|  |  |  |
| --- | --- | --- |
| Пермский филиал федерального государственного автономного образовательного учреждения высшего образования «Национальный исследовательский университет «Высшая школа экономики»  *Факультет экономики, менеджмента и бизнес-информатики*  Образовательная программа бакалавриата «Программная инженерия» | | |
|  | | |
| **Отчёт**  **по учебной практике** | | |
|  | | |
|  | | Выполнил студент группы ПИ-16-1  Валеев Марат Фаритович  (Фамилия, Имя, Отчество)  (подпись) |
| Проверил  Руководитель практики:  Преподаватель кафедры информационных технологий в бизнесе  (должность, учёная степень, учёное звание)  Сахипова Марина Станиславовна  (Фамилия, Имя, Отчество) | |  |
| (оценка) | (подпись)  (дата) |  |

Оглавление

[Введение 4](#_Toc491893261)

[Задача №1. 5](#_Toc491893262)

[Задача №2. 6](#_Toc491893263)

[Задача №3. 7](#_Toc491893264)

[Задача №4. 9](#_Toc491893265)

[Задача №5. 11](#_Toc491893266)

[Задача №6. 13](#_Toc491893267)

[Задача №7. 16](#_Toc491893268)

[Задача №8. 17](#_Toc491893269)

[Задача №9. 19](#_Toc491893270)

[Задача №10. 22](#_Toc491893271)

[Задача №11. 24](#_Toc491893272)

[Задача №12. 27](#_Toc491893273)

[Заключение 30](#_Toc491893274)

[Библиографический список 31](#_Toc491893275)

[Приложение А. Блок-схемы 32](#_Toc491893276)

[Приложение Б. Листинг программы задачи №1 35](#_Toc491893277)

[Приложение В. Листинг программы задачи №2 36](#_Toc491893278)

[Приложение Г. Листинг программы задачи №3 37](#_Toc491893279)

[Приложение Д. Листинг программы задачи №4 38](#_Toc491893280)

[Приложение Е. Листинг программы задачи №5 39](#_Toc491893281)

[Приложение Ж. Листинг программы задачи №6 40](#_Toc491893282)

[Приложение И. Листинг программы задачи №7 41](#_Toc491893283)

[Приложение К. Листинг программы задачи №8 42](#_Toc491893284)

[Приложение Л. Листинг программы задачи №9 43](#_Toc491893285)

[Приложение М. Листинг программы задачи №10 44](#_Toc491893286)

[Приложение Н. Листинг программы задачи №11 45](#_Toc491893287)

[Приложение О. Листинг программы задачи №12 46](#_Toc491893288)

Введение

Актуальность данной работы заключается в развитии и закреплении практических навыков создания программных систем с использованием современных сред разработки (Microsoft Visual Studio, язык C#), а также приобретение опыта проектирования, реализации, тестирования и отладки программных систем.

Алгоритмизация — это описание очередности выполнения различных операций, необходимых для решения той или иной задачи в форме алгоритма.

Цель данной работы заключается в закрепление полученных знаний по дисциплинам «Введение в программную инженерию», «Дискретная математика», «Компьютерный практикум по основам алгоритмизации и методам программирования», «Программирование», а также приобретение навыков и опыта алгоритмизации задач.

Задачами учебной практики являются:

1. Развитие и закрепление практических навыков построения и описания алгоритмов для решения задач из разных предметных областей (численные методы, дискретная математика, структуры данных и др).
2. Развитие и закрепление практических навыков использования языков высокого уровня и современных сред разработки для реализации построенных алгоритмов.
3. Развитие и закрепление практических навыков объектно-ориентированного программирования.
4. Развитие практических навыков оформления отчетов о проделанной работе, публичного выступления с защитой проекта.

Описание задачи: два отрезка на плоскости заданы целочисленными координатами своих концов в декартовой системе координат. Требуется определить, существует ли у них общая точка.

Входные данные: восемь вещественных чисел, пары чисел разделены строчками.

Выходные данные: в зависимости от результата выводится “Yes” или “No”.

