МИНОБРНАУКИ РФ

ФГБОУ ВПО Тверской государственный технический университет

Кафедра “Программное обеспечение”

Курсовая работа

дисциплина “Алгоритмы и структуры данных”

Тема: “ Лабораторные работы 1 сем АиСД”

Выполнил: студент группы

ПИН 17.05

Завгороднев Егор и Иванов Роман

Проверил:

Мальков А.А

Тверь 2018

Оглавление

[Введение 3](#_Toc533367898)

[Алгоритмы 4](#_Toc533367899)

[“Алгоритмы работы со строками” 4](#_Toc533367900)

[“Алгоритмы поиска” 6](#_Toc533367901)

[“Алгоритмы сортировки” 6](#_Toc533367902)

[“Стек” 7](#_Toc533367903)

[“Очередь” 7](#_Toc533367904)

[“Дек” 7](#_Toc533367905)

[“Работа со списками” 7](#_Toc533367906)

# Введение

**Цель:** Понимать, что такое алгоритмы и структуры данных, изучить и применить на практике такие структуры данных как массивы, матрицы, списки, графы, стеки, очереди, деки.

Ознакомиться с алгоритмами поиска в массиве, работы со строками, сортировки, работы со списками, поиска в графе.

**Задача:** Выполнить 12 лабораторных работ

1. Симметричные матрицы
2. Разреженные массивы
3. Алгоритмы работы со строками
4. Алгоритмы поиска
5. Алгоритмы сортировки
6. Стек
7. Очереди
8. Дек
9. Алгоритмы работы со списками
10. Алгоритмы поиска в глубину и в ширину
11. Алгоритм поиска сильно связных компонент
12. Алгоритмы поиска циклов в графе

# Алгоритмы

## “Алгоритмы работы со строками”

**Задача**: реализовать

- поиск подстроки в тексте

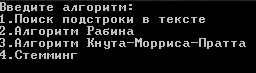
- алгоритм Рабина

- алгоритм Кнута - Морриса - Пратта

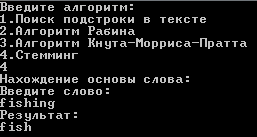
- стемминг.

**Было сделано:**

Меню программы



Демонстрация работы стемминга

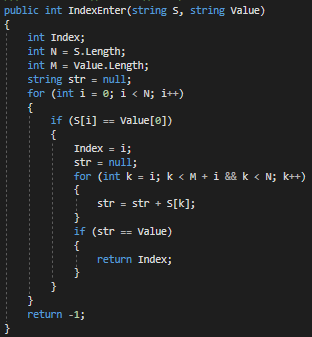


Алгоритмы:

* Поиска подстроки в строке.

Определяет индекс первого вхождения отдельного символа или подстроки в строке

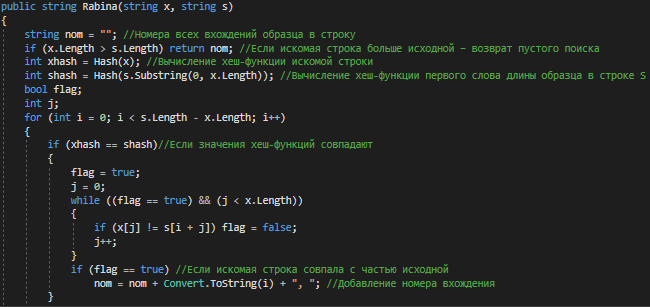
Код:

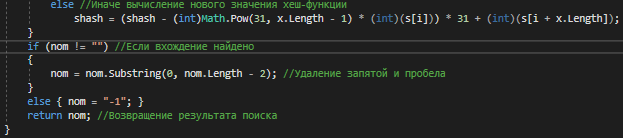


* Алгоритм Рабина

Определяет индексы всех отдельного символа или подстроки в строке

Код:

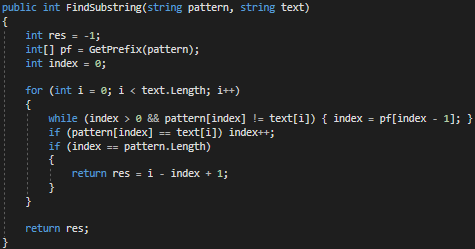




* Алгоритм Кнута – Морриса – Прата

Определяет индексы всех отдельного символа или подстроки в строке

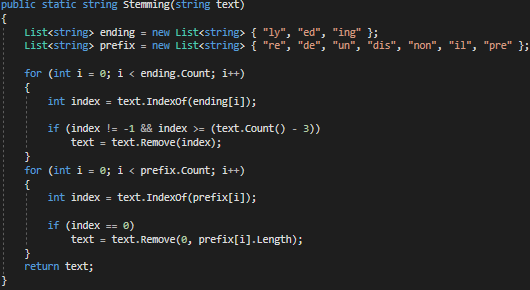
Код:



* Стемминг

Нахождение основы слова

Код:

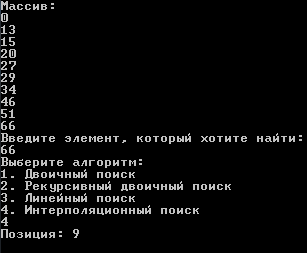


## “Алгоритмы поиска”

**Задача:** 4 алгоритма поиска элемента в массиве + рекурсивный вариант.

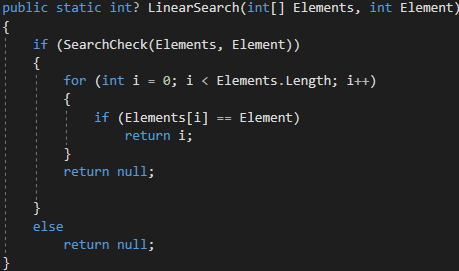
**Было сделано:**

Интерфейс:



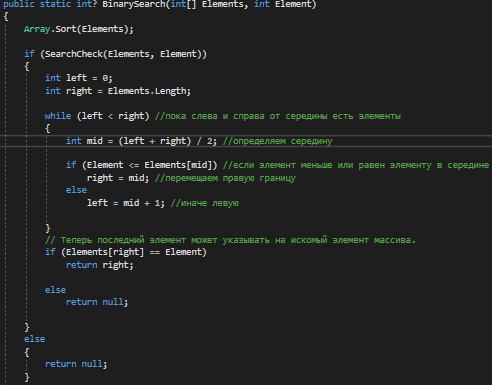
1.Линейный поиск

Самый простой метод. Перебираем последовательно все элементы массива и каждый сравниваем с искомым значением. Если они равны, то запоминаем индекс. Слабость алгоритма в том, что нам приходится перебирать все элементы, вне зависимости от того, где находится искомый элемент.



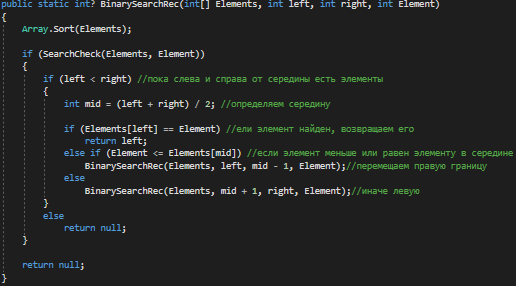
2.Двоичный поиск

Этот алгоритм подразумевает, что входной массив уже отсортирован (например, по алфавиту). Только в этом случае можно применять двоичный поиск. Здесь в каждой итерации массив делится на две части, и мы сравниваем срединное значение с искомым. Соответственно, принимаем решение, что может находиться только в левой или только в правой части массива (оказалось больше или меньше границы раздела). Отсекаем ту часть, в которой искомого значения быть не может, таким образом, сразу же уменьшая количество итераций в два раза. Так делаем на каждой следующей итерации, пока «граничное» значение не окажется тем, что мы ищем.



