

Расчет статистик массива

Выполнил:
студент 2 курса магистратуры
05182м группы
Василевский А.В.

Задача

- Поиск минимума массива
- Поиск максимума массива
- Поиск медианы массива
- => Поиск k-й порядковой статистики массива
- Поиск среднего, дисперсии, среднеквадратичного отклонения и т.д.

Теория — порядковые статистики

Дано:

1. Выборка $X = [x_1, x_2, \dots, x_N]$.
2. Число k — номер статистики.
3. k -й порядковой статистикой называется элемент $X_{(k)} = (\text{sort}_{\leq} X)[k]$.

Соответственно:

$$\min = X_{(1)}; \quad \max = X_{(N)}; \quad \text{median} = X_{(N/2)}$$

Простейший алгоритм: отсортировать выборку и выбрать k -й порядковый элемент.

Теория — непорядковые статистики

Дана выборка $X = [x_1, x_2, \dots, x_N]$. **k -ми начальным и центральным моментами** выборки X называются величины соответственно:

$$\nu_k(X) = \frac{1}{N} \sum_{i=1}^N x_i^k; \quad \mu_k(X) = \frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^k$$

1. **Среднее:** $\bar{x} = \nu_1 = \frac{1}{N} \sum_{i=1}^N x_i$
2. **Второй момент:** $D_x^2 = \nu_2 = \frac{1}{N} \sum_{i=1}^N x_i^2$
3. **Среднеквадратичное отклонение:**

$$\sigma_x = \sqrt{\mu_2}; \quad \sigma_x^2 = \frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2 = \dots = D_x^2 - \bar{x}^2$$

Порядковые статистики – анализ

- Наивный общий алгоритм (полной сортировкой, например *quicksort*) – $O(n \log n)$.
- Частные случаи (\min , \max) – $O(n)$.
- Нельзя ли быстрее и в общем случае? – Можно (*quickselect*).
 - База – быстрая сортировка.
 - Основа – отказ от сортировки ненужных подмассивов.
 - (Метод «сокращай и властуй»)
 - Итоговая сложность: $O(n)$ в среднем, но $\Omega(n^2)$ в худшем случае.

Порядковые статистики – реализация

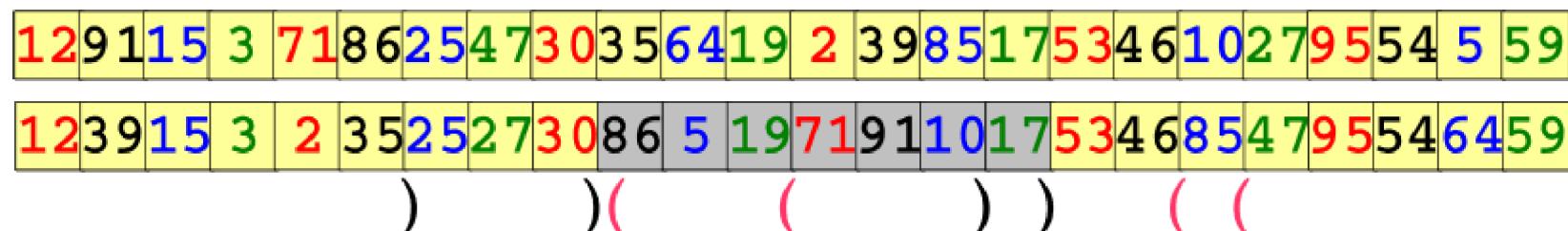
```
int findOrderStatistic(int[] array, int k) {
    int left = 0, right = array.length;
    while (true) {
        int mid = partition(array, left, right);

        if (mid == k) {
            return array[mid];
        }
        else if (k < mid) {
            right = mid;
        }
        else {
            left = mid + 1;
        }
    }
}
```

```
int partition(a: T[n], int l, int r)
T v = a[(l + r) / 2]
int i = l
int j = r
while (i ≤ j)
    while (a[i] < v)
        i++
    while (a[j] > v)
        j--
    if (i ≥ j)
        break
    swap(a[i++], a[j--])
return j
```

Порядковые статистики – параллелизация

- Наивный общий алгоритм (полной сортировкой) – так же, как и у алгоритма сортировки.
- Частные случаи (\min , \max) – параллельная редукция.
- *Quickselect* – в явном виде не параллелизуется.
 - Метод *partition* может быть распараллелен (как в *quicksort*).
 - Algorithms by Francis and Pannan, Tsigas and Zang and MCSTL.
 - Основа – разделение на ряд подпоследовательностей (1 2 3 4), применение *partition* к каждой из них параллельно, затем последовательный *partition* к средней (недоделанной) части диапазона



Порядковые статистики – известные реализации

- k-я порядковая статистика
 - `std::nth_element` – C++ STL [sequential]
 - `mcstl::parallel_nth_element` – C++ MCSTL [parallel (CPU)]
- Партиционирование
 - `std::partition` – C++ STL [sequential]
 - `cub::DevicePartition` – CUDA CUB [parallel (GPU)]
 - `thrust::partition` – C++ Thrust [parallel (GPU)]
- Min/Max
 - `std::min_element` – C++ STL [sequential]
 - `thrust::minmax_element` – C++ Thrust [parallel (GPU)]

Непорядковые статистики – анализ

Имея формулы:

$$1. \bar{x} = \frac{1}{N} \sum_{i=1}^N x_i$$

$$2. \sigma_x^2 = \left(\frac{1}{N} \sum_{i=1}^N x_i^2 \right) - \bar{x}^2 = D_x^2 - \bar{x}^2$$

можно свести задачу нахождения среднего и дисперсии к проблеме map/reduce:

$$1. \sum_{i=1}^N x_i \implies \text{foldr } (+) \ 0 \ X$$

$$2. \sum_{i=1}^N x_i^2 \implies \text{foldr } (+) \ 0 \ (\text{fmap } (^2) \ X)$$

Вычисление \bar{x} и D_x^2 можно выполнять за раз (объединить в один вызов map/reduce). Алгоритм имеет линейную сложность $O(n)$.

