|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования |
| **«МИРЭА – Российский технологический университет»** |
| **РТУ МИРЭА** |
|  |

| **Отчет по выполнению практического задания № 6** | |
| --- | --- |
| **Тема:** | |
| **«Двунаправленные динамические списки»** | |
| Дисциплина: «Структуры и алгоритмы обработки данных» | |
|  | Выполнил студент: Лелюхин Н.С. |
|  | Группа: ИКБО-74-23 |

Москва – 2024

СОДЕРЖАНИЕ

[1 ЦЕЛЬ 3](#_gjdgxs)

[2 ЗАДАНИЕ 4](#_30j0zll)

[2.1 Формулировка задачи (В списке №18, Вариант 2) 4](#_1fob9te)

[2.2 Определение списка и описание операций над списком 5](#_3znysh7)

[2.2.1 Определение структуры узла двунаправленного списка 5](#_2et92p0)

[2.2.2. Вставка узла в конец списка. 6](#_3dy6vkm)

[2.2.3 Удаление узла. 7](#_1t3h5sf)

[2.2.4 Вывод элементов списка на экран. 8](#_4d34og8)

[2.2.5 Поиск узла по номеру маршрута 8](#_2s8eyo1)

[2.2.6 Добавление нового узла с упорядочиванием по первым 4-ём цифрам телефона 9](#_17dp8vu)

[2.2.7 Удалить последний узел с заданным номером телефона 9](#_26in1rg)

[2.2.8 Вычисление общего времени разговора по заданному номеру 10](#_xyib28hu806l)

[2.3 Реализация алгоритма на языке C++ и проведение тестирования 10](#_35nkun2)

[2.3.1 Реализация на языке программирования C++ 10](#_1ksv4uv)

[2.3.2 Тестирование 18](#_44sinio)

[2.4 Вывод по заданию 22](#_2jxsxqh)

[3 ВЫВОДЫ 24](#_3j2qqm3)

[4 ЛИТЕРАТУРА 25](#_1y810tw)

# **1 ЦЕЛЬ**

Получение знаний и практических навыков управления двунаправленным списком в программах на языке С++.

# **2 ЗАДАНИЕ**

## **2.1 Формулировка задачи (В списке №18, Вариант 2)**

Разработать многомодульную программу, которая демонстрирует выполнение всех операций, определенных вариантом, над линейным двунаправленным динамическим списком.

Требования к разработке.

1. Разработать структуру узла списка, структура информационной части узла определена вариантом. Для определения структуры узла списка, используйте тип struct или class. Сохраните определение структуры узла и прототипы функций в заголовочном файле.

2. Разработайте функции для выполнения операции над линейным двунаправленным динамическим списком:

• создание списка;

• вставку узла;

• удаление узла;

• вывод списка в двух направлениях (слева направо и справа налево);

• поиск узла с заданным значением (операция должна возвращать указатель на узел с заданным значением).

3. Дополнительные операции над списком, указанные вариантом, оформите в виде функций и включите в отдельный файл с расширением cpp. Подключите к этому файлу заголовочный файл с определением структуры узла.

4. Разработайте программу, управляемую текстовым меню, и включите в меню демонстрацию выполнения всех операций задания и варианта.

5. Проведите тестирование операций.

• Оцените сложность алгоритма первой дополнительной операции.

6. Оформите отчет по разработке программы в соответствии с требованиями задания по однонаправленному списку.

Примечание: в определении информационной части узла варианта, подчеркнутое поле считать полем ключа.

Индивидуальный вариант. Тип информационной части узла: Номер телефона (из 7 цифр), время разговора (целое число), номер телефона вызываемого абонента.

Дополнительные операции:

* Добавить новый узел в список, упорядочивая узлы по первым четырем цифрам телефона в порядке возрастания.
* Удалить последний узел с заданным значением телефона.
* Подсчитать суммарное время разговора с заданного телефона

## **2.2 Определение списка и описание операций над списком**

### **2.2.1 Определение структуры узла двунаправленного списка**

Структура Node представляет собой узел двунаправленного списка. В каждом узле содержится информация о телефонном номере, времени разговора и номере вызываемого абонента. Кроме того, каждый узел содержит указатели на предыдущий (prev) и следующий (next) узлы в списке. Это позволяет эффективно организовывать связанные узлы, обеспечивая возможность двунаправленного перемещения по списку. В частности, наличие указателя на предыдущий узел (prev) облегчает выполнение операций удаления и вставки элементов в список.

Структура Node является базовым строительным блоком для реализации двунаправленного списка, обеспечивая хранение информации о разговорах, а также связи между узлами списка.

Данное описание структуры представлено в виде кода на C++ в блоке кода 1, а его изображение на рисунке 1.

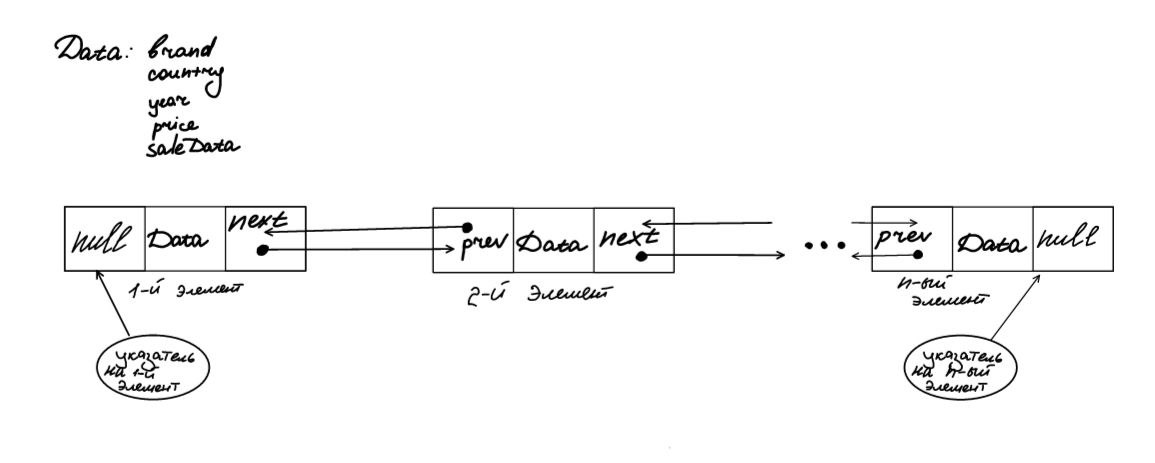


Рисунок 1 - Изображение структуры данных

| **struct** **Node** {  **int** phoneNumber;  **int** talkTime;  string calleePhoneNumber;   Node\* prev;  Node\* next; }; |
| --- |

Блок кода 1 - Структура

### 2.2.2. Вставка узла в конец списка.

Отобразим выполнение данной операции на рисунке 2.

