МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра САПР

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Алгоритмы и структуры данных»

Тема: «Списки»

Вариант №17

Студент гр. 2302 _____ Фролов А. Э.

Преподаватель: _____ Пестерев Д. О.

Санкт-Петербург

2023

1 Постановка задачи

Реализовать объект в виде двусвязного списка с набором методов/функций. Данные, хранящиеся в списке, имеют целочисленный тип int.

2 Описание реализуемого класса и методов

Двусвязный список реализован с помощью класса TwoWaysList. Структура Node является узлом (шаблоном элемента) этого списка. Указатели head и tail указывают на первый и последний элементы списка соответственно. Ниже приведено описание методов списка:

`void push_back(int digit)` - Добавление элемента в конец списка;

`void push_front(int digit)` - Добавление элемента в начало списка;

`void delete_last()` - Удаление последнего элемента;

`void delete_first()` - Удаление первого элемента;

`void push_element_by_index(int digit, int index)` - Добавление элемента по индексу;

`Node* get_element_by_index(int pos)` - Получение элемента по индексу;

`void delete_element_by_index(int pos)` - Удаление элемента по индексу;

`unsigned get_size()` - Получение размера списка;

`void clear()` - Удаление всех элементов списка;

`void set_element_by_index(int digit, int pos)` - Замена элемента по индексу на передаваемый элемент;

`bool is_clear()` - Проверка на пустоту списка;

`void reverse()` - Изменение порядка элементов на обратный;

`void push_list_by_index(TwoWayList list, int index)` - Вставка другого списка в список, начиная с индекса;

`void push_list_back(TwoWayList list)` - Вставка другого списка в конец;

`void push_list_beg(TwoWayList list)` - Вставка другого списка в начало;

`bool is_list_in(TwoWayList list)` - Проверка на содержание другого списка в списке;

`int find_first_inclusion(TwoWayList list)` - Поиск первого вхождения другого списка в список;

`int find_last_inclusion(TwoWayList list)` - Поиск последнего вхождения другого списка в список;

`void permutation(int ind1, int ind2)` - Обмен двух элементов списка по индексам = Θ (max(index1, index2));

3 Оценка временной сложности каждого метода

Ниже приведена оценка временной сложности для каждого метода с доказательствами для некоторых из них. В качестве кол-ва элементов будет использована переменная **size**, а для обозначения индекса переменная **index**.

1) Добавление элемента в конец списка = Θ(1);

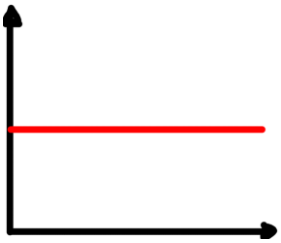
Докажем. На рисунке ниже показано количество операций, затрачивающихся в ходе работы метода:

```
void push_back(int digit)
{
    if (tail == nullptr) {      // 1 операция
        tail = new Node;
        tail->digit = digit;    } 3 операции
        head = tail;
    }
    else {
        tail->next = new Node;
        tail->next->digit = digit;  } 4 операции
        tail->next->prev = tail;
        tail = tail->next;
    }
    size++;   // 1 операция
}
```

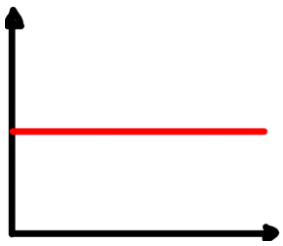
Таким образом, в среднем временная сложность алгоритма:

$$T(n) = 1 + \left(\frac{(3+4)}{2}\right) + 1 = \Theta(1)$$

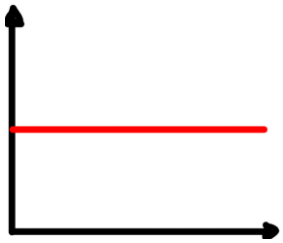
График:



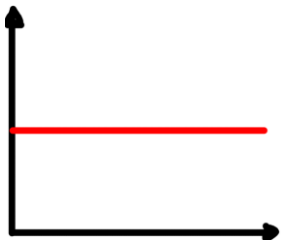
2) Добавление элемента в начало списка = Θ(1);



3) Удаление последнего элемента = Θ(1);



4) Удаление первого элемента = Θ(1);



5) Добавление элемента по индексу = Θ(index);

Докажем. Ниже показано кол-во операций.

