САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №4

по курсу «Алгоритмы и структуры данных»

Тема: Стек, очередь, связанный список.

Вариант 15

Выполнил:
Скворцов Д.А.
К3140

Проверил:
Афанасьев А.В.

Санкт-Петербург
2024 г.

Содержание отчета

Содержание отчета **2**
Задачи по варианту **3**
Задача №1. Улучшение Quick sort 3
Задача №2. Анти-quick sort 5
Задача №4. Точки и отрезки 7
Дополнительные задачи **10**
Задача №5. Индекс Хирша 10
Задача №6. Сортировка целых чисел 11
Задача №8. K ближайших точек к началу координат 14
Вывод **17**

Задачи по варианту

Задача №1. Стек

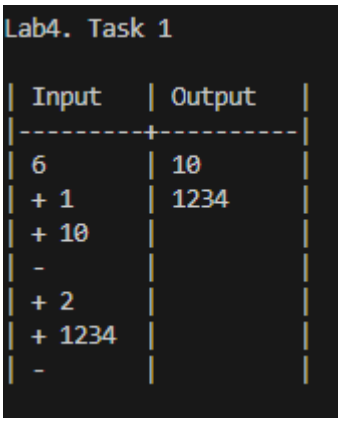
Реализуйте работу стека. Для каждой операции изъятия элемента выведите ее результат. На вход программе подаются строки, содержащие команды. Каждая строка содержит одну команду. Команда — это либо “+ N”, либо “–”. Команда “+ N”означает добавление в стек числа N, по модулю не превышающего 10^9. Команда “–”означает изъятие элемента из стека. Гарантируется, что не происходит извлечения из пустого стека. Гарантируется, что размер стека в процессе выполнения команд не превысит 10^6 элементов.

- **Формат входного файла (input.txt).** В первой строке входного файла содержится M ($1 \leq M \leq 10^6$)– число команд. Каждая последующая строка исходного файла содержит ровно одну команду.
- **Формат выходного файла (output.txt).** Выведите числа, которые удаляются из стека с помощью команды “–”, по одному в каждой строке. Числа нужно выводить в том порядке, в котором они были извлечены из стека. Гарантируется, что изъятий из пустого стека не производится.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.

Листинг кода.

```
def solution(lst):
    stack = []
    ans = []
    for command in lst:
        cmd = command.split()
        if cmd[0] == '+':
            stack.append(int(cmd[1]))
        else:
            ret = stack.pop()
            ans.append(ret)
    return ans
```

Стек реализован при помощи функционала списка. Каждой операции *push* сопоставлен метод *append*, для *pop* - *pop*.




```
.Тест 100 элементов:
Время работы: 7.209996692836285e-05 секунд
Затрачено памяти: 688 Байт
Тест 10e4 элементов:
Время работы: 0.005868299980647862 секунд
Затрачено памяти: 190.6 Килобайт
Тест 10e6 элементов:
Время работы: 0.30300079996231943 секунд
Затрачено памяти: 19.3 Мегабайт
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0000720999669283628 секунд	688 байт
Медиана диапазона значений входных данных из текста задачи	0.005868299980647862 секунд	190.6 Килобайт
Верхняя граница диапазона значений входных данных из текста задачи	0.30300079996231943 секунд	19.3 Мегабайт

Вывод по задаче:
Алгоритм работает за линейное время и проходит все тесты на память.

Задача №4. Скобочная последовательность. Версия 2

Определение правильной скобочной последовательности такое же, как и в задаче 3, но теперь у нас больше набор скобок: []{}(). Нужно написать функцию для проверки наличия ошибок при использовании разных типов скобок в текстовом редакторе типа LaTeX. Для удобства, текстовый редактор должен не только информировать о наличии ошибки в использовании скобок, но также указать точное место в коде (тексте) с ошибочной скобочкой. В первую очередь объявляется ошибка при наличии первой несовпадающей закрывающей скобки, перед которой отсутствует открывающая скобка, или ко торая не соответствует открывающей, например, ()[}- здесь ошибка укажет на }. Во вторую очередь, если описанной выше ошибки не было найдено, нужно указать на первую несовпадающую открывающую скобку, у которой отсутствует закрывающая, например, (в ([]. Если не найдено ни одной из указанный выше ошибок, нужно сообщить, что использование скобок корректно. Помимо скобок, код может содержать большие и маленькие латинские буквы, цифры и знаки препинания.

Листинг кода.

