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

Институт комплексной безопасности и специального приборостроения

Кафедра КБ-14 «Цифровые технологии обработки данных»

**ОТЧЕТ   
о выполнении лабораторной работы №1**

**«Реализация сортировки линейных структур данных»**

**по дисциплине   
«Алгоритмы и структуры данных»**

**Вариант № 3**

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 № 3.**

**Реализация связи элементов линейного списка:** **Указатели**

**Способ организации линейного связанный список: Очередь с одной головой**

**Алгоритм сортировки**: **Сравнение и подсчет**

**Теория о сортировках.**

**Сортировка сравнением и подсчетом.**

Сортировка сравнением и подсчетом – алгоритм сортировки целых неотрицательных чисел. Данный алгоритм использует массив - счетчик длиной, равной диапазону среди различных неповторяющихся сортируемых объектов. Сортировка происходит в соответствии с индексами буфера, где индексом является сам элемент, а значением – количество этих повторяющихся элементов. Алгоритм работает за линейное время O(n+k), где n – длина исходного массива, k – массива - счетчика. Алгоритм особенно эффективен для чисел небольшого диапазона. Например, если требуется отсортировать большое количество чисел с небольшим разбросом среди сортируемых значений, поэтому для эффективной работы k не должно сильно быть больше, чем n.

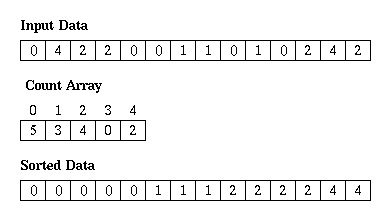


Рисунок 1. – Пример отсортированных данных.

Рисунок 1 иллюстрирует работу данного алгоритма. Первостепенно идет обход по исходному массиву в поисках максимального значения. После чего инициализируется счетчик – массив для подсчета количества каждого элемента. Длина счетчика равна единице, прибавленной к максимальному элементу. Так, в примере на рисунке максимальным элементом является четыре. Следовательно, длина счетчика равна пяти. Далее идет подсчет количество повторений каждого элемента в исходном массиве и присвоение его счетчику, где индекс – элемент, повторения которого считали, а значение – количество повторений. Например, в примере на рисунке 1 значение ноль повторяется пять раз, следовательно, нулевой элемент счетчика равен пяти, один повторяется три раза, значит, первый элемент равен трем и т.д. После завершения подсчета всех значений в исходный массив поочередно записываются индексы счетчика столько раз, сколько записаны повторения в значениях этих индексов. Так, если нулевой элемент равен пяти, то в массив запишется ноль пять раз, а если первый элемент равен 3, то один запишется 3 раза и т.д. На этом сортировка завершена

**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]]

    \_size: int

    \_n\_op: int

    def \_\_init\_\_(self) -> None:

        self.\_head = None

        self.\_size = 0

        self.\_n\_op = 0

    def push(self, value: VT) -> None:  # 2n + 9

        node = Node(value)  # 2

        if self.is\_empty():  # 1

            self.\_head = node  # 1

        else:

            if self.\_head.next != None:

                tail = self.\_head.next  # 3

                while tail.next is not None:  # n \* (

                    tail = tail.next  # 2

                # ) = 2n

            else:

                tail = self.\_head  # 2

            tail.next = node  # 2

        self.\_size += 1  # 1

        self.\_n\_op += self.\_size \* 2 + 8

    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

        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 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:  # 2n + 16

    queue.push(queue.pop())  # 2n + 16

def seek(queue: Queue[VT], pos: int) -> VT:  # 4n^2 + 32n + 2

    for \_ in range(pos):  # n \* (

        rotate(queue)  # 2n + 16

    # ) = 2n^2 + 16n

    el = queue.head  # 2

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

        rotate(queue)  # 2n + 16

    # ) = 2n^2 + 16n

    return el

def pop\_by\_pos(queue: Queue[VT], pos: int) -> VT:  # 4n^2 + 32n + 7

    for \_ in range(pos):  # n \* (

        rotate(queue)  # 2n + 16

    # ) = 2n^2 + 16n

    el = queue.pop()  # 7

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

        rotate(queue)  # 2n + 16

    # ) = 2n^2 + 16n

    return el

def push\_by\_pos(queue: Queue[VT], el: VT, pos: int) -> None:  # 4n^2 + 34n + 11

    if pos >= queue.size:  # 2

        queue.push(el)  # 2n + 9

        return

    for \_ in range(pos):  # n \* (

        rotate(queue)  # 2n + 16

    # ) = 2n^2 + 16n

    queue.push(el)  # 2n + 9

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

        rotate(queue)  # 2n + 16

    # ) = 2n^2 + 16n

def swap(queue: Queue, pos1: int, pos2: int):  # 16n^2 + 132n + 36

    temp = pop\_by\_pos(queue, pos1)  # 4n^2 + 32n + 7

    push\_by\_pos(queue, pop\_by\_pos(queue, pos2 - 1), pos1)  # 8n^2 + 66n + 18

    push\_by\_pos(queue, temp, pos2)  # 4n^2 + 34n + 11

def slice\_(queue: Queue[VT], l: int = 0, r: int = -1) -> Queue[VT]:  # 8n^2 + 57n + 6