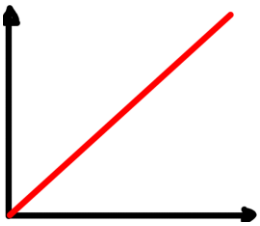
```
void push_element_by_index(int digit, int index) {
    if (head == nullptr) {
        head = new Node;
        head->digit = digit;
        tail = head;
    }
    else if (index == 0) {
        Node* tmp = head;
        head->prev = new Node;
        head->prev->digit = digit;
        head = head->prev;
        head->next = tmp;
    }
    else if (index == size) {
        Node* tmp = tail;
        tail->next = new Node;
        tail->next->digit = digit;
        tail = tail->next;
        tail->prev = tmp;
    }
    else {
        int i = 1;
        Node* tmp = head;
        while (i != index) {
            i++;
            tmp = tmp->next;
        }
        Node* tmp2 = tmp->next;
        tmp->next = new Node;
        tmp->next->digit = digit;
        tmp2->prev = tmp->next;
        tmp->next->prev = tmp;
        tmp->next->next = tmp2;
    }
    size++;
}
```

4 6 6 2*index +8 1

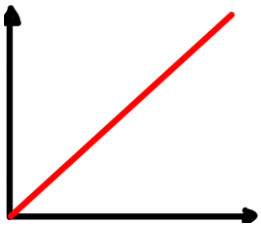
Таким образом, в среднем временная сложность алгоритма:

$$T(n) = 1 + \left(\frac{4 + 6 + 6 + (2 * index + 8)}{4}\right) = 7 + \frac{index}{2} = \Theta(index)$$

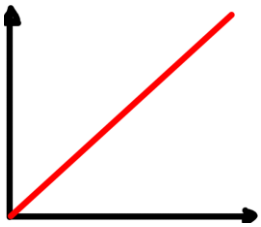
График:



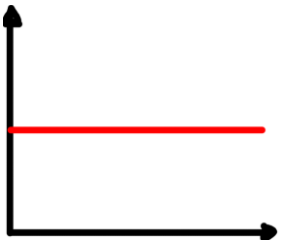
6) Получение элемента по индексу = Θ(index);



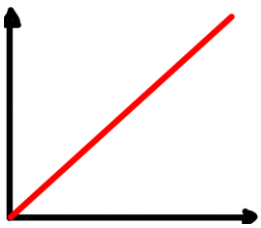
7) Удаление элемента по индексу = Θ(index);



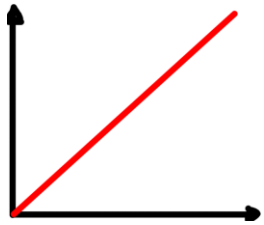
8) Получение размера списка = Θ(1);



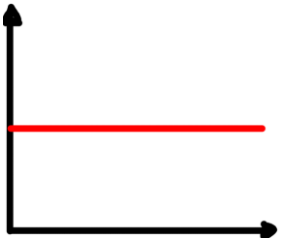
9) Удаление всех элементов списка = Θ(size);



10) Замена элемента по индексу на передаваемый элемент = Θ(index);



11) Проверка на пустоту списка = Θ(1);



12) Изменение порядка элементов на обратный = Θ(size);

Докажем. Ниже показано кол-во операций:

```
void reverse() {
    if (size > 1) {
        Node* tmp1 = head;
        Node* tmp2 = tail;
        int tmp;
        if (size % 2 == 0) {
            do {
                tmp = tmp1->digit;
                tmp1->digit = tmp2->digit;
                tmp2->digit = tmp;

                tmp1 = tmp1->next;
                tmp2 = tmp2->prev;
            }
            while (tmp2->next != tmp1);
        }
        else {
            do {
                tmp = tmp1->digit;
                tmp1->digit = tmp2->digit;
                tmp2->digit = tmp;

                tmp1 = tmp1->next;
                tmp2 = tmp2->prev;
            } while (tmp1 != tmp2);
        }
    }
}
```

4

1

$5 * \left(\frac{size}{2}\right)$

$5 * int\left(\frac{size}{2}\right)$

, где int – взятие целой части числа.

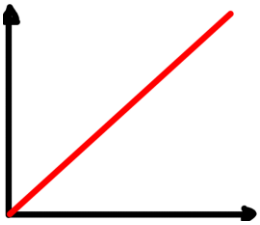
Таким образом, средняя временная сложность равна:

$$T(n) = 5 + \left(\frac{5 * \left(\frac{size}{2}\right) + 5 * int\left(\frac{size}{2}\right)}{2}\right)$$

Операцией int() мы можем пренебречь, тогда:

$$T(n) = \frac{25(\left(\frac{size}{2}\right) + \left(\frac{size}{2}\right))}{2} = \frac{25}{4} * size = \Theta(\text{size});$$

График:



13) Вставка другого списка в список, начиная с индекса = Θ(index);

