Задача D. Зарядные станции.

2030 год. Министр инфраструктуры Украины Остап Бендеренко выступил в парламенте со смелой инициативой – построить трассу международного значения Крыжополь-Жмеринка. Его проект был поддержан единогласно, поскольку, удивительному по стечению обстоятельств, дома всех n депутатов расположились вдоль предполагаемой трассы, причем все депутаты живут в разных домах. Однако, поскольку все депутаты еще с 2022 года ездят на электрокарах, остро встал вопрос о том, где же ставить зарядные станции – ведь трасса очень длинная, а электричество дорогое, поэтому заряжаться дома слишком накладно...
Естественно, каждый депутат захотел, чтобы зарядную станцию построили рядом с его домом, но тут оказалось, что бюджет проекта рассчитан лишь на m зарядных станций. Начались дебаты, и в результате бурного шестичасового обсуждения было принято следующее решение:
– Все m станций будут построены возле домов некоторых депутатов;

– Дома депутатов-"счастливчиков" будут выбраны так, чтобы общая сумма расстояний от каждого депутатского дома до ближайшей к ней зарядной станции была минимальной.

Контроль за исполнением этого решения возложили на вас как главного помощника заместителя спикера парламента, так что определять местоположение зарядных станций придется именно вам.

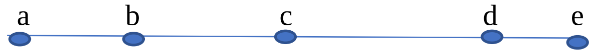
В первой строке содержатся два целых числа: количество депутатов n ($1 \le n \le 300$) и количество зарядных станций m ($1 \le m \le 30$), $m \le n$.
Вторая строка содержит n целых чисел в возрастающем порядке, являющихся координатами домов депутатов от начала трассы. Для каждой координаты x верно $1 \le x \le 10^4$.

Первая строка выходного файла должна содержать одно целое число — общую сумму расстояний от каждого депутатского дома до ближайшей к ней зарядной станции.
Вторая строка должна содержать m целых чисел в возрастающем порядке. Эти числа являются искомыми координатами зарядных станций. Если для заданного расположения домов есть несколько решений, необходимо найти любое из них.

Решение

1. Создадим вектор чисел на n элементов, где будут храниться координаты домов депутатов.
2. Имея координаты домов, мы можем сделать некоторый предподсчёт :

2.1. Заметим, что для любого количества домов, для которых мы можем разместить всего одну станцию, оптимальным вариантом будет поставить станцию на дом, находящийся посередине.
Доказательство: имеем отрезок



Поставив станцию на точку c, суммарное расстояние от каждого дома до станции будет [a,c] + [b,c] + [c,d] + [c,e] = [a,b] + [b,c] + [b,c] + [c,d] + [c,d] + [d,e] = [a,e] + [b,d]. Если поставить станцию на любую другую точку, например, на b, то суммарное расстояния уже будет [a,b] + [b,c] + [b,d] + [b,e] = [a,b] + [b,c] + [b,c] + [c,d] + [b,c] + [c,d] + [d,e]=[a,e] + [b,d] + [b,c], что, очевидно, больше предыдущего значения, так как мы на 1 раз больше вынуждены проехать подотрезок [b,c]. Это верно и для всех других точек. Если количество точек чётно, то не имеет значения, какую из двух серединных выбрать нам, суммарное расстояние не изменится. Доказывается аналогично.

2.2. Теперь мы можем посчитать суммарные длины для всех подотрезков, для которых мы хотим поставить 1 дом. Чтобы делать это эффективно, необходимо для текущего подотрезка [left, right] за O(1) высчитывать суммарное расстояние до центрального дома. Делать мы это будем следующим образом:
Допустим, мы имеем подотрезок домов с координатами 1 2 3 4 5 6 7 8 9. Суммарное расстояние до центральной точки(5) будет следующим: 5 – 1 + 5 – 2 + 5 – 3 + 5 – 4 + 6 - 5 + 7 – 5 + 8 – 5 + 9 – 5 = 4 * 5 – (1 + 2 + 3 + 4) – 4 * 5 + (6 + 7 + 8 + 9) = (6 + 7 + 8 + 9) – (1 + 2 + 3 + 4).
Очевидно, что это разница между суммой всех элементов с [middle + 1, right] и суммой элементов с [left, middle – 1]. Для вычисления суммы всех элементов на любом подотрезке за O(1) предпосчитаем также вектор префикс-сумм prefixSums, а результаты самих сумм будем записывать в двумерный массив sums размера n*n. Окончательная формула: sums[left][right] = prefixSums[right] – prefixSums[middle + 1] – prefixSums[middle – 1] + prefixSums[left].
Для подотрезка чётной длины, например, 1 2 3 4 5 6 7 8 9 10, результат будет тем же, что для предыдущего, только с добавлением разницы между последним и серединным элементом.

2.3. Параллельно с предпосчётом создадим наш двумерный вектор пар значений result, у которого в позиции result[i][j].length будет храниться минимальная суммарная длина для i последних домов, на которые собираются распределить j станций. Соответственно, во время предпосчёта мы можем заполнить все result[i][1].length.

3. Далее наш алгоритм будет работать следующим образом: мы условно пытаемся разделить каждый суффикс [left, n -1] на сумму из двух частей при помощи разделителя right: в левой части мы будем использовать

значение sums[left][right][1], а во второй res[right + 1][k – 1], где k - текущее количество станций. Из всех таких значений мы выберем минимальное и запишем оптимальную границу right в res[left][n - 1].end для восстановления ответа. То есть, зная все оптимальные суффиксы для k-1 станций, мы подбираем ему соответствующий оптимальный префикс для 1 станции, что в сумме даст выгодный подотрезок для k станций. Ответ будет храниться в res[0][m].

4. Так как мы сохраняли оптимальную границу для каждого суффикса, мы можем m раз подвинуть левую границу на оптимальную границу для текущего суффикса, параллельно выводя значения, которые находятся посередине подотрезка – координаты домов, для которых мы решили установить зарядные станции.

Подсчёт префикс-суммы выполняется за O(n), благодаря ей мы смогли предпосчитать суммы для всех подотрезков с одной станцией за O(n^2), ну а так как основной цикл алгоритма перебирает m – 1 количество станций, левую границу суффикса и границу раздела суффикса, то общая сложность алгоритма будет O(m*n^2).

По памяти мы используем 2 вектора размера O(n) – вектор входных данным и префикс-сумм, вектор сумм подотрезков размера O(n^2), а также вектор result размера O(n*m). Итого, затраты по памяти – O(n^2). В принципе, можно было не хранить массив sums и каждый раз вычислять соответствующие суммы расстояний динамически, тем самым уменьшив затраты по памяти до O(n*m), но пожертвовали памятью ради того, что не пересчитывать по несколько раз одни и те же значения.