Непорядковые статистики – реализация

```
struct moments {
    double mean;
    double stdev_sq;
};

moments get_moments(const std::vector<double> & data) {
    moments m = { 0.0, 0.0 };

    for (size_t i = 0; i < data.size(); ++i) {
        m.mean += data[i];
        m.stdev_sq += data[i] * data[i];
    }

    m.mean /= data.size();
    m.stdev_sq /= data.size();

    m.stdev_sq -= m.mean * m.mean;

    return m;
}
```

Непорядковые статистики – параллелизация

- Также, как и у map/reduce.

Непорядковые статистики – известные реализации

- cv::meanStdDev – C++ OpenCV [sequential / parallel (?)]
- numpy.mean, numpy.std – Python NumPy [sequential]
- ... (все, что реализует parallel map/reduce)...
 - Thrust
 - CUDA libraries
 - MCSTL
 - ...

Варианты использования

- Статистические расчеты
 - Нормализация данных
 - Фильтрация по среднему / дисперсии
 - Детектирование (шума/сигнала)
- Медианная фильтрация с произвольной базой
 - При фиксированной базе алгоритм можно «развернуть», экономя на циклах. Сортировку можно заменить на более подходящую (битонную).
- Поиск соседнего (в некоторой метрике, например $|a-b|$) элемента в несортированном списке

Задача 1

- Найти пересечение двух массивов A и B.
 - Наивный подход. Для каждого элемента массива A проверяем наличие такого элемента в B. Двойной цикл. Сложность: $O(A.length * B.length) = O(N^2)$ в худшем случае ($A = B$).
 - Вспомогательное множество. Преобразованием массива B в множество (Set/HashSet) можно ускорить проверку того, содержится ли элемент в B. Сложность: $O(B.length)$ на преобразование $B \rightarrow HashSet$, $O(1)$ тест присутствия элемента в B, $O(A.length)$ на итерирование по элементам массива A. Итого: $O(A.length + B.length)$. Плюс $O(B.length)$ доп. памяти.
 - Для отсортированных массивов можно придумать более простое решение со сложностью $O(\max(A.length, B.length))$ [не приводится] и без использования дополнительной памяти, введя два курсора.
 - Если A или B уже HashSet, достаточно $O(A.length)$ операций проверок.

Задача 1

```
public static void main (String[] args)
{
    List<Integer> a = Arrays.asList(1, 2, 3, 4, 5);
    List<Integer> b = Arrays.asList(0, 2, 8, 4, 9);
    List<Integer> c = Arrays.asList(2, 4);
    System.out.println(intersect1(a, b));
    System.out.println(intersect2(a, b));
    System.out.println(c);
}

public static <T> List<T> intersect1(List<T> a, List<T> b) {
    return b.stream().filter(a::contains).collect(Collectors.toList());
}

public static <T> List<T> intersect2(List<T> a, List<T> b) {
    Set<T> index = new HashSet<>(a);
    return b.stream().filter(index::contains).collect(Collectors.toList());
}
```

Задача 2

- Сжатие строки методом RLE (Run Length Encoding).
- Пример: aabcccccaa → a2bc5a3.
- Реализация: цикл с подсчетом.

Задача 2

```
public static String rle(String s) {
    StringBuilder b = new StringBuilder(s.length());
    char c0 = '\0';
    int n = 0;
    for (int i = 0; i < s.length(); ++i) {
        char c = s.charAt(i);
        ++n;
        if (c0 != c) {
            if (n > 1) b.append(n);
            b.append(c);
            n = 0;
        }
        c0 = c;
    }
    ++n;
    if (n > 1) b.append(n);
    return b.toString();
}
```

```
main :: IO ()
main = putStrLn $ (rle "aabccccccaaa")

rle :: String -> String
rle cs = go cs 1 where
    go (c : [])      n          = c : (fmt n)
    go (c : d : cs) n | (c == d) = go (c : cs) (n + 1)
                      | otherwise = (c : (fmt n)) ++ (go (d : cs) 1)
    fmt n | n > 1      = show n
    fmt _ | otherwise = ""
```

Литература

- [1] М.Т. Гудрич, Р. Тамассия – Структуры данных и алгоритмы в Java – Мн.: Новое знание (2003)
- [2] neerc.ifmo.ru: «Поиск k-ой порядковой статистики»
- [3] neerc.ifmo.ru: «Быстрая сортировка»
- [4] Википедия, Wikipedia
- [5] Leonor Frias, Jordi Petit – Parallel Partition Revisited – WEA (2008)