|  |
| --- |
| 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 № 85**

Выполнил: студент 2 курса

Группы **\_\_\_\_\_\_\_\_\_\_\_**

Шифр \_\_\_\_\_\_\_\_

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

Проверил:

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

Москва 2020 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 №85.**

**Реализация связи элементов линейного списка: Массив**

**Способ организации линейного связанный список: Очередь**

**Алгоритм сортировки: Распределяющий подсчет**

**Теория о сортировках.**

**Сортировка подсчётом** - алгоритм сортировки, в котором используется диапазон чисел сортируемого массива (списка) для подсчёта совпадающих элементов. Применение сортировки подсчётом целесообразно лишь тогда, когда сортируемые числа имеют (или их можно отобразить в) диапазон возможных значений, который достаточно мал по сравнению с сортируемым множеством, например, миллион натуральных чисел меньших 1000.

Предположим, что входной массив состоит из n n {\displaystyle n} целых чисел в диапазоне от 0 {\displaystyle 0} 0 до k - 1 k − 1 {\displaystyle k-1} , где k ∈ N {\displaystyle k\in \mathbb {N} } . Далее алгоритм будет обобщён для произвольного целочисленного диапазона. Существует несколько модификаций сортировки подсчётом, ниже рассмотрены три линейных и одна квадратичная, которая использует другой подход, но имеет то же название.

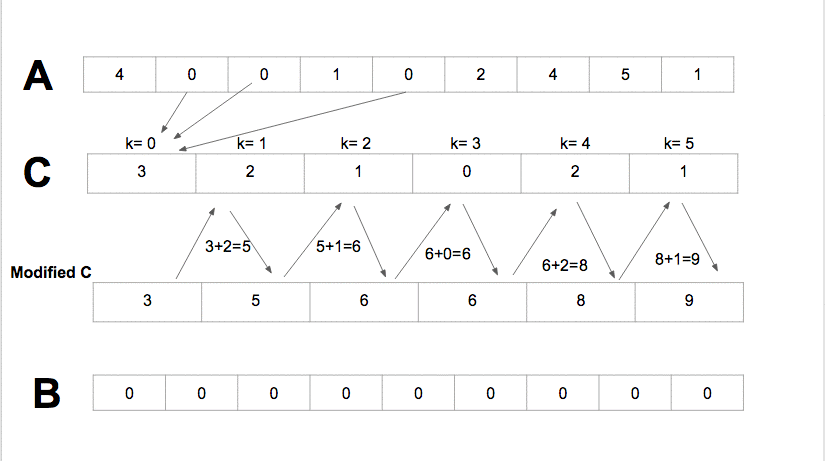


Рисунок 1. Визуализация работы алгоритма

Шаги алгоритма:

1. Создать вспомогательный массив C [0..k - 1], состоящий из нулей
2. Последовательно прочитать элементы входного массива A, для каждого A[i] увеличить C[A[i]] на единицу.
3. Пройти по массиву C, для каждого j ∈ { 0 , . . . , k − 1 } {\displaystyle j\in \{0,...,k-1\}} j в массив A последовательно записать число j C[j] раз.

**Листинг программы с расчетами.**

from typing import Generic, TypeVar, List

VT = TypeVar("VT")

class Queue(Generic[VT]):

    \_queue: List[VT]

    \_nop: int

    def \_\_init\_\_(self):

        self.\_queue = []

        self.\_nop = 0

    def push(self, value: VT): # n + 2

        self.\_nop += self.count + 2

        self.\_queue = self.\_queue.copy()

        self.\_queue.append(0)

        # Имитация работы со списком как с массивом.

        # Создание массива на элемент больше и копирование элементов

        self.\_queue[-1] = value

    def pop(self) -> VT: # n + 2

        self.\_nop += self.count + 2

        if self.is\_empty():

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

        el = self.\_queue[0]

        self.\_queue = self.\_queue[1:] # создание нового списка на элемент меньше

        return el

    def is\_empty(self) -> bool: # 1

        return len(self.\_queue) == 0

    @property

    def count(self) -> int: # 1

        return len(self.\_queue)

    @property

    def tail(self) -> VT: # 1

        return self.\_queue[-1]

    @property

    def head(self) -> VT: # 1

        return self.\_queue[0]

    @property

    def n\_op(self) -> int:

        return self.\_nop

def print\_queue(queue: Queue):

    elems = []

    for \_ in range(queue.count):

        el = queue.pop()

        elems.append(el)

        queue.push(el)

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

def rotate(queue: Queue): # 2n + 4

    queue.push(queue.pop())

def seek(queue: Queue, pos: int): # 2n \* pos + 4pos + 4n + 8 + 2n^2 + 8n - 2n \* pos - 4pos - 2n - 4 = 2n^2

    for \_ in range(pos): # pos \* (2n + 4) = 2n \* pos + 4pos

        rotate(queue)

    el = queue.pop() # n + 2

    queue.push(el) # n + 2

    for \_ in range(queue.count - pos - 1): # (n - pos - 1) \* (2n + 4) = 2n^2 + 8n - 2n \* pos - 4pos - 2n - 4

        rotate(queue)

    return el

def pop\_by\_pos(queue: Queue, pos: int): # 2n^2 + 3n - 2

    for \_ in range(pos): # 2n \* pos + 4pos

        rotate(queue)

    el = queue.pop() # n + 2

    for \_ in range(queue.count - pos): # (n - pos - 1) \* (2n + 4)

        rotate(queue)

    return el

def push\_by\_pos(queue: Queue, el, pos: int): # 2n^2 + 3n - 2

    if pos >= queue.count: # 1

        queue.push(el) # n + 2

        return

    for \_ in range(pos): # 2n \* pos + 4pos

        rotate(queue)

