парами (длина, буква) в стеке
- Расширить слово – дописать в конец отрезок из букв 0
- [222][33][1] -> [222][33][1][**0000**]
- Сократить слово – выкинуть сколько-то последних элементов стека и, возможно, сократить один
- [1111][3~~3~~][~~444~~][~~22~~] -> [1111][3]

Простая структура данных

- Перейти к следующему лексикографически: если последний отрезок состоит не из буквы x-1, отщепляем от него последнюю букву и увеличиваем на 1
  x = 5, [222][11][333] -> [222][11][33][**4**]
- Иначе превращаем его в отрезок из 0, отщепляем от предпоследнего отрезка букву и увеличиваем её на 1
  x = 5, [222][11][444] -> [222][1][**2**][**000**]
  x = 5, [222][1][444] -> [222][**2**][**000**] -> [2222][000]
- Склеиваем соседние одинаковые отрезки (их получится не более одной пары)

Простая структура данных

- Все операции работают за O(1) кроме сокращения строки, но она также работает за амортизированные O(1), так как каждый отрезок один раз добавляется в стек и не более одного раза из него исчезает
- Сложность получившегося решения – O(n log n)

Message 45 [Edit]

Сообщение во время первого тура

Clar ID:	45
Clar UUID:	b4f93308-ce38-478a-a7de-ca19f568fd88
Flags:	
Time:	2017/03/10 13:39:46
Duration:	2:39:46
IP address:	83.220.239.182
Size:	186
Sender:	**judges** (open-judge-5)
To:	Сомкина Евгения (11) (81845)
In reply to:	44 (f478d0d6-edf3-46bc-9e8f-7c40b5aa19b7)
Locale code:	1
Subject:	Названия задач

```
Subject: Re: Названия задач

> Можно заказать маффин на завтра, пожалуйста?
```

Message 45 [Edit]

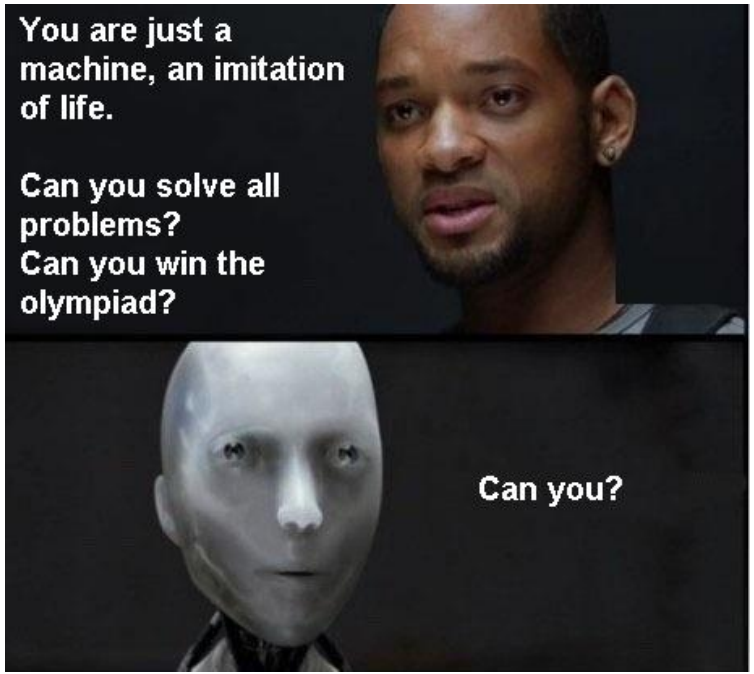
Clar ID:	45
Clar UUID:	b4f93308-ce38-478a-a7de-ca19f568fd88
Flags:	
Time:	2017/03/10 13:39:46
Duration:	2:39:46
IP address:	83.220.239.182
Size:	186
Sender:	**judges** (open-judge-5)
To:	Сомкина Евгения (11) (81845)
In reply to:	44 (f478d0d6-edf3-46bc-9e8f-7c40b5aa19b7)
Locale code:	1
Subject:	Названия задач

```
Subject: Re: Названия задач

> Можно заказать маффин на завтра, пожалуйста?

Вжух, вы истратили своё желание!
```

Задача Muffin. «Робот в поле»



