|  |
| --- |
| 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 № 28**

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 № 28.**

**Реализация связи элементов линейного списка: Указатели**

**Способ организации линейного связанный список: Стек**

**Алгоритм сортировки: Быстрая по Хоару (без медианного (pivot) элемента)**

**Теория о сортировках.**

**Быстрая сортировка Хоара (QuickSort).**

Один из самых быстрых известных универсальных алгоритмов сортировки массивов: в среднем O ( n log ⁡ n ) {\displaystyle O(n\log n)} обменов при упорядочении n {\displaystyle n} элементов; из-за наличия ряда недостатков на практике обычно используется с некоторыми доработками.

QuickSort является существенно улучшенным вариантом алгоритма сортировки с помощью прямого обмена (его варианты известны как «Пузырьковая сортировка» и «Шейкерная сортировка»), известного в том числе своей низкой эффективностью. Принципиальное отличие состоит в том, что в первую очередь производятся перестановки на наибольшем возможном расстоянии и после каждого прохода элементы делятся на две независимые группы (таким образом улучшение самого неэффективного прямого метода сортировки дало в результате один из наиболее эффективных улучшенных методов).

Рисунок иллюстрирует сортировку Хоара.

Шаги алгоритма:

1. Выбрать элемент из массива. Назовём его опорным.

2. Разбиение: перераспределение элементов в массиве таким образом, что элементы, меньшие опорного, помещаются перед ним, а большие или равные - после.

3. Рекурсивно применить первые два шага к двум подмассивам слева и справа от опорного элемента. Рекурсия не применяется к массиву, в котором только один элемент или отсутствуют 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 quick\_sort(queue: Stack[VT]) -> Stack[VT]:  # 230n^2 + 2nlog(n) - 172n + 4

    def \_quick\_sort(

        left: int, right: int

    ) -> None:  # 230n^2 + 2nlog(n) - 172n + 4

        i = left  # 1

        j = right  # 1

        partition\_index = seek(queue, (i + j) // 2)  # 46n - 40

        while i < j:  # n \* (

            while seek(queue, i) > partition\_index:  # 46n - 40

                i += 1

            while seek(queue, j) < partition\_index:  # 46n - 40

                j -= 1

            if i <= j:  # 1

                if i < j:  # 1

                    swap(queue, i, j)  # 138n - 96

                i += 1  # 1

                j -= 1  # 1

        # ) = 230n^2 - 172n

        if left < j:  # 1

            \_quick\_sort(left, j)  # ~= nlog(n)

        if i < right:  # 1

            \_quick\_sort(i, right)  # ~= nlog(n)

    \_quick\_sort(0, queue.size - 1)

    return queue

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))

        quick\_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()

            quick\_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}")

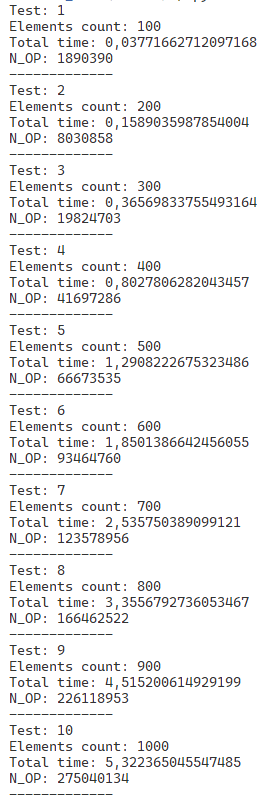
            print(f"N\_OP: {struct.n\_op}")

            print("-------------")

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во элементов | F(n) | O(F(n)) | Т(n) (сек) | N\_op |
| 100 | 2284133 | 10000 | 0,037716627 | 1890390 |
| 200 | 9168662 | 40000 | 0,158903599 | 8030858 |
| 300 | 20653341 | 90000 | 0,365698338 | 19824703 |
| 400 | 36738119 | 160000 | 0,802780628 | 41697286 |
| 500 | 57422970 | 250000 | 1,290822268 | 66673535 |
| 600 | 82707879 | 360000 | 1,850138664 | 93464760 |
| 700 | 112592836 | 490000 | 2,535750389 | 123578956 |
| 800 | 147077834 | 640000 | 3,355679274 | 166462522 |
| 900 | 186162869 | 810000 | 4,515200615 | 226118953 |
| 1000 | 229847936 | 1000000 | 5,322365046 | 275040134 |

|  |  |  |  |
| --- | --- | --- | --- |
| С1=F(n)/T(n) | С2=O(F(n))/T(n) | С3=F(n)/N\_op | С4=O(F(n))/N\_op |
| 60560366,76 | 265135,0548 | 1,208286529 | 0,005289914 |
| 57699521,05 | 251724,9471 | 1,141678952 | 0,004980788 |
| 56476442,93 | 246104,4822 | 1,04179827 | 0,004539791 |
| 45763584,46 | 199307,2508 | 0,881067393 | 0,00383718 |
| 44485574,22 | 193674,9979 | 0,861255816 | 0,003749614 |
| 44703610,7 | 194580,0101 | 0,884909763 | 0,003851719 |
| 44402176,25 | 193236,6853 | 0,911100396 | 0,003965076 |
| 43829526,66 | 190721,445 | 0,883549236 | 0,003844709 |
| 41230254,13 | 179394,0223 | 0,823296174 | 0,003582185 |
| 43185300,82 | 187886,3985 | 0,835688713 | 0,003635833 |

**Таблица результата экспериментов и графики зависимостей**

**Скриншот работы программы:**

**Выводы.**

В данной работе был реализован класс Stack для работы со стеком на указателях. Также были реализованы базовые операции над этой структурой.

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

**Литература:**

1. Левитин А. В. Глава 4. Метод декомпозиции: Быстрая сортировка // Алгоритмы. Введение в разработку и анализ — М.: Вильямс, 2006.