3.Рекурсивный двоичный поиск

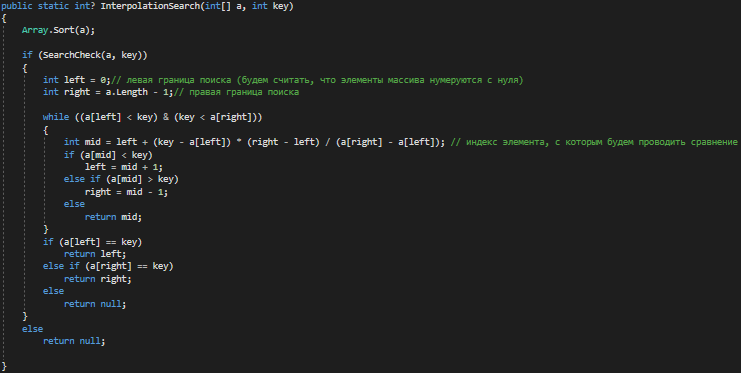
То же самое, но с применением рекурсии



4.Интерполяционный поиск

В основе интерполяционного поиска лежит операция интерполирование. Интерполирование – нахождение промежуточных значений величины по имеющемуся дискретному набору известных значений. Интерполяционный поиск работает только с упорядоченными массивами; он похож на бинарный, в том смысле, что на каждом шаге вычисляется некоторая область поиска, которая, по мере выполнения алгоритма, сужается.

Но в отличие от двоичного, интерполяционный поиск не делит последовательность на две равные части, а вычисляет приблизительное расположение ключа (искомого элемента), ориентируясь на расстояние между искомым и текущим значением элемента.



## “Алгоритмы сортировки”

**Задача:** по 2 из каждой группы(стратегии) сортировки.

- реальные данные

- сравнение времени работы сортировок (график) на одном наборе данных, но разной длины.

**Реализовано:**

Устойчивые сортировки:

Пузырьковая

Слиянием

Неустойчивые сортировки:

Быстрая сортировка

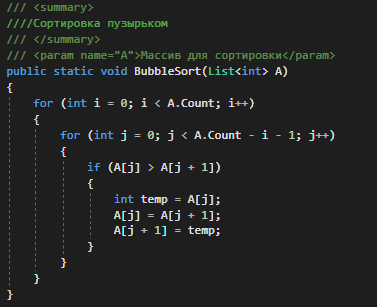
Сортировка выбором

Устойчивая (стабильная) сортировка — сортировка, которая не меняет относительный порядок сортируемых элементов, имеющих одинаковые ключи.

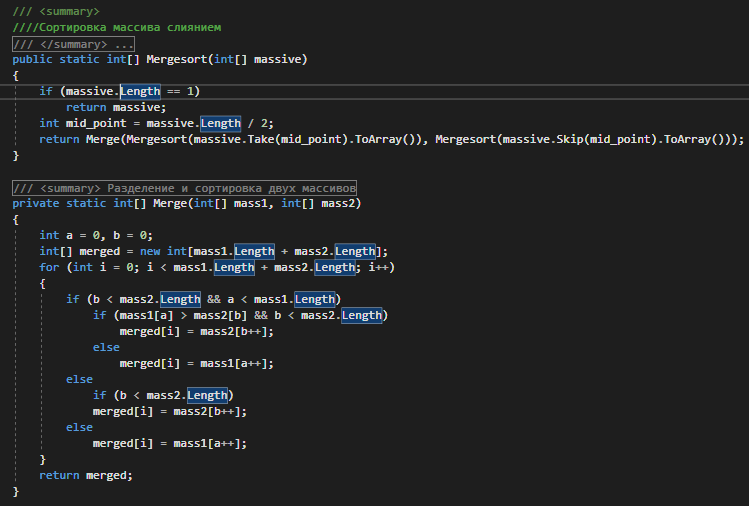
Устойчивость является очень важной характеристикой алгоритма сортировки, но, тем не менее, она практически всегда может быть достигнута путём удлинения исходных ключей за счёт дополнительной информации об их первоначальном порядке. Несмотря на кажущуюся необходимость, вытекающую из названия, устойчивость совсем не обязательна для правильности сортировки и чаще всего не соблюдается, так как для её обеспечения практически всегда необходимы дополнительная память и время.