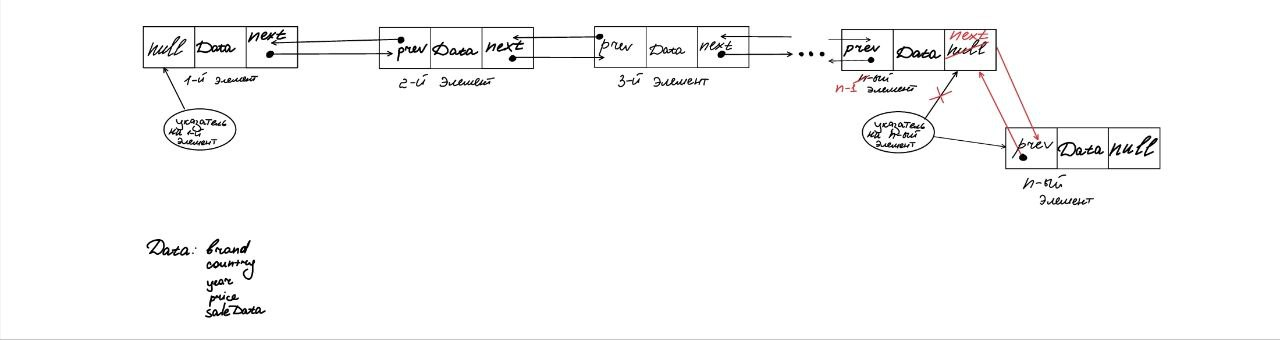


Рисунок 2 - Изображение добавление нового узла

Создадим новый узел и добавим его в конец списка. Если список пустой, новый узел становится и головой, и хвостом списка. В противном случае, новый узел добавляется после текущего хвоста, и хвост обновляется.

Реализация данного алгоритма представлена в блок коде 3. Данные для тестирования будут приведены в таблице 1.

Таблица 1 - Данные для тестирования

| № | Входные данные | Добавить | Ожидаемый результат |
| --- | --- | --- | --- |
| 1 |  | 12345,10, 98765 | 12345, 10, 98765 |
|  | 12345, 10, 98765 | 98765,10, 12345 | 12345, 10, 98765  98765, 10, 12345 |
|  | 12345, 10, 98765  98765, 10, 12345 | 13579,12 24680 | 12345, 10, 98765  98765, 10, 12345  13579,12 24680 |
|  | 12345, 10, 98765  98765, 10, 12345  13579,12 24680 | 8642,31,97531 | 12345, 10, 98765  98765, 10, 12345  13579,12 24680  08642, 31,97531 |
|  | 12345, 10, 98765  98765, 10, 12345  13579,12 24680  08642, 31,97531 | 13579, 23,24680 | 12345, 10, 98765  98765, 10, 12345  13579,12 24680  8642, 31,97531  13579, 23,24680 |

### 2.2.3 Удаление узла.

Отобразим выполнение данной операции на рисунке 3.

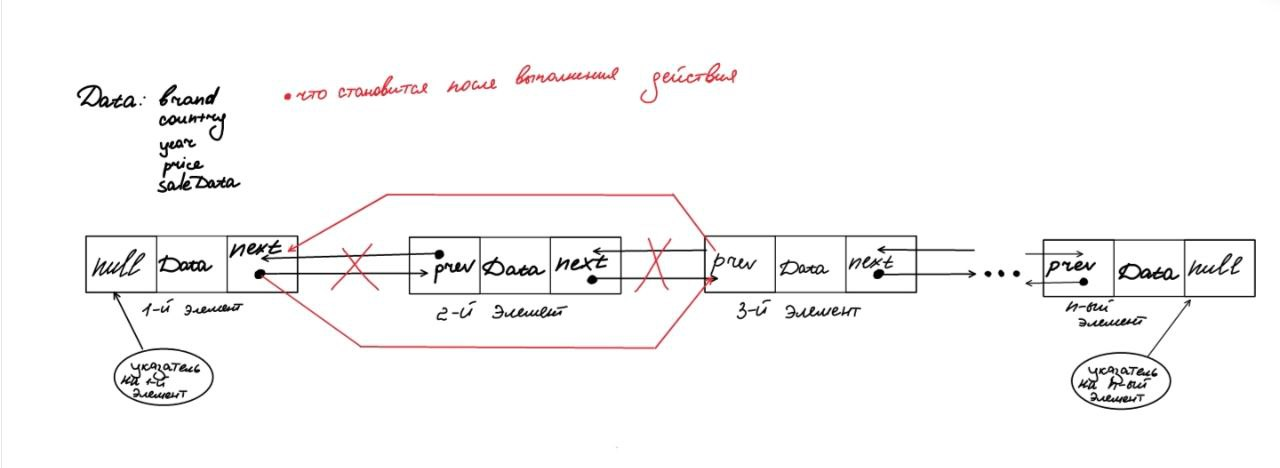


Рисунок 3 - Изображение удаления узла

Удалим узел с заданным номером телефона. Просматриваем список, пока не найдет узел с заданным номером телефона или не достигнет конца списка. Если узел найден, он удаляется из списка, обновляются ссылки на предыдущий и следующий узлы, а затем освобождается память, выделенная для удаленного узла.

Реализация данного алгоритма представлена в блок коде 3. Данные для тестирования будут приведены в таблице 2.

Таблица 2 - Данные для тестирования

| № | Входные данные | Удалить | Ожидаемый результат |
| --- | --- | --- | --- |
| 1 | 12345, 10, 98765  98765, 10, 12345  13579,12 24680  8642, 31,97531  13579, 23,24680 | 8642 | 12345, 10, 98765  98765, 10, 12345  13579,12 24680  13579, 23,24680 |

### 2.2.4 Вывод элементов списка на экран.

Функция printListLeftToRight выводит содержимое списка, начиная с головы и двигаясь вперед по ссылкам next. Для каждого узла выводятся номер телефона, время разговора и номер телефона вызываемого абонента.

