|  |
| --- |
| 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 № 10**

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 № 10.**

**Реализация связи элементов линейного списка: Указатели**

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

**Алгоритм сортировки: Алгоритм Шелла**

**Теория о сортировках.**

**Сортировка Шелла** (англ. *Shell sort*) — алгоритм сортировки, являющийся усовершенствованным вариантом сортировки вставками. Идея метода Шелла состоит в сравнении элементов, стоящих не только рядом, но и на определённом расстоянии друг от друга. Иными словами — это сортировка вставками с предварительными «грубыми» проходами. Аналогичный метод усовершенствования пузырьковой сортировки называется сортировка расчёской.

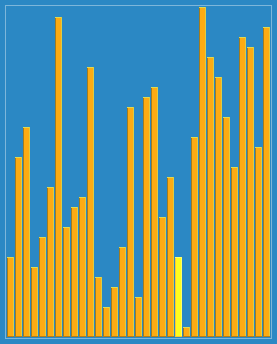


Рисунок 1. Иллюстрация сортировки алгоритмом Шелла

**Алгоритм**

При сортировке Шелла сначала сравниваются и сортируются между собой значения, стоящие один от другого на некотором расстоянии d {\displaystyle d} (о выборе значения d {\displaystyle d} см. ниже). После этого процедура повторяется для некоторых меньших значений d {\displaystyle d} , а завершается сортировка Шелла упорядочиванием элементов при d = 1 {\displaystyle d=1} (то есть обычной сортировкой вставками). Эффективность сортировки Шелла в определённых случаях обеспечивается тем, что элементы «быстрее» встают на свои места (в простых методах сортировки, например, пузырьковой, каждая перестановка двух элементов уменьшает количество инверсий в списке максимум на 1, а при сортировке Шелла это число может быть больше).

Невзирая на то, что сортировка Шелла во многих случаях медленнее, чем быстрая сортировка, она имеет ряд преимуществ:

* отсутствие потребности в памяти под стек;
* отсутствие деградации при неудачных наборах данных — быстрая сортировка легко деградирует до O(n²), что хуже, чем худшее гарантированное время для сортировки Шелла.

**Листинг программы с расчетами.**

from typing import Generic, TypeVar, Optional

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): # 5

        node = Node(value)

        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 += 4

    def pop(self) -> VT: # 6

        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 += 6

        return node.value

    def is\_empty(self) -> bool: # 1

        return self.\_size == 0

    @property

    def head(self) -> VT: # 1

        return self.\_head.value

    @property

    def tail(self) -> VT: # 1

        return self.\_tail.value

    @property

    def size(self) -> int: # 1

        return self.\_size

    @property

    def n\_op(self) -> int: # 1

        return self.\_n\_op

def print\_queue(queue: Queue):

    elems = []

    for \_ in range(queue.size):

        el = queue.pop()

        elems.append(el)

        queue.push(el)

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

def rotate(queue: Queue): # 11

    queue.push(queue.pop())

def seek(queue: Queue[VT], pos: int) -> VT: # 11n + 2

    for \_ in range(pos): # 11pos

        rotate(queue)

    el = queue.head # 2

    for \_ in range(queue.size - pos): # 11n - 11pos

        rotate(queue)

    return el

def pop\_by\_pos(queue: Queue[VT], pos: int) -> VT: # 11n + 6

    for \_ in range(pos): # 11pos

        rotate(queue)

    el = queue.pop() # 6

    for \_ in range(queue.size - pos): # 11n - 11pos

        rotate(queue)

    return el

def push\_by\_pos(queue: Queue[VT], el: VT, pos: int): # 11n - 3

    if pos >= queue.size: # 2

        queue.push(el) # 6

        return

    for \_ in range(pos): # 11pos

        rotate(queue)

    queue.push(el) # 6

    for \_ in range(queue.size - pos - 1): # (n - pos - 1) \* 11 = 11n - 11pos - 11

        rotate(queue)

