Министерство науки и высшего образования РФ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Новгородский государственный университет имени Ярослава Мудрого»

Кафедра информационных технологий и систем

**МЕТОДЫ РАЗРАБОТКИ ФУНКЦИОНАЛЬНЫХ ПРОГРАММ**

Лабораторная работа по дисциплине:

«Функциональное и логическое программирование»

Выполнил студент группы 8091:

\_\_\_\_\_\_\_\_\_\_ / Васильев И.В.

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

Принял преподаватель:

\_\_\_\_\_\_ / Михайлов Д.В.

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

**Великий Новгород**

**2021**

1. **Цель и задач**

Целью работы является изучение основных методов разработки функциональных программ с позиций строго функционального языка.

Основные задачи:

− освоить приемы нисходящего и восходящего проектирования функциональных программ;

− научиться выделять основные и вспомогательные функции с учетом разбиения задачи на подзадачи;

− овладеть приемами использования накапливающих параметров во вспомогательных функциях;

− ознакомиться с упреждающим использованием результата вызова функции.

Задание №1

Написать программу сортировки списка методом Шелла. Вычисление последовательности шагов сортировки производится в соответствии с Методом Р. Седжвика.

Задание №2

Написать программу сортировки списка методом пузырька.

Задание №3

Сравнить эффективность реализованной сортировки в Задании 2 и реализованного в Задании 1 варианта сортировки Шелла.

Задание №4

Написать программу объединения двух отсортированных списков в один. При этом порядок сортировки в списке-результате должен сохраняться.

Задание №5

Написать программу, которая в исходном списке заменяет все элементы-символы соответствующими им ASCII-кодами. Список может содержать подсписки произвольной глубины вложения.

1. **Решение:**

Задание №1

Теория:

Сортировка Шелла это, по-сути, модификация схем сортировки других алгоритмов. Фактически для сортировки элементов используются другие алгоритмы, такие как: пузырьком, вставками, выбором и т.д. Но только эти алгоритмы применяются не ко всей исходной последовательности, а к ее частям.

Сначала в исходной последовательности сортируются между собой элементы, отстоящие друг от друга на расстоянии n/2 элементов, затем на расстоянии n/4 и т.д. до тех пор пока не получим 2 последовательности, элементы которых отстоят друг от друга на расстоянии 1-го элемента. После этого делаем сортировку этой полученной последовательности выбранным методом и на выходе имеем уже полностью отсортированную последовательность.

Возникает вопрос: зачем же были предыдущие сортировки? Для того, чтобы расположить сортируемые элементы наиболее близко к своим положенным позициям. А в этом случае в последней сортировке по всей последовательности значительно сокращается количество перестановок.

Приращение в сортировке Шелла - это расстояние между сортируемыми элементами динамически меняющееся на каждом проходе. Главное требование, чтобы на последней итерации оно было равно 1. Динамика изменения этой величины очень существенно сказывается на производительности алгоритма в целом.

Очевидно, что программист может выбрать любой алгоритм уменьшения этого приращения на каждом шаге, главное, чтобы в конце оно приняло значение 1. Существует немало стратегий рассчета приращения на каждом проходе алгоритма Шелла.

Например, Р. Седжвик, предложил такую схему вычисления прирашений:  
d[i] = 9\*2i - 9\*2i/2 + 1, если i четно  
d[i] = 8\*2i - 6\*2(i+1)/2 + 1, если i нечетно

Было доказано, что используя эту схему производительность алгоритма возрастает ~ O(n7/6) в среднем и до ~ O(n4/3) в худшем случае. При расчете приращений по этому методу останавливаться следует на значении d[i-1], если 3\*d[i] > n. Обычно расчет начинается с нулевых значений i(=[0,1,2...]) и продолжается до такого i, когда 3\*d[i+1] > n, как было сказано ранее. Т.о. данная процедура рассчета запускается перед самой сортировкой Шелла и затем хранит полученную таблицу приращений в памяти, а алгоритм сортировки на каждом шаге к ней обращается за очередным значением.

Решение:

Код программы прикреплён в файле c названием “Васильев Lab31”

Пример работы программы:



Рисунок 1 При: (shell-sort '(2 3 9 2 8 4 6 8 11 12 4 6 7 8 9 999))

Задание №2

Теория:

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется перестановка элементов. Проходы по массиву повторяются {\displaystyle N-1} раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован. При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива («всплывает» до нужной позиции, как пузырёк в воде — отсюда и название алгоритма).

Решение:

Код программы прикреплён в файле c названием “Васильев Lab32”

Пример работы программы:



Рисунок 2 (buble-sort '(3 1 9 2 8 4 6 8 11 12 4 6 7 8 9 999))

Задание №3

Для вычисление затраченного времени на алгоритм использовалась функция time

Алгоритм №1: 109.39 мс

Алгоритм №2: 203.126 мс

Вывод:

В моей реализации алгоритмов, быстрее всех оказался первый алгоритм: Сортировка Шелла с шагом Р.Сэджика”

Задание №4

Для выполнения поставленной задачи, я выбрал следующий алгоритм:

Сравнить элемент в первом списке с первым элементом во втором

Если в первом значение меньше, чем во втором, записываю первое значение и перехожу на следующий элемент

Если второе значение оказалось меньше, чем в первом, записываю значение из второго, после чего сравниваю этот же элемент в первом, со следующим элементом во втором

Если мы дошли до конца списка, выписываем все оставшиеся элементы в другом списке

Решение:

Код программы прикреплён в файле c названием “Васильев Lab33”

Пример работы программы:



Рисунок 3 Вывод, при (glue-sort '( 1 2 4 7 10) '(4 8 10 11 13))

Задание №5

Решение:

Код программы прикреплён в файле c названием “Васильев Lab34”

Пример работы программы:



Рисунок 4 При вызове: (translateToASCII '("A" "C" ("A" ("C"))((("!" "#")"&"))))

1. **Вывод**

Мною были изучены основные методы разработки функциональных программ с позиции строго функционального языка.