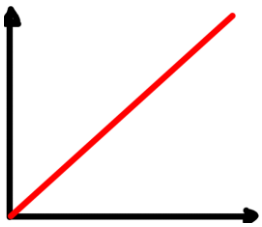
Докажем. Рассмотрим код:

```
void push_list_by_index(TwoWayList* list, int index) {
    if (index == 0) push_list_beg(list);
    else if (index == size) push_list_back(list);
    else {
        int ind = 1;
        Node* tmp = head;
        while (ind != index) {
            ind++;
            tmp = tmp->next;
        }
        tmp->next->prev = list->get_tail();
        list->get_tail()->next = tmp->next;
        tmp->next = list->get_head();
        list->get_head()->prev = tmp;
        size += list->get_size();
    }
}
```

Функции push_list_back() и push_list_beg() работают за константное время. Код в блоке else {} содержит обычные операции и цикл while, работающий (index - 1) раз. Т.к. при большом размере списка практически всегда будет исполняться код из блока else {}, то средняя временная сложность алгоритма:

$$T(n) = \Theta(\text{index})$$

График:



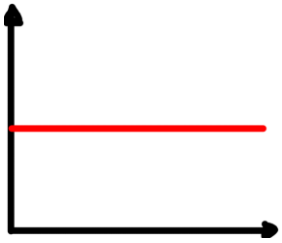
14) Вставка другого списка в конец = Θ(1);

```
void push_list_back(TwoWayList* list) {
    tail->next = list->get_head();
    list->get_head()->prev = tail;
    tail = list->get_tail();
    size += list->get_size();
}
```

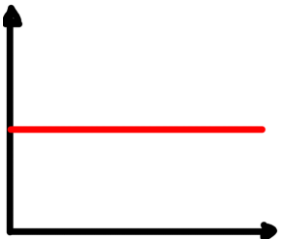
Очевидно, что цикл содержит 4 операции (т.к. функции get_head() и get_tail() просто возвращают элемент), а значит работает за константное время.

$$T(n) = \Theta(1)$$

График:



15) Вставка другого списка в начало = Θ(1);



16) Проверка на содержание другого списка в списке = Θ(size * list.get_size());

Докажем:

```
bool is_list_in(TwoWayList list) {
    int index = 0, index_tmp = 0;
    int index2 = 1;
    while (index < size) {
        if (get_element_by_index(index)->digit == list.get_element_by_index(0)->digit) {
            if (list.get_size() == 1) return 1;

            index_tmp = index++;
            while (index < size) {
                if (get_element_by_index(index)->digit == list.get_element_by_index(index2)->digit) {
                    index++;
                    index2++;
                    if (index2 == list.get_size()) return 1;
                }
                else {
                    index = index_tmp;
                    index2 = 1;
                    break;
                }
            }
        }
        index++;
    }

    return 0;
}
```

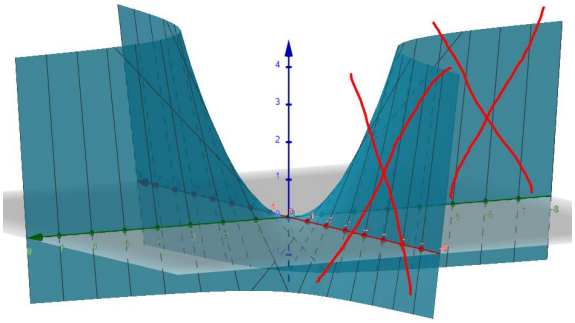
Алгоритм работает следующим образом: проходится по всем элементам основного списка, в случае совпадения текущего элемента с первым элементам проверяемого списка мы проверяем и последующие элементы. В случае, если список оказался найден, алгоритм завершается, а иначе возвращается к символу, следующему за тем, с которого начиналась проверка на наличие проверяемого списка.

Таким образом в лучшем случае алгоритм работает list.get_size() раз, а в худшем – **size * (list.get_size() - 1)**.

Для среднего случая можно считать, что алгоритм останавливается не на каждом элементе, но на оценку временной сложности это не повлияет:

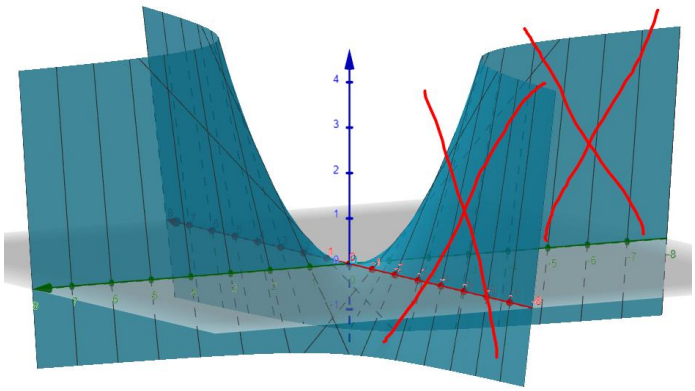
$$T(n) = \Theta(\text{size} * \text{list.get}(\text{size}))$$

