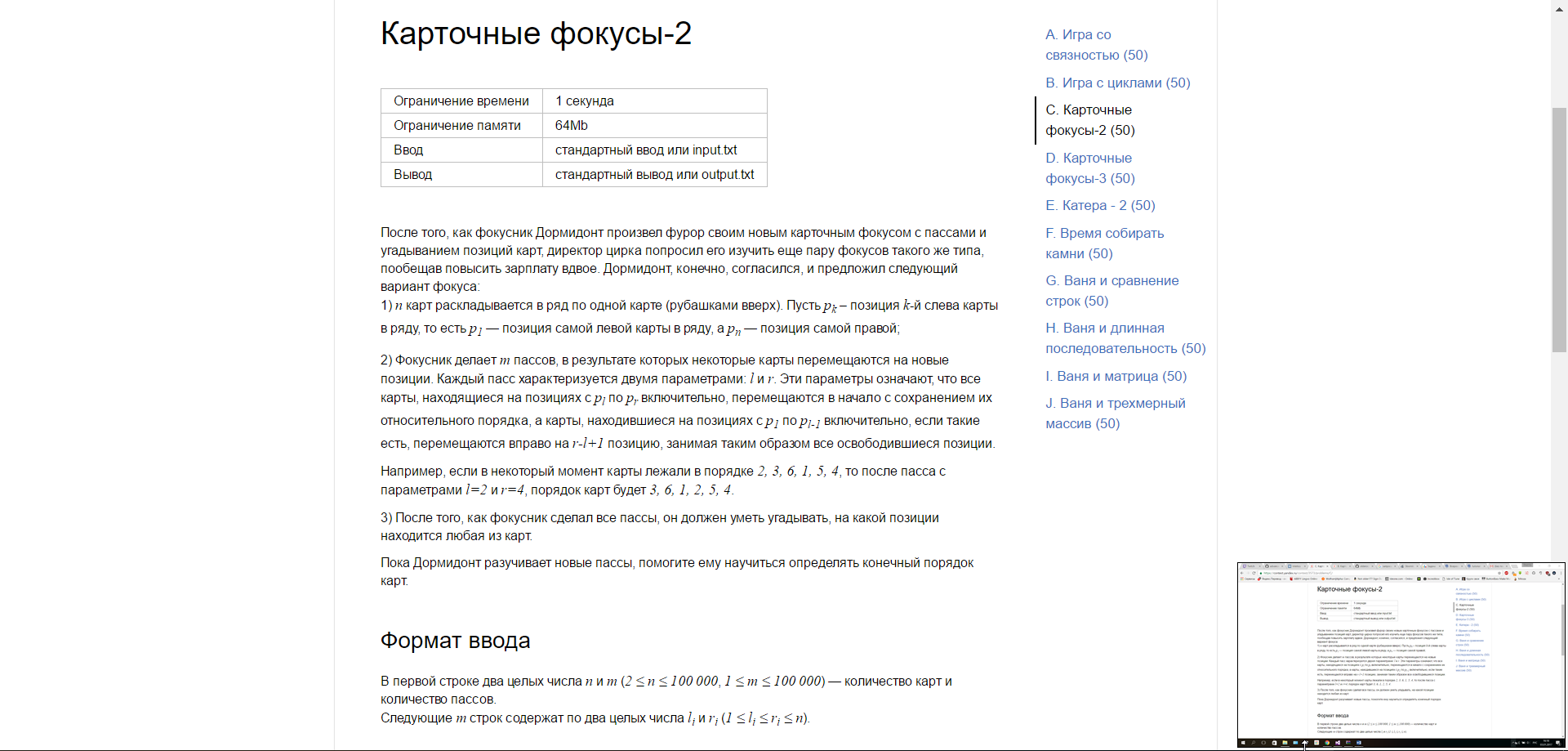
