Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Российский химико-технологический университет имени Д.И. Менделеева»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

Выполнил студент группы КС-36: Золотухин А.А.
Ссылка на репозиторий: https://github.com/MUCTR-IKT-CPP/ZolotukhinAA_36_ALG
Принял: Крашенников Роман Сергеевич
Дата сдачи: 10.03.2025

Москва
2025

Оглавление

Описание задачи

Написать свою реализацию двусвязного списка:

- добавление элемента в начало, в конец, в произвольное место;
- удаление элемента из списка.

В рамках лабораторной работы необходимо изучить и реализовать двусвязный список. Структура должна:

- использовать шаблонный подход, обеспечивая работу контейнера с произвольными данными;
- реализовывать своё итератор, предоставляющий стандартный для языка механизм работы с ним (для *C++* это операции *++* и *!=*);
- обеспечивать работу стандартных библиотек и конструкции *for each*, если она есть в языке, если их нет, то реализовать собственную функцию, использующую итератор;
- обеспечивать проверку на пустоту и подсчёт количества элементов.

Для демонстрации работы структуры необходимо создать набор тестов (под тестом понимается функция, которая создаёт структуру, проводит операцию или операции над структурой и удаляет структуру):

- заполнение контейнера 1000 целыми числами в диапазоне от -1000 до 1000 и подсчёт их суммы, среднего, минимального и максимального;
- провести проверку работы операций вставки и изъятия элементов на коллекции из 10 строковых элементов;
- заполнение контейнера 100 структур, содержащих фамилию, имя, отчество и дату рождения (от 01.01.1980 до 01.01.2020). Значения каждого поля генерируются случайно из набора заранее заданных. После заполнения необходимо найти всех людей младше 20 лет и старше 30 и создать новые структуры, содержащие результат фильтрации, проверить выполнение на правильность подсчётом количества элементов, не подходящих под условие в новых структурах.

Тесты:

1. перемешать все элементы;
2. выполнить серию тестирования сортировки из первой лабораторной на реализованном списке и сравнить производительность с полученной на массиве.

Описание метода/модели

Двусвязный список - это двунаправленный список, в котором каждый узел имеет два указателя: на следующий и предыдущий узлы, которые ссылаются на следующий и предыдущий узлы соответственно. В отличие от односвязаного списка, в котором каждый узел указывает только на следующий узел, в двусвязном списке есть дополнительный предыдущий указатель, который позволяет перемещаться как вперёд, так и назад.

Каждый узел двусвязного списка состоит из трёх полей:

- *data* - значение, хранящееся в узле;
- *next* - ссылка на следующий узел в списке;
- *prev* - ссылка на предыдущий узел в списке.

Анализ сложности основных операций над двусвязным списком:

- **Вставка в начало**: $O(1)$;
- **Вставка в конец**: $O(1)$;
- **Вставка в опредeнный узел**: $O(n)$;
- **Удаление в начале**: $O(1)$;
- **Удаление в конце**: $O(1)$;
- **Удаление в опредeнном узле**: $O(n)$;

Преимущества:

- Позволяет перемещаться как вперёд, так и назад;
- Удаление узла выполняется более эффективно и просто, поскольку у него есть указатель на предыдущий узел;
- Является динамическим по своей природе, поэтому он может увеличиваться и уменьшаться в размерах.

Недостатки:

- Для каждого узла требуется больше памяти, чем для массивов, из-за дополнительного хранилища, используемого для указателей;
- Его сложнее реализовать и поддерживать по сравнению с односвязным списком;
- Нужно пройти от головного узла к определённому узлу для вставки и удаления в определённых местах.

Выполнение задачи

Двусвязный список реализован на языке *C++*. Построение графиков с помощью программы *GNUplot*.

"main" функция работает с вызовом методов для выполнения задания

```
int main() {
  srand(time(0));

  testIntegers();
  testStrings();
  testPersons();
  testShuffle();
  simulation();

  return 0;
}

```

"Node" класс

```
template <typename T>
class Node {
  public:
    T data;
    Node<T> *prev;
    Node<T> *next;

    Node(const T &value) : data(value), prev(nullptr), next(nullptr) {}
};