    buffer = Queue[VT]()  # 2

    if r == -1:  # 1

        r = queue.size - 1  # 3

    for \_ in range(l):  # n \* (

        rotate(queue)  # 2n + 16

    # ) = 2n^2 + 16n

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

        el = queue.pop()  #  7

        buffer.push(el)  # 2n + 9

        queue.push(el)  # 2n + 9

    # ) = 4n^2 + 25n

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

        rotate(queue)  # 2n + 16

    # ) = 2n^2 + 16n

    return buffer

def counting\_sort(queue: Queue[VT]):  # 4n^2 + 38n + 4

    max\_ = queue.head  # 2

    min\_ = queue.head  # 2

    for \_ in range(queue.size):  # n

        el = queue.pop()  # 7

        if el > max\_:  # 1

            max\_ = el  # 1

        if el < min\_:  # 1

            min\_ = el  # 1

        queue.push(el)  # 2n + 9

    # ) = 2n^2 + 20n

    buffer = [0] \* (max\_ - min\_ + 1)  # 1

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

        el = queue.pop()  # 7

        buffer[el - min\_] += 1  # 2

    # ) = 9n

    for i in range(len(buffer)):  # n \* (

        for \_ in range(buffer[i]):  # k \* (

            queue.push(i + min\_)  # 2n + 9

        # ) = 2n +9

    # ) = 2n^2 + 9n

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))

        counting\_sort(struct)

        print\_queue(struct)

    elif sys.argv[1] == "tests":

        tests = 10

        step = 1000

        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()

            counting\_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}")

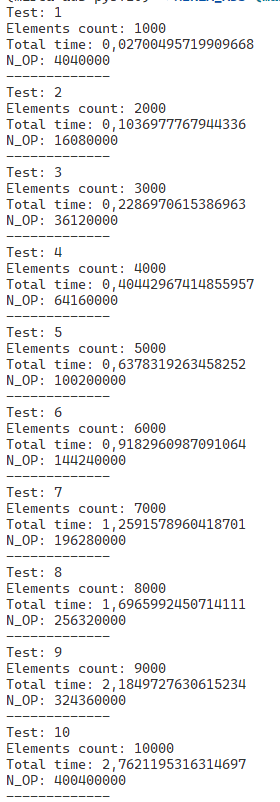
            print("-------------")

**Таблица результата экспериментов и графики зависимостей**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество элементов | F(N) | O(F(n)) | T(n) (сек) | N\_op |
| 1000 | 4038004 | 1000000 | 0,027004957 | 4040000 |
| 2000 | 16076004 | 4000000 | 0,103697777 | 16080000 |
| 3000 | 36114004 | 9000000 | 0,228697062 | 36120000 |
| 4000 | 64152004 | 16000000 | 0,404429674 | 64160000 |
| 5000 | 100190004 | 25000000 | 0,637831926 | 100200000 |
| 6000 | 144228004 | 36000000 | 0,918296099 | 144240000 |
| 7000 | 196266004 | 49000000 | 1,259157896 | 196280000 |
| 8000 | 256304004 | 64000000 | 1,696599245 | 256320000 |
| 9000 | 324342004 | 81000000 | 2,184972763 | 324360000 |
| 10000 | 400380004 | 100000000 | 2,762119532 | 400400000 |

|  |  |  |  |
| --- | --- | --- | --- |
| С1=F(n)/T(n) | С2=O(F(n))/T(n) | С3=F(n)/N\_op | С4=O(F(n))/N\_op |
| 149528250,3 | 37030238,29 | 0,999505941 | 0,247524752 |
| 155027470,2 | 38573633,14 | 0,999751493 | 0,248756219 |
| 157911972,1 | 39353369,65 | 0,999833998 | 0,249169435 |
| 158623385,2 | 39561884,36 | 0,999875374 | 0,249376559 |
| 157079004,5 | 39195278,52 | 0,99990024 | 0,249500998 |
| 157060456 | 39203041,43 | 0,999916833 | 0,249584027 |
| 155870844 | 38914897,13 | 0,999928694 | 0,249643367 |
| 151069266,8 | 37722520,62 | 0,999937594 | 0,24968789 |
| 148442126,8 | 37071400,33 | 0,999944518 | 0,249722531 |
| 144953902 | 36204081,27 | 0,99995006 | 0,24975025 |

**Скриншот работы программы:**

****

**Выводы.**

В результате данной работы был реализован класс Queue для работы с очередью с одной головой на указателях. Также были реализованы базовые операции над ней и рассчитана их алгоритмическая сложность.

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

**Литература:**

1. Алгоритмы: построение и анализ. Кормен Т.Х., Лейзерсон Ч.И.

2. https://en.wikipedia.org/wiki/Counting\_sort