Студент: Тимур Хабибуллин
Группа: 1
Дата: 5 мая 2022 г.

---

1. `extractMin()` за $o(\log(n)$.

   Предположим, что у нас есть какой-то хитрый алгоритм `extractMin()`, который всегда работает за $o(\log(n))$. Для того, чтобы он после своей работы выполнил свойство кучи, необходимо, чтобы он работал на сравнениях элементов, потому что при своей работе алгоритм обязан хоть раз нарушить свойство кучи (случай когда куча состоит из одного элемента или все элементы в куче одинаковые - не интересен) и поэтому сравнивать элементы обязательно придется. Тогда получается, что кучу с таким `extractMin()`, основанным на сравнениях, можно использовать для того, чтобы сортировать массив за $o(n\log(n))$. Однако это не возможно, поскольку мы знаем, что нижняя оценка на сортировку, остованную на сравнениях, есть $\Omega(n\log(n))$. Значит, такого хитрого алгоритма `extractMin()` не существует.

2. **Анка-пулеметчица**

   (a) Поскольку каждый патрон отклонился не более чем на k от своей изначальной позиции, самый первый партон в перемешанном массиве находится среди $1...k$ пуль. Тогда на первом шаге найдем среди первых k элементов линейным поиском первый патрон и поставим его в начало. После этого мы, по сути, пришли к исходной задаче, - второй патрон не мог сместиться влево на 1 позицию, там уже стоит 1 по порядку пуля, но мог на не более чем k вправо. Поскольку на предыдущем шаге мы переставляли пули среди $1...k$, то 2 пуля не может лежать правее $k+2$ пули, значит, она лежит в первых k пулях справа от нее. Линейно находим 2 пулю и ставим на место. Аналогично продолжаем и расставляем все пули на место за $O(nk)$.

   (b) В отличие от предыдущего пункта, вместо линейного поиска по ближайшим k элементам справа создадим min-кучу за $O(k)$ из первых k элементов и на каждом шаге за $O(\log(k))$ будем вытаскивать минимум, т.е. тот патрон, который по определенному признаку, известному только Анке-пулеметчице, должен стоять первый среди этих k, и докладывать следующий элемент справа. В итоге получим асимптотику $O(n\log(k))$.

   (c)

3. **Белоснежка и гномы**

   Пусть у нас есть множество пар $\{(a_i, b_i)\}_{i=1..n}$, где a_i - сколько времени надо укладывать $i-$го гнома, а b_i - сколько он спит. Нам надо проверить, можно ли уложить гномов так, чтобы, уложив последнего, все остальные еще спали, т.е. любой гном просыпался позже момента, когда всех уложат спать:

   $$S_n = \sum_{i=1}^{n} a_i < t_{\text{wake up}}(j),\ \forall j \in [1, n] \tag{1}$$

   Время подъема j гнома можно найти как:

   $$t_{\text{wake up}}(j) = \sum_{i \leqslant j} a_i + b_j = S_{j-1} + a_j + b_j$$

   Подставляя это в (1), получаем условие:

   $$S_n - S_{j-1} < a_j + b_j$$

Ннадо отсортировать пары так, чтобы $a_j + b_j$ убывало. Предположим, что у нас есть оптимальное решение, у которого это нарушается правило сортировки, начиная с какого-то номера, т.е пусть номер j

$$a_j + b_j < a_{j+1} + b_{j+1} \tag{2}$$

Тогда для этого оптимально решения выполнено:

$$\begin{cases} S_n - S_{j-1} < a_j + b_j \\ S_n - S_j < a_{j+1} + b_{j+1} \end{cases} \tag{3}$$

Мы хотим показать, что тогда мы всегда можем перестроить наше оптимальное решение, поменяв порядок вхождения пар местами, т.е. должно выполняться

$$\begin{cases} S_n - S_{j-1} < a_{j+1} + b_{j+1} \\ S_n - S_{j-1} - a_{j+1} < a_j + b_j \end{cases} \tag{4}$$

Если взять первое уравнение из (3) и учесть условие (2), то мы автоматически получим первое уравнение из (4):

$$S_n - S_{j-1} < a_j + b_j < a_{j+1} + b_{j+1}$$

Теперь снова возьмем первое уравнение и вычтем из него a_{j+1} и вспомним про (2):

$$S_n - S_{j-1} - a_{j+1} < b_j < a_{j+1} + b_{j+1} - a_j < a_{j+1} + b_{j+1},$$

т.е. мы получили второе уравнение в (4). Значит, мы всегда можем перестроить оптимальное решение. Тогда можно отсортировать исходную последовательность $\{(a_i, b_i)\}_{i=1..n}$ за по убыванию суммы $a_i + b_i$ за $O(n \log(n))$ и за $O(n)$ на каждом шаге проводить проверку