Функция printListRightToLeft также выводит содержимое списка, но начиная с хвоста и двигаясь назад по ссылкам prev. Это позволяет вывести список в обратном порядке.

Реализация данного алгоритма представлена в блок коде 3.

### 2.2.5 Поиск узла по номеру маршрута

Отобразим выполнение данной операции на рисунке 4.

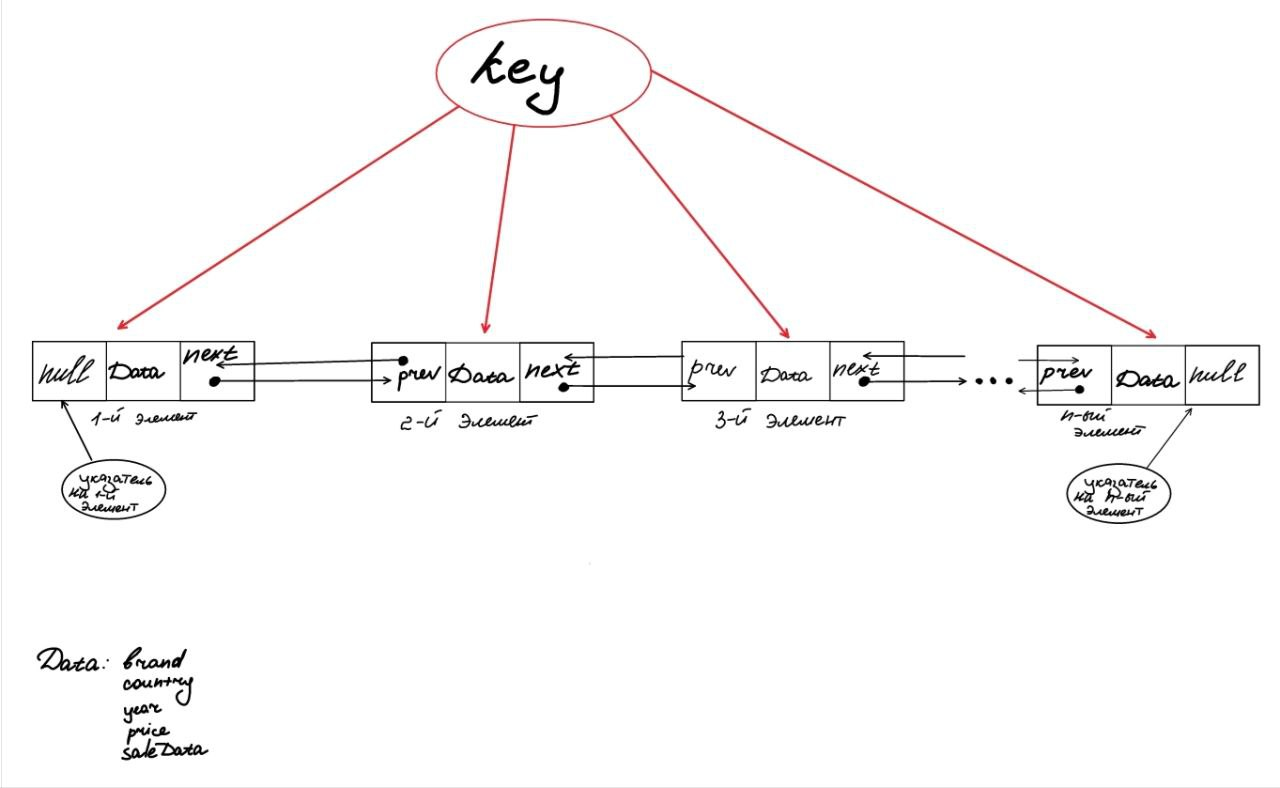


Рисунок 4 - Поиск узла с заданным значением

Начнем поиск с головы списка и перемещаемся по списку, проверяя каждый узел на совпадение с заданным номером телефона. Если узел с указанным номером телефона найден, функция возвращает указатель на этот узел. Если такой узел не найден, функция возвращает nullptr.

Реализация данного алгоритма представлена в блок коде 3.

### 2.2.6 Добавление нового узла с упорядочиванием по первым 4-ём цифрам телефона

Добавим новый узел в список, упорядочивая узлы по первым четырем цифрам телефона в порядке возрастания. Если список пуст, новый узел становится его головой и хвостом. Если номер телефона нового узла меньше, чем у головы списка, новый узел становится новой головой. В противном случае, функция находит место для вставки нового узла в упорядоченный порядок, перемещаясь по списку до того момента, пока не найдет узел с более большим номером телефона или пока не дойдет до конца списка. После этого новый узел вставляется перед узлом, чей номер телефона больше нового, или становится новым хвостом списка, если такого узла не найдено.

Реализация данного алгоритма представлена в блок коде 3. Данные для тестирования будут приведены в таблице 3.

Таблица 3 - Данные для тестирования

| № | Входные данные | Добавить | Ожидаемый результат |
| --- | --- | --- | --- |
| 1 | 12345, 10, 98765  98765, 10, 12345  13579,12 24680  13579, 23,24680 | 12345, 2, 24680 | 12345, 10, 98765  12345, 2, 24680  98765, 10, 12345  13579,12 24680  13579, 23,24680 |

### 2.2.7 Удалить последний узел с заданным номером телефона

Удалим последний узел с заданным номером телефона из списка. Она начинает поиск с хвоста списка и перемещается в обратном направлении, проверяя каждый узел на совпадение с заданным номером телефона. Если узел с указанным номером телефона найден, функция удаляет его из списка, обновляя ссылки на предыдущий и следующий узлы. Если найденный узел является единственным узлом в списке, то голова и хвост списка устанавливаются в nullptr. После удаления узла функция освобождает выделенную для него память.

Реализация данного алгоритма представлена в блок коде 3. Данные для тестирования будут приведены в таблице 4.

Таблица 4 - Данные для тестирования

| № | Входные данные | Номер телефона | Ожидаемый результат |
| --- | --- | --- | --- |
| 1 | 12345, 10, 98765  12345, 2, 24680  98765, 10, 12345  13579,12 24680  13579, 23,24680 | 13579 | 12345, 10, 98765  12345, 2, 24680  98765, 10, 12345  13579,12 24680 |

### 2.2.8 Вычисление общего времени разговора по заданному номеру

Вычислим общее время разговора для заданного номера телефона. Она перебирает все узлы в списке, начиная с головы, и для каждого узла проверяет совпадение номера телефона с заданным. Если номер телефона узла совпадает с заданным, его время разговора добавляется к общему времени. По завершении перебора всех узлов функция возвращает общее время разговора для заданного номера телефона.

