**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЕВМ**

Курсовая РАБОТА

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

Тема: **Разработка структуры данных для хранения последних команд**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3344 |  | Тукалкин В.А. |
| Преподаватель |  | Иванов Д.В. |

Санкт-Петербург

2024

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Тукалкин В.А. | | |
| Группа 3344 | | |
| Тема работы: " Разработка структуры данных для хранения последних команд" | | |
| Исходные данные:   * Храним N последних команд пользователя в терминале (N небольшое, например, N<=20). * Должна быть возможность увеличить количество хранимых команд в данный момент времени или уменьшить. * Должна быть возможность вывода очереди. * При уменьшении очереди лишние команды теряются. | | |
| Содержание пояснительной записки:   1. Содержание 2. Введение 3. Задание варианта 4. Исследование 5. Методы 6. Полученные результаты 7. Заключение 8. Список использованных источников 9. Приложение А | | |
| Предполагаемый объем пояснительной записки:  Не менее 16 страниц. | | |
| Дата выдачи задания: 05.11.2024 | | |
| Дата сдачи реферата: 10.12.2024 | | |
| Дата защиты реферата: 10.12.2024 | | |
| Студент |  | Тукалкин В.А. |
| Преподаватель |  | Иванов Д.В. |

**Аннотация**

Курсовая работа подразумевает создание структуры данных для хранения последних N команд пользователя в терминале.

Было проведено исследование и выбрана лучшая реализация структуры данных из очереди, бинарного дерева и кольцевого буфера.

**содержание**

|  |  |  |
| --- | --- | --- |
|  | СОДЕРЖАНИЕ | 5 |
|  | ВВЕДЕНИЕ | 6 |
| 1. | ЗАДАНИЕ ВАРИАНТА | 7 |
| 2. | ИССЛЕДОВАНИЕ | 8 |
| 3. | РАЗРАБОТКА | 9 |
| 4. | ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ | 10 |
| 5. | ЗАКЛЮЧЕНИЕ | 12 |
| 6. | СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ | 13 |
| 7. | ПРИЛОЖЕНИЕ А. КОД ПРОГРАММЫ | 14 |

**введение**

Целью курсовой работы является разработка структуры данных для хранения последних N команд пользователя в терминале. Программа должна поддерживать следующие команды:

1. Добавление новой команды.

2. Изменение размеров очереди.

3. Вывод всех хранимых команд.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Изучить теоретическую информацию о существующих структурах данных и выбрать наиболее подходящую для применения в данной работе.

2. Выбрать способ реализации выбранной структуры данных и продумать логику работы программы.

3. Реализовать класс соответствующей структуры данных.

4. Протестировать программу для выявления ошибок.

**1. ЗАДАНИЕ ВАРИАНТА**

Вариант 4.4

Храним N последних команд пользователя в терминале (N небольшое, например, N<=20).

Должна быть возможность увеличить количество хранимых команд в данный момент времени или, наоборот, уменьшить это количество, а также вывести все хранящиеся команды.

Уточнение: если хранилось 10 последних команд, а далее количество хранимых команд было сокращено до 5, то все не попавшие в новое количество команды считаются утерянными и при повторном расширении добавлять их обратно не нужно.

**2. Исследование**

В исследовании участвовали очередь на базе бинарного дерева, кольцевой буфер и очередь на базе массива.

Очередь — абстрактный тип данных с дисциплиной доступа к элементам «первый пришёл — первый вышел» (FIFO, англ. first in, first out).

Кольцевой буфер, или циклический буфер (англ. ring-buffer) — это структура данных, использующая единственный буфер фиксированного размера таким образом, как будто бы после последнего элемента сразу же снова идет первый.

Массив — структура данных, хранящая набор значений (элементов массива), идентифицируемых по индексу или набору индексов, принимающих целые (или приводимые к целым) значения из некоторого заданного непрерывного диапазона.

Бинарное дерево — иерархическая структура данных, в которой каждый узел имеет не более двух потомков (детей).

Исследование представлено в таблице 1 в секундах.

Таблица 1.



