Хронология работы:

В 6 главе проверить merge. Описание алгоритма.

Таблица файлов:

**Раздел 6 Sorting**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Name | doc | drn | go | pdf | jpg | md |
| buubleSort | + | + | + |  |  |  |
| selectSort | + | + | + |  |  |  |
| insertionSort | + | + | + |  |  |  |
| quickSort | + | + | + |  |  |  |
| shellSort | + | + | + |  |  |  |
| heaport | + | + | + |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

MergeSort

В языке Golang существует несколько методов сортировки, один из них - это сортировка слиянием (merge sort). Этот метод является устойчивым, т.е. он сохраняет относительный порядок равных элементов при сортировке. Сложность алгоритма сортировки слиянием составляет O(n\*log(n)).

В приведенном вами коде представлена реализация сортировки слиянием в Golang. Давайте разберем его подробнее:

1. Функция main() - это точка входа в программу. В ней создается массив arr из 10 целых чисел, выводится на печать исходный массив, затем выводится на печать отсортированный массив с помощью функции mergeSort().

2. Функция merge() - это вспомогательная функция, которая используется для слияния двух отсортированных массивов a и b в один отсортированный массив result. В этой функции используются три индекса i, j и k для обхода массивов a и b и заполнения массива result.

3. Функция mergeSort() - это рекурсивная функция, которая реализует сортировку слиянием. Она принимает на вход массив numbers и возвращает отсортированный массив. Если длина массива меньше 2, то он возвращается без изменений. Иначе вычисляется индекс middle, который делит массив на две равные части, затем рекурсивно вызывается функция mergeSort() для каждой из частей и с помощью функции merge() они сливаются в один отсортированный массив.

Как работает модуль merge?

Модуль merge в сортировке слиянием используется для слияния двух отсортированных массивов в один отсортированный массив. Алгоритм слияния выглядит следующим образом:

1. Создается пустой массив result для хранения отсортированного массива.

2. Инициализируются индексы i и j для обхода массивов a и b соответственно, а также индекс k для заполнения массива result.

3. Пока индекс k не достигнет длины массива result, выполняются следующие действия:

- Если индекс i достиг длины массива a, то элемент из массива b с индексом j добавляется в массив result и индекс j увеличивается на 1.

- Если индекс j достиг длины массива b, то элемент из массива a с индексом i добавляется в массив result и индекс i увеличивается на 1.

- Если элемент из массива a с индексом i больше или равен элементу из массива b с индексом j, то элемент из массива b с индексом j добавляется в массив result и индекс j увеличивается на 1.

- Если элемент из массива a с индексом i меньше элемента из массива b с индексом j, то элемент из массива a с индексом i добавляется в массив result и индекс i увеличивается на 1.

4. Возвращается отсортированный массив result.

Этот алгоритм гарантирует, что элементы в массиве result будут отсортированы, так как на каждом шаге выбирается наименьший элемент из оставшихся элементов в массивах a и b.

**Раздел 7 Searching**

|  |  |  |
| --- | --- | --- |
| Name | Содержание программы | Адрес: |
|  |  |  |
|  |  |  |
|  |  |  |

**Раздел 8 Деревья**

|  |  |  |
| --- | --- | --- |
| Name | Содержание программы | Адрес: |
| bstFind | Обход дерева и поиск узла |  |
|  | Деревья avl |  |
| rbtAll | Дервья rbt |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Раздел 9 ГРАФЫ**

|  |  |  |
| --- | --- | --- |
| Name | Содержание программы | Адрес: |
| graphBFS | Обход графа в ширину |  |
| graphDFS | Обход графа в глубину |  |
| graphBasic | Базовые функции |  |
| graphMiniPath | Выбор пути между вершинами |  |
| graphDijkstra | Алгоритм Дейкстра (Sym) |  |
|  |  |  |
|  |  |  |