Реализация данного алгоритма представлена в блок коде 3.

## **2.3 Реализация алгоритма на языке C++ и проведение тестирования**

### **2.3.1 Реализация на языке программирования C++**

Реализуем данный алгоритм на языке C++. Для реализации понадобятся такие библиотеки, как iostream, string.

| #ifndef HEADER\_H #define HEADER\_H #include <iostream> #include <string> **using** **namespace** std;  **struct** **Node** {  **int** phoneNumber;  **int** talkTime;  string calleePhoneNumber;   Node\* prev;  Node\* next; };  **class** **DoublyLinkedList** { **private**:  Node\* head;  Node\* tail;  **public**:  DoublyLinkedList();  ~DoublyLinkedList();   **void** **insertNode**(**int** phoneNumber, **int** talkTime, **const** std::string& calleePhoneNumber);  **void** **deleteNode**(**int** phoneNumber);  **void** **printListLeftToRight**();  **void** **printListRightToLeft**();  Node\* **findNode**(**int** phoneNumber);   // Additional operations  **void** **addNodeOrdered**(**int** phoneNumber, **int** talkTime, **const** string& calleePhoneNumber);  **void** **deleteLastNodeWithPhoneNumber**(**int** phoneNumber);  **int** **totalTalkTime**(**int** phoneNumber); };  #endif |
| --- |

Блок кода 2 – header.h

| #include "header.h"  DoublyLinkedList::DoublyLinkedList() {  head = **nullptr**;  tail = **nullptr**; }  DoublyLinkedList::~DoublyLinkedList() {  Node\* current = head;  **while** (current) {  Node\* next = current->next;  **delete** current;  current = next;  } }  **void** DoublyLinkedList::insertNode(**int** phoneNumber, **int** talkTime, **const** string& calleePhoneNumber) {  Node\* newNode = **new** Node;  newNode->phoneNumber = phoneNumber;  newNode->talkTime = talkTime;  newNode->calleePhoneNumber = calleePhoneNumber;  newNode->prev = **nullptr**;  newNode->next = **nullptr**;   **if** (!head) {  head = tail = newNode;  } **else** {  tail->next = newNode;  newNode->prev = tail;  tail = newNode;  } }  **void** DoublyLinkedList::deleteNode(**int** phoneNumber) {  Node\* current = head;  **while** (current) {  **if** (current->phoneNumber == phoneNumber) {  **if** (current == head && current == tail) {  head = tail = **nullptr**;  } **else** **if** (current == head) {  head = head->next;  head->prev = **nullptr**;  } **else** **if** (current == tail) {  tail = tail->prev;  tail->next = **nullptr**;  } **else** {  current->prev->next = current->next;  current->next->prev = current->prev;  }  **delete** current;  **return**;  }  current = current->next;  } }  **void** DoublyLinkedList::printListLeftToRight() {  Node\* current = head;  **while** (current) {  cout << "Номер телефона: " << current->phoneNumber << ", Время разговора: " << current->talkTime << ", Номер телефона вызываемого абонента: " << current->calleePhoneNumber << endl;  current = current->next;  } }  **void** DoublyLinkedList::printListRightToLeft() {  Node\* current = tail;  **while** (current) {  cout << "Номер телефона: " << current->phoneNumber << ", Время разговора: " << current->talkTime << ", Номер телефона вызываемого абонента: " << current->calleePhoneNumber << endl;  current = current->prev;  } }  Node\* DoublyLinkedList::findNode(**int** phoneNumber) {  Node\* current = head;  **while** (current) {  **if** (current->phoneNumber == phoneNumber) {  **return** current;  }  current = current->next;  }  **return** **nullptr**; }  // Additional operations  **void** DoublyLinkedList::addNodeOrdered(**int** phoneNumber, **int** talkTime, **const** string& calleePhoneNumber) {  Node\* newNode = **new** Node;  newNode->phoneNumber = phoneNumber;  newNode->talkTime = talkTime;  newNode->calleePhoneNumber = calleePhoneNumber;  newNode->prev = **nullptr**;  newNode->next = **nullptr**;   **if** (!head) {  head = tail = newNode;  } **else** **if** (phoneNumber < head->phoneNumber) {  newNode->next = head;  head->prev = newNode;  head = newNode;  } **else** {  Node\* current = head;  **while** (current->next && current->next->phoneNumber < phoneNumber) {  current = current->next;  }  **if** (!current->next) {  tail->next = newNode;  newNode->prev = tail;  tail = newNode;  } **else** {  newNode->next = current->next;  newNode->prev = current;  current->next->prev = newNode;  current->next = newNode;  }  } }  **void** DoublyLinkedList::deleteLastNodeWithPhoneNumber(**int** phoneNumber) {  Node\* current = tail;  **while** (current) {  **if** (current->phoneNumber == phoneNumber) {  **if** (current == head && current == tail) {  head = tail = **nullptr**;  } **else** **if** (current == tail) {  tail = tail->prev;  tail->next = **nullptr**;  } **else** **if** (current == head) {  head = head->next;  head->prev = **nullptr**;  } **else** {  current->prev->next = current->next;  current->next->prev = current->prev;  }  **delete** current;  **return**;  }  current = current->prev;  } }  **int** DoublyLinkedList::totalTalkTime(**int** phoneNumber) {  **int** total = 0;  Node\* current = head;  **while** (current) {  **if** (current->phoneNumber == phoneNumber) {  total += current->talkTime;  }  current = current->next;  }  **return** total; } |
| --- |

