Алгоритм Мо

Алгоритм Мо (англ. *Mo's algorithm*) — применяется для решения задач, в которых требуется отвечать на запросы $arr[l \dots r]$ на массиве *без* изменения элементов в оффлайн за время $O(Q \cdot \log Q + (N + Q) \cdot \sqrt{N})$, где Q — количество запросов, а N — количество элементов в массиве. Характерными примерами задач на этот алгоритм являются: нахождение моды на отрезке (число, которое встречается больше всех остальных), вычисление количества инверсий на отрезке.

Содержание

- 1 Алгоритм
- 2 Доказательство
- 3 Реализация
- 4 См. также
- 5 Источники информации

Алгоритм

В каждый момент времени будем хранить непрерывный отрезок $[a \dots b]$ исходного массива (будем называть его рабочим отрезком), вместе со структурой данных, которая умеет обрабатывать следующие операции:

- `addLeft(a − 1)`, `addRight(b + 1)` — операции, которые позволяют добавить элемент в рабочий отрезок слева и справа соответственно;
- `delLeft(a)`, `delRight(b)` — операции, которые позволяют удалить элемент рабочего отрезка слева и справа соответственно;
- `answer` — операция, которая позволяет получить ответ на запрос, если бы его границами был рабочий отрезок.

Изначально в качестве рабочего отрезка можно взять любой отрезок. Для удобства чтения будем считать изначальным отрезок $[1; 1)$, то есть $a = 1, b = 0$, фактически — пустой отрезок.

Запишем все запросы в массив, отсортируем их определённым способом (который будет описан ниже) будем их обрабатывать в том порядке, в котором они будут лежать в массиве после сортировки.

Допустим, что текущий рабочий отрезок — $[a \dots b]$, а первый необработанный запрос — $[l_i, r_i]$ тогда рассмотрим случаи:

- Если изначально было $a > l_i$, то будем добавлять в рабочий отрезок элементы слева по одному, пока граница не совпадёт;
- Если же это не так, то есть $a < l_i$ это значит, что в рабочем отрезке присутствуют те элементы, которых там быть не должно, и они должны быть удалены;
- При равенстве $a = l_i$ никаких действий с левой границей рабочего отрезка производить не потребуется.

Аналогично поступим с b и r_i. Для компактности и наглядности кода мы сначала расширим рабочий отрезок до отрезка $[l \dots r]$, где $l = \min(a, l_i)$, а $r = \max(b, r_i)$, а затем удалим лишние элементы при помощи операций `delLeft`, `delRight`, чтобы получить отрезок $[l_i \dots r_i]$, после чего вызовем `answer` и запомним ответ для этого запроса.

Теперь разберём поподробнее, как именно следует сортировать запросы для достижения вышеназванной асимптотики по времени.

Разделим все запросы на блоки размера K по левой границе: те запросы, для которых $1 \leqslant l_i \leqslant K$ — попадают в первую группу, те запросы, для которых $K + 1 \leqslant l_i \leqslant 2 \cdot K$ — во вторую, $2 \cdot K + 1 \leqslant l_i \leqslant 3 \cdot K$ — в третью, и так далее. Будем рассматривать все группы запросов независимо друг от друга. Если внутри каждой группы отсортировать запросы увеличению правой границы, то асимптотика по времени для обработки одной группы будет $O(N + Q_i \cdot K)$, где Q_i — количество запросов, принадлежащих группе под номером i.

Доказательство

Докажем, что на обработку одной группы суммарно уйдёт не больше чем $3 \cdot N + Q_i \cdot K$ операций `add` и `del`.

Для этого рассмотрим отдельно количество сделанных операций каждого из четырёх типов:

- изначально, до обработки группы, рабочий отрезок был $[a \ldots b]$, для обработки первого запроса может потребоваться $2 \cdot N$ операций `add`, `del`;
- `delRight` между отрезками одной группы не произойдёт ни разу, так как рабочий отрезок внутри одной группы будет только расширяться в сторону правого конца;
- `addRight` в этой группе произойдёт суммарно не больше чем N раз, так как минимальная правая граница — 1, а максимальная — N;
- для оставшихся двух операций рассмотрим два последовательных запроса $[l_i \ldots r_i]$, $[l_j \ldots r_j]$. Нетрудно заметить, что так как отрезки принадлежат одной группе, то $|l_i - l_j| < K$, следовательно, количество операций `addLeft` или `delLeft` не будет превосходить K, а суммарно для всей группы — $Q_i \cdot K$.

Таким образом, нетрудно видеть, все группы будут обработаны за время $O\left(\frac{N^2}{K} + K \cdot Q\right)$.

При выборе $K = \sqrt{N}$ с учётом сортировки по правой границе получается асимптотика времени $O(Q \cdot \log Q + (N + Q) \cdot \sqrt{N})$.

Реализация

```
struct Query:
  int l, r, index

int K = sqrt(N)
int a = 1, b = 0 // создаём пустой рабочий отрезок

bool isLess(Query a, Query b):
  if a.l / K != b.l / K:
    return a.l < b.l
  return a.r < b.r

function process(Query[Q] q):
  sort(q, isLess) // сортируем запросы, используя функцию isLess как оператор сравнения
  for i = 0 to Q - 1:
    while a > q[i].l:
      addLeft(a - 1)
      a -= 1
    while b < q[i].r:
      addRight(b + 1)
      b += 1
    while a < q[i].l:
      delLeft(a)
```

```
    a += 1
  while b > q[i].r:
    delRight(b)
    b -= 1
  result[q[i].id] = answer() // получаем ответ на [a...b]
```

Рассмотрим для наглядности решение задачи нахождения моды на отрезке:

Будем использовать код описанный выше, осталось только описать операции `addLeft`, `addRight`, `delLeft`, `delRight`. Так как в данной задаче порядок чисел на отрезке не важен, важно лишь количество вхождений каждого, то реализация отдельных функций для добавления слева и справа нам не потребуется.

Для простоты будем считать, что все числа **не превышают** N, тогда будем хранить массив $cnt[N+1]$, где $cnt[value]$ - количество вхождений числа $value$ в рабочем отрезке. Будем помимо этого массива хранить отсортированное множество $current$, в котором будут содержаться все пары вида $\langle$`cnt[value], value`$\rangle$, для ненулевых $cnt[value]$. Реализовать его можно, например, используя красно-черное дерево Тогда операции будут иметь следующий вид:

```
function add(int index):
  int value = arr[index]
  if cnt[value] > 0:
    current.erase((cnt[value], value))
  cnt[a[index]] += 1
  current.insert((cnt[value], value))

function del(int index):
  int value = arr[index]
  current.erase((cnt[value], value))
  cnt[a[index]] -= 1
  if cnt[value] > 0:
    current.insert((cnt[value], value))

function answer(): int
  return current.max.second // находим максимальную пару в множестве
```

Итоговая асимптотика решения: $O(Q \cdot \log Q + (N+Q) \cdot \sqrt{N} \cdot \log N)$.

См. также

- Корневая эвристика
- Разреженная таблица

Источники информации

- HackerEarth - Mo's algorithm (https://www.hackerearth.com/practice/notes/mos-algorithm/)

Источник — «http://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_Мо&oldid=60019»

- Эта страница последний раз была отредактирована 18 января 2017 в 19:08.