```
def solution(line):
    is_open = {'(': True, '[': True, '{': True,
               ')': False, ']': False, '}': False}
    clone = {')': '(', ']': '[', '}': '{'}
    stack = []
    for i, symb in enumerate(line):
        symb = line[i]
        if symb not in is_open:
            continue
        if is_open[symb]:
            stack.append(symb)
        elif stack and stack[-1] == clone[symb]:
            stack.pop()
        else:
            return i + 1
    if not stack:
        return 'Success'
    return len(line)
```

Используем функционал стека, чтобы запоминать порядок наложения скобок. При любом несоответствии возвращаем индекс символа, на котором всё сломалось, иначе - *“Success”*.

```
Lab4. Task 4

| Input       |   Output |
|-------------+----------|
| foo(bar[i); |       10 |
```

```
.....Тест 100 элементов:
Время работы: 2.2499996703118086e-05 секунд
Затрачено памяти: 744 Байт
Тест 10e4 элементов:
Время работы: 0.001773000054527074 секунд
Затрачено памяти: 41.2 Килобайт
Тест 10e6 элементов:
Время работы: 0.2277961999643594 секунд
Затрачено памяти: 4.0 Мегабайт
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000022499996703 секунд	744 байт
Медиана диапазона значений входных данных из текста задачи	0.001773000054527074 секунд	41.2 Килобайт
Верхняя граница диапазона значений входных данных из текста задачи	0.2277961999643594 секунд	4.0 Мегабайт

Вывод по задаче:
Алгоритм работает корректно, его асимптотическая сложность - *O(n)*. В худшем случае программа доходит до конца строки, накопив большой стэк открытых скобок.

Задача №5. Стек с максимумом

Стек- это абстрактный тип данных, поддерживающий операции Push() и Pop().Нетруднореализоватьеготакимобразом,чтобыобеэтиоперацииработал и за константное время. В этой задаче ваша цель- реализовать стек, который также поддерживает поиск максимального значения и гарантирует, что все операции по-прежнему работают за константное время. Реализуйте стек, поддерживающий операции Push(), Pop() и Max().

- **Формат входного файла (input.txt).** В первой строке входного файла содержится n ($1 \le n \le 400000$) – число команд. Последующие n строк исходного файла содержит ровно одну команду: push V, pop или max. $0 \le V \le 10^5$.
- **Формат выходного файла (output.txt).** Для каждого запроса Max выведите (в отдельной строке) максимальное значение стека.
- Ограничение по времени. 5 сек.
- Ограничение по памяти. 512 мб.

Листинг кода.

```
class Stack:
    def __init__(self):
        self.stack = []

    def push(self, element):
        self.stack.append(element)

    def pop(self):
        if not self.stack:
            return
        return self.stack.pop()

    def get_last(self):
        if not self.stack:
            return
        return self.stack[-1]


class MaxStack(Stack):
    def __init__(self):
        super().__init__()
        self.max_stack = Stack()
```

```
    def push(self, element):
        super().push(element)
        last_max = self.max_stack.get_last()
        if last_max is not None:
            self.max_stack.push(max(last_max, element))
        else:
            self.max_stack.push(element)

    def pop(self):
        self.max_stack.pop()
        return super().pop()

    def max(self):
        return self.max_stack.get_last()


def solution(commands):
    stack = MaxStack()
    res = []
    for command in commands:
        cmd = command.split()
        if cmd[0] == 'push':
            stack.push(int(cmd[1]))
        elif cmd[0] == 'pop':
            stack.pop()
        else:
            res.append(stack.max())
    return res
```

