|  |
| --- |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МИРЭА – Российский технологический университет"**  **РТУ МИРЭА** |

Институт комплексной безопасности и специального приборостроения

Кафедра КБ-3 «Управление и моделирование систем»

**ОТЧЕТ   
о выполнении лабораторной работы №1**

**«Реализация сортировки линейных структур данных»**

**по дисциплине   
«Программная реализация нелинейных структур»**

**Вариант № 90**

Выполнил: студент 2 курса

Группы **\_\_\_\_\_\_\_\_\_\_\_**

шифр \_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
*(фио студента)*

Проверил:

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

Москва 2022 г.

**Задание на лабораторную работу № 1.**

В рамках лабораторной работы №1 требуется программно реализовать (с помощью указателей (однонаправленных/двунаправленный динамический линейный связанный список, массива или используя стандартный контейнер библиотеки STL “stack” или «queue» - по варианту) абстрактный тип данных (АТД) в соответствии с заданием (стек, дек, очередь с одной головой, очередь с головой и хвостом).

Абстрактный тип данных должен позволять осуществлять только операции, присущие типу линейного связанного списка:

* получить значение первого элемента (на выходе),
* добавить элемент (на вход),
* удалить элемент из списка (на выходе),
* проверить – список пуст,
* обнулить (проинициализировать) список (конструктор, при необходимости).
* деструктор (при необходимости)

Используя разработанный АТД и указанный набор операций, необходимо реализовать заданный алгоритм сортировки последовательности элементов заданного типа, при этом следует учитывать, что разрешен доступ (чтение/извлечение) только к элементу на выходе.

На основе исходного текста программы получить аналитическую оценку трудоемкости работы алгоритма сортировки, используя О-символику для каждого реализованного метода АТД и сортировки в целом.

**Вариант № 90.**

**Реализация связи элементов линейного списка: Указатели**

**Способ организации линейного связанный список: Очередь с головй и хвостом**

**Алгоритм сортировки: Фиксированное двухпутевое слияние**

**Теория о сортировках.**

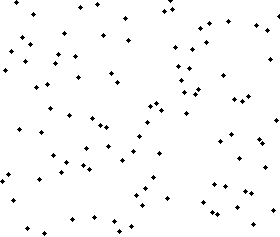
**Сортировка слиянием** (англ. *merge sort*) — алгоритм сортировки, который упорядочивает списки (или другие структуры данных, доступ к элементам которых можно получать только последовательно, например — потоки) в определённом порядке. Эта сортировка — хороший пример использования принципа «разделяй и властвуй». Сначала задача разбивается на несколько подзадач меньшего размера. Затем эти задачи решаются с помощью рекурсивного вызова или непосредственно, если их размер достаточно мал. Наконец, их решения комбинируются, и получается решение исходной задачи.

Рисунок иллюстрирует сортировку слиянием.

**Алгоритм**

1. Сортируемый массив разбивается на две части примерно одинакового размера;
2. Каждая из получившихся частей сортируется отдельно, например — тем же самым алгоритмом;
3. Два упорядоченных массива половинного размера соединяются в один.

1.1. — 2.1. Рекурсивное разбиение задачи на меньшие происходит до тех пор, пока размер массива не достигнет единицы (любой массив длины 1 можно считать упорядоченным).

3.1. Соединение двух упорядоченных массивов в один.  
Основную идею слияния двух отсортированных массивов можно объяснить на следующем примере. Пусть мы имеем два уже отсортированных по возрастанию подмассива. Тогда:  
3.2. Слияние двух подмассивов в третий результирующий массив.  
На каждом шаге мы берём меньший из двух первых элементов подмассивов и записываем его в результирующий массив. Счётчики номеров элементов результирующего массива и подмассива, из которого был взят элемент, увеличиваем на 1.  
3.3. «Прицепление» остатка.  
Когда один из подмассивов закончился, мы добавляем все оставшиеся элементы второго подмассива в результирующий массив.

**Листинг программы с расчетами.**

from typing import Generic, Optional, TypeVar

VT = TypeVar("VT")

class Node(Generic[VT]):

    \_value: VT

    next: Optional["Node[VT]"]

    def \_\_init\_\_(self, value: VT, next: Optional["Node[VT]"] = None):

        self.\_value = value

        self.next = next

    @property

    def value(self) -> VT:

        return self.\_value

    def \_\_str\_\_(self) -> str:

        return str(self.\_value)

class Queue(Generic[VT]):

    \_head: Optional[Node[VT]]

    \_tail: Optional[Node[VT]]

    \_size: int

    \_n\_op: int

    def \_\_init\_\_(self):

        self.\_head = None

        self.\_tail = None

        self.\_size = 0

        self.\_n\_op = 0

    def push(self, value: VT):  # 7

        node = Node(value)  # 2

        if self.is\_empty():  # 1

            self.\_head = node  # 1

            self.\_tail = node  # 1

        else:

            self.\_tail.next = node  # 2

            self.\_tail = node  # 1

        self.\_size += 1  # 1

        self.\_n\_op += 7

    def pop(self) -> VT:  # 7

        if self.is\_empty():  # 1

            raise Exception("Can't pop from empty queue!")

