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

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.

объём	бинарное дерево		кольцевой буфер		очередь	
	добавление	ресайз	добавление	ресайз	добавление	ресайз
10	0.0009958744049072266	0.0	0.0	0.0	0.0	0.0
1000	0.05303239822387695	0.0010020732879638672	0.0009999275207519531	0.0009987354278564453	0.0	0.0
1000000	25.42512798309326	0.024342732879638672	0.17341852188110352	0.1554861068725586	0.1225895881652832	0.0
100000000	∞		19.42512798309326		∞	

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:

```
1 asd
2 asd
3 asd
4 asd
5 asd
print_history
1 asd
2 asd
3 asd
4 asd
5 asd
```

Пример изменение размера в меньшую сторону:

```
1 asd
2 asd
3 asd
4 asd
5 asd
set_size 3
print_history
3 asd
4 asd
5 asd
```

Пример изменение размеров в большую сторону:

```
1 asd
2 asd
3 asd
4 asd
5 asd
set_size 3
print_history
3 asd
4 asd
5 asd
set_size 5
print_history
3 asd
4 asd
5 asd
```

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()
```