От обычного класса *Stack* унаследован класс *MaxStack,* в котором присутствует внутренний стек, сохраняющий максимум для каждого добавленного элемента. Таким образом, операция нахождения максимума, как и операции *pop, push,* работает за O(1).

```
Lab4. Task 5

| Input    | Output    |
|----------+-----------|
| 5        | 2         |
| push 2   | 2         |
| push 1   |           |
| max      |           |
| pop      |           |
| max      |           |
```

```
........Тест 100 элементов:
Время работы: 0.00015799998072907329 секунд
Затрачено памяти: 1.7 Килобайт
Тест 10e4 элементов:
Время работы: 0.017147700011264533 секунд
Затрачено памяти: 232.0 Килобайт
Тест 4*10e5 элементов:
Время работы: 0.2899153999751434 секунд
Затрачено памяти: 9.2 Мегабайт
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00015799998072907329 секунд	1.7 Килобайт
Медиана диапазона значений входных данных из текста задачи	0.017147700011264533 секунд	232.0 Килобайт
Верхняя граница диапазона значений входных данных из текста задачи	0.2899153999751434 секунд	9.2 Мегабайт

Вывод по задаче:

Алгоритм работает корректно, для последовательности команд время его работы - $O(n)$. Памяти затрачивается относительно мало. Такой алгоритм можно использовать в задачах, требующих частого нахождения максимума или любого другой характеристики данных в стеке.

Задача №9. Поликлиника

Очередь в поликлинике работает по сложным правилам. Обычные пациенты при посещении должны вставать в конец очереди. Пациенты, которым "только справку забрать встают ровно в ее середину, причем при нечетной длине очереди они встают сразу за центром. Напишите программу, которая отслеживает порядок пациентов в очереди.

- **Формат входного файла (input.txt).** В первой строке записано одно целое число n ($1 \le n \le 10^5$)-число запросов к вашей программе. В следующих n строках заданы описания запросов в следующем формате:
  - «+i» – к очереди присоединяется пациент i ($1 \le i \le N$) и встает в ее конец;
  - «*i» – пациент i встает в середину очереди ($1 \le i \le N$);
  - «-» – первый пациент в очереди заходит к врачу. Гарантируется, что на момент каждого такого запроса очередь будет не пуста.
- **Формат выходного файла (output.txt).** Для каждого запроса третьего типа в отдельной строке выведите номер пациента, который должен зайти к шаманам.

Листинг кода.

```
class Node:
    def __init__(self, value, next=None, prev=None):
        self.val = value
        self.next = next
        self.prev = prev

    def put_between(self, node1: 'Node', node2: 'Node'):
        if node1 is None and node2 is None:
            return

        if node2 is not None:
            self.next = node2
            node2.prev = self

        if node1 is not None:
            self.prev = node1
            node1.next = self
```

```
class QueueMid:
    def __init__(self):
        self.head = None
        self.tail = None
        self.to_mid = None
        self.len = 0

    def put(self, node: Node):
        if self.head is None:
            self.head = node
            self.to_mid = self.head
        elif self.tail is None:
            self.tail = node
            self.tail.next = self.head
            self.head.prev = self.tail
            self.to_mid = self.head
        else:
            prev_tail = self.tail
            self.tail = node
            self.tail.next = prev_tail
            prev_tail.prev = self.tail

            if self.len % 2 == 0:
                self.to_mid = self.to_mid.prev

        self.len += 1

    def put_to_mid(self, node: Node):
        if self.head is None or self.tail is None:
            self.put(node)
            return
        else:
            prev_from_mid = self.to_mid.prev
            next_from_mid = self.to_mid
            node.put_between(prev_from_mid, next_from_mid)

        if self.len % 2 == 0:
            self.to_mid = self.to_mid.prev

        self.len += 1

    def get(self):
        prev_head = self.head
```

```
        if prev_head is not None:
            new_head = self.head.prev
            if new_head is not None:
                new_head.next = None
            self.head = new_head

            self.len -= 1

        return prev_head

    def print(self):
        cur = self.head
        res = []
        while cur != None:
            res.append(cur.val)
            cur = cur.prev
        print(res)


def solution(lst):
    queue = QueueMid()
    ans = []
    for command in lst:
        cmd = command.split()
        if cmd[0] == '+':
            queue.put(Node(int(cmd[1])))
        elif cmd[0] == '*':
            queue.put_to_mid(Node(int(cmd[1])))
        else:
            ret = queue.get()
            ans.append(ret.val)
    return ans
```

Реализован вспомогательный класс *Node,* чтобы упростить операции в основном классе - *QueueMid.* Помимо стандартных двух указателей на начало и конец, у неё есть третий, отвечающий за текущий средний элемент, за который необходимо ставить ноду. После добавления проверок на пустоту некоторых переменных класс работает как стандартная очередь.