```

"LinkedList" класс

```
template <typename T>
class LinkedList {
  protected:
    int size;
    Node<T> *head;
    Node<T> *tail;

  public:
    LinkedList() : size(0), head(nullptr), tail(nullptr) {}

    void insertFront(const T &value) {
      Node<T> *newNode = new Node<T>(value);
      if (head == nullptr) {
        head = newNode;
        tail = newNode;
      } else {
        newNode->next = head;
        head->prev = newNode;
        head = newNode;
      }

      size++;
    }

    bool isEmpty() const { return size == 0; }
```

```
26
27        int getSize() const { return size; }
28
29        class Iterator {
30          private:
31            Node<T> *current;
32
33          public:
34            Iterator(Node<T> *node) : current(node) {}
35
36            T& operator*() const { return current->data; }
37
38            Iterator& operator++() {
39              if (current)
40                current = current->next;
41
42              return *this;
43            }
44
45            Iterator& operator--() {
46              if (current)
47                current = current->prev;
48
49              return *this;
50            }
51
52            bool operator!=(const Iterator &other) const { return current != other.
    current; }
53        };
54
55        Iterator begin() const { return Iterator(head); }
56
57        Iterator end() const { return Iterator(nullptr); }
58    };
59
```

"DoublyLinkedList" класс

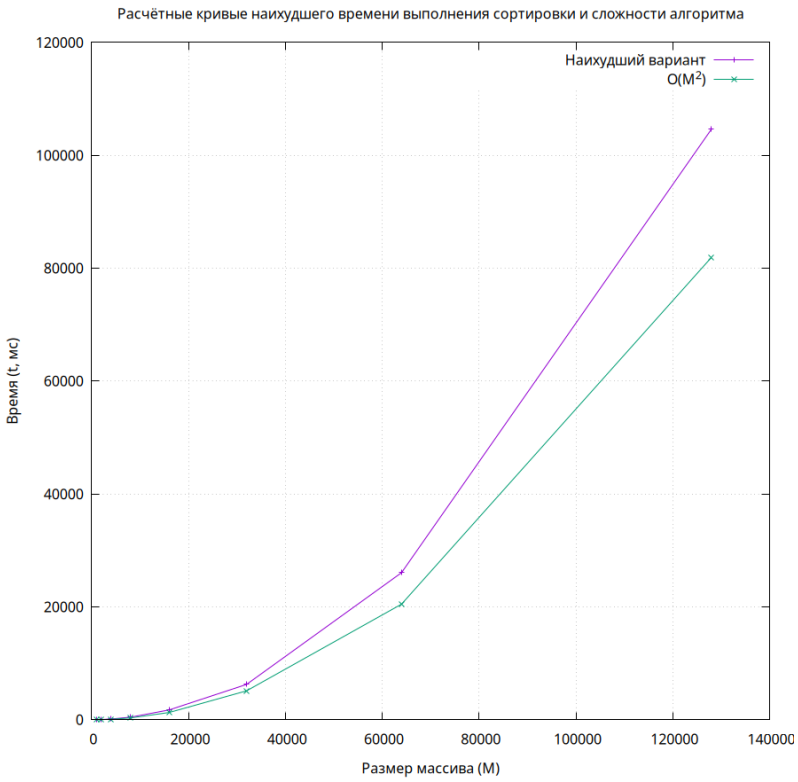
```
1     template <typename T>
2     class DoublyLinkedList : public LinkedList<T> {
3       public:
4         void insertEnd(const T& value) {
5           Node<T> *newNode = new Node<T>(value);
6           if (this->head == nullptr) {
7             this->head = newNode;
8             this->tail = newNode;
9           } else {
10            newNode->prev = this->tail;
11            this->tail->next = newNode;
12            this->tail = newNode;
13          }
14
15          this->size++;
16        }
17
18        void insertAtPosition(const T &value, int position) {
19          if (position < 0 || position > this->size)
20            throw std::out_of_range("The position is out of range.");
21
22          if (position == 0)
```