**3. Разработка**

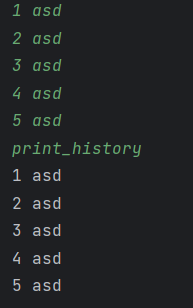
Класс CommandHistory:

* def \_\_init\_\_(self, max\_size=10) – инициализация очереди
* def add\_command(self, command) – добавление команды в очередь
* def set\_max\_size(self, new\_size) – установление максимального размера очереди
* def trim\_history(self) – удаление лишних команд
* def get\_history(self) – получение очереди
* def \_\_str\_\_(self) – вывод очереди

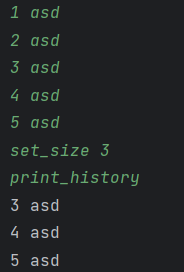
**4. ПРИмеры работы программы**

Программа выполняет все задачи согласно требованиям. В файле test.py хранятся тесты программы.

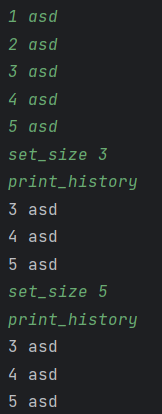
Пример вывод хранящихся команд:



Пример изменение размера в меньшую сторону:



Пример изменение размеров в большую сторону:



**заключение**

Программа успешно реализована и успешно выполняет поставленные задачи. В процессе выполнения работы был проведен анализ структур данных, была выбрана и реализована наиболее подходящая из них, очередь.

Программа поддерживает следующие операции:

1. добавление команды

2. изменение размеров очереди

3. вывод всей очереди

**список использованных источников**

1. Лекционные материалы // se.moevm.info URL: https://se.moevm.info/doku.php/courses:algorithms\_structures:start (дата обращения: 02.12.2024).

2. Кольцевой буфер // РУВИКИ. URL: https://ru.ruwiki.ru/wiki/Кольцевой\_буфер (дата обращения: 02.12.2024).

**приложение А**

**Код программы**

Название файла: CommandHistory.py

class CommandHistory:

def \_\_init\_\_(self, max\_size=10):

self.max\_size = max\_size

self.commands = []

def add\_command(self, command):

if len(self.commands) >= self.max\_size:

self.commands.pop(0) # Удаляем самую старую команду

self.commands.append(command)

def set\_max\_size(self, new\_size):

if new\_size < 0:

raise ValueError("Размер не может быть отрицательным")

self.max\_size = new\_size

self.trim\_history()

def trim\_history(self):

if len(self.commands) > self.max\_size:

self.commands = self.commands[-self.max\_size:]

def get\_history(self):

return self.commands

def \_\_str\_\_(self):

return "\n".join(self.commands)

Название файла: main.py

from modules.CommandHistory import CommandHistory

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

commandHistory = CommandHistory(100)

while True:

command = input()

if command == "print\_history":

print(commandHistory)

continue

elif command.split()[0] == "set\_size":

commandHistory.set\_max\_size(int(command.split()[1]))

continue

elif command == "break\_program\_history":

break

else:

commandHistory.add\_command(command)

Название файла: test.py

from modules.CommandHistory import CommandHistory

def test\_add\_command():

history = CommandHistory(5)

history.add\_command("ls")

history.add\_command("cd /home")

assert history.get\_history() == ["ls", "cd /home"], "test\_add\_command failed"

def test\_add\_command\_overflow():

history = CommandHistory(5)

for i in range(7):

history.add\_command(f"command{i}")

assert history.get\_history() == ["command2", "command3", "command4", "command5", "command6"], "test\_add\_command\_overflow failed"

def test\_set\_max\_size\_increase():

history = CommandHistory(5)

for i in range(5):

history.add\_command(f"command{i}")

history.set\_max\_size(7)

assert history.max\_size == 7, "test\_set\_max\_size\_increase failed"

assert history.get\_history() == ["command0", "command1", "command2", "command3", "command4"], "test\_set\_max\_size\_increase failed"

def test\_set\_max\_size\_decrease():

history = CommandHistory(5)

for i in range(5):

history.add\_command(f"command{i}")

history.set\_max\_size(3)

assert history.max\_size == 3, "test\_set\_max\_size\_decrease failed"

assert history.get\_history() == ["command2", "command3", "command4"], "test\_set\_max\_size\_decrease failed"

def test\_set\_max\_size\_zero():

history = CommandHistory(5)

for i in range(5):

history.add\_command(f"command{i}")

history.set\_max\_size(0)

assert history.max\_size == 0, "test\_set\_max\_size\_zero failed"

test\_add\_command()

test\_add\_command\_overflow()

test\_set\_max\_size\_increase()

test\_set\_max\_size\_decrease()

test\_set\_max\_size\_zero()