Решение: отрезки пересекаются тогда, когда концы каждого отрезка лежат по разные стороны от другого отрезка. Пользуемся косым произведением векторов. Если оба произведения косых произведений меньше либо равны нулю, то концы каждого из отрезков лежат по разные стороны относительного концов другого отрезка. Но, требуется проверить, что, хотя бы один из концов одного отрезка принадлежит другому отрезку, в случае если отрезки части одной прямой (произведение косых произведений равно нулю) [1-3]. Описание алгоритма решения приведено в виде блок-схемы (рис. 1.1). Код основной части (см. прил. Б).



Рисунок 1.1. Блок-схема задачи №1

Тестирование: Тестирование выполняется на сервере ресурса ACMP рис. (1.2).

C:\Users\user\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Снимок.png

Рисунок 1.2. Результаты тестирования задачи №1

Заключение: вводятся координаты четырех точек, задающих отрезки. В первой строке содержатся координаты первого конца первого отрезка, во второй - второго конца первого отрезка, в третьей и четвёртой - координаты концов второго отрезка. Все координаты – целые числа, не превосходящие 10000 по абсолютной величине. Выводится слово "Yes", если общая точка есть, или слово "No" - в противном случае.



Описание задачи: в компании «Gold&Silver Soft» разработан новый алгоритм кодирования натуральных чисел. Этот алгоритм основан на операции циклического сдвига числа. Циклическим сдвигом натурального числа N относительно числа K назовем операцию переноса K крайних слева цифр в десятичной записи числа N к цифре крайней справа. Обозначим через cyclic(N, K) число, получаемое из N посредством циклического сдвига относительно числа K.

В разработанном алгоритме предлагается для кодирования натурального числа N использовать число code(N) = N + cyclic(N, K) (где K – известный параметр). Вы работаете в «Gold&Silver Soft» и Вам была поручена разработка алгоритма декодирования, то есть восстановления числа N по числам code(N) и K.

Входные данные: два целых числа, разделённые строкой.

Выходные данные: одно целое число.

Решение: перебор цифр до нахождения нужного числа с проверкой на работоспособность конкретного варианта (то есть, если очевидно, что вариант не подходит, то переходим к следующему). Сначала требуется перебор длины числа N, так как она может быть равна длине конечного числа или на единицу меньше. Затем требуется перебор переносов в разрядах. Далее перебираются индексы цифры, с которого требуется начать и значения для этой цифры, исходя из значения подбираются и остальные цифры числа, пока число не будет найдено [1-3]. Описание алгоритма решения приведено в виде блок-схемы (см. прил. А.1., А.2). Код основной части (см. прил. В).

Тестирование: Тестирование выполняется на сервере ресурса ACMP рис. (1.2).

C:\Users\user\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Прохождение тестов.png

Рисунок 2.1. Результаты тестирования задачи №2

Заключение: в первой строке вводится целое число code(N). Во второй строке вводится целое число K. Ограничения: 10K+1 ≤ code(N) ≤ 1018, 1 ≤ K ≤ 17. Выводится целое число N. Если существует несколько вариантов восстановления числа N по числам code(N) и K, то выводится любой из них.



Описание задачи: пусть D –заштрихованная часть плоскости (рис. 3, в) и пусть u определяется по x и y следующим образом: u=x-y, если (x, y) ∈ D и u=xy+7 в противном случае [4].

Входные данные: два вещественных числа, разделённые пробелом

Выходные данные: вещественное число.

Решение: вывод формулы принадлежности точки к заштрихованной области на основе предоставленного рисунка и с помощью неё определяем ответ [1-3]. Описание алгоритма решения приведено в виде блок-схемы (рис. 3.1). Код основной части (см. прил. Г).



Рисунок 3.1. Блок-схема задачи №3

Тестирование [5]: для тестирования понадобилось три теста (табл. 3.1.) и шесть критериев чёрного ящика (см. табл. 3.2).

Таблица 3.1. Тесты



Таблица 3.2. Критерии чёрного ящика



Заключение: вводятся координаты x и y вещественного типа данных. Выводится u, так же вещественного типа данных.



Описание задачи: Вычислить бесконечную сумму с заданной точностью ε(ε > 0). Считать, что требуемая точность достигнута, если несколько первых слагаемых и очередное слагаемое оказалось по модулю меньше, чем ε, это и все последующие слагаемые можно уже не учитывать [4]. Вычислить:

Входные данные: вещественное число.