График:

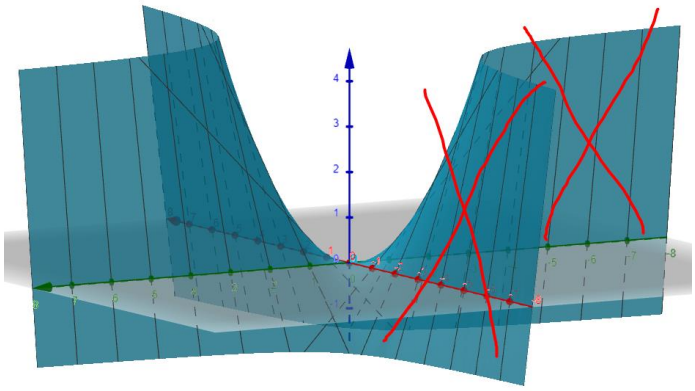


Как мы можем заметить, при росте обоих параметров функция растет параболически, а иначе линейно.

17) Поиск первого вхождения другого списка в список = Θ(size * list.get_size());



18) Поиск последнего вхождения другого списка в список = Θ(size * list.get_size());



19) Обмен двух элементов списка по индексам = Θ (index1 * index2);

Докажем. Рассмотрим код:

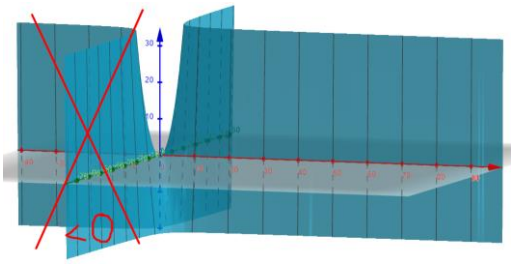
```
    void permutation(int ind1, int ind2) {
        Node* tmp1 = get_element_by_index(ind1);
        Node* tmp2 = get_element_by_index(ind2);
        int tmp = tmp1->digit;
        tmp1->digit = tmp2->digit;
        tmp2->digit = tmp;
    }
};
```

В методе дважды используется алгоритм `get_element_by_index()`, временная сложность которого равна Θ(index). Поэтому временная сложность данного алгоритма:

$$T(n) = \Theta(\text{index1} * \text{index2})$$

, где index1 и index2 – индексы первого и второго элемента соответственно.

График:



4 Пример работы программы.

```
Выберите одну из возможных комманд:
        1) добавление элемента в конец списка;
        2) добавление элемента в начало списка;
        3) удаление последнего элемента;
        4) удаление первого элемента;
        5) добавление элемента по индексу;
        6) получение элемента по индексу;
        7) удаление элемента по индексу;
        8) получение размера списка;
        9) удаление всех элементов списка;
        10) замена элемента по индексу на передаваемый элемент;
        11) проверка на пустоту списка;
        12) изменение порядка элементов на обратный;
        13) вставка другого списка в список, начиная с индекса;
        14) вставка другого списка в конец;
        15) вставка другого списка в начало;
        16) проверка на содержание другого списка в списке;
        17) поиск первого вхождения другого списка в список;
        18) поиск последнего вхождения другого списка в список;
        19) обмен двух элементов списка по индексам.
Ваш выбор: 15

Введите размер списка, который хотите добавить: 3
        Введите 1 элемент списка: 344
        Введите 2 элемент списка: 242
        Введите 3 элемент списка: 64

Введите индекс, начиная с которого будет начинаться новый список (от 0 до 0): 0
Элементы текущего списка:
344 242 64

Хотите продолжить? (да/нет)
Ваш ответ:
```

Рисунок 1 – Контрольный пример №1.

```
        2) добавление элемента в начало списка;
        3) удаление последнего элемента;
        4) удаление первого элемента;
        5) добавление элемента по индексу;
        6) получение элемента по индексу;
        7) удаление элемента по индексу;
        8) получение размера списка;
        9) удаление всех элементов списка;
        10) замена элемента по индексу на передаваемый элемент;
        11) проверка на пустоту списка;
        12) изменение порядка элементов на обратный;
        13) вставка другого списка в список, начиная с индекса;
        14) вставка другого списка в конец;
        15) вставка другого списка в начало;
        16) проверка на содержание другого списка в списке;
        17) поиск первого вхождения другого списка в список;
        18) поиск последнего вхождения другого списка в список;
        19) обмен двух элементов списка по индексам.
Ваш выбор: 18

Введите размер списка, который хотите найти: 2
        Введите 1 элемент списка: 1
        Введите 2 элемент списка: 2

Индекс последнего вхождения: 1
Элементы текущего списка:
2 1 2 3

Хотите продолжить? (да/нет)
Ваш ответ:
```

Рисунок 2 – Контрольный пример №2.

