Отчет по лабораторной работе № 25-26 по курсу “Алгоритмы и структуры данных ”

Студент группы М80-101Б-22, Шляхтуров Александр Викторович, № по списку 27

Контакты email: shliakhturov@gmail.com

Работа выполнена: «25» апреля 2023г.

Преподаватель: каф. 806 Крылов Сергей Сергеевич

Входной контроль знаний с оценкой _____

Отчет сдан « » _____202 __ г., итоговая оценка _____

Подпись преподавателя _____

1. **Тема:** Структуры данныхы
2. **Цель работы:** Научиться реализовывать структуры данных и алгоритмы сортировки
3. **Задание** Реализовать линейный список и алгоритм сортировки слиянием

4. **Оборудование**:

Оборудование ПЭВМ студента, если использовалось:

Процессор **AMD Ryzen 5 5500U 2.10 GHz, 6 ядер** с ОП **8192 Мб**, ТТН **512000 Мб**. Мониторы **Lenovo.**

5. **Программное обеспечение:**

Программное обеспечение ЭВМ студента, если использовалось:

Операционная система семейства **Linux**, наименование **Ubuntu** версия **20.04.5,** интерпретатор команд bash версия 5.0.17(1).

Система программирования CLion версия 2021.1.3

Редактор текстов nano версия 6.2

Утилиты операционной системы WinRar, Microsoft Word.

Прикладные системы и программы Ubuntu wsl, Clion, Google Chrome

Местонахождение и имена файлов программ и данных на домашнем компьютере /home/artur

6. **Идея, метод, алгоритм** решения задачи (в формах: словесной, псевдокода, графической [блок-схема, диаграмма, рисунок, таблица] или формальные спецификации с пред- и постусловиями)

Node.h

```
#ifndef NODE_H
#define NODE_H

class Node {
public:
    int data;
    Node* next;
    Node(int data);
};

#endif
```

Linkedlist.h

```
#ifndef LINKED_LIST_H
#define LINKED_LIST_H

#include <bits/stdc++.h>
```

```
using namespace std;

class LinkedList {
public:
    struct Node {
        int data;
        Node* next;
        Node(int data) : data(data), next(NULL) {}
    };

    Node* head;
    LinkedList();
    void insert(int data, int pos);
    void remove(int pos);
    void print();
    void merge_sort();

private:
    Node* merge_sort_rec(Node* node);
    Node* merge(Node* first, Node* second);
};

#endif
```

Node.cpp

```
#include "Node.h"
#include <bits/stdc++.h>
using namespace std;

Node::Node(int data) {
    this->data = data;
    next = NULL;
}
```

LINKEDLIST.cpp

```
#include <bits/stdc++.h>
using namespace std;

#include "LinkedList.h"
#include "Node.h"

LinkedList::LinkedList() {
    head = nullptr;
}

void LinkedList::insert(int data, int pos) {
    Node* new_node = new Node(data);
    if (head == NULL) {
        // если список пустой, добавляем элемент в начало
        head = new_node;
    }
    else if (pos == 0) {
```

```
        // если позиция равна 0, добавляем элемент в начало
        new_node->next = head;
        head = new_node;
    }
    else {
        // ищем элемент, предшествующий позиции вставки
        Node* prev = head;
        for (int i = 0; i < pos - 1 && prev->next != NULL; i++) {
            prev = prev->next;
        }
        // если позиция больше длины списка, добавляем элемент в конец
        if (prev->next == NULL) {
            prev->next = new_node;
        }
        else {
            // вставляем элемент между prev и prev->next
            new_node->next = prev->next;
            prev->next = new_node;
        }
    }
    }

void LinkedList::remove(int pos) {
    // проверяем, что список не пустой
    if (head == NULL) {
        return;
    }

    Node* to_delete;

    if (pos == 0) {
        // если удаляем первый элемент, просто обновляем указатель на голову списка
        to_delete = head;
        head = head->next;
    }
    else {
        // ищем элемент, предшествующий позиции удаления
        Node* prev = head;
        for (int i = 0; i < pos - 1 && prev->next != NULL; i++) {
            prev = prev->next;
        }
        // проверяем, что позиция не превышает размер списка
        if (prev->next == NULL) {
            return;
        }
        // удаляем элемент
        to_delete = prev->next;
        prev->next = to_delete->next;
    }

    // освобождаем память, выделенную для удаляемого элемента
    delete to_delete;
}
```

```
void LinkedList::print() {
     Node* curr = head;
        while (curr != NULL) {
            cout << curr->data << " ";
            curr = curr->next;
        }
        cout << endl;
    }

void LinkedList::merge_sort() {
    head = merge_sort_rec(head);
}

LinkedList::Node* LinkedList::merge_sort_rec(Node* node)  {
   if (node == NULL || node->next == NULL) {
            return node;
        }

        // разделяем список на две части
        Node* fast = node->next;
        Node* slow = node;
        while (fast != NULL) {
            fast = fast->next;
            if (fast != NULL) {
                slow = slow->next;
                fast = fast->next;
            }
        }
        Node* second_half = slow->next;
        slow->next = NULL;

        // сортируем каждую часть рекурсивно
        Node* first_half_sorted = merge_sort_rec(node);
        Node* second_half_sorted = merge_sort_rec(second_half);

        // объединяем две отсортированные части
        Node* merged_list = merge(first_half_sorted, second_half_sorted);
        return merged_list;
    }

LinkedList::Node* LinkedList::merge(Node* first, Node* second) {
    if (first == NULL) {
            return second;
        }
        if (second == NULL) {
            return first;
        }

        if (first->data < second->data) {
            first->next = merge(first->next, second);
            return first;
        }
```

```
        else {
            second->next = merge(first, second->next);
            return second;
        }
    }
```