Выходные данные: вещественное число.

Решение: считаем до заданной точности и выводим ответ [1-3]. Описание алгоритма решения приведено в виде блок-схемы (рис. 4.1). Код основной части (см. прил. Д).



Рисунок 4.1. Блок-схема задачи №4

Тестирование [5]: для тестирования понадобилось два теста (см. табл. 4.1.) и четыре критериев чёрного ящика (см. табл. 4.2).

Таблица 4.1. Тесты



Таблица 4.2. Критерии чёрного ящика



Заключение: вводится точность (ε) вещественного типа данных. Выводится бесконечная сумма, так же вещественного типа данных.



Описание задачи: дана действительная квадратная матрица порядка n. Найти наибольшее из значений элементов, расположенных в заштрихованной части матрицы (правый нижний угол вместе с диагональю) [4].

Входные данные: целое число и ряд вещественных чисел.

Выходные данные: вещественное число.

Решение: обходим только нужные нам элементы и ищем среди них максимум, в конце выводим [1-3]. Описание алгоритма решения приведено в виде блок-схемы (рис. 5.1). Код основной части (см. прил. Е).



Рисунок 5.1. Блок-схема задачи №5

Тестирование [5]: для тестирования понадобилось три теста (см. табл. 5.1.) и шесть критериев чёрного ящика (см. табл. 5.2).

Таблица 5.1. Тесты



Таблица 5.2. Критерии чёрного ящика



Заключение: вводится размерность матрицы и сама матрица. Выводится максимум из заданной части матрицы.



Описание задачи: ввести . Построить последовательность чисел = + + . Довести ее до <= М. Напечатать последовательность, N. Сообщить, выполняется ли равенство = М [4].

Входные данные: четыре вещественных числа, разделённых пробелами.

Выходные данные: последовательность вещественных чисел, целое число, сообщение о выполнении равенства.

Решение: сперва проверяем первые элементы относительно M. Затем вычисляем и выводим числовую последовательность [1-3]. Описание алгоритма решения приведено в виде блок-схемы (рис. 6.1). Код основной части (см. прил. Ж).



Рисунок 6.1. Блок-схема задачи №6

Тестирование [5]: для тестирования понадобилось девять тестов (табл. 6.1.) и двадцать критериев чёрного ящика (табл. 6.2. и см. табл. 6.3).

Таблица 6.1. Тесты



Таблица 6.2. Критерии чёрного ящика (входные данные)



Таблица 6.3. Критерии чёрного ящика (выходные данные)



Заключение: вводятся элементы . Выводится последовательность длины N, само N и сообщение о том, что выполняется ли равенство = М.



Описание задачи: выписать все булевы функции от 3 аргументов, которые не линейны. Выписать их вектора в лексикографическом порядке [4].

Входные данные: отсутствуют.

Выходные данные: список булевых функций.

Решение: проверяем линейность поверхностно (у линейных функций среди значений нулей и единиц всегда четное число), затем по методу Паскаля и выводим нелинейные [1-3]. Описание алгоритма решения приведено в виде блок-схемы (рис. 7.1). Код основной части (см. прил. и).



Рисунок 7.1. Блок-схема задачи №7

Тестирование: не требуется.

Заключение: ничего не вводится. Выводятся все булевы функции от 3 аргументов, которые не линейны, в лексикографическом порядке.



Описание задачи: граф задан матрицей смежности. Выяснить, является ли он деревом. Для тестирования программы разработать генератор тестов, который позволит сгенерировать набор входных данных, используемых при тестировании [4].

Входные данные: целое число.

Выходные данные: матрицы и сообщения под ними.

Решение: если в ряду одни нули (значит граф не связный) или кол-во рёбер больше n-1 (у деревьев кол-во рёбер на 1 меньше, чем вершин), то граф не является деревом [1‑3]. Описание алгоритма решения приведено в виде блок-схемы (рис. 8.1). Код основной части (см. прил. К).



Рисунок 8.1. Блок-схема задачи №8

Тестирование [5]: для тестирования понадобилось два теста (табл. 8.1.) и четыре критерия чёрного ящика (табл. 8.2).