$$S_n - S_{j-1} < a_j + b_j$$

4. **Антиклика**

Будем раскрашивать вершины из антиклики в синий, а остальные - в красный, и класть пемеченные вершины в соответствующие `set`-ы, в котором проверка на наличие элемента есть $O(1)$. Возьмем первую вершину и покрасим ее в синий, а ее соседей - в красный, положим в нужные `set`-ы. Затем возьмем следующую непокрашенную вершину и сделаем аналогично - ее покрасим в синий, а смежные с ней - в красный, и тд. На каждом шаге мы добавляем в антиклику одну вершину и красим не более чем $d + 1$ вершину, проходять не более чем по d ребрам очередной синей вершины. В результате на каждом шаге мы тратим $O(d + 1)$ времени. Всего мы сделаем не менее $\frac{n}{d-1}$ действие и добавим в антиклику как минимум $\frac{n}{d-1}$ вершину. При этом мы можем несколько раз покрасить одну вершину в красный цвет, но не более того, потому что у любой синей вершины все ее смежные вершины - красные, т.е непокрашенных среди них нет.
При поиске непокрашенных вершин мы пройдем всего лишь 1 раз по всем вершинам, потому что мы не возвращаемся к тем вершинам, которые уже прошли, а проверка на наличие в `set`-е - константное.
Тогда общее время работы алгоритма $O\left((d+1)\frac{n}{d+1}\right) = O(n)$

5. **Идейная**

Будем придерживаться жадной стратегии. Пусть мы находимся в точке cur, у нас есть b бензина и мы добрались до этой точки эффективным образом. Поскольку емкость бака меньше k, то мы никогда не можем за раз уехать дальше, чем k. Поэтому найдем среди следующих k километров

заправку с минимальной ценой c_i. Если нам хватает топлива, то едем сразу туда и заправляемся так, чтобы нам минимально хватило до следующей остановки в следующих k километрах и минимальной ценой. Если же нам не хватает топлива, т.е.

$$b < (i - cur),$$

то выгоднее дозаправиться на станции, у которой наименьшая цена среди ближайших b километров, минимально заправиться и докаться до i. Действительно, если дозаправиться на заправке с координатой j, то в этой точке останется бензина

$$b_j = b - (j - cur),$$

а тогда для того, чтобы доехать до i заправки с пустым баком надо заплатить

$$c_j(i - j - b_j) = c_j(i - j - b + j - cur) = c_j(i - cur - b).$$

Видно, для того, чтобы заплатить меньше всего денег на дозаправку надо выбирать заправку с наименьшей ценой c_j из доступных, т.е. ближайших b.

Почему надо заправляться на i заправке, т.е. на минимальной в пределах ближайших k километров, вместо того, чтобы попробовать найти заправку подальше еще более дешевую? (наверно это очевидно, но я решил написать)
Предположим, что есть заправка $j + l$ (j все также одна из ближайших b заправок):

$$c_j > c_i > c_{j+l},\ l \leqslant k,\ i < cur + k < j + l$$

Будем считать, что все также бензина не хватает до i заправки, иначе стратегия как в предыдущих рассуждениях. Хотим добраться до $j + l$ заправки. Есть 2 стратегии:

(a) минимально дозаправиться на j остановке, потом на i и докатиться и полностью заправиться в $j + l$. Тогда на первую дозаправку мы уже посчитали денежные траты, на вторую надо $c_i(j + l - i)$ денег, того
$$c_j(i - cur - b) + c_i(j + l - i)$$

(b) минимально дозаправиться, но побольше, на j заправке и доехать сразу до $j+l$. Но тогда для того, чтобы доехать до i заправки мы потратим все также $c_j(i-cur-b)$ денег, а на оставшемся участке, фактически, мы будем тратить более дорогой $j-$й бензин. Наши расходы

$$c_j(i - cur - b) + c_j(j + l - i)$$

оказываются очевидно больше.
Можно еще на каждом шаге смотреть, когда там конец поездки и если $n - cur \leqslant k$, то заправить $n - cur$ бензина, чтобы не переплачивать.

На каждом шаге мы находим минимум в ближайших k километрах, сделаем $O(n/k)$ операций. Тогда асимптотика алгоритма $O(n)$

6. **Станок**

Будем считать, что первый станок непрерывно работает, нечего ему лениться. Для того, чтобы работа обоих станков была минимальной, надо сделать так, чтобы второй станок как можно меньше ждал очередную деталь, потому что он и так отработает все время $b_1 + ... + b_n$, однако чем дольше он ждет новую деталь, тем больше увеличивается время.
Пусть взята 1 деталь, ее первый станок сделает за a_1, сразу передадим ее на второй станок и

возьмем вторую, она сделается за a_2. Простой может прозойти, если $a_2 > b_1$ - второй станок уже все сделал, а вторая деталь еще не вышла из первого. Если же наоборот, то второй станок работает на пределе своих возможностей и никак ускорить его мы не можем. И так аналогично и для всех деталей. Поэтому время простоя i детали

$$\tau_i = max(0, a_i - b_{i-1})$$

Тогда получаем оценку сверху на сумму всего времени простоя

$$T_n \leqslant a_1 + a_2 - b_1 + a_3 - b_2 + ... + a_n - b_{n-1} = S_n(a) - S_{n-1}(b)$$

, где $S-$ префискная сумма. Для того, чтобы минимизировать T, необходимо уменьшить первую сумму и увеличить вторую. Это можно добиться тем, чтобы вначале подавать вначале детали с маленькими a_i и большими b_i, а в конце - с . Т.е. надо отсортировать по $\min(a_i, b_i)$