|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ | | | | | | |
|  | | |  | | |  | | |
|  | ФГАОУ ВО «Пермский государственный национальный исследовательский университет» | | | | | | |  |
|  | | |  | | |  | | |
|  | | ОТЧЕТ  О ВЫПОЛНЕНИИ ЗАДАНИЯ №6  ПО ДИСЦИПЛИНЕ СЯиТП  по теме:  «Сравнение парадигм программирования»  Вариант 4 | | | | |  | |
|  | | |  | | |  | | |
|  |  | | |  | Работу выполнил  Студент гр. МФТИ-1,  2 курс        Бешляга Г.К. | | |  |
|  | | | Пермь 2023    **1.** Постановка задачи  Многочлен задается списком пар (коэффициент, степень). Вычислить произведение двух многочленов. Ответ вывести в естественном виде (например, x^5+4x^3-x+1). Исходные многочлены могут быть не отсортированы по степени.  2. Алгоритм  Реализация на C#:   1. Представим каждое слагаемое многочлена в виде кортежа с коэффициентом перед x и степенью x 2. Представим два многочлена в виде двух списков кортежей с коэффициентами и степенями x 3. Проходим по каждому слагаемое первого полинома и умножим его на каждое слагаемое второго полинома 4. Просуммируем коэффициенты слагаемых с одинаковыми степенями и удалим слагаемые с нулевыми коэффициентами 5. Отсортируем результаты по убыванию степеней и вернем список кортежей, представляющих умножение полиномов     Реализация на Haskell:   * 1. Создаем новый класс Polynomial, являющийся кортежем из двух целых чисел   2. Создаем функцию multiplyPolynomials, принимающую два полинома poly1 и poly2 в виде списков пар (коэффициент, степень).   3. Внутри функции создается список multiplied, в котором каждый элемент представляет собой произведение пары элементов из poly1 и poly2. Каждый элемент в multiplied имеет вид (c1\*c2, e1+e2), где c1 и c2 - коэффициенты, а e1 и e2 - степени соответствующих элементов.   4. Затем список multiplied сортируется по второму элементу (степени) и группируется с помощью функции groupBy, чтобы объединить элементы с одинаковыми степенями.   5. После группировки, каждая группа суммируется с помощью функции map, чтобы получить один элемент суммы для каждой степени.   6. Затем происходит фильтрация списка, чтобы удалить элементы с нулевым коэффициентом, и сортировка по убыванию степеней.   7. Результат умножения полиномов возвращается в виде списка пар (коэффициент, степень).   3. Сравнение использованных ЯП    **Трудоёмкость разработки**  Haskell:   * Прочтение документации – 4 часа * Написание кода – 9 часов * Откладка и тестирование программы – 4 часа   Итог: на написание программы на функциональном языке Haskell было потрачено 17 часов  C#:   * Написание кода – 3 часа * Откладка и тестирование программы – 2 часа   Итог: на написание программы на объектно-ориентированном языке C# был потрачен 5 часов.    **Трудоемкость тестирования и отладки**  Haskell:   * Ошибки синтаксиса. На исправление потребовалось достаточно много времени ввиду того, что Haskell не указывал явно на ошибку или делал это неточно. В остальных случаях проблем не происходило. * С тестированием программ проблем не возникло.   С#:   * Проблем с тестированием и отладкой не возникло.   **Объём полученного кода**  Haskell:  Строки алгоритма: 23 строки  Строки тестов: 47 строк  C#:  Строки алгоритма: 50 строк  Строки тестов: 205 строк  **Быстродействие, скорость работы**  Для анализа рассмотрим тесты для проверки корректности работы программ.  Рассмотрим результаты работы программы и время их выполнения путем запуска каждого теста 10000 раз (в миллисекундах):  Для следующих тестов будут рассмотрены следующие пары множителей   * Тест 1:   Polynomial 1: (x+1)  Polynomial 2: (x+1)  Haskell:  Время выполнение программы: 1,5 ms  С#:  Время выполнение программы: 14 ms   * Тест 2:   Polynomial 1: Многочлен с 50 маленькими положительными коэффициентами и отсортированными степенями    Polynomial 2: Многочлен с 50 маленькими положительными коэффициентами и отсортированными степенями  Haskell:  Время выполнение программы: 6 ms  С#:  Время выполнение программы: 593 ms   * Тест 3:   Polynomial 1: Многочлен с 50 маленькими положительными коэффициентами и неотсортированными степенями  Polynomial 2: Многочлен с 50 маленькими положительными коэффициентами и неотсортированными степенями  Haskell:  Время выполнение программы: 7 ms  С#:  Время выполнение программы: 595 ms   * Тест 4:   Polynomial 1: Многочлен с 50 большими положительными коэффициентами и неотсортированными степенями  Polynomial 2: Многочлен с 50 большими положительными коэффициентами и неотсортированными степенями  Haskell:  Время выполнение программы: 9 ms  С#:  Время выполнение программы: 1937 ms   * Тест 5:   Polynomial 1: Многочлен с 50 большими коэффициентами разного знака и неотсортированными степенями  Polynomial 2: Многочлен с 50 большими коэффициентами разного знака и неотсортированными степенями  Haskell:  Время выполнение программы: 12 ms  С#:  Время выполнение программы: 2130 ms | | |  | | |

Исходя из вышеперечисленных данных можно сделать вывод о том, что программа, реализованная на C#, работает медленнее чем Haskell на больших размерах полиномов, при отсортированности степеней и при отрицательных коэффициентах.

В целом программа на Haskell конкретно для данного алгоритма работает быстрее на любых массивах данных, объем кода на Haskell занимает меньше строк, но на изучение Haskell было потрачено больше времени чем на C#