Блок кода 3 – header.cpp

| #include <iostream> #include "header.h"  **using** **namespace** std;  **void** **printMenu**() {  cout << "====== Меню ======" << endl;  cout << "1. Вставить узел" << endl;  cout << "2. Удалить узел" << endl;  cout << "3. Вывести список вперед" << endl;  cout << "4. Вывести список назад" << endl;  cout << "5. Найти узел по номеру телефона" << endl;  cout << "6. Добавить узел упорядоченно" << endl;  cout << "7. Удалить последний узел с номером телефона" << endl;  cout << "8. Посчитать общее время разговора по номеру телефона" << endl;  cout << "9. Выход" << endl;  cout << "==================" << endl; }  **int** **main**() {  DoublyLinkedList callList;   **int** choice;  **do** {  printMenu();  cout << "Выберите операцию: ";  cin >> choice;   **switch** (choice) {  **case** 1: {  **int** phoneNumber, talkTime;  string calleePhoneNumber;  cout << "Введите номер телефона: ";  cin >> phoneNumber;  cout << "Введите время разговора (в минутах): ";  cin >> talkTime;  cout << "Введите номер вызываемого абонента: ";  cin >> calleePhoneNumber;  callList.insertNode(phoneNumber, talkTime, calleePhoneNumber);  cout << "Узел успешно вставлен." << endl;  **break**;  }  **case** 2: {  **int** phoneNumber;  cout << "Введите номер телефона для удаления: ";  cin >> phoneNumber;  callList.deleteNode(phoneNumber);  cout << "Узел с номером телефона " << phoneNumber << " успешно удален." << endl;  **break**;  }  **case** 3: {  cout << "Список вперед:" << endl;  callList.printListLeftToRight();  cout << endl;  **break**;  }  **case** 4: {  cout << "Список назад:" << endl;  callList.printListRightToLeft();  cout << endl;  **break**;  }  **case** 5: {  **int** phoneNumber;  cout << "Введите номер телефона для поиска: ";  cin >> phoneNumber;  Node\* foundNode = callList.findNode(phoneNumber);  **if** (foundNode) {  cout << "Найден узел с номером телефона " << phoneNumber << endl;  } **else** {  cout << "Узел с номером телефона " << phoneNumber << " не найден" << endl;  }  **break**;  }  **case** 6: {  **int** phoneNumber, talkTime;  string calleePhoneNumber;  cout << "Введите номер телефона: ";  cin >> phoneNumber;  cout << "Введите время разговора (в минутах): ";  cin >> talkTime;  cout << "Введите номер вызываемого абонента: ";  cin >> calleePhoneNumber;  callList.addNodeOrdered(phoneNumber, talkTime, calleePhoneNumber);  cout << "Узел успешно добавлен." << endl;  **break**;  }  **case** 7: {  **int** phoneNumber;  cout << "Введите номер телефона для удаления последнего узла: ";  cin >> phoneNumber;  callList.deleteLastNodeWithPhoneNumber(phoneNumber);  cout << "Последний узел с номером телефона " << phoneNumber << " успешно удален." << endl;  **break**;  }  **case** 8: {  **int** phoneNumber;  cout << "Введите номер телефона для подсчета времени разговора: ";  cin >> phoneNumber;  **int** totalTalkTime = callList.totalTalkTime(phoneNumber);  cout << "Общее время разговора по номеру телефона " << phoneNumber << ": " << totalTalkTime << " минут" << endl;  **break**;  }  **case** 9:  cout << "Выход из программы." << endl;  **break**;  **default**:  cout << "Неверный выбор. Пожалуйста, выберите существующую операцию." << endl;  **break**;  }  } **while** (choice != 9);   **return** 0; } |
| --- |

Блок кода 4 – main.cpp

### **2.3.2 Тестирование**

Проведем тестирование основываясь на данные из таблиц 1-4. Результаты тестирования представлены на рисунках 5-13.