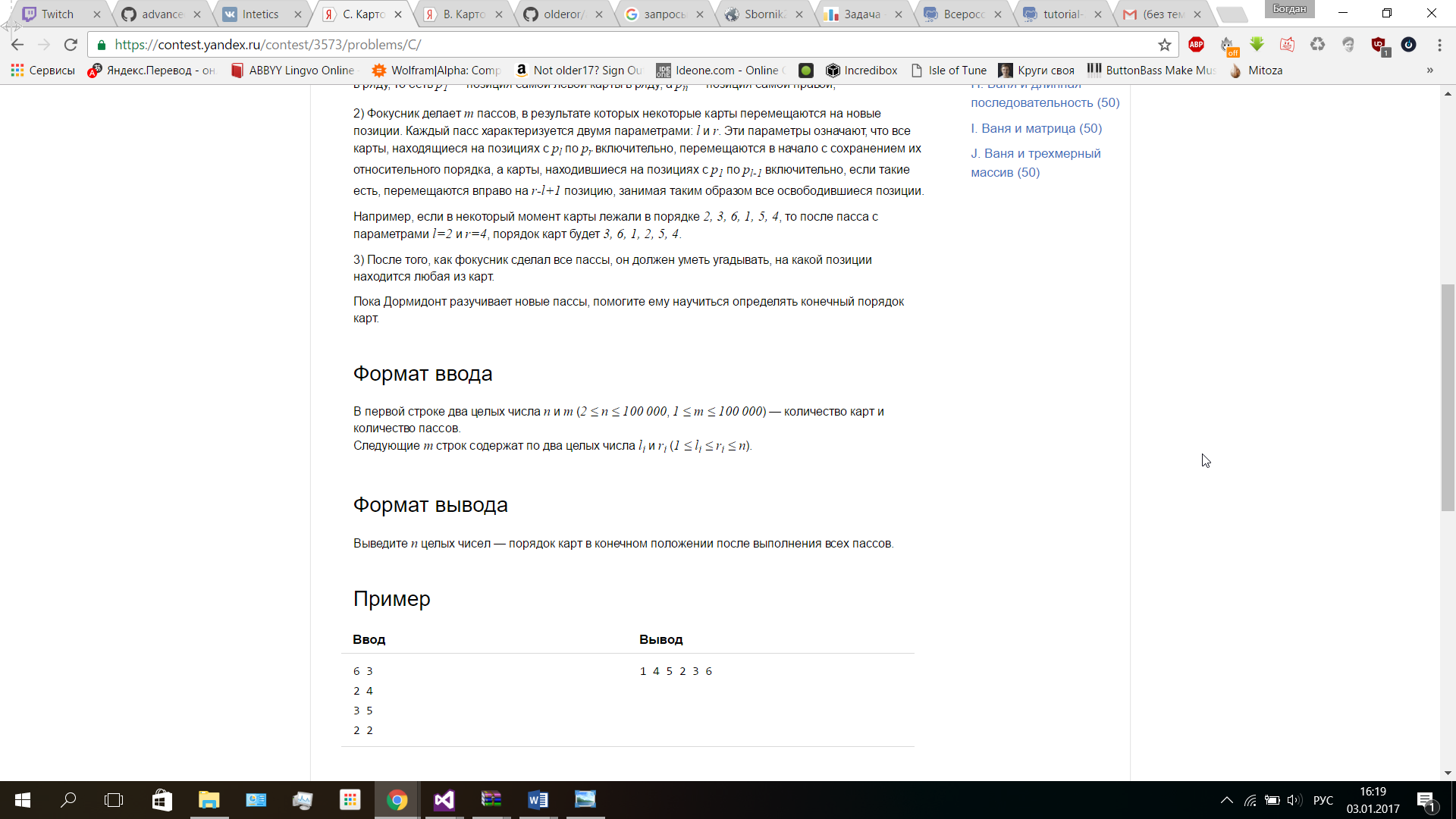
Задача «Карточные фокусы-2»