Автор задачи: Михаил Тихомиров, МФТИ

Разработчик задачи: Александр Проскурин, МФТИ

Формальная постановка

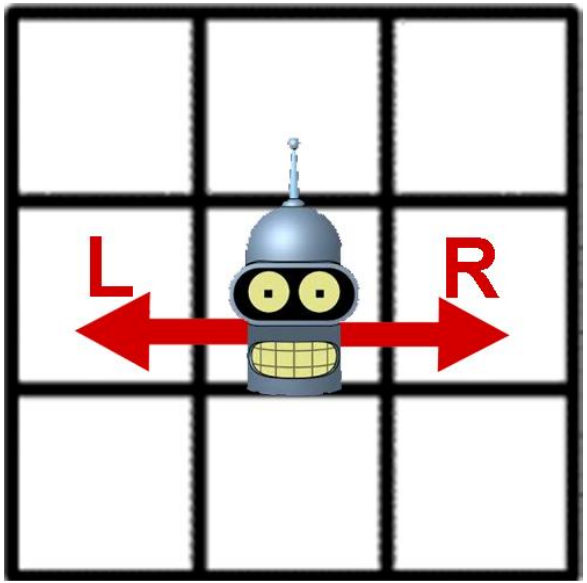
- Дана строка длины *n*, состоящая из символов «L», «U», «R», «D», обозначающих команды движения влево, вверх, вправо и вниз
- В произвольный момент можно перестать выполнять команды определённого типа
- Требуется так выбрать эти моменты времени, чтобы дойти до заданной точки *(x, y)*, либо сказать, что это невозможно

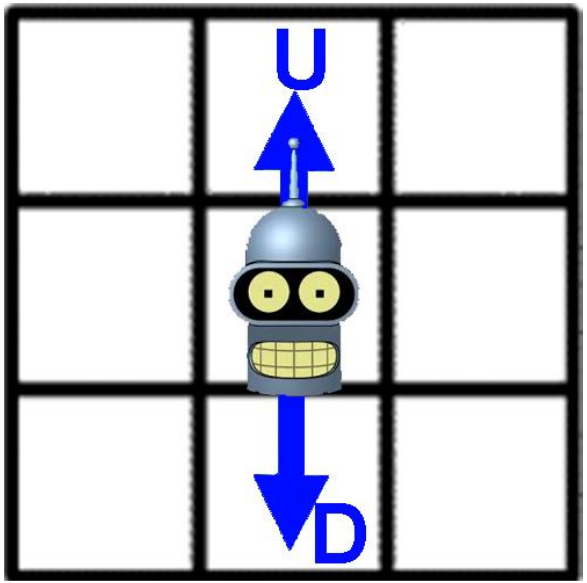
Решение на 32 балла

- Переберём моменты времени, в которые нужно перестать выполнять команды каждого типа
- После этого проверим, дойдет ли робот после выполнения команд до точки *(x, y)*
- Решение за $O(n^5)$

Ключевая идея

- Заметим, что команды, влияющие на разные координаты, можно рассматривать отдельно



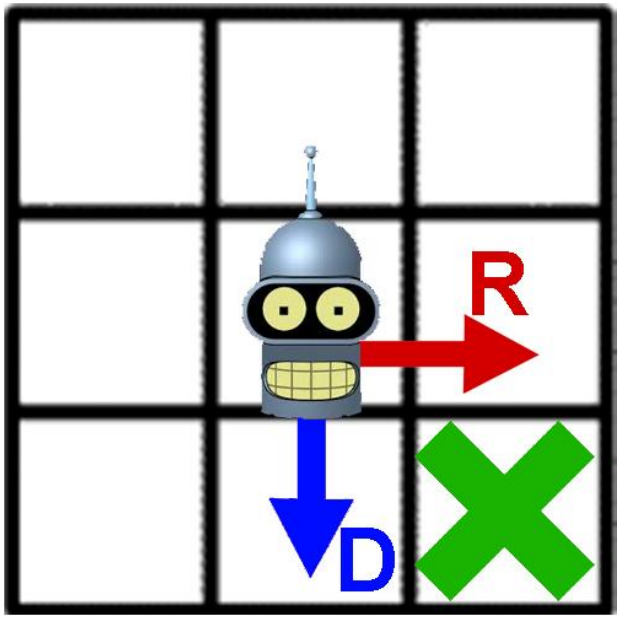


Решение на 61 балл

- Найдем ответ для первой координаты
- Для этого переберём позицию в строке, в которой нужно прекратить использовать команду «L»
- Таким образом мы зафиксировали сдвиг влево
- Переберём, сколько первых команд «R» нужно выполнить, чтобы дойти до нужной координаты x
- Аналогично найдём ответ для второй координаты
- Решение за $O(n^2)$

Следующая идея

- Заметим, что одну из команд для каждой координаты можно вообще не использовать



Решение на 100 баллов

- Найдём ответ для первой координаты
- Если $x = 0$, то $ans_L = ans_R = 0$
- Если $x > 0$, то $ans_L = 0$, а чтобы найти ans_R, переберём, сколько первых команд «R» нужно выполнить
- Если $x < 0$, то $ans_R = 0$, а чтобы найти ans_L, переберём, сколько первых команд «L» нужно выполнить
- Аналогично найдём ответы ans_U и ans_D для второй координаты
- Решение за $O(n)$