MAIN.cpp

```
#include "Node.h"
#include <bits/stdc++.h>
using namespace std;
#include "LinkedList.h"
#include "Node.h"

int main() {
    void menu();
    cout << "mklist - создать список" << endl;
    cout << "q - выход" << endl;
    cout << "insert <value> <position> - добавить элемент value на позицию position" <<
endl;
    cout << "printlist - вывести список" << endl;
    cout << "mergesort - отсортировать слиянием" << endl;
    cout << "remove <position> - удалить элемент на позиции position" << endl << endl;
    string s;
    LinkedList list;
    int value;
    int position;
    int list_exist = 0;
    while(true) {

        cin >> s;

        // Создание списка
        if (list_exist == 0 && s == "mklist") {
            list_exist = 1;
            cout << "Список создан" << endl;
            continue;
        }
        if (list_exist && s == "mklist") {
            cout << "Список уже создан" << endl;
            continue;
        }

        // Добавление значения
        if (list_exist && s == "insert") {
            cin >> value;
            cin >> position;
            list.insert(value, position);
            cout << "Добавлен элемент " << value << " на позицию " << position  <<
endl;
            continue;}

        if (list_exist == 0 && s == "insert"){
            cout << "Список не создан"  << endl;
```

```
            continue;
            }

        // Удаление элемена
        if (list_exist && s == "remove") {
            cin >> position;
            list.remove(position);
            cout << "Удален элемент на позиции " << position  << endl;
            continue;}

        if (list_exist == 0 && s == "remove"){
            cout << "Список не создан"  << endl;
            continue;
            }

         // Вывод списка
        if (list_exist == 1  && s == "printlist"){
            cout << "Список: ";
            list.print();
            continue;
            }

        // if (list_exist == 0  && s == "printlist"){
        //     cout << "Список не создан"  << endl;
        //     continue;
        //     }

         // Сортировка списка
        if (list_exist == 1  && s == "mergesort"){
            list.merge_sort();
            cout << "Список отсортирован " << endl;
            continue;
            }
        if (list_exist == 0 && s == "mergesort"){
            cout << "Список не создан" << endl;
            continue;
            }

         // Пользователь захотел выйти
        if (s == "q") {
            break;
        }

    }
}
```

MAKEFILE

```
CC=g++
CFLAGS=-c -Wall
LDFLAGS=
SOURCES=main.cpp LinkedList.cpp Node.cpp
OBJECTS=$(SOURCES:.cpp=.o)
EXECUTABLE=program

all: $(SOURCES) $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
    $(CC) $(LDFLAGS) $(OBJECTS) -o $@

.cpp.o:
    $(CC) $(CFLAGS) $< -o $@

clean:
    rm -f $(OBJECTS) $(EXECUTABLE)
```

alexander@DESKTOP-KNBCFCI:~/makefile26$ ./program
mklist - создать список
q - выход
insert <value> <position> - добавить элемент value на позицию position
printlist - вывести список
mergesort - отсортировать слиянием
remove <position> - удалить элемент на позиции position

mklist
Список создан
insert 1 1
Добавлен элемент 1 на позицию 1
insert 3 2
Добавлен элемент 3 на позицию 2
insert 5 3
Добавлен элемент 5 на позицию 3
insert 4 4
Добавлен элемент 4 на позицию 4
printlist
Список: 1 3 5 4
remove 0
Удален элемент на позиции 0
printlist
Список: 3 5 4
mergesort
Список отсортирован
printlist
Список: 3 4 5

7. Распечатка протокола (подклеить листинг окончательного варианта программы с тестовыми примерами, подписанный преподавателем).

9. Дневник отладки должен содержать дату и время сеансов отладки и основные события (ошибки в сценарии и программе, нестандартные ситуации) и краткие комментарии к ним. В дневнике отладки приводятся сведения об использовании других ЭВМ, существенном участии преподавателя и других лиц в написании и отладке программы.

10. Замечания автора по существу работы

11. Выводы

Я научился преобразовывать деревья в выражения и работать с ними. Узнал про алгоритм сортировочной станции Дейкстры и то, как происходит работа с математическими выражениями.

Недочёты при выполнении задания могут быть устранены следующим образом: --

Подпись студента

_____