|  |
| --- |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МИРЭА – Российский технологический университет"**  **РТУ МИРЭА** |

Институт комплексной безопасности и специального приборостроения

Кафедра КБ-14 «Цифровые технологии обработки данных»

**ОТЧЕТ   
о выполнении лабораторной работы №1**

**«Реализация сортировки линейных структур данных»**

**по дисциплине   
«Алгоритмы и структуры данных»**

**Вариант № 6**

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 № 6.**

**Реализация связи элементов линейного списка:** **Массив**

**Способ организации линейного связанный список: Стек**

**Алгоритм сортировки**: **Сравнение и подсчет**

**Теория о сортировках.**

**Сортировка сравнением и подсчетом.**

Сортировка сравнением и подсчетом – алгоритм сортировки целых неотрицательных чисел. Данный алгоритм использует массив - счетчик длиной, равной диапазону среди различных неповторяющихся сортируемых объектов. Сортировка происходит в соответствии с индексами буфера, где индексом является сам элемент, а значением – количество этих повторяющихся элементов. Алгоритм работает за линейное время O(n+k), где n – длина исходного массива, k – массива - счетчика. Алгоритм особенно эффективен для чисел небольшого диапазона. Например, если требуется отсортировать большое количество чисел с небольшим разбросом среди сортируемых значений, поэтому для эффективной работы k не должно сильно быть больше, чем n.

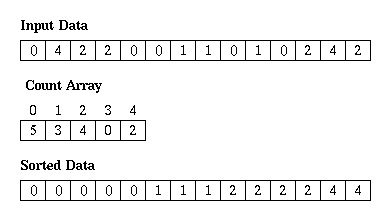


Рисунок 1. – Пример отсортированных данных.

Рисунок 1 иллюстрирует работу данного алгоритма. Первостепенно идет обход по исходному массиву в поисках максимального значения. После чего инициализируется счетчик – массив для подсчета количества каждого элемента. Длина счетчика равна единице, прибавленной к максимальному элементу. Так, в примере на рисунке максимальным элементом является четыре. Следовательно, длина счетчика равна пяти. Далее идет подсчет количество повторений каждого элемента в исходном массиве и присвоение его счетчику, где индекс – элемент, повторения которого считали, а значение – количество повторений. Например, в примере на рисунке 1 значение ноль повторяется пять раз, следовательно, нулевой элемент счетчика равен пяти, один повторяется три раза, значит, первый элемент равен трем и т.д. После завершения подсчета всех значений в исходный массив поочередно записываются индексы счетчика столько раз, сколько записаны повторения в значениях этих индексов. Так, если нулевой элемент равен пяти, то в массив запишется ноль пять раз, а если первый элемент равен 3, то один запишется 3 раза и т.д. На этом сортировка завершена

**Листинг программы с расчетами.**

from typing import Generic, List, TypeVar

VT = TypeVar("VT")

class Stack(Generic[VT]):

    \_stack: List[VT]

    \_size: int

    \_n\_op: int

    def \_\_init\_\_(self, max\_size: int = 10000):

        self.\_stack = [None for \_ in range(max\_size)]

        self.\_n\_op = 0

        self.\_size = 0

    def push(self, el: VT):  # 5

        self.\_stack[self.size] = el  # 3

        self.\_size += 1  # 2

        self.\_n\_op += 5

    def pop(self) -> VT:  # 9

        if self.is\_empty():  # 2

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

        el = self.\_stack[self.\_size - 1]  #  5

        self.\_size -= 1  # 2

        self.\_n\_op += 9

        return el

    def is\_empty(self) -> bool:  # 2

        return self.\_size == 0

    @property

    def n\_op(self) -> int:

        return self.\_n\_op

    @property

    def size(self) -> int:  # 1

        return self.\_size

    @property

    def top(self) -> VT:  # 1

        return self.\_stack[self.\_size - 1]

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

    buffer = Stack[VT](max\_size=stack.size)

    elements = []

    for \_ in range(stack.size):

        el = stack.pop()

        elements.append(el)

        buffer.push(el)

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

    for \_ in range(buffer.size):

        stack.push(buffer.pop())