Таблица 8.1. Тесты



Таблица 8.2. Критерии чёрного ящика



Заключение: вводится количество генерируемых матриц смежности графов. Выводятся сгенерированные матрицы смежности графов и сообщения о том, являются ли они (графы) деревьями.



Описание задачи: создать циклический список, с возможностью поиска и удаления элементов (всё через рекурсию). В информационные поля элементов заносятся номера с 1 до N (N водится с клавиатуры). Первый элемент в списке, имеющий номер 1, оказывается в хвосте списка (последним) [4].

Входные данные: выбор пункта меню и целое число, если необходимо.

Выходные данные: список целых чисел, целое число или сообщение об ошибке, сообщение об успешной операции или об ошибке, выходные данные зависят от выбранного действия.

Решение: создание класса циклического списка и методов его создания, поиска и удаления его элементов [1-3]. Описание алгоритма решения приведено в виде блок-схемы (рис. 9.1, см. рис.9.2. и см. прил.А.3). Код основной части (см. прил. Л).



Рисунок 9.1. Функция CreateList



Рисунок 9.2. Функция FimdElIndex

Тестирование [5]: для тестирования понадобилось три теста (табл. 9.1.) и двенадцать критериев чёрного ящика (см. табл. 9.2).

Таблица 9.1. Тесты



Таблица 9.2. Критерии чёрного ящика



Заключение: вводится длина циклического списка, значение элемента для поиска в списке, значение элемента для удаления в списке. Выводится циклический список заданной длины, позиция заданного элемента или сообщение об ошибке, сообщение об удаление элемента или об ошибке, соответственно.



Описание задачи: написать метод уничтожения дерева [4].

Входные данные: целое число.

Выходные данные: отсутствуют.

Решение: создание класса бинарного дерева с деконструктором, генерация дерева, присваивание ему null [1-3]. Описание алгоритма решения приведено в виде блок‑схемы (рис. 10.1). Код основной части (см. прил. М).



Рисунок 10.1. Функция Destruct

Тестирование [5]: для тестирования понадобилось два теста (табл. 10.1.) и четыре критерия чёрного ящика (см. табл. 10.2).

Таблица 10.1. Тесты



Таблица 10.2. Критерии чёрного ящика



Заключение: вводится количество элементов в дереве. Выходные данные отсутствуют.



Описание задачи: доказать, что матрица [a,y]i=1..10, j=1..10 может служить ключом шифра, если из элементов: a[i,j], a[10-i+1, j], a[i, 10-j+1], a[10-i+1, 10-j+1], только один равен нулю [4].

Входные данные: отсутствуют.

Выходные данные: матрица, два текста.

Решение: создание генераторов текста и матрицы-ключа, методов шифрования и дешифрования [1-3]. Описание алгоритма решения приведено в виде блок-схемы (рис. 11.1. и см. рис. 11.2). Код основной части (см. прил. Н).



Рисунок 11.1. Функция Encrypt



Рисунок 11.2. Функция Decrypt

Тестирование [5]: для тестирования понадобилось один тест (табл. 11.1.) и двенадцать критериев чёрного ящика (см. табл. 11.2).

Таблица 11.1. Тесты



Таблица 11.2. Критерии чёрного ящика



Заключение: ничего не вводится. Выводится зашифрованный и расшифрованный текст, с помощью сгенерированных текста и матрицы-ключа для шифрования.



Описание задачи: выполнить сравнение двух методов сортировки (простыми вставками и с помощью двоичного дерева) одномерных массивов, содержащих n элементов, по количеству пересылок и сравнений. Провести анализ методов сортировки для трех массивов: упорядоченного по возрастанию, упорядоченного по убыванию и неупорядоченного.

Найти в литературе теоретические оценки сложности каждого из методов и сравнить их с оценками, полученными на практике. Сделать выводы о том, насколько отличаются теоретические и практические оценки количества операций, объяснить почему это происходит. Сравнить оценки сложности двух алгоритмов [4].

Входные данные: целое число.

Выходные данные: текст.

Решение: генерация массивов трёх типов (количество элементов вводит пользователь), их сортировка с помощью простых вставок и двоичным деревом. Также вывод счётчиков пересылок и сравнений, теоретической сложности и объяснения их различия [1-3]. Описание алгоритма решения приведено в виде блок-схемы (рис. 12.1. и см. рис. 12.2). Код основной части (см. прил. О).



