Задача 1. Обоснование алгоритма

Выполнил: Сиволобцев Роман

Для меня, именно вторая задача является основной в тестовом задании от Wargaming Forge. Поэтому для первой задачи был так же выбран Python - как язык программирования.

Для работы с данными в этой задаче нужны были массивы. Первым делом подключаем библиотека numpy для продуктивной работы с ними. Следующим шагом была выгрузка данным в эти самые массивы, этот функционал поддерживает все тот же numpy. Заранее я посмотрел границы хранящихся числовых значений в файлах, на основании чего был выбран тип данных массивов. Для рейтингов игроков - np.uint16, для id игроков - np.uint32.

Дабы разбить команды по парам, нужно знать рейтинги команд. Поэтому следующим шагом был подсчет рейтинга каждой команды. Я уточнял у организаторов как именно высчитывать разницу рейтинга между двумя командами. На что получил ответ - как обычная разница сум рейтингов игроков каждой команды. Следующий шаг - высчитывание этих самых рейтингов каждой команды.

В дальнейшем команды нужно будет отсортировать по рейтингу, поэтому я пытался придумать как совместить подсчет рейтинга каждой команды и сортировку команд по их рейтингам. Но, к сожалению, ничего не придумал. Поэтому следующим шагом я просто посчитал рейтинг для каждой команды. Так как это массивы, то обращение к его i-му элементу - задача сложности O(1). Соответственно задача формирования массива рейтинга всех команд сложности - O(n).

Теперь нужно разбить эти команды по парам в соответствии с их рейтингами. Это можно сделать если отсортировать команды по рейтингу. Забегая на перед - это самая трудоемкая часть алгоритма, которая и определяет его скорость. В библиотеки numpy представлены 3 вида довольно популярных сортировок, которые имеют сложность O(n*logn). quicksort, mergesort, heapsort. Практическим путем я посмотрел какая из них лучше всего работает на данных массивах и получил следующие значения(Рис. 1).

На данном этапе я имел отсортированный массив рейтингов команд и тут задача распадается на две - когда количество команд четное и не четное.

```
**************************************************
test_A
--- mergesort 0.012508869171142578 seconds ---
--- quicksort 0.009506464004516602 seconds ---
--- heapsort  0.01801156997680664 seconds ---
**************************************************
test_B
--- mergesort 0.0010008811950683594 seconds ---
--- quicksort 0.0010008811950683594 seconds ---
--- heapsort  0.0010004043579101562 seconds ---
**************************************************
test_C
--- mergesort 0.0010004043579101562 seconds ---
--- quicksort 0.0004999637603759766 seconds ---
--- heapsort  0.0015010833740234375 seconds ---
**************************************************
test_D
--- mergesort 0.0009999275207519531 seconds ---
--- quicksort 0.0010006427764892578 seconds ---
--- heapsort  0.0010008811950683594 seconds ---
```

Рис. 1: Скорость работы сортировок на разных наборах данных

Первый случай является тривиальным. Докажем это введя предположение:

Предположение. Если команд четное количество, то оптимальным разбиение их на пары с целью минимизации дистанции является разбиение - {(1,2), (3,4) ... (2i,2i+1) ... (n-1,n)}. Представим отсортированные рейтинги команд как точки на прямой (Рис. 2). Формирование пар начнем с крайних точек множества. Берем точку №1 с наименьшим рейтингом и ищем кратчайшее растоянии до остального множества. Очевидно, что это будет точка №2, так как массив заранее отсортирован. Выкидываем из множества точки, которые уже разбиты на пары и берем следующую крайнюю точку - №3. Повторяя эту процедуру пока не закончатся точки мы и получим наше оптимальное разбиение.