    queue.push(el) # n + 2

    for \_ in range(queue.count - pos - 1): # (n - pos - 1) \* (2n + 4)

        rotate(queue)

def swap(queue: Queue, pos1: int, pos2: int): # 6n^2 + 9n - 7

    temp = pop\_by\_pos(queue, pos1) # 2n^2 + 3n - 2

    push\_by\_pos(queue, pop\_by\_pos(queue, pos2 - 1), pos1) # 2n^2 + 3n - 3

    push\_by\_pos(queue, temp, pos2) # 2n^2 + 3n - 2

def count\_sort(queue: Queue[int]): # 1 + 1 + n^2 + 6n + 2 + 3 + n^2 + 7n + 2k + kln + 4kl = 2n^2 + n \* (15 + kl) + 4kl + 7

    max\_elem = None # 1

    min\_elem = 0 # 1

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

        el = queue.head # 1

        if max\_elem is None or el > max\_elem: # 3

            max\_elem = el # 1

        elif el < min\_elem: # 1

            min\_elem = el # 1

        rotate(queue) # n + 2

    # ) = n^2 + 6n

    bias = abs(min\_elem) + 1 # 2

    counter = [0] \* (max\_elem + bias) # 3

    while not queue.is\_empty(): # n \* (

        counter[bias + queue.head - 1] += 1 # 5

        queue.pop() # n + 2

    # ) = n^2 + 7n

    for i in range(len(counter)): # k \* (

        if counter[i] == 0: # 2

            continue

        for \_ in range(counter[i]): # l \* (

            queue.push(i - bias + 1) # n + 4

        # ) = ln + 4l

    # ) 2k + kln + 4kl

if \_\_name\_\_ == '\_\_main\_\_':

    import time

    from random import randint

    tests = 10

    step = 5000

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

        queue = Queue[int]()

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

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

        start\_time = time.time()

        count\_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}")

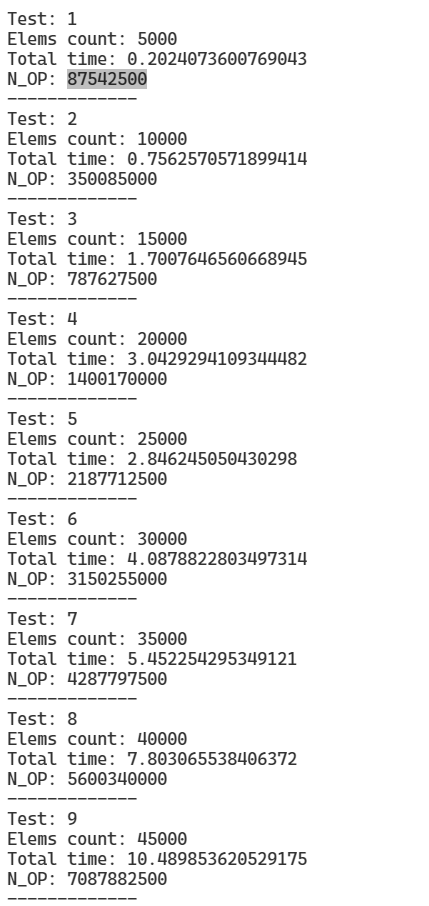
        print(f"N\_OP: {queue.n\_op}")

        print("-------------")

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во элементов | F(n) | O(F(n)) | Т(n) (сек) | N\_op |
| 5000 | 50075007 | 25000000 | 0,2024 | 87542500 |
| 10000 | 200150007 | 100000000 | 0,7562 | 350085000 |
| 15000 | 450225007 | 225000000 | 1,7007 | 787627500 |
| 20000 | 800300007 | 400000000 | 3,0429 | 1400170000 |
| 25000 | 1250375007 | 625000000 | 2,8462 | 2187712500 |
| 30000 | 1800450007 | 900000000 | 4,0878 | 3150255000 |
| 35000 | 2450525007 | 1225000000 | 5,4522 | 4287797500 |
| 40000 | 3200600007 | 1600000000 | 7,803 | 5600340000 |
| 45000 | 4050675007 | 2025000000 | 10,4898 | 7087882500 |
| 50000 | 5000750007 | 2500000000 | 13,5473 | 8750425000 |

|  |  |  |  |
| --- | --- | --- | --- |
| С1=F(n)/T(n) | С2=O(F(n))/T(n) | С3=F(n)/N\_op | С4=O(F(n))/N\_op |
| 247406161,1 | 123517786,6 | 0,572007962 | 0,285575578 |
| 264678665,7 | 132240148,1 | 0,571718317 | 0,285644915 |
| 264729233,3 | 132298465,3 | 0,571621746 | 0,285668035 |
| 263005687,7 | 131453547,6 | 0,571573457 | 0,285679596 |
| 439313824,4 | 219591033,7 | 0,571544482 | 0,285686533 |
| 440444739,7 | 220167327,2 | 0,571525164 | 0,285691158 |
| 449456184,1 | 224679945,7 | 0,571511366 | 0,285694462 |
| 410175574,4 | 205049340 | 0,571501017 | 0,28569694 |
| 386153692,8 | 193044672 | 0,571492968 | 0,285698867 |
| 369132595,2 | 184538616,6 | 0,571486529 | 0,285700409 |

**Таблица результата экспериментов и графики зависимостей**

**Скриншот работы программы:**

**Выводы.**

По результатам экспериментов было установлено, что алгоритм сортировки подсчетом для очереди на массиве имеет квадратичную 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: Сортировка подсчётом // Алгоритмы. Введение в разработку и анализ — М.: Вильямс, 2006.