МИНОБРНАУКИ РОССИИ

Санкт-Петербургский государственный

электротехнический университет

«ЛЭТИ» им. В.И. Ульянова (Ленина)

Кафедра МО ЭВМ

Отчет

по Курсовой работе

по дисциплине «Алгоритмы и структуры данных»
Тема: Поиск и использование оптимальной структуры данных для хранения последних команд

Студент гр. 3382 _____ Миллер С.С.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2024

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Миллер С.Е.

Группа 3382

Тема работы: Поиск и использование оптимальной структуры данных для хранения последних команд

Исходные данные:

Вариант 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..

Для выбора структуры данных нужно провести подробное исследование и зафиксировать его в отчете.(Т.е. По каким причинам была выбрана эта структура данных? Почему именно эта структура лучше в данном случае? Если у вас хэш-таблица, то почему именно такой метод решения коллизий был выбран? и т.д.) Простыми словами: необходимо привести аргументы, подтверждающие, что эта структура данных является лучшей для вашей ситуации.
Для демонстрации работы необходимо сделать промежуточные выводы работы вашей программы и визуализацию вашей структуры данных.

Предполагаемый объем пояснительной записки:
Не менее -- страниц.

Дата выдачи задания: 06.11.2024

Дата сдачи реферата: 15.12.2024

Дата защиты реферата: 17.12.2024

Студент _____ Миллер С.Е

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, двусвязного списка и кольцевого буфера.

SUMMARY

The course work involves creating a data structure to store the last N user commands in terminal. A study was conducted and the best implementation of the data structure from the queue, binary tree, linked list and ring buffer was selected.

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 команд

Для этого необходимо:

1. Изучить теоретическую информацию о существующих структурах данных и выбрать наиболее подходящую для применения в данной работе.
2. Продумать логику работы программы и способ реализации структуры.
3. Реализовать структуру данных.
4. Тестирование программы

ВЫБОР ОПТИМАЛЬНОЙ СТРУКТУРЫ ДАННЫХ

В исследование учувствовало несколько структур данных, а именно: очередь на базе бинарного дерева, очередь на базе массива, кольцевой буффер.

Из задания понятно, что мы должны использовать структуру данных

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).

Выбор пал на очередь реализованной на базе массива, так как это оптимальный выбор. Из выбранных на анализ структур данных, реализация очереди на базе массива более простая. Эта структура данных оптимальна по времени и памяти.

	Добавление элемента	Изменение размера
Кольцевой буфер	O(1)	O(n)
Бинарное дерево	O(log n)	O(log n)
очередь	O(1)	O(n)

ОПИСАНИЕ ФУНКЦИЙ

1. def __init__(self, max_size=10) – инициализация очереди
2. def add_command(self, command) – добавление команды в очередь

   Если количество команд превышает допустимый размер max_size, самая старая команда удаляется
3. def set_size(self, new_size) – установление максимального размера очереди

   Если новый размер меньше текущего, история обрезается с конца
4. def trim_history(self) – удаление лишних команд
5. def get_commands(self) – получение очереди

   Возвращает текущий список команд.

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:

- Добавление команды
- Изменение размера очереди
- Вывод всей очереди

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1) Онлайн-курс «Алгоритмы и Структуры Данных» // URL: https://e.moevm.info/course/view.php?id=45

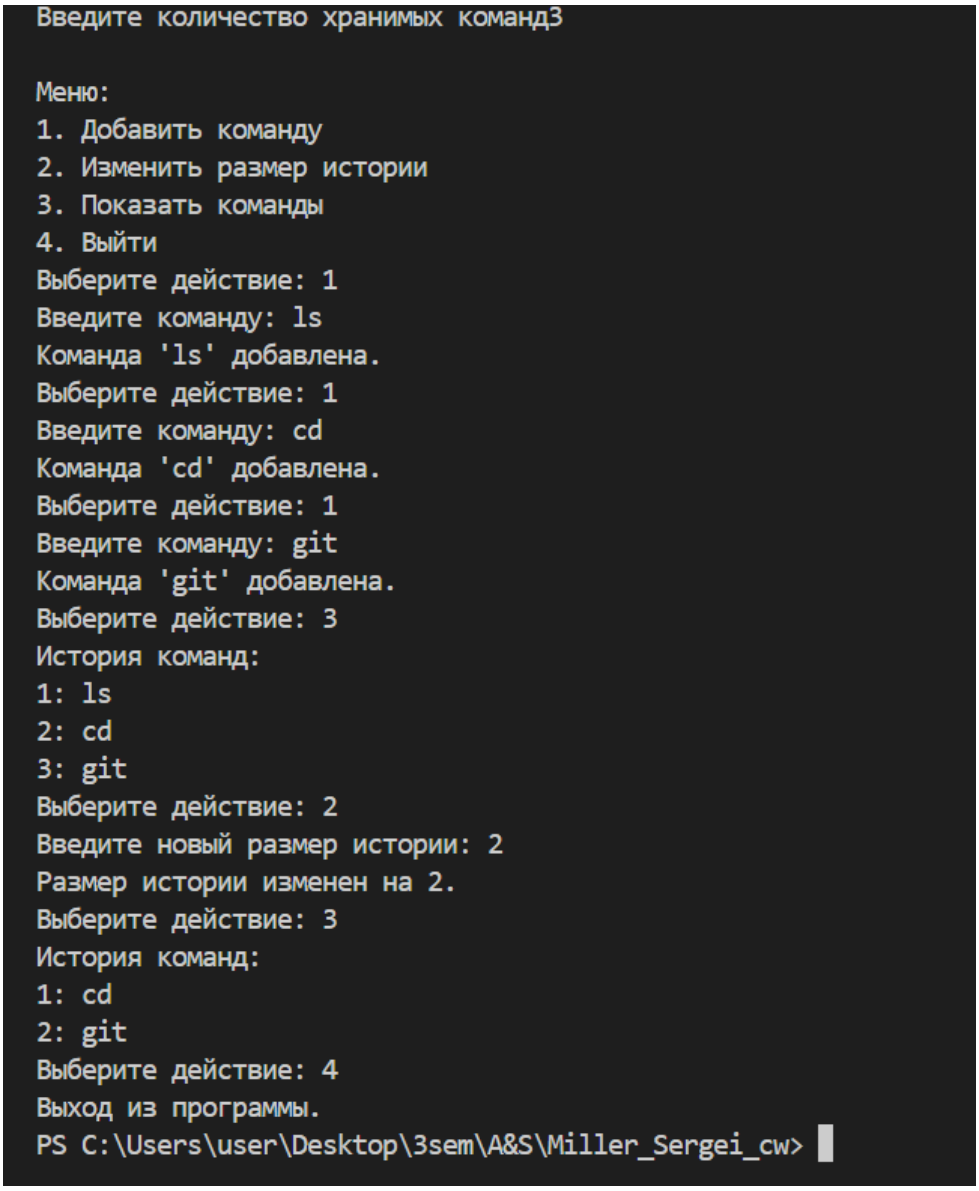
2) Кольцевой буффер // РУВИКИ. URL: https://ru.wikipedia.org/wiki/Кольцевой_буфер