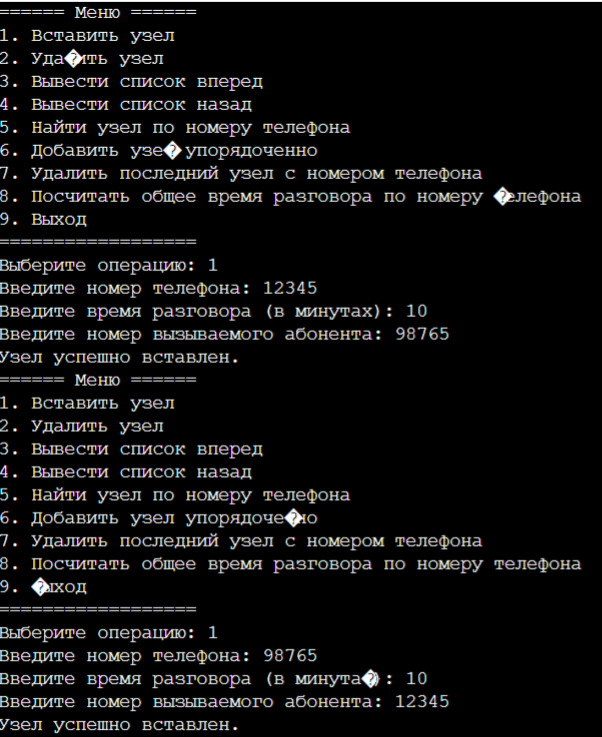


Рисунок 5 - Тестирование программы

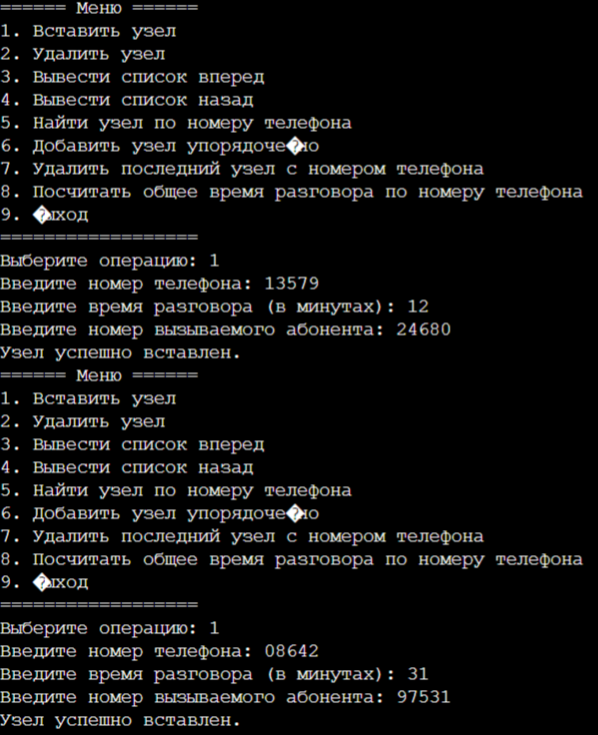


Рисунок 6 - Тестирование программы

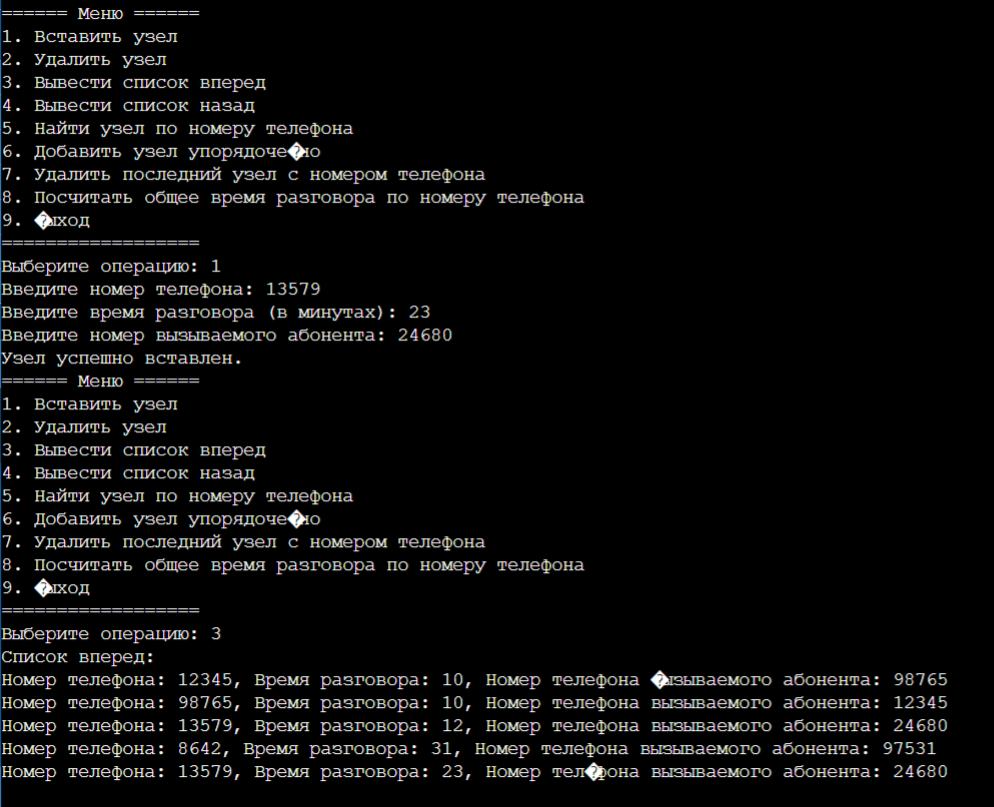


Рисунок 7 - Тестирование программы

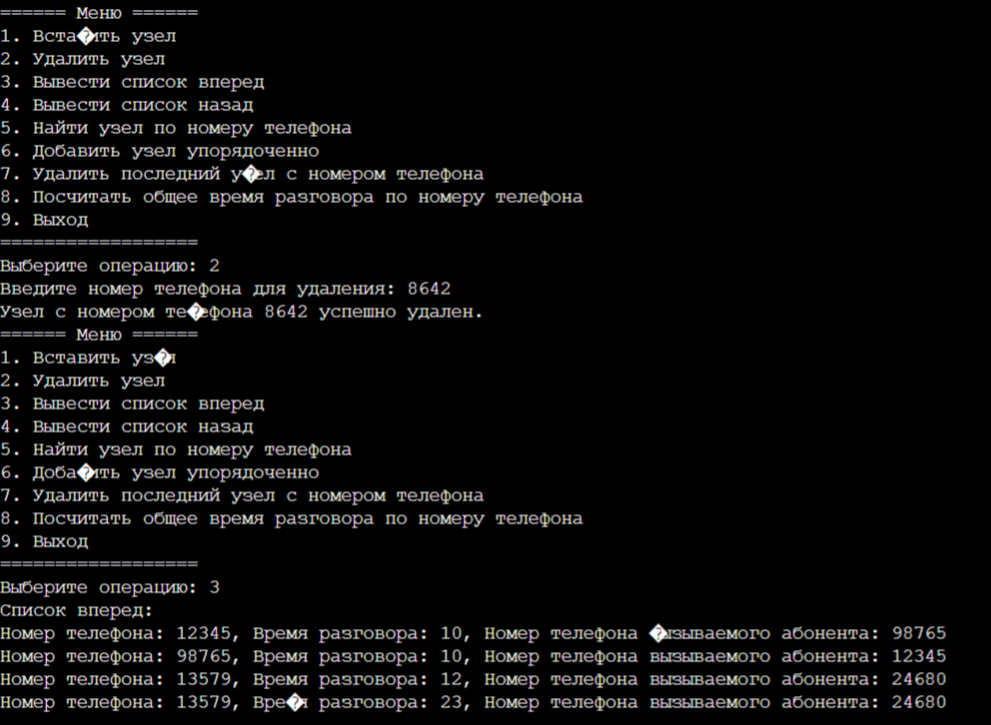


Рисунок 8 - Тестирование программы

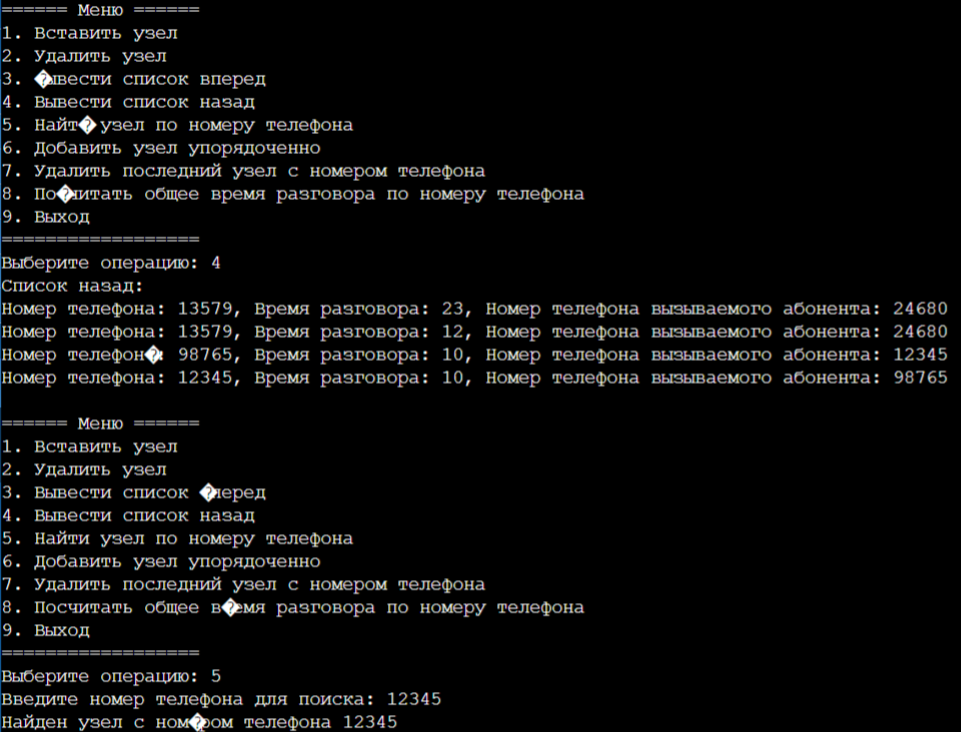


Рисунок 9 - Тестирование программы

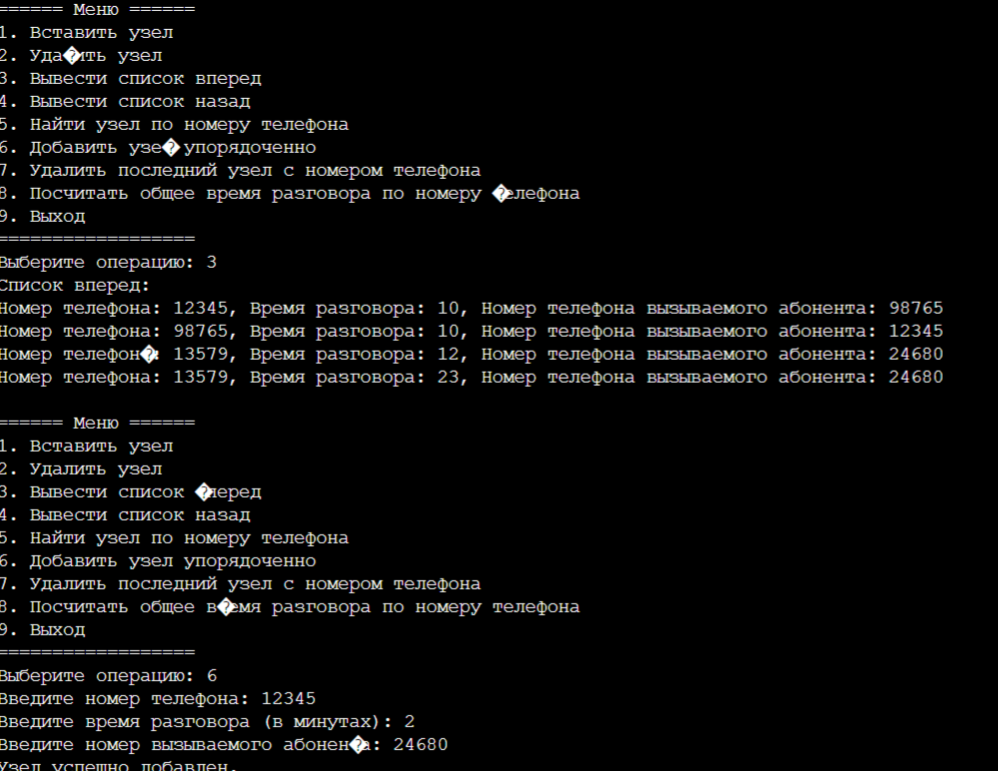


Рисунок 10 - Тестирование программы

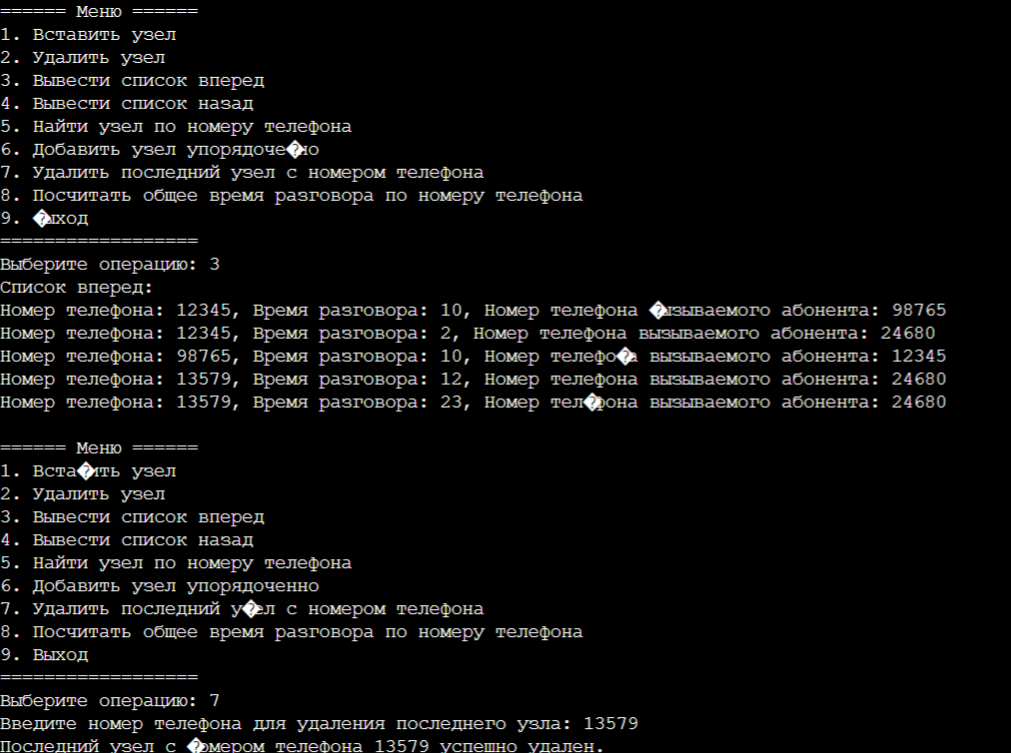


Рисунок 11 - Тестирование программы

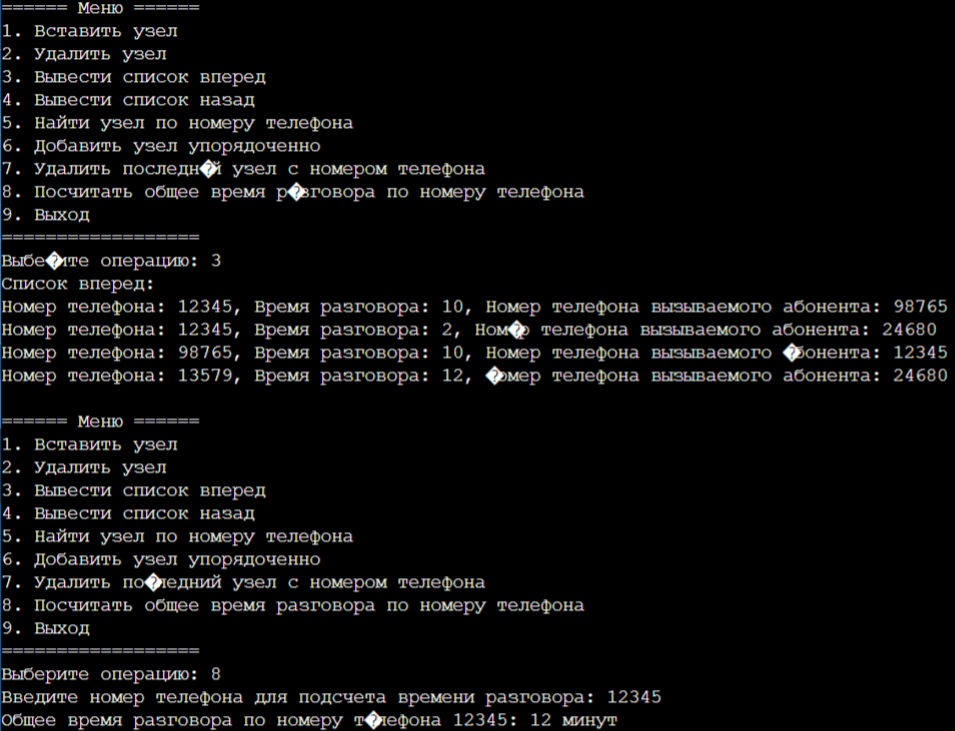


Рисунок 12 - Тестирование программы

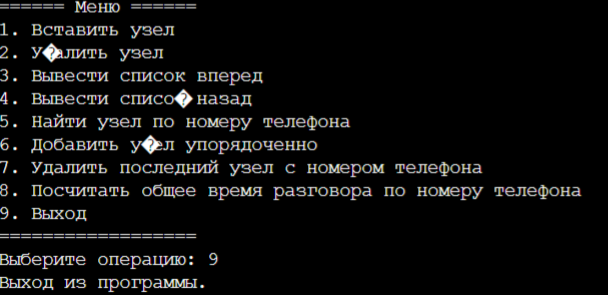


Рисунок 13 - Тестирование программы

Проведение тестирования на основе таблиц 1-4, показало, что программа выполнена верно, так как совпала с ожидаемым результатом.

## **2.4 Вывод по заданию**

Двунаправленный список представляет собой структуру данных, где каждый элемент содержит ссылки как на предыдущий, так и на следующий элемент. Это обеспечивает возможность двунаправленного перемещения по списку, что полезно для множества операций, таких как вставка, удаление и поиск элементов.

Алгоритм вставки нового узла в список с упорядочиванием узлов по первым четырем цифрам телефона в порядке возрастания будет иметь теоретическую сложность O(n), где n - количество узлов в списке.