Задача Napoleon. «Застройка мегаполиса»

Автор задачи: Максим Ахмедов, МГУ
Разработчик задачи: Игорь Краскевич, НИУ ВШЭ

Формальная постановка

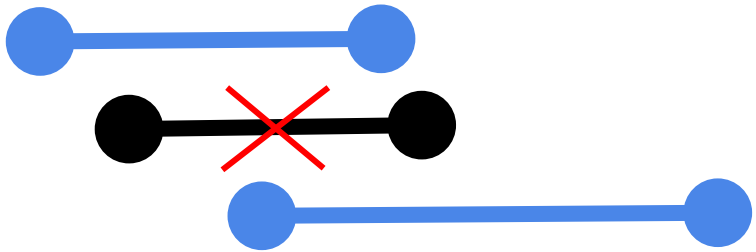
- Дан набор отрезков [l[i], r[i]] на прямой, у каждого есть какая-то толщина h[i]
- Нужно выбрать их подмножество так, чтобы заданный отрезок был полностью покрыт
- При этом нужно минимизировать толщину покрытия

Решение на 8 баллов

- Переберем все подмножества отрезков
- Для каждого их них проверим, что оно покрывает заданный отрезок целиком
- Если это так, то посчитаем его толщину и обновим ответ
- Сложность такого решения $O(2^n \cdot poly(n))$

Решение на 15 баллов

- Если толщина всех отрезков 1, то ответ всегда -1, 1 или 2
- Почему это так? Пусть какая-то точка покрыта тремя отрезками. В этом случае хотя бы один из трех отрезков можно выкинуть



Решение на 15 баллов

- Проверим, можно ли получить ответ 1. Для этого построим ориентированный граф, добавляя ребро из одного отрезка в другой, если они состыковываются, но не перекрываются, и найдем в нем путь
- Если ответ не 1, то проверим, можно ли вообще покрыть весь отрезок. Снова построим граф (на этот раз и с перекрывающимися отрезками)
- Граф можно строить наивно, перебирая все пары отрезков. Решение за $O(N^2)$

Решение на 31 балл (толщина 1)

- Будем использовать те же идеи, что и в предыдущем пункте
- Заметим, что отрезки можно добавлять в порядке возрастания их правых концов и хранить самую правую достижимую точку
- Тогда, после сортировки, достаточно одного линейного прохода
- Такое решение работает за $O(N \log N)$

Решение на ~~44~~ 0 баллов

- Отсортируем отрезки по их правому концу
- Пусть d[i] — минимальная толщина такого покрытия, что отрезок i в нем самый правый, и все левее него уже покрыто
- Переходы: d[i] = min(max(d[j], h[i] + h[j])) по всем j, таким, что l[j] <= l[i] <= r[j] < r[i]
- d[i] = min(d[i], min(max(d[j], h[i])) по всем j, таким, что r[j] = l[i] - 1) в случае соприкасания

Решение на ~~44~~ 0 баллов

- Пусть нужно покрыть отрезок [1, 5] и даны отрезки [1, 3], [2, 4], [3, 5] с толщинами 3, 2 и соответственно 3.
- $d[1] = 3, d[2] = 5,$
  $d[3] = min(max(d[1], h[1] + h[3], max(d[2], h[2] + h[3])) = 5$
- Но правильный ответ, очевидно, 6
- Дело в том, что данная динамика не учитывает того факта, что новый отрезок может перекрываться, например, с предпоследним

Решение на 44 балла

- Сделаем бинарный поиск по ответу
- Используем тот факт, что есть оптимальный ответ, в котором любая точка покрыта не более чем двумя отрезками
- Отсортируем отрезки по правой границе

Решение на 44 балла

- Отсортируем отрезки по правой границе
- Посчитаем динамику *d[i]* — минимальная правая граница предпоследнего отрезка такого покрытия, что *i* — последний отрезок в покрытии, и все левее уже корректно покрыто
- Переход: переберем все предыдущие отрезки *j*, и для всех таких *j,* что *i*-ый и *j*-тый отрезок пересекаются или стыкуются и *d[j]* < *l[i]*, обновим *d[i]*

Решение на 44 балла

- Пусть текущий кандидат на ответ равен X. Тогда формула для пересчета динамики выглядит так:
  $d[i] = \min(r[j])$ по j, таким, что $d[j] < l[i] <= r[j] < r[i]$,
  и $w[j] <= X - w[i]$
- Не забудем про стыкующиеся отрезки:
  $r[j] = l[i] - 1$, $w[i] <= X$, $d[j] != \infty$
- Сложность такого решения $O(N^2 \log Ans)$