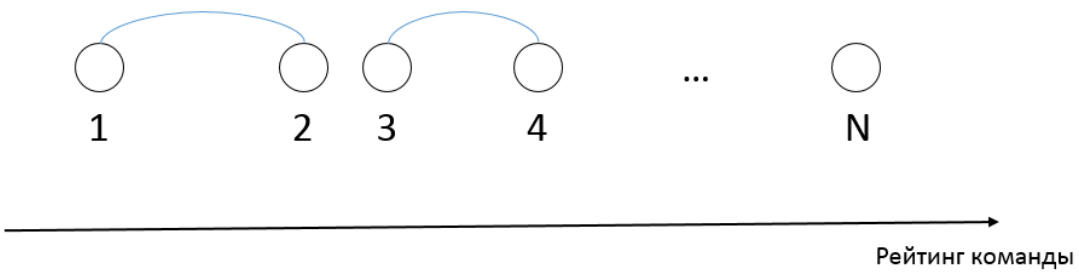

Рис. 2: Разбиение четного количества точек по парам.

Рассмотрим ситуацию, когда точек не четное количество. Получается два типа разбиения точек по парам, начиная с начала, назовем его I, и начиная с конца, назовем его J. В случае с I последняя точка останется без пары, в J - первая. Рис. 3.

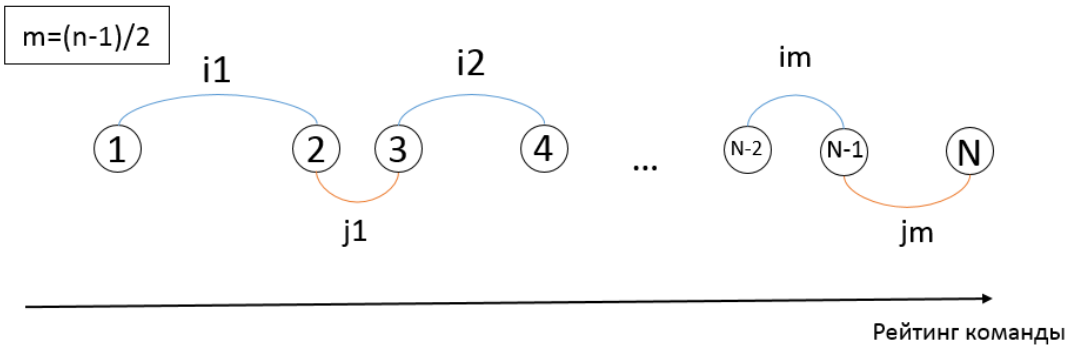

Рис. 3: Разбиение не четного количества точек по парам. Формирование I и J множеств

Теоретически начнем убирать вершины и смотреть что станет с этим множеством, а также множествами I и J. И так, выкидываем k-ую вершину и разбиваем по парам как в предыдущем пункте. Оказывается, что это разбиение представляет собой множество отрезков I стоящих левее k-ой вершины + множество отрезков J стоящих правее k-ой вершины. Нюанс только в четности позиции k. Если k не четное, то это пересечение представляет меньшую сумму и не включает в себе дополнительный отрезок. См. Рис. 4-5

Таким образом в отсортированном массиве рейтингов команд имеет смысл выкидывать только не четные вершины. Причем просчитав один раз значения объектов из множеств I и J, можно найти дистанцию без любой другой не четной вершины как $\sum_{s=1}^{k} i_k + \sum_{s=k}^{m} j_k$, где k-номер не четной вершины которую пытаются выкинуть.

Таким образом переформировав массивы I, J по правилам $I = \{0, i_1, i_1 + i_2, ... \sum_{s=1}^{m} i_s\}$, $J = \{\sum_{s=1}^{m} j_s ..., j_{n-1} + j_n, j_n, 0\}$ и просуммировав их, можно

