|  |
| --- |
| 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 № 33**

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 № 33.**

**Реализация связи элементов линейного списка: Указатели**

**Способ организации линейного связанный список: Дек**

**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, TypeVar

VT = TypeVar("VT")

class Node(Generic[VT]):

    value: VT

    next: 'Node' = None

    prev: 'Node' = None

    def \_\_init\_\_(self, value: VT):

        self.value = value

    def \_\_repr\_\_(self) -> str:

        return str(self.value)

class Dequeue(Generic[VT]):

    \_head: Node[VT] = None

    \_tail: Node[VT] = None

    \_size: int = 0

    \_n\_op: int = 0

    def is\_empty(self) -> bool:

        return self.\_size == 0

    def push\_back(self, value: VT):

        node = Node[VT](value)

        if self.is\_empty():

            self.\_head = node # 1

            self.\_tail = node # 1

            self.\_n\_op += 3

        else:

            self.\_tail.next = node # 2

            node.prev = self.\_tail # 2

            self.\_tail = node # 1

            self.\_n\_op += 6

        self.\_size += 1 # 1

    def push\_front(self, value: VT):

        node = Node[VT](value)

        if self.is\_empty():

            self.\_head = node # 1

            self.\_tail = node # 1

            self.\_n\_op += 3

        else:

            self.\_head.prev = node # 2

            node.next = self.\_head # 2

            self.\_head = node # 1

            self.\_n\_op += 6

        self.\_size += 1 # 1

    def pop\_back(self) -> VT:

        if self.is\_empty():

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

        node = self.\_tail # 1

        self.\_tail = node.prev # 2

        self.\_tail.next = None # 2

        node.prev = None # 2

        self.\_size -= 1 # 1

        self.\_n\_op += 8

        return node.value

    def pop\_front(self) -> VT:

        if self.is\_empty():

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

        node = self.\_head # 1

        self.\_head = node.next # 2

        self.\_head.prev = None # 2

        node.next = None # 2

        self.\_size -= 1 # 1

        self.\_n\_op += 8

        return node.value

    def \_\_repr\_\_(self) -> str:

        repr\_str = ""

        node = self.\_head

        while node is not None:

            repr\_str += str(node)

            if node.next is not None:

                repr\_str += ", "

            node = node.next

        return "dequeue<[" + repr\_str + "]>"

    @property

    def size(self) -> int:

        return self.\_size

    @property

    def head(self) -> VT:

        return self.\_head.value

    @property

    def tail(self) -> VT:

        return self.\_tail.value

    @property

    def n\_op(self) -> int:

        return self.\_n\_op

def print\_dequeue(dequeue: Dequeue):

    elems = []

    for \_ in range(dequeue.size):

        elems.append(dequeue.head)

        rotate\_left(dequeue)

    print("Dequeue[" + ", ".join(map(str, elems)) + "]")

def rotate\_left(dequeue: Dequeue): # 8 + 6 = 14

    dequeue.push\_back(dequeue.pop\_front())

def rotate\_right(dequeue: Dequeue): # 8 + 6 = 14

    dequeue.push\_front(dequeue.pop\_back())

def seek(dequeue: Dequeue[VT], index: int) -> VT: # 14n + 8

    if dequeue.is\_empty(): # 1

        raise Exception("Can't seek empty dequeue!")

    if index >= dequeue.size: # 2

        raise Exception("Index out of range!")

    if index >= dequeue.size // 2: # 3

        for \_ in range(dequeue.size - index - 1): # 14 \* (n / 2) = 7n

            rotate\_right(dequeue)

        node = dequeue.tail # 2

        for \_ in range(dequeue.size - index - 1): # 7n

            rotate\_left(dequeue)

        return node

    else:

        for \_ in range(index): # 14 \* (n / 2) = 7n

            rotate\_left(dequeue)

        node = dequeue.head # 2

        for \_ in range(index): # 14 \* (n / 2) = 7n

            rotate\_right(dequeue)

        return node

def swap(dequeue: Dequeue, pos1: int, pos2: int):

    # # Σ(0, left\_el\_pos)(14) + Σ(0, left\_el\_pos)(14) + Σ(0, right\_el\_pos - left\_el\_pos - 1)(14) + 4 + 3 + 7 + 6 + 6 =

    # = 28n + 26

    left\_el\_pos, right\_el\_pos = min(pos1, pos2), max(pos1, pos2) # 4

    if left\_el\_pos < 0 or right\_el\_pos >= dequeue.size: # 3

        raise Exception("Invalid position argument!")

    if left\_el\_pos < dequeue.size // 2: # 3

        for \_ in range(left\_el\_pos): # Σ(0, left\_el\_pos)(14)

            rotate\_left(dequeue) # 14

        left\_el = dequeue.pop\_front() # 7

        for \_ in range(right\_el\_pos - left\_el\_pos - 1): # Σ(0, right\_el\_pos - left\_el\_pos - 1)(14)

            rotate\_left(dequeue) # 14

        right\_el = dequeue.pop\_front() # 7

        dequeue.push\_front(left\_el) # 6

        for \_ in range(right\_el\_pos - left\_el\_pos - 1): # Σ(0, right\_el\_pos - left\_el\_pos - 1)(14)

            rotate\_right(dequeue)

        dequeue.push\_front(right\_el) # 6

        for \_ in range(left\_el\_pos): # Σ(0, left\_el\_pos)(14)

            rotate\_right(dequeue)

    else:

        for \_ in range(dequeue.size - right\_el\_pos - 1):

            rotate\_right(dequeue)

        left\_el = dequeue.pop\_back()

        for \_ in range(right\_el\_pos - left\_el\_pos - 1):

            rotate\_right(dequeue)

        right\_el = dequeue.pop\_back()

        dequeue.push\_back(left\_el)

        for \_ in range(right\_el\_pos - left\_el\_pos - 1):

            rotate\_left(dequeue)

        dequeue.push\_back(right\_el)

        for \_ in range(dequeue.size - right\_el\_pos - 1):

            rotate\_left(dequeue)

def quick\_sort(dequeue: Dequeue):

    def \_quick\_sort(left: int, right: int):

