Алгоритмы и структуры данных-1

SET 2. Задача A4.

Осень 2024. Клычков М. Д.

Пункт 1. Представим описание алгоритма в виде программы на C++:

```
1  long long CINV(std::vector<long long> &arr) {
2      // Base case
3      if (arr.size() == 1) {
4          return 0;
5      }
6
7      // 1. Divide
8      size_t mid = arr.size() / 2;
9      std::vector<long long> left(arr.begin(), arr.begin() + mid);
10     std::vector<long long> right(arr.begin() + mid, arr.end());
11
12     // 2. Conquer
13     auto count_left = CINV(left);
14     auto count_right = CINV(right);
15
16     // 3. Combine
17     long long count_merge{};
18     for (auto iter = arr.begin(), iter_left = left.begin(), iter_right = right.begin();
19             iter != arr.end(); ++iter) {
20         if (iter_left != left.end() &&
21               (iter_right == right.end() || *iter_left <= *iter_right)) {
22             *iter = *iter_left;
23             ++iter_left;
24         } else {
25             *iter = *iter_right;
26             count_merge += std::distance(iter_left, left.end());
27             ++iter_right;
28         }
29     }
30
31     return count_left + count_right + count_merge;
32 }
```

Рис. 1: `CINV` algorithm

Неформально опишу, что происходит в этом алгоритме. Для того чтобы посчитать количество инверсий, здесь применяется принцип *разделяй и властвуй*, а именно «модифицированный» алгоритм `MergeSort`. Помимо сортировки на каждом этапе `COMBINE` мы подсчитываем количество инверсий, а ответом будет являться их сумма.

Как же вычисляется это количество? Представим, что при объединении двух упорядоченных подмассивов в один (*merge*) текущий элемент из правого подмассива меньше, чем текущий элемент левого подмассива. Тогда оставшиеся (еще не обработанные) и само текущее числа левого подмассива образуют инверсию с рассматриваемым правым. Найденное число инверсий прибавляется к общему (строка 26 кода).

При таком подходе сохранятся все инверсии, которые образовывались элементами из разных подмассивов, поэтому искомое число инверсий будет получено.

Опишем суть шагов `DIVIDE`, `CONQUER` и `COMBINE`, характерных для `DaC` алгоритмов.

1. `DIVIDE`: Разделяем массив на два подмассива (если число элементов нечетное, количество элементов в правом подмассиве больше на 1).

2. `CONQUER`: Рекурсивно применяем алгоритм для полученных на этапе `DIVIDE` подмассивах. Сохраняем результат – количество инверсий в левой и правой половинах. Базовый случай (*остановка рекурсии*) происходит, когда длина подмассива становится равной единице, в таком случае количество инверсий равно 0.

3. `COMBINE`: Считаем количество инверсий между двумя упорядоченными подмассивами и собираем из них упорядоченный исходный массив. Если текущий элемент левого подмассива меньше, то он копируется в исходный массив на нужную позицию, но если элемент из правого подмассива больше, то помимо добавления в начальный массив, происходит увеличение счетчика инверсий. Как именно это происходит и почему это верно описано выше.

Выразим рекуррентное соотношение, описывающее функцию временной сложности $T(n)$. В рекурсивной ветке вычислений происходит два вызова подзадача вдвое меньшего размера, а в нерекурсивной – *merge* двух подмассивов, производящийся проходом цикла до n. Тогда можем записать (опускаем округление вниз):

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) \tag{1}$$

Воспользуемся мастер-теоремой: $a = 2, b = 2, k = 1, f(n) = 1, \log_b a = 1$. Тогда нам подходит случай $k = log_b a$, следовательно получим:

$$T(n) = O(n \cdot \log_2 n) \tag{2}$$

Стоит отметить, что разработанный алгоритм изменяет исходный массив, поэтому если есть необходимость сохранить начальный порядок, асимптотика не изменится, так как $T(n) = O(n \cdot \log_2 n) + O(n) = O(n \cdot \log_2 n)$

Заметим, что полученное общее число инверсий не всегда равно минимальному числу, достаточному для получения отсортированного массива. Например, в массиве $A = [5, 3, 2, 4, 1]$ всего 8 инверсий, однако, чтобы получить отсортированный массив достаточно сделать только две: $5 \leftrightarrow 1$ и $3 \leftrightarrow 2$.

Пункт 2. Сохраним общую схему решения, основанную на модификации `MergeSort`. Хотим, чтобы шаги `DIVIDE` и `CONQUER` должны остаться неизменными, попытаемся изменить только `COMBINE`.

Чтобы суть алгоритма (*поиск числа инверсий между двумя половинами*) осталась неизменной, попробуем сохранить встроенную в этот алгоритм сортировку. Но уже не получится так легко подсчитывать инверсии в одно и то же время, что и *merge* двух половин. Тогда добавим дополнительный проход, который будет считать только инверсии, а за ним будет уже соединение двух упорядоченных подмассивов.

Таким образом в нерекурсивную ветку вычислений рекурренты добавится терм n, но он никак не повлияет на асимптотику:

$$T'(n) = T(n) + n = 2T\left(\frac{n}{2}\right) + O(n) + n = 2T\left(\frac{n}{2}\right) + O(n) = O(n \cdot \log_2 n) \tag{3}$$

Также опишем этот алгоритм кодом 2

```cpp
long long CSINV(std::vector<long long> &arr) {
    // Base case
    if (arr.size() == 1) {
        return 0;
    }

    // 1. Divide
    size_t mid = arr.size() / 2;
    std::vector<long long> left(arr.begin(), arr.begin() + mid);
    std::vector<long long> right(arr.begin() + mid, arr.end());

    // 2. Conquer
    auto count_left = CSINV(left);
    auto count_right = CSINV(right);

    // 3. Combine

    // 3.1. Combine - Count inversions which starts at left half and ends at right part
    long long count_merge{};
    for (auto iter_left = left.begin(), iter_right = right.begin();
            iter_left != left.end() && iter_right != right.end();) {
        if (iter_left != left.end() &&
              (iter_right == right.end() || *iter_left <= 2 * *iter_right)) {
            ++iter_left;
        } else {
            count_merge += std::distance(iter_left, left.end());
            ++iter_right;
        }
    }

    // 3.2. Combine - Merge two halfs to make array sorted
    for (auto iter = arr.begin(), iter_left = left.begin(), iter_right = right.begin();
            iter != arr.end(); ++iter) {
        if (iter_left != left.end() &&
              (iter_right == right.end() || *iter_left <= *iter_right)) {
            *iter = *iter_left;
            ++iter_left;
        } else {
            *iter = *iter_right;
            ++iter_right;
        }
    }

    return count_left + count_right + count_merge;
}
```

Рис. 2: `CSINV` algorithm