|  |
| --- |
| 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 № 12**

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 № 12.**

**Реализация связи элементов линейного списка: Указатели**

**Способ организации линейного связанный список: Стек**

**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.

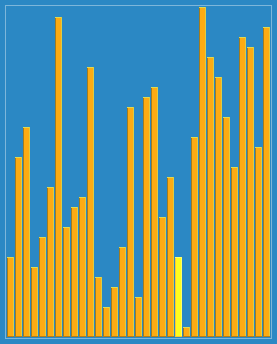


Рисунок 1. Иллюстрация сортировки алгоритмом Шелла

**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, 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 Stack(Generic[VT]):

    \_top: Optional[Node[VT]]

    \_size: int

    \_n\_op: int

    def \_\_init\_\_(self) -> None:

        self.\_top = None

        self.\_size = 0

        self.\_n\_op = 0

    def push(self, value: VT) -> None:  # 14

        node = Node(value)  # 2

        if self.is\_empty():  # 2

            self.\_top = node  # 3

        else:

            node.next = self.\_top  # 3

            self.\_top = node  # 2

        self.\_size += 1  # 2

        self.\_n\_op += 14

    def pop(self) -> VT:  # 9

        if self.is\_empty():  # 2

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

        node = self.\_top  # 2

        self.\_top = node.next  # 3

        self.\_size -= 1  # 2

        self.\_n\_op += 9

        return node.value

    def is\_empty(self) -> bool:  # 2

        return self.\_size == 0

    @property

    def top(self) -> VT:  # 2

        return self.\_top.value

    @property

    def size(self) -> int:  # 1

        return self.\_size

    @property

    def n\_op(self) -> int:

        return self.\_n\_op

def print\_stack(stack: Stack[VT]) -> None:

    buffer = Stack[VT]()

    elems = []

    for \_ in range(stack.size):

        el = stack.pop()

        elems.append(el)

        buffer.push(el)

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

    for \_ in range(buffer.size):

        el = buffer.pop()

        stack.push(el)

def seek(stack: Stack[VT], pos: int) -> VT:  # 46n - 40

    swap\_stack = Stack[VT]()  # 2

    stack\_size = stack.size  # 2

    for \_ in range(stack\_size - pos - 1):  # (n - 1) \* (

        swap\_stack.push(stack.pop())  # 23

    # ) = 23n - 23

    el = stack.top  # 2

    for \_ in range(stack\_size - pos - 1):  # (n - 1) \* (

        stack.push(swap\_stack.pop())  # 23

    # ) = 23n - 23

    return el

def push\_by\_pos(stack: Stack[VT], el: VT, pos: int) -> None:  # 46n - 28

    swap\_stack = Stack()  # 2

    stack\_size = stack.size  # 2

    for \_ in range(stack\_size - pos):  # n \* (

        swap\_stack.push(stack.pop())  # 23

    # ) = 23n - 23

    stack.push(el)  # 14

    for \_ in range(stack\_size - pos):  # n \* (

        stack.push(swap\_stack.pop())  # 23

    # ) = 23n - 23

def pop\_by\_pos(stack: Stack[VT], pos: int) -> VT:  # ~= 46n - 40

    swap\_stack = Stack()

    stack\_size = stack.size

    for \_ in range(stack\_size - pos - 1):

        swap\_stack.push(stack.pop())

    el = stack.pop()

    for \_ in range(stack\_size - pos - 1):

        stack.push(swap\_stack.pop())

    return el

def swap(stack: Stack[VT], pos1: int, pos2: int) -> None:  # 138n - 96

    temp = pop\_by\_pos(stack, pos1)  # 46n - 40

    push\_by\_pos(stack, pop\_by\_pos(stack, pos2 - 1), pos1)  # 46n - 28

    push\_by\_pos(stack, temp, pos2)  # 46n - 28

def reverse(stack: Stack[VT]) -> Stack[VT]:  # 23n

    out = Stack[VT]()  # 2

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

        out.push(stack.pop())  # 23

    # ) = 23n

    return out

def slice\_stack(stack: Stack[VT], l: int = 0, r: int = -1) -> Stack[VT]:  # 83n - 49

