1 Введние. Алгоритм Мисра-Граеса

1.1 Разогревочные задачи

Задача 1. *Дана последовательность $\sigma = \langle a_1, a_2, \cdots, a_m \rangle$, где каждый $a_i \in [n] = \{1, 2, \cdots, n\}$. Обозначим частоту появления элемента x через $f_\sigma[x] = \#\{i \mid a_i = x\}$. Известно, что $\exists x : f_\sigma[x] = 1$ и для всех остальных значений $y \neq x$ $f_\sigma[y] = 0$ mod 2. Требуется найти x за один проход по последовательности, используя $O(\log n + \log m)$ памяти.*

Заведем ячейку `res` и при обработке очередного элемента будем обновлять значение `res := res XOR` a_i. Поскольку `XOR` двух одиковых значений равен нулю, и `0 XOR a = a`, то после обработки всего потока в `res` будет лежать искомый x.

Примечание 1. *Операция `XOR` для двух бит b_1 и b_2 определяется как $b_1 + b_2$ `mod` 2. Операция `XOR` для двух целых чисел определяется побитово.*

Задача 2. *Дана последовательность $\sigma = \langle a_1, a_2, \cdots, a_m \rangle$, где каждый $a_i \in [n]$. Известно, что $\exists x : f_\sigma[x] = 0$ и для всех остальных значений $y \neq x$ $f_\sigma[y] = 1$. Требуется найти x за один проход по последовательности, используя $O(\log n + \log m)$ памяти.*

Заведем ячейку `res`, которая будет хранить сумму всех элементов в последовательности σ. При обработке очередного элемента a_i обновим значение `res := res +` a_i. Поскольку все элементы, кроме одного, присутствуют в потоке ровно один раз, то $x + \sum_{i=1}^{m} a_i = \frac{n(n+1)}{2}$. Вычтем из $\frac{n(n+1)}{2}$ значение `res` и получим ответ.

Упражнение 1. *Решите Задачу 2 через побитовый XOR.*

Задачу 2 можно обобщить, как если бы отсутствовал не один, а два элемента. В этом случае мы могли бы подсчитать сумму всех элементов последовательности $S_1 = \sum_i a_i$ и сумму их квадратов $S_2 = \sum_i a_i^2$. Тогда для недостающих элементов можно было бы составить два уравнения:

$$\begin{cases} x_1 + x_2 & = & \sum_i i - S_1 \\ x_1^2 + x_2^2 & = & \sum_i i^2 - S_2 \end{cases}$$

Решение данной системы сводится к решению квадратного уравнения.

Примечание 2. *Обобщение Задачи 2 до k пропавших элементов имеет красивое решение без повышения порядка элементов. Мы рассмотрим решение этой задачи позже в курсе.*

1.2 Введение

Мы рассмотрели две задачи, в каждой из которых нам разрешается пройти по некоторой последовательности ровно один раз. Данное ограничение возникает естественным образом в реальной жизни. Давайте представим себе поток данных, который проходит через роутер. Нас интересует какая-нибудь статистика об адресах доставки: сколько различных адресов используется, куда посылается наибольшее число пакетов и т.п. У нас нет возможности хранить всю историю пакетов. Нет возможности хранить и статистику по всем адресам, поскольку всего их может быть 2^32 или 2^64.

Отсюда возникают ограничения на потоковые данные. Нам требуется решить некоторую задачу за один проход, используя $\overline{o}(n + m)$ памяти. Идеальным решением абстрактной потоковой задачи считается решение за $O(\log n + \log m)$ памяти. Нам требуется хотя бы константное число ячеек для хранения длины последовательности и ее элементов. Более реалистичным обычно оказываются оценки вида $O(\text{poly}(\log n, \log m))$, где poly – некоторый полином. Такая оценка не очень велика по отношению к объему данных, но достаточна, чтобы иметь возможность придумать разумный алгоритм.

Сегодня мы будем рассматривать задачи, где будут использоваться более одного прохода. Такое допущение связано с тем, что иногда случайный доступ по последовательности разрешен, но крайне дорог. Как пример, данные могут храниться на вторичных или третичных носителях, и обращение по случайному адресу может оказаться роскошью.

1.3 Задача о частом элементе

Задача 3. *Дана последовательность $\sigma = \langle a_1, a_2, \cdots, a_m\rangle$, где каждый $a_i \in [n] = \{1, 2, \cdots, n\}$. Известно, что $\exists x : f_\sigma[x] \geq \lfloor \frac{m}{2} \rfloor + 1$. Требуется найти x за один проход по последовательности, используя $O(\log n + \log m)$ памяти.*

Алгоритм 1. Будем хранить две ячейки: `res` и `cnt`. На первой стадии заполним ячейки нулями. Ячейка `res` будет хранить текущего кандидата, а ячейка `cnt` отражать, насколько это похоже на правду. Перед проходом по потоку, `cnt` равна нулю.

При обработке очередного элемента `y` будем смотреть на счетчик `cnt`. Если `cnt` равен нулю, то обновим `res`. Если `y` равен `res`, то увеличим `cnt` на 1, иначе уменьшим.

```
def init():
  res = 0
  cnt = 0

def process(y):
  if cnt == 0:
    res = y
  if y == res:
    cnt++
  else:
    cnt--
```

Теорема 1. *Алгоритм 1 корректен.*

Доказательство. Пусть `x` – правильный ответ. Введем величину

$$M = \begin{cases} \texttt{cnt} & \text{, если } \texttt{res} = \texttt{x}; \\ -\texttt{cnt} & \text{, иначе.} \end{cases}$$

Заметим, что изначально M равно 0. Если очередной элемент $\texttt{y} = \texttt{x}$, то M увеличивается на единицу: при $\texttt{res} = \texttt{x}$ – очевидно; при $\texttt{res} \neq \texttt{x}$ `cnt` уменьшается, значит, $M = -\texttt{cnt}$ увеличивается. Поскольку $f_\sigma[x] \geq \lfloor \frac{m}{2} \rfloor$, то по завершении обработки потока $M > 0$. Значит, в `res` будет лежать x. □

В Задаче 3 у нас есть гарантия, что частый элемент существует. Давайте поймём насколько эта гарантия существенна для однопроходного алгоритма.