def seek(stack: Stack[VT], pos: int) -> VT:  # 28n - 22

    buffer = Stack[VT](max\_size=stack.size)  # 2

    stack\_size = stack.size  # 2

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

        buffer.push(stack.pop())  # 14

    # ) = 14n - 14

    el = stack.top  # 2

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

        stack.push(buffer.pop())  # 14

    # ) = 14n - 14

    stack.\_n\_op += buffer.n\_op

    return el

def push\_by\_pos(stack: Stack[VT], el: VT, pos: int) -> None:  # 28n + 9

    buffer = Stack[VT](max\_size=stack.size)  # 2

    stack\_size = stack.size  # 2

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

        buffer.push(stack.pop())  # 14

    # ) = 14n

    stack.push(el)  # 5

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

        stack.push(buffer.pop())  # 14

    # ) = 14n

    stack.\_n\_op += buffer.n\_op

def pop\_by\_pos(stack: Stack[VT], pos: int) -> VT:  # 28n - 15

    buffer = Stack[VT](max\_size=stack.size)  # 2

    stack\_size = stack.size  # 2

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

        buffer.push(stack.pop())  # 14

    # )= 14n - 14

    el = stack.pop()  # 9

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

        stack.push(buffer.pop())  # 14

    # ) = 14n - 14

    stack.\_n\_op += buffer.n\_op

    return el

def swap(stack: Stack[VT], pos1: int, pos2: int) -> None:  # 112n - 12

    temp = pop\_by\_pos(stack, pos1)  #  28n - 15

    push\_by\_pos(stack, pop\_by\_pos(stack, pos2 - 1), pos1)  # 56n - 6

    push\_by\_pos(stack, temp, pos2)  # 28n + 9

def reverse(stack: Stack[VT]) -> Stack[VT]:  # 14n + 2

    out = Stack[VT](max\_size=stack.size)  # 2

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

        out.push(stack.pop())  # 14

    # ) = 14n

    out.\_n\_op += stack.n\_op

    return out

def slice\_(stack: Stack[VT], l: int = 0, r: int = -1) -> Stack[VT]:  # 62n + 18

    stack\_size = stack.size  # 2

    slice\_stack = Stack[VT](max\_size=stack.size)  # 2

    buffer = Stack[VT](max\_size=stack.size)  # 2

    if r == -1:  # 1

        r = stack.size - 1  # 3

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

        buffer.push(stack.pop())  # 14

    # ) = 14n - 14

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

        slice\_stack.push(stack.top)  # 6

        buffer.push(stack.pop())  # 14

    # ) = 20n + 20

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

        stack.push(buffer.pop())  # 14

    # ) = 14n

    slice\_stack.\_n\_op += buffer.n\_op

    slice\_stack.\_n\_op += stack.n\_op

    return reverse(slice\_stack)  # 14n + 2

def counting\_sort(stack: Stack[VT]) -> Stack[VT]: # 33n + 5

    max\_ = stack.top  # 2

    min\_ = max\_  # 1

    swap\_stack = Stack[VT](max\_size=stack.size)  # 2

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

        el = stack.pop()  # 9

        if el > max\_:  # 1

            max\_ = el  # 1

        if el < min\_:  # 1

            min\_ = el  # 1

        swap\_stack.push(el)  # 5

    # ) = 16n

    buffer = [0] \* (max\_ - min\_ + 1)  # 1

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

        el = swap\_stack.pop()  # 9

        buffer[el - min\_] += 1  # 3

        stack.\_n\_op += 3

    # ) = 12n

    for i in range(len(buffer) - 1, -1, -1):  # n \* (

        for \_ in range(buffer[i]):  # 1 \* (

            stack.push(i + min\_)  # 5

        # ) = 5

    # ) = 5n

    stack.\_n\_op += swap\_stack.n\_op

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))

        counting\_sort(struct)

        print\_stack(struct)

    elif sys.argv[1] == "tests":

        tests = 10

        step = 10000

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

            struct = Stack[int](max\_size=test\_num \* step)

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

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

            start\_time = time.time()

            counting\_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}".replace(".", ","))

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

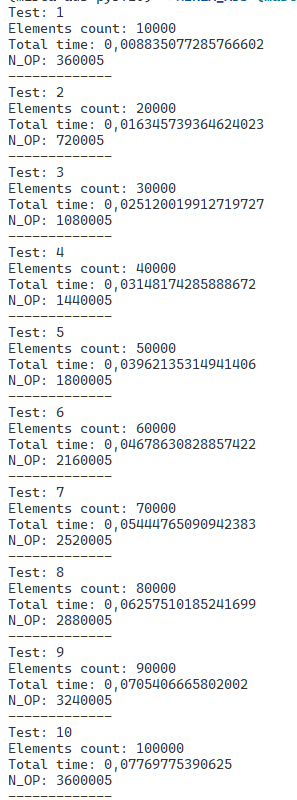
            print("-------------")

**Таблица результата экспериментов и графики зависимостей**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество элементов | F(N) | O(F(n)) | T(n) (сек) | N\_op |
| 10000 | 330005 | 10000 | 0,008835077 | 360005 |
| 20000 | 660005 | 20000 | 0,016345739 | 720005 |
| 30000 | 990005 | 30000 | 0,02512002 | 1080005 |
| 40000 | 1320005 | 40000 | 0,031481743 | 1440005 |
| 50000 | 1650005 | 50000 | 0,039621353 | 1800005 |
| 60000 | 1980005 | 60000 | 0,046786308 | 2160005 |
| 70000 | 2310005 | 70000 | 0,054447651 | 2520005 |
| 80000 | 2640005 | 80000 | 0,062575102 | 2880005 |
| 90000 | 2970005 | 90000 | 0,070540667 | 3240005 |
| 100000 | 3300005 | 100000 | 0,077697754 | 3600005 |

|  |  |  |  |
| --- | --- | --- | --- |
| С1=F(n)/T(n) | С2=O(F(n))/T(n) | С3=F(n)/N\_op | С4=O(F(n))/N\_op |
| 37351682,31 | 1131852,012 | 0,916667824 | 0,027777392 |
| 40377800,31 | 1223560,437 | 0,916667245 | 0,027777585 |
| 39410995,83 | 1194266,569 | 0,916667052 | 0,027777649 |
| 41929222,47 | 1270577,686 | 0,916666956 | 0,027777681 |
| 41644337,43 | 1261945,795 | 0,916666898 | 0,027777701 |
| 42320180,25 | 1282426,466 | 0,91666686 | 0,027777713 |
| 42426164,61 | 1285638,569 | 0,916666832 | 0,027777723 |
| 42189383,99 | 1278463,76 | 0,916666811 | 0,02777773 |
| 42103443,93 | 1275859,789 | 0,916666795 | 0,027777735 |
| 42472334,58 | 1287038,492 | 0,916666782 | 0,027777739 |

**Скриншот работы программы:**

****

**Выводы.**

В результате данной работы был реализован класс Stack для работы со стеком на массиве. Также были реализованы базовые операции над ним и рассчитана их алгоритмическая сложность.По результатам экспериментов было установлено, что алгоритм сортировки сравнением иподсчетом для стека на массиве имеет линейную зависимость от числа элементов

**Литература:**

1. Алгоритмы: построение и анализ. Кормен Т.Х., Лейзерсон Ч.И.

2. https://en.wikipedia.org/wiki/Counting\_sort