Сортировка Пузырьком

Принцип действий прост: обходим массив от начала до конца, попутно меняя местами неотсортированные соседние элементы. В результате первого прохода на последнее место «всплывёт» максимальный элемент. Теперь снова обходим неотсортированную часть массива (от первого элемента до предпоследнего) и меняем по пути неотсортированных соседей. Второй по величине элемент окажется на предпоследнем месте. Продолжая в том же духе, будем обходить всё уменьшающуюся неотсортированную часть массива, запихивая найденные максимумы в конец.

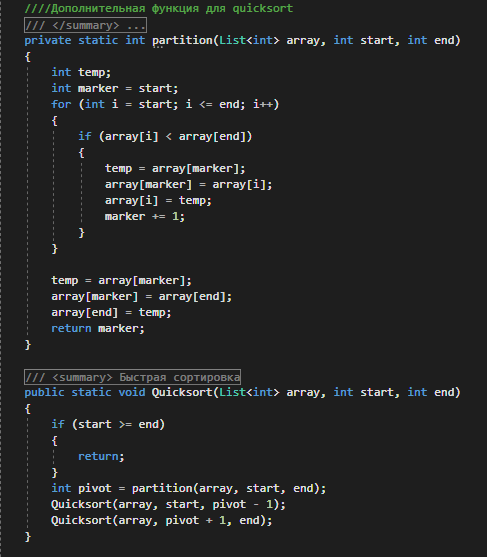


Сортировка слиянием (англ. *merge sort*) — алгоритм сортировки, который упорядочивает списки (или другие структуры данных, доступ к элементам которых можно получать только последовательно, например — потоки) в определённом порядке. Эта сортировка — хороший пример использования принципа «разделяй и властвуй». Сначала задача разбивается на несколько подзадач меньшего размера. Затем эти задачи решаются с помощью рекурсивного вызова или непосредственно, если их размер достаточно мал. Наконец, их решения комбинируются, и получается решение исходной задачи.

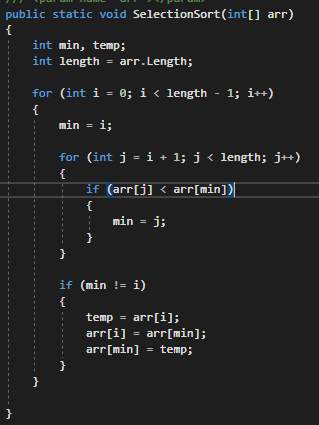


Неустойчивые сортировки

**Быстрая сортировка**, сортировка Хоара (англ. quicksort), часто называемая qsort (по имени в стандартной библиотеке языка Си) — широко известный алгоритм сортировки, разработанный английским информатиком Чарльзом Хоаром во время его работы в МГУ в 1960 году. Один из самых быстрых известных универсальных алгоритмов сортировки массивов: в среднем O(nlog n) обменов при упорядочении n элементов; из-за наличия ряда недостатков на практике обычно используется с некоторыми доработками.



**Сортировка выбором** (Selection sort) — алгоритм сортировки. Может быть как устойчивый, так и неустойчивый. На массиве из n элементов имеет время выполнения в худшем, среднем и лучшем случае Θ(n2), предполагая что сравнения делаются за постоянное время



## “Стек”

**Задача:** реализовать стек, не используя встроенный класс Stack

Методы класса "Стек":

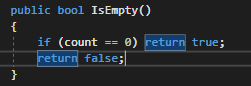
- добавление

- чтение

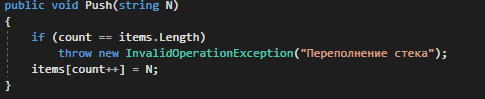
**Реализовано:**

Стек (stack) — это структура данных, представляющая собой специализированным образом организованный список элементов. Доступ к элементам стека осуществляется по принципу LIFO (Last In First Out) — последним пришел, первым вышел. Принцип работы данной структуры данных схож с магазином автоматического огнестрельного оружия. Патроны помещаются в магазин сверху вниз, а используется всегда только верхний патрон.

