Речь к презентации о сортировках слиянием

В. С. Верхотуров*

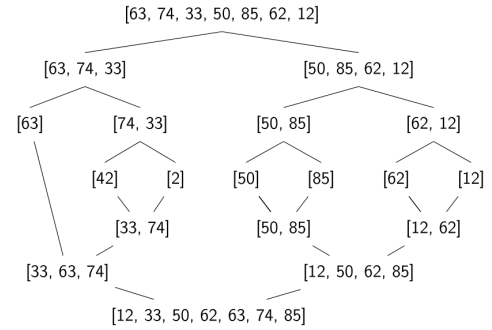
I. СОДЕРЖАНИЕ

СОДЕРЖАНИЕ

II. СЛАЙД 3

Определение

Сортировка слиянием — это алгоритм «разделяй и властвуй».



3/17

Сортировка слиянием — это алгоритм «разделяй и властвуй».

1. Он последовательно делит входной список длины n пополам, пока не останется n списков размера 1 (верхняя часть графа);
2. Затем пары списков объединяются вместе с меньшим первым элементом среди пары списков, добавляемых на каждом шаге (нижняя часть графа).

III. СЛАЙДЫ 4, 5

$$\left\{\begin{matrix}33 & 63 & 74\\ 12 & 50 & 62 & 85\end{matrix}\right. \quad (1)$$

$$12 \quad \left\{\begin{matrix}33 & 63 & 74\\ 50 & 62 & 85\end{matrix}\right. \quad (2)$$

$$12 \quad 33 \quad \left\{\begin{matrix}63 & 74\\ 50 & 62 & 85\end{matrix}\right. \quad (3)$$

$$12 \quad 33 \quad 50 \quad \left\{\begin{matrix}63 & 74\\ 62 & 85\end{matrix}\right. \quad (4)$$

$$12 \quad 33 \quad 50 \quad 62 \quad \left\{\begin{matrix}63 & 74\\ 85\end{matrix}\right. \quad (5)$$

4/17

$$12 \quad 33 \quad 50 \quad 62 \quad 63 \quad \left\{\begin{matrix}74\\ 85\end{matrix}\right. \quad (6)$$

$$12 \quad 33 \quad 50 \quad 62 \quad 63 \quad 74 \quad \left\{\begin{matrix}\\ 85\end{matrix}\right. \quad (7)$$

$$12 \quad 33 \quad 50 \quad 62 \quad 63 \quad 74 \quad 85 \quad \left\{\begin{matrix}\\ \end{matrix}\right. \quad (8)$$

5/17

Принцип объединения списков (merge).

1. Даны два массива. Гарантируется, что каждый из массивов отсортирован;

* РТУ МИРЭА; Электропочта:valery.verkhoturov1505@gmail.com

2. На каждой итерации 1–8 сравниваются первые элементы массива. Наименьший элемент достаётся из исходного массива и добавляется в конец отсортированного массива. Итерации заканчиваются, когда все элементы перешли в отсортированный массив.

IV. СЛАЙД 6

Реализация однопоточного алгоритма на CPython

```
1  from random import randint
2  from numbers import Number
3
4  def merge(arrays: list[list[Number]]) -> list[Number]:
5    assert len(arrays) == 2
6    x, y = arrays
7    index_x = index_y = 0
8    out = []
9    while index_x < len(x) and index_y < len(y):
10     if x[index_x] < y[index_y]:
11       out.append(x[index_x])
12       index_x += 1
13     else:
14       out.append(y[index_y])
15       index_y += 1
16   out += x[index_x:] + y[index_y:]
17   return out
18
19 def merge_sort(arr : list[Number]) -> list[Number]:
20   if len(arr) <= 1:
21     return arr
22   if len(arr) == 2:
23     return arr if arr[0] < arr[1] else [arr[1], arr[0]]
24   mid = len(arr) // 2
25   return merge(merge_sort(arr[:mid]), merge_sort(arr[mid:]))
26
27
28 def test_merge_sort():
29   input_array = [randint(1, 100) for i in range(10)]
30   print(merge_sort(input_array))
31
32 test_merge_sort()
33
```

6/17

Код на Python читается снизу вверх. Разобран на слайдах 7, 8, 9.

V. СЛАЙД 7

Реализация однопоточного алгоритма на CPython. Часть 1

```
1  def test_merge_sort():
2    input_array = [randint(1, 100)
3        for i in range(10)]
4    print(merge_sort(input_array))
5
6  test_merge_sort()
7
```

7/17

Строка 1: Определение функции, проверяющей работоспособность алгоритма;

2, 3: Создание массива из 10 элементов псевдослучайных чисел от 1 до 100;

4: Сортировка, вывод отсортированного массива;

6: Вызов функции, определённой в строке 1, проверяющей работоспособность алгоритма.