```
Lab4. Task 9

| Input   | Output   |
|---------+----------|
| 10      | 1        |
| + 1     | 3        |
| + 2     | 2        |
| * 3     | 5        |
| -       | 4        |
| + 4     |          |
| * 5     |          |
| -       |          |
| -       |          |
| -       |          |
| -       |          |
```

```
..Тест 100 элементов:
Время работы: 7.139996159821749e-05 секунд
Затрачено памяти: 4.7 Килобайт
Тест 1000 элементов:
Время работы: 0.0016130000003613532 секунд
Затрачено памяти: 43.4 Килобайт
Тест 10e5 элементов:
Время работы: 0.06775749998632818 секунд
Затрачено памяти: 4.2 Мегабайт
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0000713999615982175 секунд	4.7 Килобайт
Медиана диапазона значений входных данных из текста задачи	0.0016130000003613532 секунд	43.4 Килобайт
Верхняя граница диапазона значений входных данных из текста задачи	0.06775749998632818 секунд	4.2 Мегабайт

Вывод по задаче:

Алгоритм работает за линейное время, будучи практически равным по скорости выполнения обычной очереди. Также память практически не тратится, поскольку значения существенно ограничены условием.

Дополнительные задачи

Задача №7. Максимум в движущейся последовательности

Задан массив из n целых чисел - a1, ..., an и число m < n , нужно найти максимум среди последовательности ("окна") {ai,...,ai+m−1} для каждого значения 1 ≤ i ≤ n−m+1. Простой алгоритм решения этой задачи за O(nm) сканирует каждое "окно" отдельно. Ваша цель - алгоритм за O(n)

- **Формат входного файла (input.txt).** В первой строке содержится целое число n ($1 \le n \le 10^5$) – количество чисел в исходном массиве, вторая строка содержит n целых чисел a1,..., an этого массива, разделенных пробелом ($0 \le ai \le 10^5$). В третьей строке - целое число m - ширина "окна" (1≤m≤n).
- **Формат выходного файла (output.txt).** Нужно вывести max ai,..., ai+m−1 для каждого 1≤i≤n−m+1.
- Ограничение по времени. 5 сек.
- Ограничение по памяти. 512 мб.

Листинг кода.

```
class Stack:
    def __init__(self):
        self.stack = []

    def push(self, element):
        self.stack.append(element)

    def pop(self):
        if not self.stack:
            return
        return self.stack.pop()

    def peek(self):
        if not self.stack:
            return
        return self.stack[-1]
```

```
    def print(self, arg='', **kvargs):
        print(arg, self.stack, **kvargs)


class MaxStack(Stack):
    def __init__(self):
        super().__init__()
        self.max_stack = Stack()

    def push(self, element):
        super().push(element)
        last_max = self.max_stack.peek()
        if last_max is not None:
            self.max_stack.push(max(last_max, element))
        else:
            self.max_stack.push(element)

    def pop(self):
        self.max_stack.pop()
        return super().pop()

    def max(self):
        return self.max_stack.peek()

    def is_empty(self):
        return self.peek() is None

    def print(self):
        super().print(arg='stack:', end=' ')
        self.max_stack.print(arg='max:')


class QueueMax():
    def __init__(self):
        self.input_stack = MaxStack()
        self.output_stack = MaxStack()

    def peek(self):
        if self.output_stack.is_empty():
            while not self.input_stack.is_empty():
                self.output_stack.push(self.input_stack.pop())
        return self.output_stack.peek()
```

```
    def pop(self):
        if self.output_stack.is_empty():
            while not self.input_stack.is_empty():
                self.output_stack.push(self.input_stack.pop())
        return self.output_stack.pop()

    def put(self, value):
        self.input_stack.push(value)

    def peek_max(self):
        if self.input_stack.max() is None:
            return self.output_stack.max()
        if self.output_stack.max() is None:
            return self.input_stack.max()
        return max(self.input_stack.max(), self.output_stack.max())

    def print(self):
        print('inp/out')
        self.input_stack.print()
        self.output_stack.print()


def solution(n, lst, m):
    queue = QueueMax()
    for i in range(m):
        queue.put(lst[i])

    ans = [queue.peek_max()]
    for i in range(m, n):
        queue.pop()
        queue.put(lst[i])
        ans.append(queue.peek_max())

    return ans
```