    stack\_size = stack.size  # 2

    slice\_stack = Stack[VT]()  # 2

    swap\_stack = Stack[VT]()  # 2

    if r == -1:  # 2

        r = stack.size - 1  # 3

    for \_ in range(stack\_size - r - 1):  # (n - 1) \* (

        swap\_stack.push(stack.pop())  # 23

    # ) = 23n - 23

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

        slice\_stack.push(stack.top)  # 14

        swap\_stack.push(stack.pop())  # 23

    # ) = 37n - 37

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

        stack.push(swap\_stack.pop())  # 23

    # ) = 23n

    return reverse(slice\_stack)  # 23n

def shell\_sort(stack: Stack[VT]) -> Stack[VT]:  # 138n^2 \* log(n) - 90n \* log(n) + 6

    last\_index = stack.size - 1  # 3

    step = stack.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(stack, delta) < seek(

                stack, j

            ):  # k (where k << n) \* (

                swap(stack, delta, j)  # 138n - 96

                j = delta  # 1

                delta = j - step  # 2

            # ) = 138n - 93

        # ) = 138n^2 - 90n

        step //= 2

    # ) = 138n^2 \* log(n) - 90n \* log(n) + 6

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 = Stack[int]()

        for \_ in range(20):

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

        shell\_sort(struct)

        print\_stack(struct)

    elif sys.argv[1] == "tests":

        tests = 10

        step = 100

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

            struct = Stack[int]()

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

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

            start\_time = time.time()

            shell\_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}")

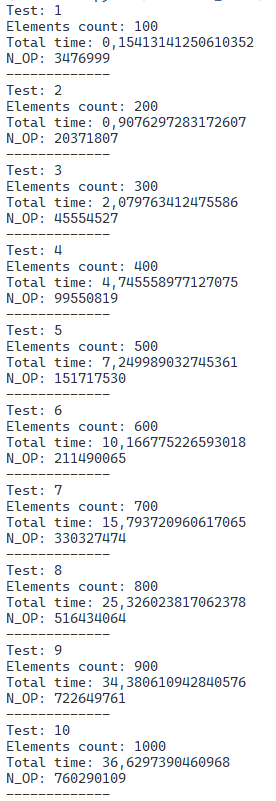
            print(f"N\_OP: {struct.n\_op}")

            print("-------------")

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во элементов | F(n) | O(F(n)) | Т(n) (сек) | N\_op |
| 100 | 9108733 | 66439 | 0,154131413 | 3476999 |
| 200 | 42056503 | 305754 | 0,907629728 | 20371807 |
| 300 | 101979756 | 740594 | 2,079763412 | 45554527 |
| 400 | 190545172 | 1383017 | 7,249989033 | 99550819 |
| 500 | 308916104 | 2241446 | 5,728699207 | 151717530 |
| 600 | 457989362 | 3322375 | 10,16677523 | 211490065 |
| 700 | 638495475 | 4631093 | 15,79372096 | 330327474 |
| 800 | 851051027 | 6172068 | 25,32602382 | 516434064 |
| 900 | 1096189551 | 7949163 | 34,38061094 | 722649761 |
| 1000 | 1374381317 | 9965784 | 48,64275503 | 760290109 |

|  |  |  |  |
| --- | --- | --- | --- |
| С1=F(n)/T(n) | С2=O(F(n))/T(n) | С3=F(n)/N\_op | С4=O(F(n))/N\_op |
| 59097186,54 | 431051,4049 | 2,619711089 | 0,019108019 |
| 46336629,84 | 336871,1249 | 2,064446357 | 0,015008695 |
| 49034306,22 | 356095,1586 | 2,238630554 | 0,016257302 |
| 26282132,43 | 190761,2528 | 1,914049264 | 0,013892573 |
| 53924301,55 | 391266,1479 | 2,036126633 | 0,014773811 |
| 45047652,98 | 326787,4675 | 2,165536061 | 0,015709365 |
| 40427172,08 | 293223,7094 | 1,932916652 | 0,014019704 |
| 33603815,32 | 243704,5786 | 1,647937436 | 0,011951318 |
| 31883946,25 | 231210,6314 | 1,516902946 | 0,011000021 |
| 28254594,46 | 204877,0527 | 1,807706427 | 0,01310787 |

**Таблица результата экспериментов и графики зависимостей**

**Скриншот работы программы:**

**Выводы.**

В данной работе был реализован класс Stack предоставляющий интерфейс для взаимодействия со стеком на указателях. Также были реализованные базовые операции над ним и вспомогательные операции для сортировки.

По результатам экспериментов было установлено, что алгоритм сортировки Шелла для стека на указателях имеет зависимость от числа элементов

**Литература:**

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