def swap(queue: Queue, pos1: int, pos2: int): # 44n + 6

    temp = pop\_by\_pos(queue, pos1) # 11n + 6

    push\_by\_pos(queue, pop\_by\_pos(queue, pos2 - 1), pos1) # 22n + 3

    push\_by\_pos(queue, temp, pos2) # 11n - 3

def shell\_sort(queue: Queue): # 66n^2 \* log(n) + 12n \* log(n) + 6

    last\_index = queue.size - 1 # 3

    step = queue.size // 2 # 3

    while step > 0: # log(n) \* (

        for i in range(step, last\_index + 1, 1): # n \* (

            j = i # 1

            delta = j - step # 2

            while delta >= 0 and seek(queue, delta) > seek(queue, j): # k \* (22n + 3 +

                swap(queue, delta, j) # 44n + 6

                j = delta # 1

                delta = j - step # 2

            # ) = k \* (66n + 12) ~= 66n + 12

        step //= 2

     # ) = 66n^2 \* log(n) + 12n \* log(n)

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

    import time

    from random import randint

    tests = 10

    step = 100

    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()

        shell\_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}")

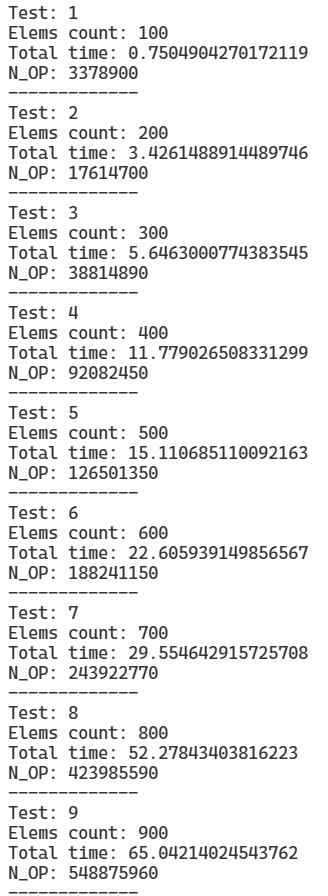
        print(f"N\_OP: {queue.n\_op}")

        print("-------------")

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во элементов | F(n) | O(F(n)) | Т(n) (сек) | N\_op |
| 100 | 4392923,713 | 66438,5619 | 0,7504 | 3378900 |
| 200 | 20198131,6 | 305754,2476 | 3,4261 | 17614700 |
| 300 | 48908812,77 | 740593,6821 | 5,6463 | 38814890 |
| 400 | 91320617,87 | 1383016,99 | 11,7790 | 92082450 |
| 500 | 147989241,4 | 2241446,071 | 15,1106 | 126501350 |
| 600 | 219343185,6 | 3322374,729 | 22,6059 | 188241150 |
| 700 | 305731563,5 | 4631093,445 | 29,5546 | 243922770 |
| 800 | 407449072,5 | 6172067,961 | 52,2784 | 423985590 |
| 900 | 524750737,3 | 7949162,765 | 65,0421 | 548875960 |
| 1000 | 657861358,2 | 9965784,285 | 69,2312 | 584283780 |

|  |  |  |  |
| --- | --- | --- | --- |
| С1=F(n)/T(n) | С2=O(F(n))/T(n) | С3=F(n)/N\_op | С4=O(F(n))/N\_op |
| 5854109,425 | 88537,52918 | 1,300104683 | 0,019662778 |
| 5895371,296 | 89242,65129 | 1,146663389 | 0,017357903 |
| 8662099,564 | 131164,4231 | 1,26005285 | 0,019080144 |
| 7752832,827 | 117413,7864 | 0,991726631 | 0,015019333 |
| 9793736,94 | 148336,0073 | 1,169862941 | 0,017718752 |
| 9702917,627 | 146969,3632 | 1,165224424 | 0,017649567 |
| 10344635,47 | 156696,1977 | 1,253394931 | 0,018985901 |
| 7793832,108 | 118061,5314 | 0,960997454 | 0,014557259 |
| 8067862,774 | 122215,6536 | 0,95604613 | 0,014482621 |
| 9502382,715 | 143949,3218 | 1,125927812 | 0,017056411 |

**Таблица результата экспериментов и графики зависимостей**

**Скриншот работы программы:**

**Выводы.**

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

**Литература:**

1. Д. Кнут. Искусство программирования. Том 3. Сортировка и поиск, 2-е изд.