Очередь с максимумом реализована на основе двух стеков с максимумом, каждый из которых хранит максимумы значений на каждом этапе добавления в очередь. Стеки отвечают за изменение порядка данных при выходе, чтобы очередь выбрасывала значения из “окна” последовательно. При помощи очереди легко получается сделать срез данного массива и пройти по нему, собирая информацию о каждом “локальном максимуме”.

```
Lab4. Task 7

| Input           | Output    |
|-----------------+-----------|
| 8               | 7 7 5 6 6 |
| 2 7 3 1 5 2 6 2 |           |
| 4               |           |
```

```
.....Тест 100 элементов:
Время работы: 0.00017720001051202416 секунд
Затрачено памяти: 2.8 Килобайт
Тест 1000 элементов:
Время работы: 0.002266200026497245 секунд
Затрачено памяти: 15.6 Килобайт
Тест 10e5 элементов:
Время работы: 0.2047137000481598 секунд
Затрачено памяти: 1.4 Мегабайт
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000177200010512024 секунд	2.8 Килобайт
Медиана диапазона значений входных данных из текста задачи	0.002266200026497245 секунд	15.6 Килобайт
Верхняя граница диапазона значений входных данных из текста задачи	0.2047137000481598 секунд	1.4 Мегабайт

Вывод по задаче:

Правильная реализация очереди позволила сократить время нахождения максимума до константного. Таким образом, мы затрачиваем дополнительную память, но оптимизируем алгоритм для тех ситуаций, когда в задаче необходимо часто узнавать “локальный максимум”.

Задача №12. Строй новобранцев

В этой задаче n новобранцев, пронумерованных от 1 до n, разделены на два множества: строй и толпа. Вначале строй состоит из новобранца номер 1, все остальные составляют толпу. В любой момент времени строй стоит в один ряд по прямой. Товарищ сержант может использовать четыре команды. Вот они.

- "I, встать в строй слева от J."Эта команда заставляет новобранца номер I, находящегося в толпе, встать слева от новобранца номер J, находящегося в строю.
- "I, встать в строй справа от J." Эта команда действует аналогично предыдущей, за исключением того, что I встает справа от J.
- "I, выйти из строя." Эта команда заставляет выйти из строя новобранца но мер I. После этого он присоединяется к толпе.
- "I, назвать соседей." Эта команда заставляет глубоко задуматься новобранца номер I, стоящего в строю, и назвать номера своих соседей по строю, сначала левого, потом правого. Если кто-то из них отсутствует (новобранец находится на краю ряда), то вместо соответствующего номера он должен назвать 0.

Известно, что ни в каком случае строй не остается пустым. Иногда строй становится слишком большим, и товарищ сержант уже не может проверять сам, правильно ли отвечает новобранец. Поэтому он попросил вас написать программу, которая помогает ему в нелегком деле обучения молодежи и выдает правильные ответы для его команд.

- **Формат входного файла (input.txt).** В первой строке находятся два числа N ($1 \leq N \leq 75000$) и M ($1 \leq M \leq 75000$) - количество новобранцев и команд соответственно. Следующие M строк содержат команды, одна команда на строку. Каждая команда- одна из следующих:
  - left I J - соответствует команде "I, встать в строй слева от J."
  - right I J - "I,встать в строй справа от J."
  - leave I - "I, выйти из строя."
  - name I - "I, назвать соседей."

  Гарантируется, что все команды корректны, например, leave I не будет заставлять выйти из строя новобранца, стоящего в толпе. Также гарантиру ется, что строй никогда не будет пустым.
- **Формат выходного файла (output.txt).** Для каждой строки, содержащей name I, выведите в отдельной строке два числа - номера

левого и правого соседа новобранца номер I. Если кто-то из соседей отсутствует, выведите ноль вместо его номера.

Листинг кода.