3) Сайт se moevm info (методические указания по курсовым работам) // URL:https://docs.google.com/spreadsheets/d/1B68c3-GJX8Z5nXowrpTJVW2nCC0MgZkpuime0bDJN2A/edit?usp=sharin

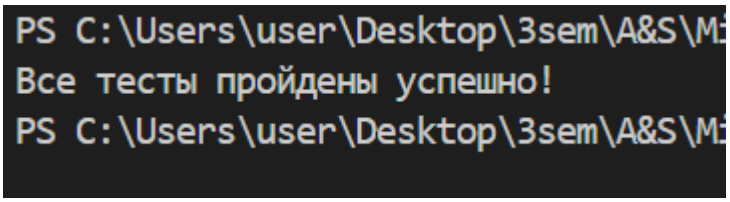
ПРИЛОЖЕНИЕ А

ТЕСТИРОВАНИЕ ПРОГРАММЫ

1) Пример работы программы.



2) Тестирование программы. Результат работы файла test.py



ПРИЛОЖЕНИЕ А
КОД ПРОГРАММЫ

Main.py

```
from ComandHistory import *


# Пример использования
if __name__ == "__main__":
    history_len = 20
    history_leын = input('Введите количество хранимых команд ')

    history = CommandHistory(history_len)

    print("\nМеню:")
    print("1. Добавить команду")
    print("2. Изменить размер истории")
    print("3. Показать команды")
    print("4. Выйти")
    while True:
        choice = input("Выберите действие: ")

        if choice == "1":
            command = input("Введите команду: ")
            history.add_command(command)
            print(f"Команда '{command}' добавлена.")

        elif choice == "2":
            try:
                new_size = int(input("Введите новый размер истории: "))
                history.set_size(new_size)
                print(f"Размер истории изменен на {new_size}.")
            except ValueError:
                print("Пожалуйста, введите корректное число.")

        elif choice == "3":
            commands = history.get_commands()
            print("История команд:")
            for i, cmd in enumerate(commands, start=1):
                print(f"{i}: {cmd}")

        elif choice == "4":
            print("Выход из программы.")
            break

        else:
            print("Неверный выбор. Пожалуйста, выберите действие из меню.")
```

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)
        self.commands.append(command)
```

```
    def set_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_commands(self):
        return self.commands
```

Название файла: test.py

```
from ComandHistory import *

def test_add_command():
    """Тест на добавление команд в историю."""
    history = CommandHistory(3)
    history.add_command("ls")
    history.add_command("cd /")
    history.add_command("mkdir test")
    assert history.get_commands() == ['ls', 'cd /', 'mkdir test'],
"test_add_command failed"

def test_exceeding_limit():
    """Тест на удаление старых команд при превышении лимита."""
    history = CommandHistory(2)
    history.add_command("ls")
    history.add_command("cd /")
    history.add_command("mkdir test")
    assert history.get_commands() == ["cd /", "mkdir test"],
"test_exceeding_limit failed"

def test_reduce_size():
    """Тест на уменьшение размера истории."""
    history = CommandHistory(3)
    history.add_command("ls")
    history.add_command("cd /")
    history.add_command("mkdir test")
    history.set_size(2)  # Изменение максимального размера истории
    assert history.get_commands() == ["cd /", "mkdir test"], "test_reduce_size
failed"

def test_increase_size():
    """Тест на увеличение размера истории."""
    history = CommandHistory(2)
    history.add_command("ls")
    history.add_command("cd /")
    history.set_size(4)  # Увеличение максимального размера
    history.add_command("mkdir test")
    history.add_command("touch file.txt")
    assert history.get_commands() == ["ls", "cd /", "mkdir test", "touch
file.txt"], "test_increase_size failed"

def test_empty_history():
    """Тест на пустую историю команд."""
    history = CommandHistory(3)
    assert history.get_commands() == [], "test_empty_history failed"
```

```
# Запуск тестов
test_add_command()
test_exceeding_limit()
test_reduce_size()
test_increase_size()
test_empty_history()
print("Все тесты пройдены успешно!")
```