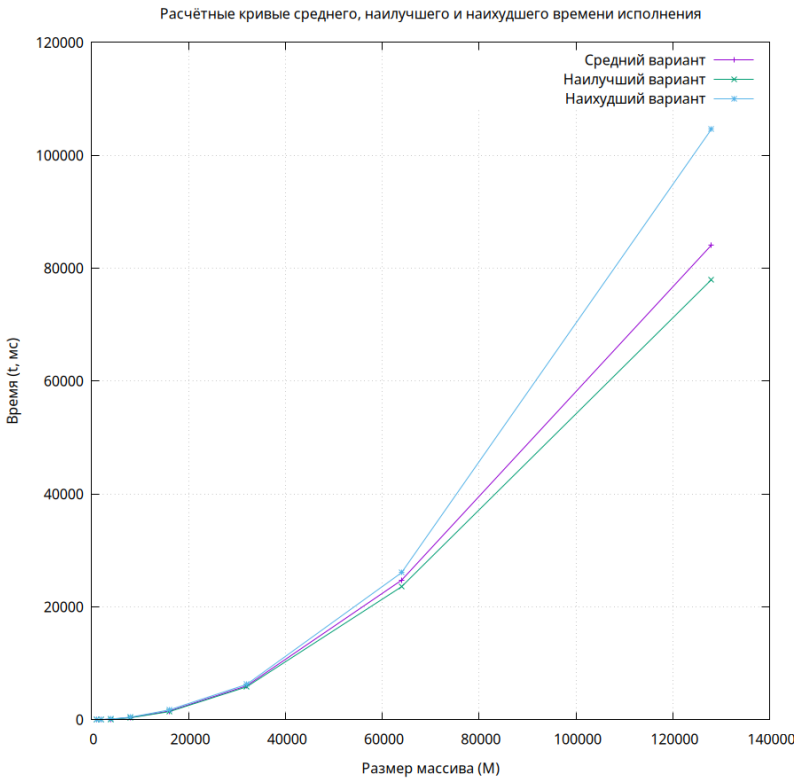
```
23            this->insertFront(value);
24          else if (position == this->size)
25            this->insertEnd(value);
26          else {
27            Node<T> *newNode = new Node<T>(value);
28            Node<T> *current = this->head;
29
30            for (int i = 0; i < position - 1; i++)
31              current = current->next;
32
33            newNode->next = current->next;
34            newNode->prev = current;
35            current->next->prev = newNode;
36            current->next = newNode;
37
38            this->size++;
39          }
40        }
41
42        void deleteNode(const T &value) {
43          Node<T> *current = this->head;
44          while (current != nullptr) {
45            if (current->data == value) {
46              if (current->prev != nullptr)
47                current->prev->next = current->next;
48              else
49                this->head = current->next;
50
51              if (current->next != nullptr)
52                current->next->prev = current->prev;
53              else
54                this->tail = current->prev;
55
56              delete current;
57
58              this->size--;
59
60              break;
61            }
62
63            current = current->next;
64          }
65        }
66
67        void shuffle() {
68          if (this->size <= 1)
69            std::cerr << "There isn't anything to shuffle!";
70
71          std::vector<T> elements;
72          for (typename DoublyLinkedList<T>::Iterator it = this->begin(); it != this
    ->end(); ++it)
73            elements.push_back(*it);
74
75          std::random_device rd;
76          std::mt19937 gen(rd());
77          std::shuffle(elements.begin(), elements.end(), gen);
78
79          this->clear();
80          for (const T &element : elements)
```

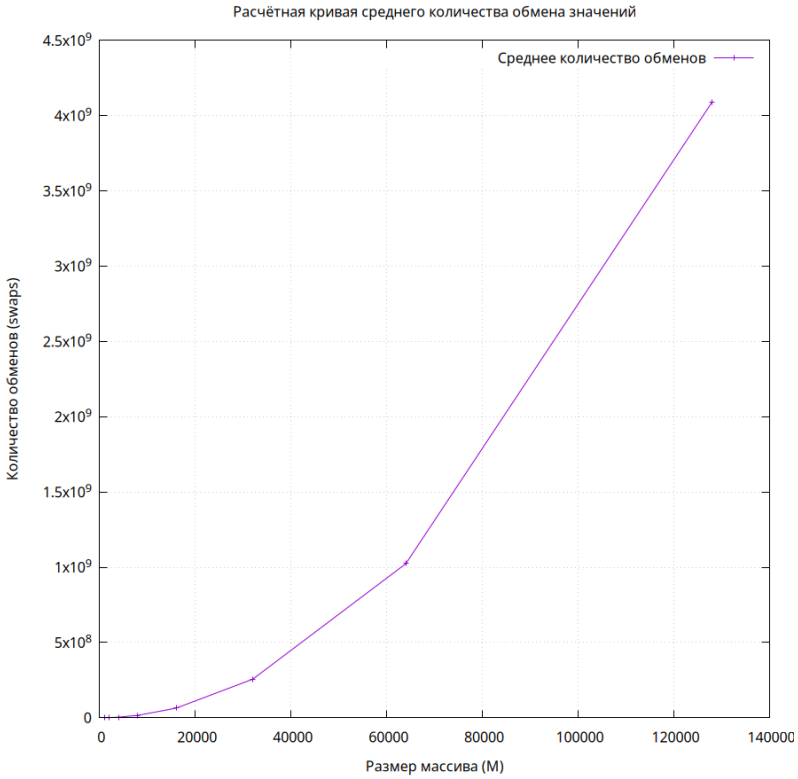
```
            this->insertEnd(element);
        }

        void clear() {
          while (this->head != nullptr) {
            Node<T>* temp = this->head;
            this->head = this->head->next;
            delete temp;
          }
          this->tail = nullptr;
          this->size = 0;
        }
    };