        node = self.\_head  # 1

        self.\_head = node.next  # 2

        if self.\_head is None:  # 1

            self.\_tail = None  # 1

        self.\_size -= 1  # 1

        self.\_n\_op += 7

        return node.value

    def is\_empty(self) -> bool:  # 2

        return self.\_size == 0

    @property

    def head(self) -> VT:  # 2

        return self.\_head.value

    @property

    def tail(self) -> VT:  # 2

        return self.\_tail.value

    @property

    def size(self) -> int:  # 1

        return self.\_size

    @property

    def n\_op(self) -> int:

        return self.\_n\_op

def print\_queue(queue: Queue[VT]):

    elements = []

    for \_ in range(queue.size):

        el = queue.pop()

        elements.append(el)

        queue.push(el)

    print("Queue[" + ", ".join(map(str, elements)) + "]")

def rotate(queue: Queue[VT]) -> None:  # 14

    queue.push(queue.pop())

def seek(queue: Queue[VT], pos: int) -> VT:  # 28n + 2

    for \_ in range(pos):  # n \* (

        rotate(queue)  # 14

    # ) = 14n

    el = queue.head  # 2

    for \_ in range(queue.size - pos):  # n \* (

        rotate(queue)  # 14

    # ) = 14n

    return el

def pop\_by\_pos(queue: Queue[VT], pos: int) -> VT:  # 28n + 7

    for \_ in range(pos):  # n \* (

        rotate(queue)  # 14

    # ) = 14n

    el = queue.pop()  # 7

    for \_ in range(queue.size - pos):  # n \* (

        rotate(queue)  # 14

    # ) = 14n

    return el

def push\_by\_pos(queue: Queue[VT], el: VT, pos: int) -> None:  # 28n + 7

    if pos >= queue.size:  # 2

        queue.push(el)  # 7

        return

    for \_ in range(pos):  # n \* (

        rotate(queue)  # 14

    # ) = 14n

    queue.push(el)  # 7

    for \_ in range(queue.size - pos - 1):  # n \* (

        rotate(queue)  # 14

    # ) = 14n

def swap(queue: Queue, pos1: int, pos2: int):  # 112n + 28

    temp = pop\_by\_pos(queue, pos1)  # 28n + 7

    push\_by\_pos(queue, pop\_by\_pos(queue, pos2 - 1), pos1)  # 56n + 14

    push\_by\_pos(queue, temp, pos2)  # 28n + 7

def slice\_(queue: Queue[VT], l: int = 0, r: int = -1) -> Queue[VT]:  # 49n + 6

    buffer = Queue[VT]()  # 2

    if r == -1:  # 1

        r = queue.size - 1  # 3

    for \_ in range(l):  # n \* (

        rotate(queue)  # 14

    # ) = 14n

    for \_ in range(r - l + 1):  # n \* (

        el = queue.pop()  #  7

        buffer.push(el)  # 7

        queue.push(el)  # 7

    # ) = 21n

    for \_ in range(queue.size - r - 1):  # n \* (

        rotate(queue)  # 14

    # ) = 14n

    return buffer

def merge(queue1: Queue[VT], queue2: Queue[VT]) -> Queue[VT]:  # 74n + 3

    out = Queue[VT]()  # 1

    while queue1.size > 0 and queue2.size > 0:  # n \* (

        if queue1.head < queue2.head:  # 3

            out.push(queue1.pop())  # 14

        else:  # 1

            out.push(queue2.pop())  # 14

    # ) = 32n

    while queue1.size > 0:  # n \* (

        out.push(queue1.pop())  # 14

    # ) = 14n

    while queue2.size > 0:  # n \* (

        out.push(queue2.pop())  # 14

    # ) = 14n

    out.\_n\_op += queue1.n\_op + queue2.n\_op

    return out  # 14n + 2

def fixed\_two\_way\_merge\_sort(queue: Queue[VT]) -> Queue[VT]:  # 2n \* log(n) + 116n + 13

    if queue.size <= 1:  # 1

        return queue  # 1

    queue\_size = queue.size  # 2

    left\_queue = Queue[VT]()  # 2

    right\_queue = Queue[VT]()  # 2

    for \_ in range(queue\_size // 2):  # (n / 2) \* (

        left\_queue.push(queue.pop())  # 14

    # ) = 7n

    for \_ in range(queue\_size - queue\_size // 2):  # (n / 2) \* (

        right\_queue.push(queue.pop())  # 14

    # ) = 7n

    left\_queue = fixed\_two\_way\_merge\_sort(left\_queue)  # ~= nlog(n)