VI. СЛАЙД 8

Реализация однопоточного алгоритма на CPython. Часть 2

```
1  def merge_sort(arr : list[Number]) -> list[Number]: #
       T^{merge}(n)
2    if len(arr) <= 1:
3      return arr
4    if len(arr) == 2:
5      return arr
6        if arr[0] < arr[1]
7        else [arr[1], arr[0]]
8    mid = len(arr) // 2
9    return merge(merge_sort(arr[:mid]),
10         merge_sort(arr[mid:])) # 2 * T^{sort}(n / 2)
11
```

8/17

1: Определении функции, которая разделяет исходный массив пополам в рекурсии, сливает и возвращает;

2–3: Если длинна массива равна 0 или 1, то массив возвращается без изменений;

4–7: Массив из 2 элементов сортируется. Если первый элемент меньше второго, то возвращается без изменений, иначе элементы меняются местами, изменённый массив возвращается;

8, 9: Если массив состоит больше, чем из двух элементов, то находится индекс посередине массива, массив делится на две половины `arr[:mid]` и `arr[mid:]`, каждая половина рекурсивно сортируется $T^{sort}(n\times 2)$, вместе сливаются $2\times T^{sort}(n\times \times 2)$.

VII. СЛАЙД 9

Реализация однопоточного алгоритма на CPython. Часть 3

```
1  def merge(arrays: list[list[Number]]) \
2          -> list[Number]:
3    assert len(arrays) == 2
4    x, y = arrays
5    index_x = index_y = 0
6    out = []
7    while index_x < len(x) and \
8        index_y < len(y):
9      if x[index_x] < y[index_y]:
10       out.append(x[index_x])
11       index_x += 1
12     else:
13       out.append(y[index_y])
14       index_y += 1
15   out += x[index_x:] + y[index_y:]
16   return out
17
```


1, 2: Определении функции слияния, которая принимает массив с двумя сортированными массивами и возвращает один отсортированный массив;

3: Проверка, находятся ли в принимаемом массиве два внутренних массива,

4: x — первый внутренний массив, y — второй внутренний массив;

5: Индексы первых элементов массивов x, y, которые будут сравниваться, равны 0;

6: Создание пустого выходного массива;

7–8: Пока в массиве x И y есть хотя бы по одному элементу, то:

9-14: Добавить в выходной массив наименьшее значение из первых элементов массивов x, y, увеличить индекс начально элемента того массива, из которого было взято значение;

15: Добавить оставшийся элемент к выходному массиву;

16: Вернуть из функции выходной массив.

Примечание: на слайдах 4, 5 говорилось об извлечении элемента из исходного, что потребовало бы создание нового массива размера $n-1$ и копирование элементов исходного массива. Эффективнее увеличить индекс на 1 и принимать его за индекс первого элемента на следующей итерации.

VIII. СЛАЙД 10

Вычислительная сложность

Худшая выч. сложность:

$$O(n \log n) \tag{9}$$

Лучшая выч. сложность:

$$O(n \log n) \tag{10}$$

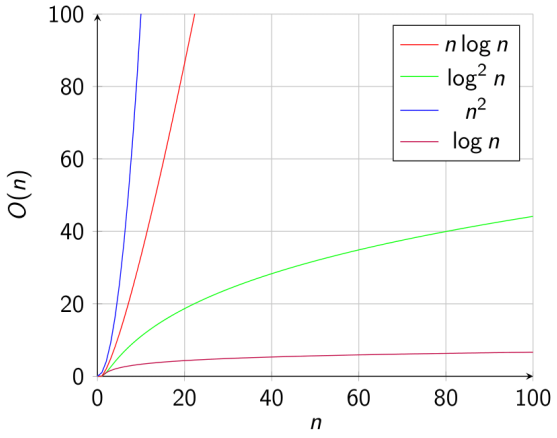
Средняя выч. сложность:

$$O(n \log n) \tag{11}$$

Временная сложность:

$$T^{sort}(n) = 2T^{sort}\left(\frac{n}{2}\right) + T^{merge}(n) = \\ = 2T^{sort}\left(\frac{n}{2}\right) + \Theta(n \log n) \tag{12}$$

IX. СЛАЙД 11



Средние выч. сложности:

$n \log n$**:** Выч. сложность однопоточного алгоритма сортировки слиянием;

$\log^2 n$**:** Сортировка Бетчера (Bitonic sorter);

n^2**:** Сортировка пузырьком;

$\log n$**:** Параллельная сортировка слиянием.

X. СЛАЙД 12

Параллельные вычисления в CPython (side-stepping the GIL (Global Interpreter Lock))

Рис.: Иллюстрация GIL

- `multiprocessing.Process`, `multiprocessing.Pool` https://docs.python.org/3/library/multiprocessing.html,
- C-расширение, расширение на Cython
- `os.system("python child.py")`,

или...