Задача 4. *Дана последовательность $\sigma = \langle a_1, a_2, \cdots, a_m\rangle$, где каждый $a_i \in [n] = \{1, 2, \cdots, n\}$. За один проход по последовательности требуется найти x такой, что $f_\sigma[x] \geq \lfloor \frac{m}{2} \rfloor + 1$.*

Теорема 2. *Пусть $m \leq n$. Любой детерминированный алгоритм для Задачи 4 использует $\Omega(m \cdot (logn - logm + 1))$ памяти. В частности, если $m = n$, то памяти необходимо $\Omega(n)$.*

Доказательство. Разделим последовательность на две равные части. Возьмем произвольное множество $S \subseteq [n]$ из $\frac{m}{2}$ элементов и расположим его в первой. Запустим алгоритм на этой части. Заметим, что теперь из памяти алгоритма можно извлечь все множество S. Чтобы проверить, лежит ли элемент x в S, достаточно записать его $\frac{m}{2}$ раз во второй части. Положительный ответ алгоритма при проходе по второй части означает, что $x \in S$, и наоборот.

Всего различных множеств из $m/2$ элементов $\binom{n}{m/2} \geq \left(\frac{n}{m/2}\right)^{m/2}$. Значит, памяти потребуется $\log\left(\frac{n}{m/2}\right)^{m/2} = \Omega(m(\log n - \log m + 1))$.

□

Из Теоремы 2 следует, что при отсутствии гарантий, экономичное по памяти решение поиска частого элемента требует хотя бы два прохода. В завершении этой лекции мы рассмотрим общий случай Задачи 4.

Задача 5. *Даны число k и последовательность $\sigma = \langle a_1, a_2, \cdots, a_m \rangle$, где каждый $a_i \in [n] = \{1, 2, \cdots, n\}$. Элемент x называется частым, если $f_\sigma[x] \geq \lfloor \frac{m}{k} \rfloor + 1$. Необходимо за два прохода найти все частые элементы, используя $O(k \cdot (\log n + \log m))$ памяти.*

Алгоритм для Задачи 5 похож на Алгоритм 1. Алгоритм использует два прохода. После первого прохода алгоритм оставляет не больше $k-1$ кандидата в ответы. На втором проходе алгоритм честно подсчитывает частоту каждого кандидата и возвращает ответ.

Мы распишем подробно первый проход и докажем, что после него все частые элементы войдут в множество кандидатов.

Алгоритм 2 (Первый проход Мисра-Граеса). Алгоритм поддерживает ассоциативный массив `cnt`, который для каждого элемента хранит его условную частоту. В `cnt` хранятся только ненулевые значения. Если элемент `y` отсутствует в `cnt` как ключ, мы считаем, что `cnt[y] = 0`. Изначально `cnt` пуст.

Алгоритм действует следующим образом. Если очередной элемент `y` присутствует в `cnt`, то увеличиваем соответствующий счетчик на 1. Иначе, если в `cnt` меньше чем $k-1$ ключ, то добавим `y` в `cnt`. Иначе, вычтем из всех счетчиков в `cnt` по единице. Если какой-то счетчик обнулился, сотрем соответствующий ключ.

```
1  def process(y):
2    if y in keys(cnt):
3      cnt[y]++
4    else if |keys(cnt)| < k - 1:
5      cnt[y] = 1
6    else:
7      # cnt[y] = 1
8      for k in keys(cnt):
9        cnt[k]--
10       if cnt[k] == 0:
11         cnt.erase(k)
```

Теорема 3. *После первого прохода в множестве ключей* `cnt` *будут содержатся все частые элементы.*

Доказательство. Раскомментируем строку 7 и заметим, что работа алгоритма не изменится. Далее по коду строки 9 и 11 удалят `y` из `cnt` обратно.

Рассмотрим произвольный частый элемент `x`. Заметим, что при обработке очередного `y` = `x`, `cnt[x]` увеличивается на 1, как это видно на строках 3, 5 и 7. Всего инкрементов будет хотя бы $\lfloor \frac{m}{k} \rfloor + 1$.

Мы покажем, что `cnt[x]` уменьшается не больше $\lfloor \frac{m}{k} \rfloor$ раз. Заметим, что декремент `cnt[x]` выполняется по одному разу на каждый проход по строкам 8 – 11. Введем величину $M = \sum_{\texttt{k} \in \texttt{keys(cnt)}} \texttt{cnt[k]}$. При общем декременте величина M уменьшается ровно на k. По ходу выполнения кода величина M увеличвается на 1 на каждом из m элементов. Поскольку изначально величина M равна нулю, то всего общих декрементов может быть не больше $\lfloor \frac{m}{k} \rfloor$.

Значит, по завершении работы алгоритма `cnt[x]` будет положительным, а `x` будет лежать в ключах `cnt`. □

Алгоритм 3 (Мисра-Граеса). Запустить первый проход согласно Алгоритму 2. Запустить второй проход, во время которого для каждого ключа `k` в `cnt` посчитать его частоту. Вернуть все частые элементы.

1.4 Литература

1. J. Misra, David Gries, Finding repeated elements, Science of Computer Programming