Рисунок 12.1. Функция InsertionSort



Рисунок 12.2. Функция Insert

Тестирование [5]: для тестирования понадобилось два теста (табл. 12.1.) и четыре критерия чёрного ящика (см. табл. 12.2).

Таблица 12.1. Тесты



Таблица 12.2. Критерии чёрного ящика



Заключение: вводятся количество элементов для сортировки. Выводится сравнение двух методов сортировки (простыми вставками и с помощью двоичного дерева) одномерных массивов по количеству пересылок и сравнений для трёх массивов: упорядоченного по возрастанию, упорядоченного по убыванию и неупорядоченного. Теоретические оценки сложности и полученные на практике, также объяснение их различия.

Заключение

Подводя итоги, были спроектированы, разработаны и протестированы двенадцать задач; развиты и закреплены практические навыки построения и описания алгоритмов для решения задач из разных предметных областей (численные методы, дискретная математика, структуры данных и др). Были развиты и закреплены практические навыки использования языков высокого уровня и современных сред разработки для реализации построенных алгоритмов. Были развиты и закреплены практические навыки объектно-ориентированного программирования. Были развиты практические навыки оформления отчетов о проделанной работе и публичного выступления с защитой проекта. По окончанию работы получены приложения выполняющие поставленные задачи. Таким образом все задачи были выполнены, и как следствие – цель.

Библиографический список

1. Мацяшек Л.А. Практическая программная инженерия на основе учебного примера: пер. с англ. – М.: БИНОМ. Лаборатория знаний, 2011. — 948 с.
2. Терехов А.Н. Технология программирования: учебное пособие. / А.Н. Терехов: – 2-е изд. – М.: БИНОМ. Лаборатория знаний, 2012. — 149 с.
3. Троелсен Э. Язык программирования C# 2010 и платформа .NET 4: пер. с англ. – М.: Издательский дом "Вильямс", 2012. — 1392 c.
4. Абрамов С.А., Гнездилова Г.Г., Капустина Е.Н., Селюн М.И. Задачи по программированию. М.: Наука, 1988. — 569 c.
5. Плаксин М.А., Тестирование и отладка программ для профессионалов будущих и настоящих / М.А. Плаксин — М.: Бином. Лаборатория знаний, 2007. — 169 c.
6. Блок-схемы



Рисунок А.1. Функция Decode



Рисунок А.2. Функция FoundNDigits



Рисунок А.3. Функция RemoveEl

1. Листинг программы задачи №1

public static void DoLineCross(string[] coords)

{ // Получает координаты

long x1 = long.Parse(coords[0]), x2 = long.Parse(coords[2]), x3 = long.Parse(coords[4]), x4 = long.Parse(coords[6]), // Координаты х и у

y1 = long.Parse(coords[1]), y2 = long.Parse(coords[3]), y3 = long.Parse(coords[5]), y4 = long.Parse(coords[7]);

bool pseudoscalarProduct1 = ((x2 - x1) \* (y4 - y1) - (x4 - x1) \* (y2 - y1)) \* ((x2 - x1) \* (y3 - y1) - (x3 - x1) \* (y2 - y1)) <= 0, // Произведение косых произведений: [P1P2, P1P4] \* [P1P2, P1P3], P1-P4, имеют координаты с соответсвующими индексами, требуется для того, чтобы проверить, что концы каждого из отрезков лежат по разные стороны относительного концов другого отрезка

pseudoscalarProduct2 = ((x4 - x3) \* (y1 - y3) - (x1 - x3) \* (y4 - y3)) \* ((x4 - x3) \* (y2 - y3) - (x2 - x3) \* (y4 - y3)) <= 0, // Произведение косых произведений: [P3P4, P3P1] \* [P3P4, P3P2], P1-P4, имеют координаты с соответсвующими индексами, требуется для того, чтобы проверить, что концы каждого из отрезков лежат по разные стороны относительного концов другого отрезка

scalarProduct1 = (x1 - x3) \* (x2 - x3) + (y1 - y3) \* (y2 - y3) <= 0, // Скалярное произведение (P3P1,P3P2)