Наиболее известная реализация Python является CPython (интерпретатор), которая имеет глобальную блокировку интерпретатора, не позволяю распараллелить программу. Асинхронные функции, инструменты стандартных библиотек `threadig`, `concurrency` работают конкурентно. Необходимы инструменты, которые обходят блокировщик (side-stepping the GIL):

стандартная библиотека multiprocessing: позволяет разделить задачи по процессам (Process создаёт 1 процесс, Pool — много);

C-расширение: распараллеливание кода на Си, вызванного из CPython;

Cython: Реализация Python, имеющая возможность обхода блокировщика, возможно написание расширение для CPython;

Консольная команда: Запуск подпроцесса с помощью консольной команды.

Примечание: стандартная библиотека `subprocess` также предоставляет возможность создания подпроцесса с другой программой, не подходит для конкретной задачи.

XI. СЛАЙД 13

Многопроцессная реализация алгоритма слияния

```
1  import multiprocessing
2  import math
3
4  def parallel_merge_sort(arr: list[Number]) \
5                          -> list[Number]:
6    processes = multiprocessing.cpu_count()
7    with multiprocessing.Pool(processes = processes) \
8                                              as pool:
9      size = math.ceil(len(arr) / processes)
10     arr = [arr[i * size:(i + 1) * size]
11            for i in range(processes)]
12     arr = pool.map(merge_sort, arr)
13     while len(arr) > 1:
14       extra = arr.pop() if len(arr) % 2 == 1 else None
15       arr = [[arr[i], arr[i + 1]]
16              for i in range(0, len(arr), 2)]
17       arr = pool.map(merge, arr) + \
18             ([extra] if extra else [])
19   return arr[0]
20
```


На слайде 14 представлена часть кода с 6 по 14 строку, на слайде 15 представлена часть кода с 13 по 19.

Примечание: Строка 7 на данном слайде является строкой 4 на слайде 14.

XII. СЛАЙД 14

Многопроцессная реализация алгоритма слияния. Часть 1

```
1  # input arr = [38, 25, 77, 45, 46, 64, 88, 76, 97, 35]
2
3  processes = multiprocessing.cpu_count() # processes=8
4  pool = multiprocessing.Pool(processes = processes)
5  size = math.ceil(len(arr) / processes) # size = 2
6  arr = [arr[i * size:(i + 1) * size]
7         for i in range(processes)]
8  # arr = [[38, 25], [77, 45], [46, 64], [88, 76],
9  #         [97, 35], [], [], []]
10 arr = pool.map(merge_sort, arr) # T^{sort}(n / 2)
11 # arr = [[25, 38], [45, 77], [46, 64], [76, 88],
12 #         [35, 97], [], [], []]
13
```


1: Дан массив из 10 элементов;

3: Определение, сколько компьютер имеет ядер;

4: Создание бассейна процессов с размером, равным кол-ву ядер;

5: Определение, какого размера будут массивы, отданные процессам;

6–9: Разбиение исходного массива на массивы, количество которых равно кол-ву ядер;

10-12: Применение для каждого полученного после разбития массива функции `merge_sort` параллельно (в зависимости от размера бассейна `pool` доступных ядер). `arr` становится равен массиву с массивами с отсортированными элементами.

XIII. СЛАЙД 15

Многопроцессная реализация алгоритма слияния. Часть 2

```
# arr = [[25, 38], [45, 77], [46, 64], [76, 88],
#        [35, 97], [], [], []]
while len(arr) > 1:
  extra = arr.pop() if len(arr) % 2 == 1 else None
  arr = [[arr[i], arr[i + 1]]
         for i in range(0, len(arr), 2)]
  # arr = [[[25, 38], [47, 77]], [[46, 64], [76, 88]],
  #        [[35, 97], []], [[], []]]
  arr = pool.map(merge, arr) + \
        ([extra] if extra else []) # T^{merge}(n)
  # Iteration 1 out: [[25, 38, 45, 77],
  #                   [46, 64, 76, 88],
  #                   [35, 97], []]
  # Iteration 2 out:
  # [[25, 38, 45, 46, 64, 76, 77, 88], [35, 97]]
  # Iteration 3 out:
  # [[25, 35, 38, 45, 46, 64, 76, 77, 88, 97]]
return arr[0]

```


Отсортированные массивы необходимо слить.

3: Пока не остался один внутренний массив:

4: Если пары для последнего внутреннего массива не нашлось, то он будет `extra`;

5–8: Внутренние массивы объединить в пары;

9–13: Для каждой пары применить функцию `merge` и добавить `extra` массив с элементами;

18: Вернуть слитый массив.

XIV. СЛАЙД 16

Временная сложность

$$T^{sort}(n) = T^{sort}\left(\frac{n}{2}\right) + T^{merge}(n) = T^{sort}\left(\frac{n}{2}\right) + \Theta(\log n) \quad (13)$$