Решение Евченко Богдана ПИ-15-1

https://github.com/olderor/advanced3





Решение.

1. Для решения задачи построим декартово дерево по неявному ключу. В качестве ключей будем использовать индексы элементов в массиве. Однако явно хранить эти значения ключа мы не будем
   1. Построим дерамиду, заполнив ее значениями от 1 до n.
2. Для каждого нового запроса.
   1. Будем делить дерамиду на 3 части. Часть до l, от l до r и от r. Для этого разделим дерамиду сначала по индексу l, а потом вторую часть результата разделим еще по индексу (r – l).
   2. Соответственно условию, смерджим дерамиды в таком порядке: сначала часть от l до r с частью до l, результат смерджим с частью от r.
3. Таким образом будем перемещать отрезки карт на нужные позиции, после всех перемещений получим результат. Обойдем дерево и выведем каждый из элементов.

Доказательство

Как известно, декартово дерево — это структура данных, объединяющая в себе бинарное дерево поиска и бинарную кучу. При реализации же декартова дерева по неявному ключу модифицируем эту структуру. А именно, оставим в нем только приоритет, а вместо ключа будем использовать следующую величину: количество элементов в нашей структуре, находящихся левее нашего элемента. Иначе говоря, будем считать ключом порядковый номер нашего элемента в дереве. Для ключей выполняется условие двоичного дерева поиска. Можно представить, что ключи – числа от 0 до  и неявно расставить их по структуре дерева.

Заметим, что при этом сохранится структура двоичного дерева поиска по этому ключу (то есть модифицированное декартово дерево так и останется декартовым деревом). Однако, с этим подходом появляется проблема: операции добавления и удаления элемента могут поменять нумерацию, и при наивной реализации на изменение всех ключей потребуется  времени, где — количество элементов в дереве. Для того, чтобы добиться будем хранить вспомогательную величину - количество вершин в поддереве нашей вершины.

Выполняя операции не будем обращать внимания на ключи в принципе: в процессе выполнения алгоритма сравниваются лишь приоритеты. Сделаем предположение, что в деревьях есть явные ключи, и они упорядочены (все ключи первого дерева не превосходят ключей второго). Это предположение заставит выполнить операцию merge деревьев обычным образом, то есть ключи первого дерева окажутся по структуре дерева раньше, чем ключи второго (после слияния элементы второго дерева автоматически приобретут неявные номера от до , где — количество элементов в первом дереве, – во втором), при этом же сохранится условие дерева поиска. Вывод: программа ни разу не обращается к ключу, поэтому реализация merge для декартова дерева по неявному ключу вообще не будет отличаться от реализации того же merge в обычном декартовом дереве. Рассмотрим работу этой операции подробнее. Новым корнем будет тот элемент (это либо элемент в корне первого дерева, либо в корне второго), приоритет которого является максимальным. Пусть приоритет первого больше второго, тогда первый элемент является корнем результата. Все элементы второго дерева, добавляемые к первому, имеют ключи, большие ключа полученного корня, поэтому они все будут помещены в правое поддерево корня первого дерева, то есть нужно выполнить операцию merge для правого поддерева первого с вторым. Аналогично, если приоритет второго больше первого. Поскольку высота дерева -  , то операция merge будет работать за . Операция split сравнивает ключи, вместо них будем использовать их неявное представление – индексы массива. Следовательно, операции split и merge будут работать за , как и в обычном декартовом дереве.

В пункте 2 каждый раз будем перемещать отрезки карт соответственно условию. Выполнив все перемещения, получим результат – конечные положения карт.

Итоги

Оценка памяти. Для решения задачи потребовалось создать дерамиду, представляя элемент массива узлом дерамиды. Следовательно, полная сложность алгоритма по памяти равна

Оценка времени работы. Для решения задачи потребовалось: построить дерамиду, используя последовательные вызовы Insert, который работает за следовательно, полное построение работает за ; для каждого запроса необходимо перестроить дерамиду, выполнив две операции Split и две операции Merge, которые также работают за следовательно, на каждый запрос уходит Поскольку всего запросов m, на обработку всех запросов уйдет В конце нужно обойти дерево и вывести все элементы за Следовательно, полная сложность алгоритма по времени работы равна

.

Итог:

* затраты по памяти.
* затраты по времени.