scalarProduct2 = (x1 - x4) \* (x2 - x4) + (y1 - y4) \* (y2 - y4) <= 0, // Скалярное произведение (P4P1,P4P2)

scalarProduct3 = (x3 - x1) \* (x4 - x1) + (y3 - y1) \* (y4 - y1) <= 0, // Скалярное произведение (P1P3,P1P4)

scalarProduct4 = (x3 - x2) \* (x4 - x2) + (y3 - y2) \* (y4 - y2) <= 0; // Скалярное произведение (P2P3,P2P4)

if (pseudoscalarProduct1 && pseudoscalarProduct2 && // Если оба произведения косых произведений <=0 значит концы каждого из отрезков лежат по разные стороны относительного концов другого отрезка

(scalarProduct1 || scalarProduct2 || scalarProduct3 || scalarProduct4)) // Но, требуется проверить, что хотя бы один из концов одного отрезка принадлежит другому отрезку, в случае если отрезки части одной прямой (произведение косых произведений=0)

Console.WriteLine("Yes"); // Записывает "Yes", если общая точка есть

else Console.WriteLine("No"); // Или "No" - в противном случае

} // Выясняет пересекаются ли отрезки с помощью угловых коэффициентов

1. Листинг программы задачи №2

static long Decode(long code, int k)

{ // Число после кодирования и кол-во сдвигов

for (int nLen = code.ToString().Length - 1; nLen <= code.ToString().Length; nLen++) // Длина N может быть меньше на единицу либо равно cyclic(N, K), поэтому перебираются оба варианта

{

int[] sumDigits = LongToIntArr(code, nLen); // Массив с цифрами числа code

int[] nDigits = new int[nLen]; // Цифры N

for (int transfComb = 0; transfComb < Math.Pow(2, nLen); transfComb++) // Перебор 2^nLen (перебор комбинаций переносов в разряды)

{

for (int i = 0; i < nLen; i++) // Отмечаем каждый элемент как непроверенный

nDigits[i] = -1;

for (int start = 0; start < nLen; start++) // Перебираем элементы, от которых отталкиваемся

if (nDigits[start] == -1) // Если элемент ещё не посещён, то требуется подобрать ему нужное значение

{

bool digitsFound = false; // Найдены ли цифры

for (int i = 0; i <= 9; i++) // Перебор цифр

{

if (start == nLen - 1 && i == 0) // Первая цифра N не может быть равна нулю

continue;

nDigits[start] = i;

if (digitsFound = FoundNDigits(start, nDigits, k, transfComb, sumDigits)) break;

}

if (!digitsFound) // Если цифры не найдены, разрываем цикл

break;

}

long n = IntArrToLong(nDigits); // Переводит цифры в число типа long

if (IsNCorrect(n, nDigits, code, k)) // Если N+cyclic(N,k)==code, значит N найдено

return n;

}

}

return -1; // Недостижимо, если данные корректны

} // Декодориует code

static bool FoundNDigits(int start, int[] nDigits, int k, int transfComb, int[] sumDigits)

{ // Начальный индекс, цифры N, кол-во сдвигов, комбинация переносов и цифры code

int cur = start; // Текущий индекс

int nLen = nDigits.Length;

while (true) // Ищем значения пока не дойдём до первого или пока не найдётся ошибка

{

int next = (cur + nLen - k) % nLen, // Следующий индекс

carryFrom = (transfComb >> cur) & 1, // Есть ли перенос из этого разряда (бинарный сдвиг быстрее возведения в степень)

carryTo = cur > 0 ? (transfComb >> (cur - 1)) & 1 : 0, // Есть ли перенос в текущий разряд, у последних цифр перенос в этот разряд невозможен

nextDigitValue = carryFrom \* 10 + sumDigits[cur] - carryTo - nDigits[cur]; // Следующее значение

if (nextDigitValue < 0 || nextDigitValue > 9) // Если следуещее значение некорректно, значит и начальное тоже => требуется продолжить перебор

return false;

if (next == nLen - 1 && nextDigitValue == 0) // Если следующий элемент первый и равен нулю, выходим

return true;

if (next == start && nextDigitValue != nDigits[start]) // Если следующий индекс равен начальному и следующее значение не равно начальному, то начальная цифра не подходит

return false;

nDigits[next] = nextDigitValue; // Присваиваем следующему индексу следующее значение

cur = next; // Текущее становится следующим

if (cur == start) // Если пришло к началу, то перебор останавливается

return true;