При вставке нового узла алгоритм должен пройти по списку и сравнить первые четыре цифры телефонного номера нового узла с первыми четырьмя цифрами телефонных номеров узлов в списке. Затем новый узел будет вставлен в нужное место в порядке возрастания первых четырех цифр телефонного номера.

Поскольку каждая операция вставки требует просмотра только части списка (не всего), сложность алгоритма вставки будет линейной и зависеть от количества узлов в списке.

# 

# **3 ВЫВОДЫ**

В ходе практической работы были выполнены следующие задачи:

- Получены знания по управления двунаправленным списком в программах на языке С++;

- Получены практические навыки управления двунаправленным списком в программах на языке С++;

- Проведён анализ структуры в соответствии с индивидуальным вариантом;

- Проведён анализ операций в соответствии с индивидуальным вариантом;

- Была реализована программа для структуры в соответствии с индивидуальным вариантом;

- Были реализованы программы для операций в соответствии с индивидуальным вариантом;

- Было реализовано меню выбора способа ввода и реализации функций;

- Было оценена сложность первой дополнительной операции;

- Проведено тестирование программы с различными данными и способами ввода;

Таким образом, главную цель практической работы, а именно получение знаний и практических навыков управления двунаправленным списком в программах на языке С++, можно считать выполненной.