    right\_queue = fixed\_two\_way\_merge\_sort(right\_queue)  # ~= nlog(n)

    buff = merge(left\_queue, right\_queue)  # 88n + 5

    for \_ in range(buff.size):  # n \* (

        queue.push(buff.pop())  # 14

    # ) = 14n

    queue.\_n\_op += buff.n\_op

    return queue

if \_\_name\_\_ == "\_\_main\_\_":

    import sys

    import time

    from random import randint

    if len(sys.argv) < 2 or sys.argv[1] not in ["example", "tests"]:

        print(f"Usage: python3 {sys.argv[0]} [example/tests]")

        exit(1)

    if sys.argv[1] == "example":

        struct = Queue[int]()

        for \_ in range(20):

            struct.push(randint(-10000, 10000))

        fixed\_two\_way\_merge\_sort(struct)

        print\_queue(struct)

    elif sys.argv[1] == "tests":

        tests = 10

        step = 5000

        for test\_num in range(1, tests + 1):

            struct = Queue[int]()

            for \_ in range(test\_num \* step):

                struct.push(randint(-10000, 10000))

            start\_time = time.time()

            fixed\_two\_way\_merge\_sort(struct)

            total\_time = time.time() - start\_time

            print(f"Test: {test\_num}")

            print(f"Elements count: {test\_num \* step}")

            print(f"Total time: {total\_time}")

            print(f"N\_OP: {struct.n\_op}")

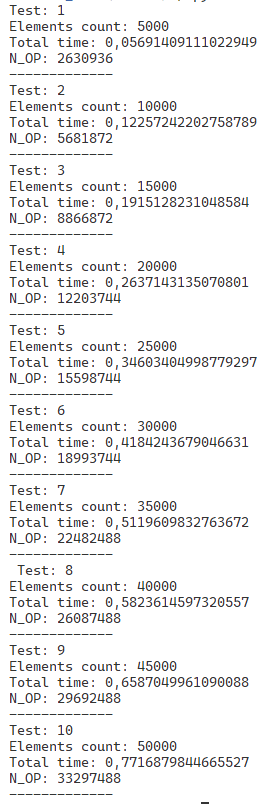
            print("-------------")

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во элементов | F(n) | O(F(n)) | Т(n) (сек) | N\_op |
| 5000 | 702890 | 61439 | 0,056914091 | 2630936 |
| 10000 | 1425767 | 132877 | 0,122572422 | 5681872 |
| 15000 | 2156193 | 208090 | 0,191512823 | 8866872 |
| 20000 | 2891521 | 285754 | 0,263714314 | 12203744 |
| 25000 | 3630495 | 365241 | 0,34603405 | 15598744 |
| 30000 | 4372373 | 446180 | 0,418424368 | 18993744 |
| 35000 | 5116668 | 528327 | 0,511960983 | 22482488 |
| 40000 | 5863030 | 611508 | 0,58236146 | 26087488 |
| 45000 | 6611200 | 695594 | 0,658704996 | 29692488 |
| 50000 | 7360977 | 780482 | 0,771687984 | 33297488 |

|  |  |  |  |
| --- | --- | --- | --- |
| С1=F(n)/T(n) | С2=O(F(n))/T(n) | С3=F(n)/N\_op | С4=O(F(n))/N\_op |
| 12350019,3 | 1079496,495 | 0,26716352 | 0,023352359 |
| 11632039,44 | 1084070,312 | 0,250932659 | 0,023386152 |
| 11258740,86 | 1086559,74 | 0,243174058 | 0,023468267 |
| 10964598,23 | 1083575,039 | 0,236937246 | 0,023415294 |
| 10491727,69 | 1055505,988 | 0,232742779 | 0,023414771 |
| 10449614,86 | 1066334,278 | 0,230200717 | 0,023490906 |
| 9994253,231 | 1031968,007 | 0,227584585 | 0,023499506 |
| 10067682,01 | 1050049,733 | 0,224744904 | 0,023440681 |
| 10036663,46 | 1056001,831 | 0,222655655 | 0,023426588 |
| 9538799,613 | 1011395,848 | 0,221067038 | 0,023439667 |

**Таблица результата экспериментов и графики зависимостей**

**Скриншот работы пограммы:**



**Выводы.**

По результатам данной работы был реализован класс для работы с очередью с головой и хвостом. Очередь была реализована через указатели. Были реализованы базовые операции для очереди и рассчитана их алгоритмическая сложность. Так же был реализован алгоритм сортировки слиянием. После аналитической оценки и экспериментов было установлено, алгоритм сортировки слиянием для очереди с головой и хвостом на указателях имеет зависимость от числа элементов.

**Литература:**

1. Левитин А. В. Глава 6. Метод декомпозиции: Сортировка слиянием // Алгоритмы. Введение в разработку и анализ — М.: Вильямс, 2006.