|  |
| --- |
| 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 № 68**

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 № 68.**

**Реализация связи элементов линейного списка: Указатели**

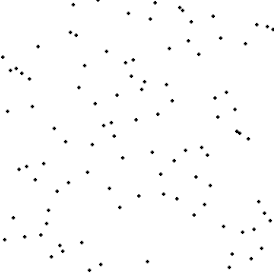
**Способ организации линейного связанный список: Стек**

**Алгоритм сортировки: Простой выбор**

**Теория о сортировках.**

**Сортировка Простым Выбором.**

Может быть как устойчивый, так и неустойчивый. На массиве из *n* элементов имеет время выполнения в худшем, среднем и лучшем случае Θ(*n*2), предполагая, что сравнения делаются за постоянное время.

Рисунок иллюстрирует сортировку простым выбором.

Шаги алгоритма:

1. Находим номер минимального значения в текущем списке
2. Производим обмен этого значения со значением первой неотсортированной позиции (обмен не нужен, если минимальный элемент уже находится на данной позиции)
3. Теперь сортируем хвост списка, исключив из рассмотрения уже отсортированные элементы

**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 selection\_sort(stack: Stack[VT]) -> None:  # 46n^3 + 149n^2 - 136n + 1

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

        min\_el\_pos = start\_pos  # 1

        min\_el = seek(stack, min\_el\_pos)  # 46n - 40

        for check\_el\_pos in range(start\_pos + 1, stack.size):  # n \* (

            check\_el = seek(stack, check\_el\_pos)  # 46n - 40

            if check\_el < min\_el:  # 1

                min\_el = check\_el  # 1

                min\_el\_pos = check\_el\_pos  # 1

        # ) = 46n^2 - 37n

        if min\_el\_pos != start\_pos:  # 1

            swap(stack, start\_pos, min\_el\_pos)  # 138n - 96

    # ) = 46n^3 - 37n^2 + 138n^2 - 96n + 46n^2 - 40n = 46n^3 + 149n^2 - 136n + 1

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

    import time

    from random import randint

    tests = 10

    step = 100

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

        stack = Stack[int]()

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

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

        start\_time = time.time()

        selection\_sort(stack)

        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: {stack.n\_op}")

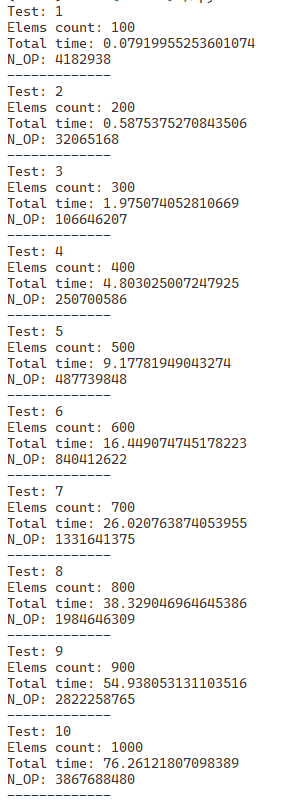
        print("-------------")

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во элементов | F(n) | O(F(n)) | Т(n) (сек) | N\_op |
| 100 | 47476401 | 1000000 | 0,078565359 | 4171990 |
| 200 | 373932801 | 8000000 | 0,580495834 | 32036165 |
| 300 | 1255369201 | 27000000 | 1,954221725 | 106554552 |
| 400 | 2967785601 | 64000000 | 4,81104517 | 250799279 |
| 500 | 5787182001 | 125000000 | 9,289477825 | 487748473 |
| 600 | 9989558401 | 216000000 | 16,33177757 | 840420350 |
| 700 | 15850914801 | 343000000 | 25,69459438 | 1331566119 |
| 800 | 23647251201 | 512000000 | 39,17893815 | 1984807677 |
| 900 | 33654567601 | 729000000 | 57,29567027 | 2822214375 |
| 1000 | 46148864001 | 1000000000 | 75,80766273 | 3867488863 |

|  |  |  |  |
| --- | --- | --- | --- |
| С1=F(n)/T(n) | С2=O(F(n))/T(n) | С3=F(n)/N\_op | С4=O(F(n))/N\_op |
| 604291783,7 | 12728255,95 | 11,37979741 | 0,239693767 |
| 644161041,1 | 13781321,98 | 11,67220861 | 0,24971778 |
| 642388314,8 | 13816241,86 | 11,78146947 | 0,253391333 |
| 616869203,3 | 13302722,74 | 11,83330994 | 0,255184147 |
| 622982487,3 | 13456084,65 | 11,86509507 | 0,256279634 |
| 611663877,8 | 13225749,56 | 11,88638329 | 0,257014243 |
| 616896868,1 | 13349111,29 | 11,90396374 | 0,257591414 |
| 603570497,8 | 13068245,95 | 11,91412724 | 0,257959502 |
| 587384132,9 | 12723474,51 | 11,92487995 | 0,258307805 |
| 608762522,7 | 13191278,61 | 11,93251374 | 0,258565709 |

**Таблица результата экспериментов и графики зависимостей**

**Скриншот работы программы:**



**Выводы.**

В данной работе был реализован класс Stack для работы со стеком на указателях, так же были реализованы базовые операции над ним.

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

**Литература:**

1. *Левитин А. В.* Глава 3. Метод грубой силы: Сортировка выбором // Алгоритмы. Введение в разработку и анализ — М.: Вильямс, 2006