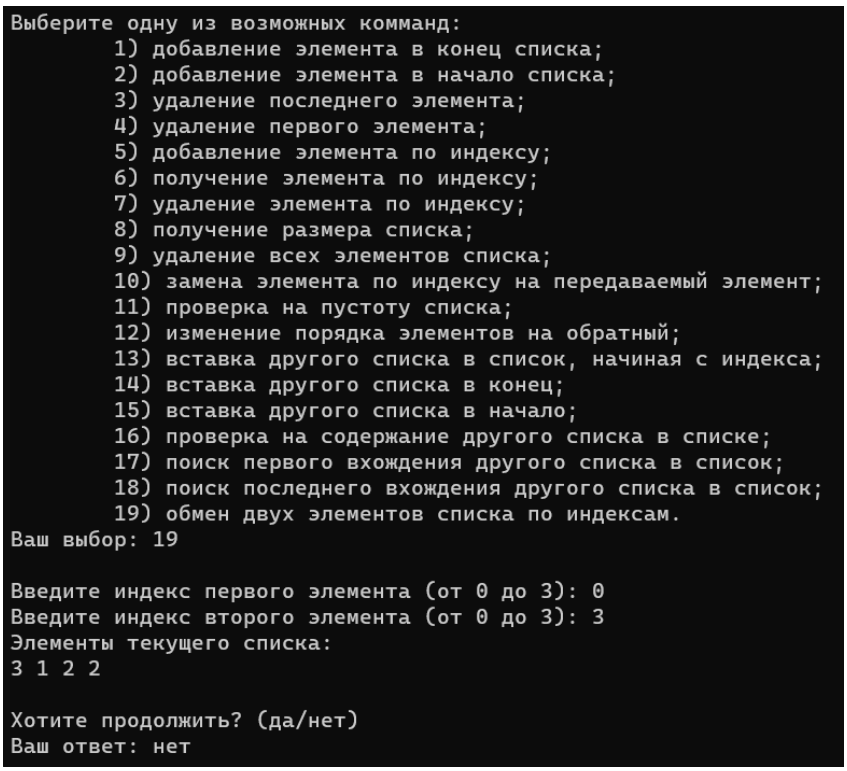


Рисунок 3 – Контрольный пример №3.

