|  |
| --- |
| 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 № 34**

Выполнил: студент 2 курса

Группы **\_\_\_\_\_\_\_\_\_\_\_**

шифр \_\_\_\_\_\_\_\_

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

Проверил:

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

Москва 2021 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 № 34.**

**Реализация связи элементов линейного списка: Указатели**

**Способ организации линейного связанный список: Очередь с головой и хваостом**

**Алгоритм сортировки: Быстрая по Хоару (с медианным элементом)**

**Теория о сортировках.**

**Быстрая сортировка Хоара (QuickSort).**

Один из самых быстрых известных универсальных алгоритмов сортировки массивов: в среднем O ( n log ⁡ n ) {\displaystyle O(n\log n)} обменов при упорядочении n {\displaystyle n} элементов; из-за наличия ряда недостатков на практике обычно используется с некоторыми доработками.

QuickSort является существенно улучшенным вариантом алгоритма сортировки с помощью прямого обмена (его варианты известны как «Пузырьковая сортировка» и «Шейкерная сортировка»), известного в том числе своей низкой эффективностью. Принципиальное отличие состоит в том, что в первую очередь производятся перестановки на наибольшем возможном расстоянии и после каждого прохода элементы делятся на две независимые группы (таким образом улучшение самого неэффективного прямого метода сортировки дало в результате один из наиболее эффективных улучшенных методов).

Рисунок иллюстрирует сортировку Хоара.

Шаги алгоритма:

1. Выбрать элемент из массива. Назовём его опорным.

2. Разбиение: перераспределение элементов в массиве таким образом, что элементы, меньшие опорного, помещаются перед ним, а большие или равные - после.

3. Рекурсивно применить первые два шага к двум подмассивам слева и справа от опорного элемента. Рекурсия не применяется к массиву, в котором только один элемент или отсутствуют элементы.

**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 quick\_sort(dequeue: Queue[VT]) -> Queue[VT]:  # 168n^2 + 2n \* log(n) + 64n + 4

    def \_quick\_sort(left: int, right: int):  # 168n^2 + 2n \* log(n) + 64n + 4

        i = left  # 1

        j = right  # 1

        pivot = seek(dequeue, (i + j) // 2)  # 28n + 2

        while i < j:  # n \* (

            while seek(dequeue, i) < pivot:  # 28n + 2

                i += 1

            while seek(dequeue, j) > pivot:  # 28n + 2

                j -= 1

            if i <= j:  # 1

                if i < j:  # 1

                    swap(dequeue, i, j)  # 112n + 28

                i += 1  # 1

                j -= 1  # 1

        # ) = 168n^2 + 36n

        if left < j:  # 1

            \_quick\_sort(left, j)  # ~= nlog(n)

        if i < right:  # 1

            \_quick\_sort(i, right)  # ~= nlog(n)

    \_quick\_sort(0, dequeue.size - 1)

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))

        quick\_sort(struct)

        print\_queue(struct)

    elif sys.argv[1] == "tests":

        tests = 10

        step = 100

        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()

            quick\_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}".replace(".", ","))

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

            print("-------------")

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во элементов | F(n) | O(F(n)) | Т(n) (сек) | N\_op |
| 100 | 1687733 | 10000 | 0,081605196 | 2153676 |
| 200 | 6735862 | 40000 | 0,374687195 | 10018946 |
| 300 | 15144141 | 90000 | 1,006764412 | 25762282 |
| 400 | 26912519 | 160000 | 1,789447308 | 45508596 |
| 500 | 42040970 | 250000 | 2,919700384 | 73940804 |
| 600 | 60529479 | 360000 | 4,501622915 | 111733636 |
| 700 | 82378036 | 490000 | 6,451433897 | 156994208 |
| 800 | 107586634 | 640000 | 8,510436535 | 210207732 |
| 900 | 136155269 | 810000 | 11,077636 | 274051442 |
| 1000 | 168083936 | 1000000 | 13,47227955 | 329575960 |

|  |  |  |  |
| --- | --- | --- | --- |
| С1=F(n)/T(n) | С2=O(F(n))/T(n) | С3=F(n)/N\_op | С4=O(F(n))/N\_op |
| 20681682,71 | 122541,2166 | 0,783652124 | 0,004643224 |
| 17977293,15 | 106755,7166 | 0,672312391 | 0,003992436 |
| 15042388,38 | 89395,29341 | 0,587841609 | 0,003493479 |
| 15039570,58 | 89413,08264 | 0,591372212 | 0,003515819 |
| 14399069,85 | 85625,2242 | 0,568576043 | 0,003381083 |
| 13446145,92 | 79971,15857 | 0,541730143 | 0,003221948 |
| 12768949,82 | 75952,10736 | 0,524720222 | 0,003121134 |
| 12641729,21 | 75201,78282 | 0,511811022 | 0,003044607 |
| 12291004,03 | 73120,29387 | 0,496823763 | 0,002955649 |
| 12476280,27 | 74226,48828 | 0,510000595 | 0,003034202 |

**Таблица результата экспериментов и графики зависимостей**

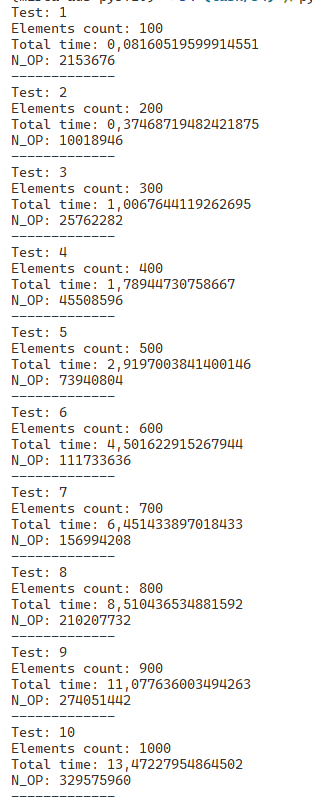
**Скриншоты работы программы:**

Рисунок 3. Результат работы программы в режиме example

Рисунок 2. Результат работы программы в режиме tests

**Выводы.**

В данной работе был реализован класс Queue предоставляющий удобный интерфейс для работы с очередью с головой и хвостом с типом связи на указателях. Так же были реализованы вспомогательные функции для удобного описания алгоритма быстрой сортировки для очереди.

По результатам экспериментов было установлено, что алгоритм сортировки Хоара для очереди с головой и хвостом на указателях имеет зависимость от числа элементов

**Литература:**

1. Левитин А. В. Глава 4. Метод декомпозиции: Быстрая сортировка // Алгоритмы. Введение в разработку и анализ — М.: Вильямс, 2006.