}

} // Найдены ли цифры числа N

1. Листинг программы задачи №3

static void Main(string[] args)

{

Console.Write("Введите координату х: ");

double x = ReadDouble(); // Последовательное чтение х и у

Console.Write("Введите координату y: ");

double y = ReadDouble();

if (Math.Pow(x, 2) + Math.Pow(y - 1, 2) <= 1 & y <= 1 - Math.Pow(x, 2)) // Если точка принадлежит заданной области (см. задание в учебнике), то u=x-y, иначе u=xy+7

Console.WriteLine("u = "+(x-y));

else Console.WriteLine("u = " + (x\*y+7));

Console.ReadLine();

}

1. Листинг программы задачи №4

static double CountInfSum()

{

Console.Write("Введите точность: ");

double eps = ReadDouble(), sum = 1, nextEl = 2;

if (eps > sum) return sum; // Если эпсилон будет больше единицы

long fact = 1; // Переменная для хранения факториала

for (int i = 1; fact > 0 && Math.Abs(nextEl) > eps; i++) // Считать до заданной точности или пока факториал не сойдёт с ума

{

sum += nextEl;

fact \*= i;

nextEl = Math.Pow(-2, i) / fact;

}

return sum;

}

1. Листинг программы задачи №5

static double MatrixsLeftBottomPartMax(double[,]matrix)

{

int n = matrix.GetLength(0);

double max = int.MinValue;

for (int i = 0; i < n; i++) // Обход всех рядов

for (int j = n - 1 - i; j < n; j++) // Обход столбцов из заданной части матрицы (правый нижний угол вместе с диагональю)

if (matrix[i, j] > max)

max = matrix[i, j];

return max;

} // Поиск максимума из заданной части матрицы (правый нижний угол вместе с диагональю)

1. Листинг программы задачи №6

static void NumericalSequence(double a1, double a2, double a3, double m, int n)

{

double a4 = (a1 + a2 + a3)/2;

if (a4 == m)

{

Console.Write(a4 + "\nN = " + n + "\nКонец последовательности, аN=M");

return;

}

if (a4 > m)

{

Console.Write("\nN = " + (n-1) + "\nКонец последовательности, аN не равно M");

return;

}

Console.Write(a4 + " "); // Если элемент меньше М, то печатаем его и идём дальше

NumericalSequence(a2, a3, a4, m, n+1);

} // Вычисление и вывод числовой последовательности

1. Листинг программы задачи №7

static void WriteAllNonLinearVectorsFrom3Arguments()

{

bool[] vector = new bool[8];

Console.WriteLine("Вектора нелинейных булевых функций от 3-х аргументов:");

for (int i = 0; i < Math.Pow(2, 8); i++)

{

for (int j = 0, k = (int)Math.Pow(2, 7); j < vector.Length; j++, k/=2)

vector[j]= i / k % 2 != 0;

if (((vector[0] ? 1 : 0) + (vector[1] ? 1 : 0) + (vector[2] ? 1 : 0) +

(vector[3] ? 1 : 0) + (vector[4] ? 1 : 0) + (vector[5] ? 1 : 0) +

(vector[6] ? 1 : 0) + (vector[7] ? 1 : 0)) % 2!=0) WriteVector(vector); // У линейных ф-ий среди значений нулей и единиц всегда четное число

else if (!IsVectorLinear(vector)) WriteVector(vector); // Иначе проверяем на линейность и выписываем, если оказался нелинейным

}

Console.ReadKey();

} // Выводит все нелинейные вектора от 3-х аргументов

static bool IsVectorLinear(bool[] vector)

{ // Вектор, который требуется проверить

bool[,] pasc = new bool[3, vector.Length]; // Таблица из метода Паскаля (со второй строки, тк 1-ая - это вектор)

for (int i = 0; i < vector.Length; i++) // Заполнение 2-го ряда таблицы (снизу показан код без цикла)

pasc[0, i] = vector[i/2\*2] ^ (i % 2 != 0 & vector[i]);