5 Листинг

Программа представлена на языке C++.

```
#include <iostream>
#include <Windows.h>
#include <string>
using namespace std;

int GetNumberOfCommand();

struct Node
{
	int digit;
	Node* next = nullptr;
	Node* prev = nullptr;
};

class TwoWayList
{
	Node* head, * tail;
	unsigned size;
public:
	TwoWayList() {
		head = tail = nullptr;
		size = 0;
```

```
}
Node* get_head() { return head; }
Node* get_tail() { return tail; }
void call_func_by_number(int i) {
    int digit, pos, count, j;
    string str;

    if (i == 1 || i == 2) {
            while (1) {
                    try {
                            cout << "Enter the number you want to add: ";
                            cin >> str;
                            digit = stoi(str);
                            if (i == 1) push_back(digit);
                            else push_front(digit);
                            out_list();
                            break;
                    }
                    catch (const exception& e) {
                            cout << "Incorrect number entered.\n";
                    }
            }
    }
    else if (i == 3 || i == 4) {
            if (size == 0) cout << "There if no elements in the list.\n";
            else {
                    if (i == 3) delete_last();
                    else delete_first();
                    out_list();
            }
    }
    else if (i == 5) {
            while (1) {
                    try {
                            cout << "Enter the number you want to add: ";
                            cin >> str;
                            digit = stoi(str);
                            break;
                    }
                    catch (const exception& e) {
                            cout << "Incorrect digit entered.\n";
                    }
            }

            while (1) {
                    try {
```

```
                cout << "Enter the index (from 0 to " << size << "): ";
                cin >> str;
                pos = stoi(str);
                if (pos < 0 || pos > size) throw exception();
                else break;
            }
            catch (const exception& e) {
                cout << "Incorrect digit entered.\n";
            }
        }
        push_element_by_index(digit, pos);
        out_list();
    }
    else if (i == 6 || i == 7 || i == 10) {
        if (size == 0) cout << "There is no elements in the list.\n";
        else {
            while (1) {
                try {
                    cout << "Enter the index " <<
                        "(from 0 to " << (size - 1) << ") : ";
                    cin >> str;
                    pos = stoi(str);
                    if (pos < 0 || pos >= size) throw exception();
                    else break;
                }
                catch (const exception& e) {
                    cout << "Incorrect digit entered.\n";
                }
            }
            if (i == 6)
                cout << "The element you got: " << get_element_by_index(pos)->digit <<
endl;
            else if (i == 7) delete_element_by_index(pos);
            else {
                while (1) {
                    try {
                        cout << "Enter the new number: ";
                        cin >> str;
                        digit = stoi(str);
                        set_element_by_index(digit, pos);
                        break;
                    }
                    catch (const exception& e) {
                        cout << "Incorrect digit entered.\n";
                    }
                }
```

```
                }
                out_list();
        }
}
else if (i == 8) {
        cout << "The size of the current list: " << get_size() << endl;
}
else if (i == 9) {
        clear();
        cout << "The list is cleared.\n";
        out_list();
}
else if (i == 11) {
        if (is_clear()) cout << "The list is empty.\n";
        else cout << "The list is not empty.\n";
}
else if (i == 12) {
        if (size == 0) cout << "There is no elements in the list.\n";
        else {
                reverse();
                out_list();
        }
}
else if (i ==  13 || i == 14 || i == 15 || i == 16 || i == 17 || i == 18) {
        while (1) {
                try {
                        if (i == 16)
                                cout << "Enter the size of the list you want to check for: ";
                        else
                                cout << "Enter size of your list: ";
                        cin >> str;
                        count = stoi(str);
                        break;
                }
                catch (const exception& e) {
                        cout << "Incorrect digit entered.\n";
                }
        }

        TwoWayList NewList;
        j = 1;
        while (j < (count + 1)) {
                while (1) {
                        try {
                                cout << "\tEnter " << j << " element of your list: ";
```

```
                                        cin >> str;
                                        digit = stoi(str);
                                        NewList.push_back(digit);
                                        break;
                                }
                                catch (const exception& e) {
                                        cout << "Incorrect digit entered.\n";
                                }
                        }
                        j++;
                }
                cout << endl;

                if (i == 13) {;
                        while (1) {
                                try {
                                        cout << "Enter the index from which the new list will start (from 0
to " << size << "): ";
                                        cin >> str;
                                        pos = stoi(str);
                                        break;
                                }
                                catch (const exception& e) {
                                        cout << "Incorrect digit entered.\n";
                                }
                        }
                }

                if (i == 13) push_list_by_index(&NewList, pos);
                else if (i == 14) push_list_back(&NewList);
                else if (i == 15) push_list_beg(&NewList);
                else if (i == 16) {
                        if (is_list_in(NewList)) cout << "Your list is here.\n";
                        else cout << "Your list is not here.\n";
                }
                else if (i == 17)