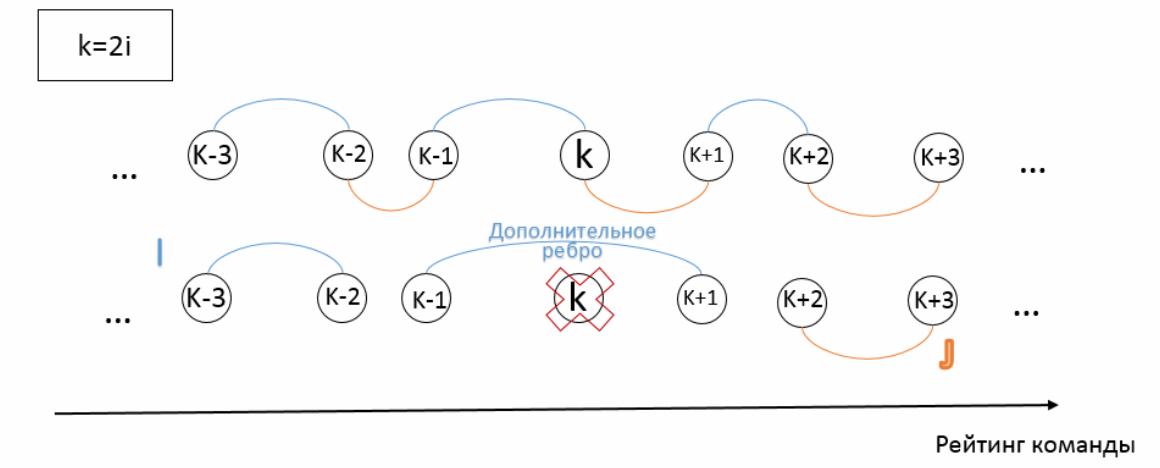

Рис. 4: Разбиение не четного количества точек по парам. Выбрасывание четной вершины

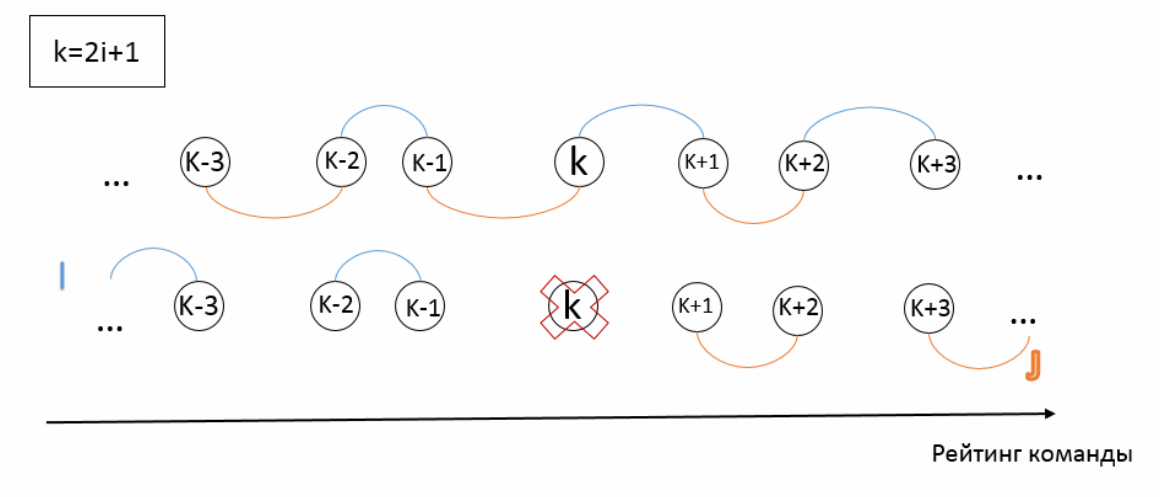

Рис. 5: Разбиение не четного количества точек по парам. Выбрасывание не четной вершины

одним проходим найти команду которую нужно оставить без пары для оптимального разбиения. Номер минимального элемента в массиве I+J равен номеру не четного элемента в отсортированном массиве. Так как все вычисления линейные, то сложность их упирается в O(n). А это означает что сложность всего алгоритма упирается в сортировку.

По поводу времени исполнения кода. На практике оказалось что поиск команды которая останется без пары занимает чуть меньше времени чем сортировка массива. Это связано с тем, что в 2-4 наборах данных находится не сильно много команд. А большую часть времени в исполнении занимает загрузка данных из файла. Это значение в разы превышает время работы самого алгоритма.