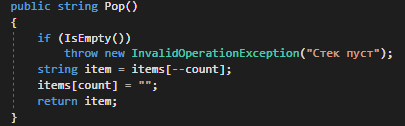
Проверка на пустоту



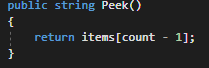
Поместить в стек



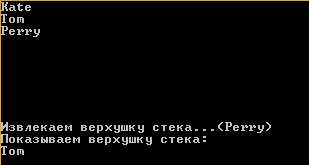
Извлечь верхушку



Показать верхушку



Пример работы стека



## “Очередь”

**Задача:** реализовать очередь, не используя встроенный класс Queue

- "простая"

- кольцевая

- с приоритетами.

## “Дек”

**Задача:** реализовать структуру данных дек

Методы класса "Дек":

- добавление

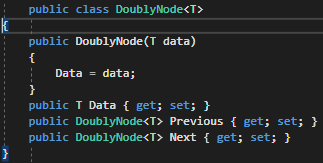
- чтение

- проверка на пустоту.

Реализация:

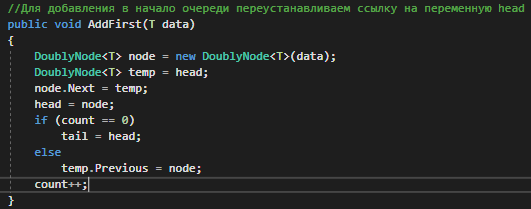
Дек (deque) представляет двустороннюю очередь, в которой элементы можно добавлять как в начало, так и в конец. Удаление также может идти как с начала, так и с конца.

Для начала создадим вспомогательный класс.

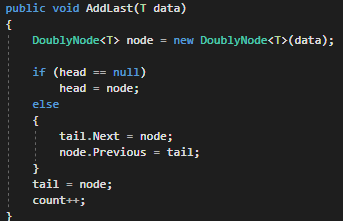


В отличие от класса очереди здесь определены два метода для добавления и два для удаления.

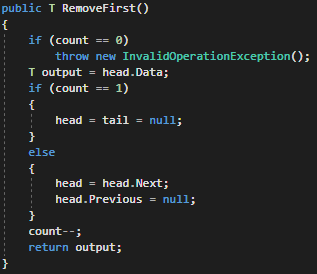
Для добавления в начало очереди переустанавливаем ссылку на переменную head:



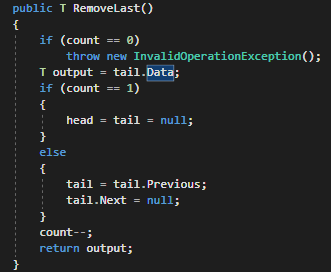
Добавление элемента в конец дека вызывает аналогичную переустановку переменной tail:



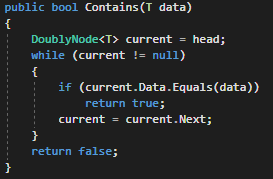
При удалении из начала дека надо переустановить ссылку на первый элемент:



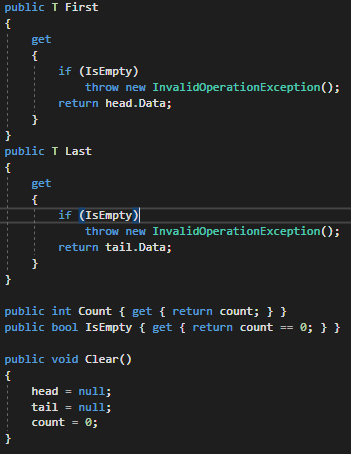
При удалении последнего элемента переустанавливается переменная tail на предпоследний элемент:



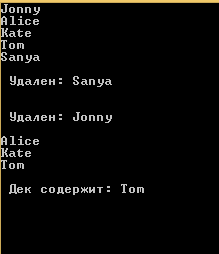
Есть метод для проверки на наличие элемента



Так же присутствуют методы проверки на пустоту, очищения Дека.



Пример работы



## “Работа со списками”

Задача: реализовать список, не используя встроенный класс List

Методы класса "Список":

- добавление

- удаление

- копирование части списка

- слияние двух списков (2 варианта: с и без создания нового списка)

- проверка на пустоту.