                        cout << "Index of the first inclusion: " << find_first_inclusion(NewList) << endl;
                else
                        cout << "Index of the last inclusipon: " << find_last_inclusion(NewList) << endl;
                out_list();
        }
        else if (i == 19) {
                while (1) {
                        try {
                                cout << "Enter the index of the first element (from 0 to " << (size - 1) << "): ";
                                cin >> str;
```

```
                pos = stoi(str);
                if (pos < 0 || pos >= size) throw exception();
                else break;
            }
            catch (const exception& e) {
                cout << "Incorrect digit entered.\n";
            }
        }
        while (1) {
            try {
                cout << "Введите индекс второго элемента (от 0 to " << (size - 1) << "): ";
                cin >> str;
                j = stoi(str);
                if (j < 0 || j >= size) throw exception();
                else break;
            }
            catch (const exception& e) {
                cout << "Incorrect digit entered.\n";
            }
        }
        permutation(pos, j);
        out_list();
    }
}
void out_list()
{
    Node* tmp = head;
    cout << "Elements of the current list:\n";
    while (tmp != nullptr) {
        cout << tmp->digit << " ";
        tmp = tmp->next;
    }
    cout << "\n\n";
}
void push_back(int digit)
{
    if (tail == nullptr) {
        tail = new Node;
        tail->digit = digit;
        head = tail;
    }
    else {
        tail->next = new Node;
        tail->next->digit = digit;
        tail->next->prev = tail;
        tail = tail->next;
```

```
    }
    size++;
}
void push_front(int digit)
{
    if (head == nullptr) {
            head = new Node;
            head->digit = digit;
            tail = head;
    }
    else {
            head->prev = new Node;
            head->prev->digit = digit;
            head->prev->next = head;
            head = head->prev;
    }
    size++;
}
void delete_last() {
    if (size != 0) {
            Node* tmp = tail;
            tail = tail->prev;
            delete tmp;
            if (tail != nullptr)
                    tail->next = nullptr;
            size--;
    }
}
void delete_first() {
    if (size != 0) {
            Node* tmp = head;
            head = head->next;
            delete tmp;
            if (head != nullptr)
                    head->prev = nullptr;
            size--;
    }
}
void push_element_by_index(int digit, int index) {
    if (head == nullptr) {
            head = new Node;
            head->digit = digit;
            tail = head;
    }
    else if (index == 0) {
            Node* tmp = head;
```

```
                head->prev = new Node;
                head->prev->digit = digit;
                head = head->prev;
                head->next = tmp;
        }
        else if (index == size) {
                Node* tmp = tail;
                tail->next = new Node;
                tail->next->digit = digit;
                tail = tail->next;
                tail->prev = tmp;
        }
        else {
                int i = 1;
                Node* tmp = head;
                while (i != index) {
                        i++;
                        tmp = tmp->next;
                }
                Node* tmp2 = tmp->next;
                tmp->next = new Node;
                tmp->next->digit = digit;
                tmp2->prev = tmp->next;
                tmp->next->prev = tmp;
                tmp->next->next = tmp2;
        }
        size++;
}
Node* get_element_by_index(int pos) {
        int i = 0;
        Node* tmp = head;
        while (i != pos) {
                tmp = tmp->next;
                i++;
        }
        return tmp;
}
void delete_element_by_index(int pos) {
        int i = 0;
        Node* tmp = head;
        while (i != pos) {
                tmp = tmp->next;
                i++;
        }
        if (tmp == head) {
                Node* tmp = head;
```

```
                head = head->next;
                delete tmp;
                head->prev = nullptr;
        }
        else if (tmp == tail) {
                Node* tmp = tail;
                tail = tail->prev;
                delete tmp;
                tail->next = nullptr;
        }
        else {
                tmp->prev->next = tmp->next;
                tmp->next->prev = tmp->prev;
                delete tmp;
        }
        size--;
}
unsigned get_size() {
        return size;
}
void clear() {
        Node* tmp = head, * tmp2;
        while (tmp != nullptr) {
                tmp2 = tmp;
                tmp = tmp->next;
                delete tmp2;
        }
        head = tail = nullptr;
        size = 0;
}
void set_element_by_index(int digit, int pos) {
        int i = 0;
        Node* tmp = head;
        while (i != pos) {
                tmp = tmp->next;
                i++;
        }
        tmp->digit = digit;
}
bool is_clear() {
        if (size == 0) return 1;
        else return 0;
}
void reverse() {
        if (size > 1) {
                Node* tmp1 = head;
```

```
                Node* tmp2 = tail;
                int tmp;
                if (size % 2 == 0) {
                        do {
                                tmp = tmp1->digit;
                                tmp1->digit = tmp2->digit;
                                tmp2->digit = tmp;

                                tmp1 = tmp1->next;
                                tmp2 = tmp2->prev;
                        }
                        while (tmp2->next != tmp1);
                }
                else {
                        do {
                                tmp = tmp1->digit;
                                tmp1->digit = tmp2->digit;
                                tmp2->digit = tmp;

                                tmp1 = tmp1->next;
                                tmp2 = tmp2->prev;
                        } while (tmp1 != tmp2);
                }
        }
}
void push_list_by_index(TwoWayList* list, int index) {
        if (index == 0) push_list_beg(list);
        else if (index == size) push_list_back(list);
        else {
                int ind = 1;
                Node* tmp = head;
                while (ind != index) {
                        ind++;
                        tmp = tmp->next;
                }
                tmp->next->prev = list->get_tail();
                list->get_tail()->next = tmp->next;
                tmp->next = list->get_head();