```
class Node:
    def __init__(self, value, next=None, prev=None):
        self.val = value
        self.next = next
        self.prev = prev

    def put_between(self, node1: 'Node', node2: 'Node'):
        if node2 is not None:
            self.next = node2
            node2.prev = self

        if node1 is not None:
            self.prev = node1
            node1.next = self

    def pop(self):
        if self.prev is not None:
            self.prev.next = self.next
        if self.next is not None:
            self.next.prev = self.prev
        self.next = None
        self.prev = None

    def name(self):
        res = [(0 if self.prev is None else self.prev.val),
               (0 if self.next is None else self.next.val)]
        return res


class LinkedListDictionary:
    def __init__(self, n):
        self.dct = {i: Node(i) for i in range(1, n + 1)}

    def put(self, node_put_idx, node2_idx, turn):
        if node_put_idx is not None:
            node_put = self.dct[node_put_idx]
        if node2_idx is not None:
            node2 = self.dct[node2_idx]
```

```
            if turn == LEFT:
                node1 = node2.prev
                node_put.put_between(node1, node2)
            else:
                node1 = node2.next
                node_put.put_between(node2, node1)

    def pop(self, node_idx):
        node = self.dct[node_idx]
        node.pop()

    def name(self, node_idx):
        node = self.dct[node_idx]
        return node.name()


def solution(n, commands):
    linked_list = LinkedListDictionary(n)
    linked_list.put(1, None, LEFT)
    ans = []
    for command in commands:
        cmd = command.split()
        if cmd[0] == 'left':
            node1_idx = int(cmd[1])
            node2_idx = int(cmd[2])
            linked_list.put(node1_idx, node2_idx, LEFT)
        elif cmd[0] == 'right':
            node1_idx = int(cmd[1])
            node2_idx = int(cmd[2])
            linked_list.put(node1_idx, node2_idx, RIGHT)
        elif cmd[0] == 'leave':
            node_idx = int(cmd[1])
            linked_list.pop(node_idx)
        else:
            node_idx = int(cmd[1])
            ans.append(linked_list.name(node_idx))
    return ans
```

Реализован класс связного списка с оптимизацией, позволяющей быстро обращаться по любому номеру. В словаре хранится ссылки на ноды,

которые, в свою очередь, ссылаются на своих соседей слева и справа (если таковые имеются).

```
Lab4. Task 12

| Input      | Output    |
|------------+-----------|
| 3 3        | 2 3       |
| left 2 1   |           |
| right 3 1  |           |
| name 1     |           |
```

```
..Тест 100 элементов:
Время работы: 0.00012919999426230788 секунд
Затрачено памяти: 14.4 Килобайт
Тест 1000 элементов:
Время работы: 0.0014013000181876123 секунд
Затрачено памяти: 161.2 Килобайт
Тест 75000 элементов:
Время работы: 0.09809339995263144 секунд
Затрачено памяти: 12.5 Мегабайт
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00012919999426230788 секунд	14.4 Килобайт
Медиана диапазона значений входных данных из текста задачи	0.0014013000181876123 секунд	161.2 Килобайт
Верхняя граница диапазона значений входных данных из текста задачи	0.09809339995263144 секунд	12.5 Мегабайт

Вывод по задаче:
Время работы алгоритма - O(n), но из-за своей структуры он расходует много памяти на сохранение ссылок на объекты. Хранение объектов в словаре позволяет добиться константного времени выполнения для каждой из 4 представленных команд.

Вывод

Такие структуры данных, как стек, очередь и связный список позволяют решать многие классы задач, если правильно применить абстракцию. Они существенно уменьшают время выполнения программы, сокращая время многих стандартных команд до константного, но ценой, иногда довольно больших, затрат по памяти.