# **4 ЛИТЕРАТУРА**

1. Бхаргава А. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих. – СПб: Питер, 2017. – 288 с.

2. Вирт Н. Алгоритмы + структуры данных = программы. – М.: Мир, 1985. – 406 с.

3. Кнут Д.Э. Искусство программирования, том 3. Сортировка и поиск, 2-е изд. – М.: ООО «И.Д. Вильямс», 2018. – 832 с.

4. Кораблин Ю.П. Структуры и алгоритмы обработки данных: учебно-методическое пособие / Ю.П. Кораблин, В.П. Сыромятников, Л.А. Скворцова. – М.: РТУ МИРЭА, 2020. — 219 с.

5. Кормен Т.Х. и др. Алгоритмы: построение и анализ, 3-е изд. – М.: ООО «И.Д.Вильямс», 2013. – 1328 с.

6. Макконнелл Дж. Основы современных алгоритмов. Активный обучающий метод. 3-е доп. изд., - М.: Техносфера, 2018. – 416 с.

7. Седжвик Р. Фундаментальные алгоритмы на C++. Анализ/Структуры данных/Сортировка/Поиск. – К.: Издательство «Диасофт», 2001. – 688 с.

8. Скиена С. Алгоритмы. Руководство по разработке, - 2-е изд. – СПб: БХВ-Петербург, 2011. – 720 с.

9. Хайнеман Д. и др. Алгоритмы. Справочник с примерами на C, C++, Java и Python, 2-е изд. – СПб: ООО «Альфа-книга», 2017. – 432 с.

10. AlgoList – алгоритмы, методы, исходники [Электронный ресурс]. URL: http://algolist.manual.ru/ (дата обращения 15.03.2022).

11. Алгоритмы – всё об алгоритмах / Хабр [Электронный ресурс]. URL: https://habr.com/ru/hub/algorithms/ (дата обращения 15.03.2022).