```

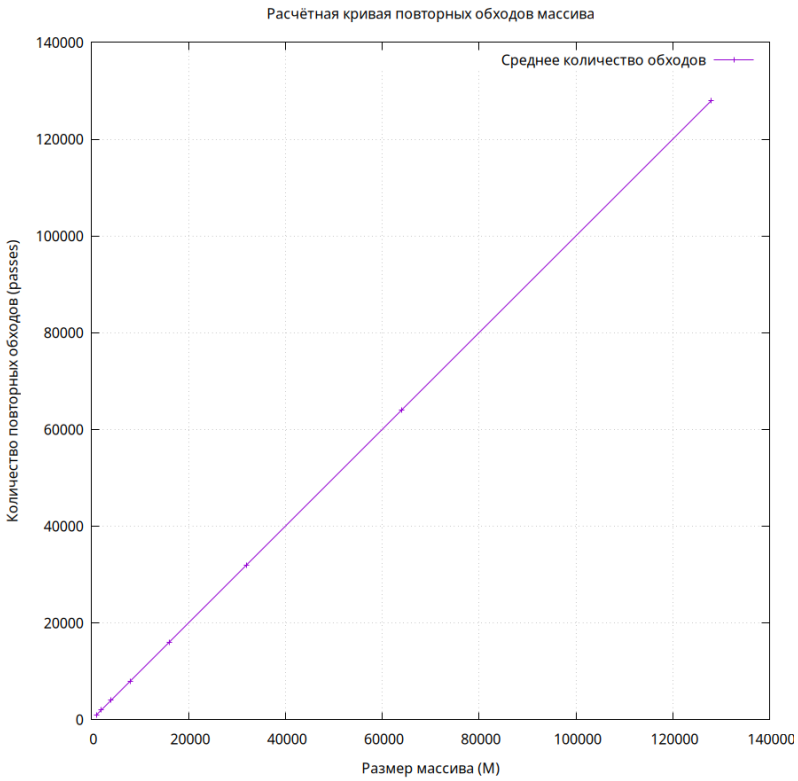
"testIntegers" функция

```
    void testIntegers() {
      DoublyLinkedList<int> list;

      int sum = 0;
      int minValue = 1001;
      int maxValue = -1001;

      for (int i = 0; i < 1000; i++) {
        int value = getRandomNumber(-1000, 1000);
        list.insertEnd(value);
        sum += value;

        if (value < minValue) minValue = value;
        if (value < maxValue) maxValue = value;
      }

      double average = static\_cast<double>(sum) / list.getSize();

      std::cout << "Test: Integers" << std::endl;
      std::cout << "Sum: " << sum << std::endl;
      std::cout << "Average: " << average << std::endl;
      std::cout << "Min: " << minValue << std::endl;
      std::cout << "Max: " << maxValue << std::endl;
    }

```


Расчётные кривые наихудшего времени выполнения сортировки и сложности алгоритма
Наихудший вариант
O(M2)
Время (t, мс)
Размер массива (M)
0
20000
40000
60000
80000
100000
120000
140000


Расчётные кривые среднего, наилучшего и наихудшего времени исполнения
Средний вариант
Наилучший вариант
Наихудший вариант
Время (t, мс)
Размер массива (M)
0
20000
40000
60000
80000
100000
120000
140000


Расчётная кривая среднего количества обмена значений
Среднее количество обменов
Количество обменов (swaps)
Размер массива (M)
0
5x10^8
1x10^9
1.5x10^9
2x10^9
2.5x10^9
3x10^9
3.5x10^9
4x10^9
4.5x10^9
0
20000
40000
60000
80000
100000
120000
140000


Расчётная кривая повторных обходов массива
Среднее количество обходов
Количество повторных обходов (passes)
Размер массива (M)
0
20000
40000
60000
80000
100000
120000
140000

Выводы

Дву