        # 14n + 8 + 28n^2 \* log(n) + 32n \* log(n) + 2 + 4log(nlog(n)) =

        # 28n^2 \* log(n) + 46n \* log(n) + 4log(nlog(n)) + 10

        i = left # 1

        j = right # 1

        pivot = seek(dequeue, (i + j) // 2) # 14n + 8

        while i < j: # ((i + j) / 2) \* (

            while seek(dequeue, i) < pivot: # n \* (i + j + 1)

                i += 1 # 1

            while seek(dequeue, j) > pivot: # n \* (i + j + 1)

                j -= 1 # 1

            if i <= j: # 1

                if i < j: # 1

                    swap(dequeue, i, j) # 28n + 26

                i += 1 # 1

                j -= 1 # 1

        # ) = ((i + j) / 2) \* (2n \* (i + j + 1) + 6 + 28n + 26) ~= 28n^2 \* log(n) + 32n \* log(n)

        if left < j: # 1

            \_quick\_sort(left, j) # ~= 2log(nlog(n))

        if i < right: # 1

            \_quick\_sort(i, right) # ~= 2log(nlog(n))

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

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

    import time

    from random import randint

    tests = 10

    step = 200

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

        dequeue = Dequeue[int]()

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

            dequeue.push\_back(randint(-10000, 10000))

        start\_time = time.time()

        quick\_sort(dequeue)

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

        print(f"Test: {test\_num}")

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

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

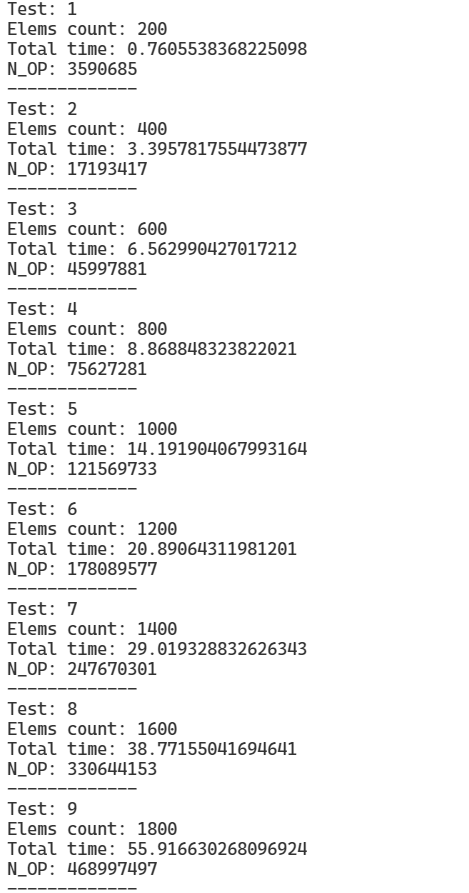
        print(f"N\_OP: {dequeue.n\_op}")

        print("-------------")

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во элементов | F(n) | O(F(n)) | Т(n) (сек) | N\_op |
| 200 | 8631484,722 | 305754,2476 | 0,8063 | 4031097 |
| 400 | 38883569,71 | 1383016,99 | 3,6824 | 17991837 |
| 600 | 93281257,54 | 3322374,729 | 6,6299 | 40709885 |
| 800 | 173172848,5 | 6172067,961 | 9,807 | 79832685 |
| 1000 | 279500439,2 | 9965784,285 | 15,157 | 123350169 |
| 1200 | 412990654,7 | 14729498,91 | 22,4098 | 184485337 |
| 1400 | 574235579,2 | 20484373,78 | 30,9869 | 259720325 |
| 1600 | 763735055,7 | 27248271,85 | 44,3141 | 372268253 |
| 1800 | 981921667,7 | 35036651,06 | 52,3007 | 433081925 |
| 2000 | 1229176750 | 43863137,14 | 62,8536 | 530641061 |

|  |  |  |  |
| --- | --- | --- | --- |
| С1=F(n)/T(n) | С2=O(F(n))/T(n) | С3=F(n)/N\_op | С4=O(F(n))/N\_op |
| 10705053,61 | 379206,5578 | 2,141224764 | 0,075848894 |
| 10559300,92 | 375574,8942 | 2,161178411 | 0,076869137 |
| 14069783,49 | 501119,8855 | 2,291366275 | 0,081611007 |
| 17658085,91 | 629353,3151 | 2,169197347 | 0,077312544 |
| 18440353,58 | 657503,7464 | 2,265910468 | 0,080792628 |
| 18429020,1 | 657279,3561 | 2,238609645 | 0,079841028 |
| 18531559,44 | 661065,6045 | 2,210976669 | 0,078870892 |
| 17234583,48 | 614889,4335 | 2,051571816 | 0,073195261 |
| 18774541,6 | 669907,88 | 2,267288499 | 0,080900747 |
| 19556186,91 | 697861,9703 | 2,316399616 | 0,082660654 |

**Таблица результата экспериментов и графики зависимостей**

**Скриншот работы программы:**

**Выводы.**

По результатам экспериментов было установлено, что алгоритм сортировки Хоара для дека 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.