|  |
| --- |
| 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 № 97**

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 №97.**

**Реализация связи элементов линейного списка: Указатели**

**Способ организации линейного связанный список: Дек**

**Алгоритм сортировки: Пирамидальная сортировка**

**Теория о сортировках.**

**Пирамидальная сортировка** (англ. *Heapsort*, «Сортировка кучей») — алгоритм сортировки, работающий в худшем, в среднем и в лучшем случае (то есть гарантированно) за O ( n log ⁡ n ) {\displaystyle O(n\log n)} операций при сортировке n {\displaystyle n} элементов. Количество применяемой служебной памяти не зависит от размера массива (то есть O ( 1 ) {\displaystyle O(1)} ). Может рассматриваться как усовершенствованная сортировка пузырьком, в которой элемент всплывает (min-heap) / тонет (max-heap) по многим путям.

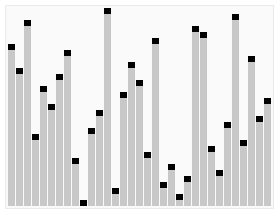


Рисунок 1. Визуализация работы алгоритма

**Алгоритм**

Сортировка пирамидой использует бинарное сортирующее дерево. Сортирующее дерево — это такое дерево, у которого выполнены условия:

1. Каждый лист имеет глубину либо d {\displaystyle d} d, либо d − 1 {\displaystyle d-1} d - 1, d {\displaystyle d} d — максимальная глубина дерева.
2. Значение в любой вершине не меньше (другой вариант — не больше) значения её потомков

Удобная структура данных для сортирующего дерева — такой массив Array, что Array[0] — элемент в корне, а потомки элемента Array[i] являются Array[2i+1] и Array[2i+2].

Алгоритм сортировки будет состоять из двух основных шагов:

1. Выстраиваем элементы массива в виде сортирующего дерева.

Этот шаг требует операций

2. Будем удалять элементы из корня по одному за раз и перестраивать дерево. То есть на первом шаге обмениваем Array[0] и Array[n-1], преобразовываем Array[0], Array[1], … , Array[n-2] в сортирующее дерево. Затем переставляем Array[0] и Array[n-2], преобразовываем Array[0], Array[1], … , Array[n-3] в сортирующее дерево. Процесс продолжается до тех пор, пока в сортирующем дереве не останется один элемент. Тогда Array[0], Array[1], … , Array[n-1] — упорядоченная последовательность.

Этот шаг требует O ( n log ⁡ n ) {\displaystyle O(n\log n)} операций.

**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 heapify(dequeue: Dequeue, n: int, i: int):  #  ~= (84n + 62) \* log(n \* log(n))

    largest = i  # 1

    l = 2 \* i + 1  # 3

    r = 2 \* i + 2  # 3

    if l < n and seek(dequeue, i) < seek(dequeue, l):  # 28n + 18

        largest = l  # 1

    if r < n and seek(dequeue, largest) < seek(dequeue, r):  # 28n + 18

        largest = r  # 1

    if largest != i:  # 1

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

        heapify(dequeue, n, largest)  # ~log(n \* log(n))

def heap\_sort(

    dequeue: Dequeue,

):  #  ~= 126n^2 \* log(n \* log(n)) + 28n^2 + 93n \* log(n \* log(n)) + 26n

    n = dequeue.size  # 1

    for i in range(

        n // 2 - 1, -1, -1

    ):  # (n / 2) \* (84n + 62) \* log(n \* log(n)) = 42n^2 \* log(n \* log(n)) + 31n \* log(n \* log(n))

        heapify(dequeue, n, i)

    for i in range(n - 1, 0, -1):  # n \* (

        swap(dequeue, 0, i)  # 28n + 26

        heapify(dequeue, i, 0)  # (84n + 62) \* log(n \* log(n))

    # ) ~= 28n^2 + 26n + 84n^2 \* log(n\*log(n)) + 62n \* log(n\*log(n))

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

    import time

    from random import randint

    tests = 10

    step = 100

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

        queue = Dequeue[int]()

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

            queue.push\_front(randint(-10000, 10000))

        start\_time = time.time()

        heap\_sort(queue)

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

        print(f"Test: {test\_num}")

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

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

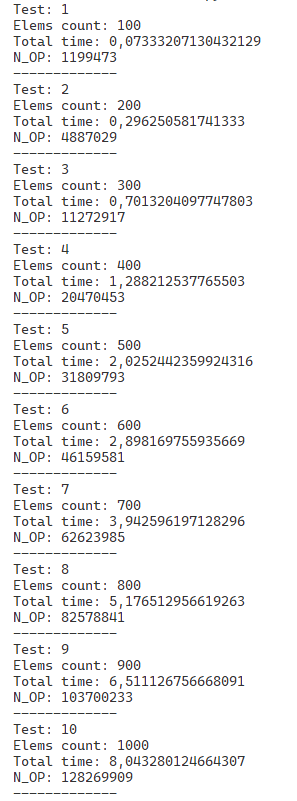
        print(f"N\_OP: {queue.n\_op}")

        print("-------------")

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во элементов | F(n) | O(F(n)) | Т(n) (сек) | N\_op |
| 100 | 12183401 | 93759 | 0,073332071 | 1199473 |
| 200 | 698898 | 423126 | 0,296250582 | 4887029 |
| 300 | 1673172 | 1014255 | 0,70132041 | 11272917 |
| 400 | 3102860 | 1880885 | 1,288212538 | 20470453 |
| 500 | 5004743 | 3032554 | 2,025244236 | 31809793 |
| 600 | 7391770 | 4476587 | 2,898169756 | 46159581 |
| 700 | 10274499 | 6218938 | 3,942596197 | 62623985 |
| 800 | 13661843 | 8264618 | 5,176512957 | 82578841 |
| 900 | 17561518 | 10617958 | 6,511126757 | 103700233 |
| 1000 | 21980327 | 13282768 | 8,043280125 | 128269909 |

|  |  |  |  |
| --- | --- | --- | --- |
| С1=F(n)/T(n) | С2=O(F(n))/T(n) | С3=F(n)/N\_op | С4=O(F(n))/N\_op |
| 166140141,7 | 1278550,69 | 10,15729468 | 0,078166637 |
| 184424495,6 | 1428271,535 | 11,1797708 | 0,086581494 |
| 186274908 | 1446208,25 | 11,58869482 | 0,089972752 |
| 187794489,5 | 1460073,51 | 11,81797081 | 0,09188292 |
| 192410767,7 | 1497376,694 | 12,25027771 | 0,09533396 |
| 198345686,2 | 1544625,63 | 12,45330778 | 0,096980674 |
| 202442904,6 | 1577371,293 | 12,74512674 | 0,09930601 |
| 204818088 | 1596560,951 | 12,83916647 | 0,100081551 |
| 209128675,5 | 1630740,506 | 13,13076427 | 0,10239088 |
| 211715866,2 | 1651411,79 | 13,275834 | 0,103553263 |

**Таблица результата экспериментов и графики зависимостей**

**Скриншот работы программы:**

**Выводы.**

По результатам экспериментов было установлено, что алгоритм сортировки подсчетом для дека на массиве имеет зависимость от числа элементов. Чем больше количество чисел и меньше диапазон, тем ближе сложность будет приближаться к асимптотической оценке.

**Литература:**

1. Левитин А. В. Глава 7. Пространственно-временной компромисс: Пирамидальная сортировка // Алгоритмы. Введение в разработку и анализ — М.: Вильямс, 2006.