Решение на 100 баллов

- Упорядочим отрезки по левому концу
- Будем поддерживать множество *допустимых* отрезков, из которых сейчас можно сделать переход
- При рассмотрении нового отрезка какие-то отрезки надо удалить из допустимых, так как они больше не пересекаются и не стыкуются с текущим
- Какие-то отрезки надо добавить, так как теперь предпоследний отрезок больше не перекрывается текущим

Решение на 100 баллов

- Для вычисления *d* для текущего отрезка из множества допустимых нужно выбрать такой, что сумма его толщины и толщины текущего отрезка не превосходит кандидата на ответ, а его правая граница минимальна

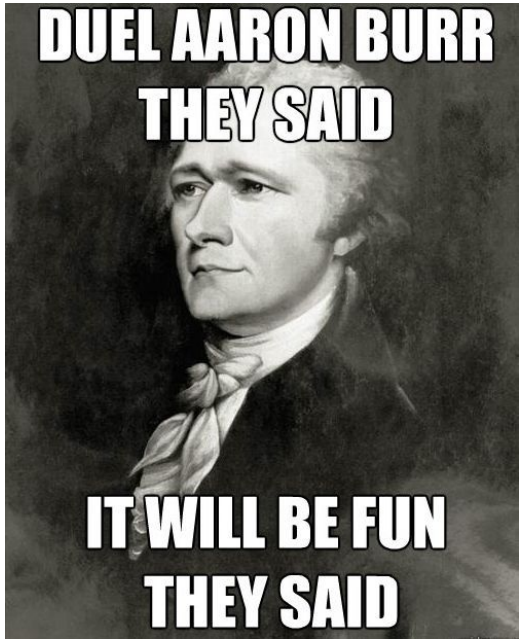
Решение на 100 баллов

- Будем поддерживать дерево отрезков на минимум. Для каждого отрезка из доступного множества будем хранить значение его толщины в точке с координатой его правой границы
- Тогда вставка — это обновление в точке, удаление — присвоение в точке бесконечности

Решение на 100 баллов

- Поиск оптимального перехода сводится к поиску самого левого элемента, значение которого не превосходит *(X - w[i])*
- Это можно сделать спуском по дереву отрезков, все операции с одним отрезком делаются за $O(\log N)$
- Необходимо также аккуратно рассмотреть случай соприкасания
- Общая сложность решения составляет $O(N \log N \log Ans)$

Задача Panna Cotta. «Поиграем?»



Автор задачи: Михаил Пядёркин, МГУ
Разработчик задачи: Андрей Гаркавый, МФТИ

Формальная постановка

За наименьшее число запросов вида f(x) (в виде сокращенной дроби) найти загаданное число n.

$$f(x) = \frac{\left[\frac{n}{x}\right]}{n}$$

Отметим, что знаменатель дроби всегда является делителем *n*

Решение на 30 баллов, ровно 60 запросов

- Замечаем, что если числитель *f(x)* — ноль, то *n* < *x*.
- Если нет, то *n* >= *x*.
- То есть можно применить бинарный поиск.

- Работает за логарифм операций: 60 запросов.

$$\frac{\left[\frac{n}{x}\right]}{n}$$

Решение на 30 баллов, 31-60 запросов

- Как выбрать ответ после нескольких запросов?
  Давайте возьмем максимальный знаменатель.
- Как проверить, что этот ответ подходит?
  Давайте сделаем запрос на один больше.
- Так появляется решение: делаем запросы на один больше макс. знаменателя, пока числитель не ноль.

- Это работает чуть лучше логарифма

$$\frac{\left[\frac{n}{x}\right]}{n}$$

Решение на 40 баллов, 21-30 запросов

- Чуть улучшим прошлое решение.
- Пусть *b* - макс. знаменатель.
- Давайте брать не x = (*b* + *1)* как следующий запрос, а x = (*2b - 1).*

- Работает еще чуть быстрее!

$$\frac{\left[\frac{n}{x}\right]}{n}$$

Решение на 69 баллов, 8 запросов

- Чуть улучшим позапрошлое решение.
- Заметим, что если мы сделали много запросов, то логичнее брать как ответ не максимальный знаменатель, а НОК всех знаменателей.

- Это делает порядка log(n) / log log (n) действий в силу закона распределения простых чисел

$$\frac{\left[\frac{n}{x}\right]}{n}$$

Решение на 100 баллов, 6 запросов

- Вернемся к идее с бин. поиском.
- Мы помним, что если числитель *f(x)* — ноль, то *n* < *x.*
- Если нет, то *x* <= *n*.
- Но если при этом *x* <= *n* < *2x*, то знаменатель точно равен *n*.
- Тогда давайте делать бинарный поиск по степеням двоек. Надо найти место, где *x* <= *n* < *2x.*
- Всего степеней двоек около 60.
  Решение за log log 10^{18} ~ 6 операций

$$\frac{\left[\frac{n}{x}\right]}{n}$$

Спасибо за внимание!