                list->get_head()->prev = tmp;
                size += list->get_size();
        }
}
void push_list_back(TwoWayList* list) {
        tail->next = list->get_head();
        list->get_head()->prev = tail;
        tail = list->get_tail();
```

```
    size += list->get_size();
}
void push_list_beg(TwoWayList* list) {
    head->prev = list->get_tail();
    list->get_tail()->next = head;
    head = list->get_head();
    size += list->get_size();
}
bool is_list_in(TwoWayList list) {
    int index = 0, index_tmp = 0;
    int index2 = 1;
    while (index < size) {
        if (get_element_by_index(index)->digit == list.get_element_by_index(0)->digit) {
            if (list.get_size() == 1) return 1;

            index_tmp = index++;
            while (index < size) {
                if (get_element_by_index(index)->digit == list.get_element_by_index(index2)->digit) {
                    index++;
                    index2++;
                    if (index2 == list.get_size()) return 1;
                }
                else {
                    index = index_tmp;
                    index2 = 1;
                    break;
                }
            }
        }
        index++;
    }

    return 0;
}
int find_first_inclusion(TwoWayList list) {
    int index = 0, index_tmp = 0;
    int index2 = 1;
    while (index < size) {
        if (get_element_by_index(index)->digit == list.get_element_by_index(0)->digit) {
            if (list.get_size() == 1) return index;

            index_tmp = ++index;
            while (index < size) {
                if (get_element_by_index(index)->digit == list.get_element_by_index(index2)->digit) {
```

```
                                        index++;
                                        index2++;
                                        if (index2 == list.get_size()) return (index_tmp - 1);
                                }
                                else {
                                        index = index_tmp;
                                        index2 = 1;
                                        break;
                                }
                        }
                }
                index++;
        }

        return -1;
}
int find_last_inclusion(TwoWayList list) {
        int index = size-1, index_tmp = size-1;
        int index2 = list.get_size() - 2;
        while (index >= 0) {
                if (get_element_by_index(index)->digit == list.get_element_by_index(list.get_size() - 1)->digit) {
                        if (list.get_size() == 1) return index;

                        index_tmp = --index;
                        while (index >= 0) {
                                if (get_element_by_index(index)->digit ==
list.get_element_by_index(index2)->digit) {
                                        index--;
                                        index2--;
                                        if (index2 == -1) return (index+1);
                                }
                                else {
                                        index = index_tmp;
                                        index2 = list.get_size() - 2;
                                        break;
                                }
                        }
                }
                index--;
        }

        return -1;
}
void permutation(int ind1, int ind2) {
        Node* tmp1 = get_element_by_index(ind1);
        Node* tmp2 = get_element_by_index(ind2);
```

```
            int tmp = tmp1->digit;
            tmp1->digit = tmp2->digit;
            tmp2->digit = tmp;
        }
};

int main()
{
        TwoWayList List;
        bool end = false;
        string str;
        setlocale(LC_ALL, "ru");
        SetConsoleCP(1251); // для чтения русских букв

        while (!end) {
            List.call_func_by_number(GetNumberOfCommand());
            while (1) {
                        cout << "Do you want to continue? (yes/no)\n" <<
                                "Your answer: ";
                        cin >> str;
                        if (str == "Yes" || str == "yes") {
                                cout << "\n";
                                break;
                        }
                        else if (str == "No" || str == "no") {
                                end = true;
                                break;
                        }
                        else {
                                cout << "The answer is not defined.\n";
                        }
            }
        }
        List.clear();

        return 0;
}

int GetNumberOfCommand() {
        cout << "Select one of the possible commands:\n" <<
            "\t1) add an element to the end;\n" <<
            "\t2) add an element to the start;\n" <<
            "\t3) delete the last element;\n" <<
            "\t4) delete the first element;\n" <<
            "\t5) add an element by index;\n" <<
            "\t6) get an element by index;\n" <<
```

```
            "\t7) delete an element by index;\n" <<
            "\t8) get the list size;\n" <<
            "\t9) delete all elements;\n" <<
            "\t10) replace an element by index;\n" <<
            "\t11) check for list emptiness;\n" <<
            "\t12) reverse the order of elements;\n" <<
            "\t13) insert another list starting from the index;\n" <<
            "\t14) insert another list at the end;\n" <<
            "\t15) insert another list at the beginning;\n" <<
            "\t16) check for the inclusion of another list;\n" <<
            "\t17) search for the index of the first inclusion of another list;\n" <<
            "\t18) search for the index of the last inclusion of another list;\n" <<
            "\t19) exchange of two list items by indexes.\n" <<
            "Your choice: ";
        try {
            int i; string str;
            cin >> str;
            i = stoi(str);
            if (i < 1 || i > 19) {
                    throw exception();
            }
            cout << "\n";
            return i;
        }
        catch (const exception& e) {
            cout << "Incorrect number entered.\n";
        }
}
```

6 Выводы

В результате данной лабораторной работы была написана программа, имеющая множество методов для работы со списками, а также имеющая понятный пользовательский интерфейс для работы.

Также была высчитана средняя временная сложность для каждого алгоритма с доказательством для некоторых из них.

7 Ссылка на репозиторий

https://github.com/afrlfff/university-student-tasks/tree/master/algoritms-and-data-structures/lab1-list