//pasc[0, 0] = vector[0];

//pasc[0, 1] = vector[0] ^ vector[1];

//pasc[0, 2] = vector[2];

//pasc[0, 3] = vector[2] ^ vector[3];

//pasc[0, 4] = vector[4];

//pasc[0, 5] = vector[4] ^ vector[5];

//pasc[0, 6] = vector[6];

//pasc[0, 7] = vector[6] ^ vector[7];

for (int i = 0; i < vector.Length; i++) // Заполнение 3-го ряда таблицы (снизу показан код без цикла), двойным комментарием помечены ненужные ячейки (не нужны, тк важно лишь есть ли конъюнкция)

pasc[1, i] = pasc[0, i/2%2 != 0 ? i-2 : i] ^ (i/2%2 != 0 & pasc[0, i]);

////pasc[1, 0] = pasc[0, 0];

//pasc[1, 1] = pasc[0, 1];

//pasc[1, 2] = pasc[0, 0] ^ pasc[0, 2];

//pasc[1, 3] = pasc[0, 1] ^ pasc[0, 3];

////pasc[1, 4] = pasc[0, 4];

//pasc[1, 5] = pasc[0, 5];

//pasc[1, 6] = pasc[0, 4] ^ pasc[0, 6];

//pasc[1, 7] = pasc[0, 5] ^ pasc[0, 7];

for (int i = 0; i < vector.Length; i++) // Заполнение 4-го ряда таблицы (снизу показан код без цикла), двойным комментарием помечены ненужные ячейки (не нужны, тк важно лишь есть ли конъюнкция)

pasc[2, i] = pasc[1, i/4%2 != 0 ? i - 4 : i] ^ (i/4%2 != 0 & pasc[1, i]);

////pasc[2, 0] = pasc[1, 0];

////pasc[2, 1] = pasc[1, 1];

////pasc[2, 2] = pasc[1, 2];

//pasc[2, 3] = pasc[1, 3];

////pasc[2, 4]= pasc[1, 0] ^ pasc[1, 4];

//pasc[2, 5]= pasc[1, 1] ^ pasc[1, 5];

//pasc[2, 6] = pasc[1, 2] ^ pasc[1, 6];

//pasc[2, 7] = pasc[1, 3] ^ pasc[1, 7];

if (pasc[2, 3] || pasc[2, 5] || pasc[2, 6] || pasc[2, 7]) return false; // Эти ячейки отвечают за элементы с конъюнкцией в полиноме Жегалкина по таблицы по методу Паскаля

return true;

} // Линеен ли вектор

1. Листинг программы задачи №8

static bool IsTree(bool[,] adjacencyMatr)

{ // Матрица смежности

int edges = 0, n = adjacencyMatr.GetLength(0); // edges - количество рёбер и n - размерность матрицы и количество вершин

for (int i = 0; i < n; i++) // Проходимся повсем рядам

{

bool onlyZeroes = true; // Маркер того, что в ряду нет единиц

for (int j = 0; j < n; j++) // По всем столбцам

{

if (!adjacencyMatr[i, j]) continue; // Пропускаем нули

onlyZeroes = false; // Если 1, то меняем маркер и прибавляем рёбра

edges++;

}

if (onlyZeroes || edges / 2 >= n) return false; // Если в ряду одни нули (значит граф не связный) или кол-во рёбер больше n-1 (у деревьев кол-во рёбер на 1 меньше, чем вершин), то граф не является деревом

}

return true;

} // Проверяет является ли граф заданный матрицей смежности деревом

1. Листинг программы задачи №9

private Element CreateList(Element first, Element current, int length)

{

// Первый элемент, текущий, значение для присваивания и длина списка

if (length==0)

{

current.Next = first; // Соединяем конец с началом

return first; // Делаем, так, чтобы 1 была в конце

}

current.Next = new Element(length);

return CreateList(first, current.Next, length-1);

} // Рекурсивно создаёт циклический список

private int FindElIndex(Element el, int index, int value)

{

if (el.Value == value) return index;

if (index == Length) return -1;

el = el.Next;

return FindElIndex(el, index + 1, value);

} // Вспомогательная рекурсивная функция для поиска элемента по его значению

private int RemoveEl(int index, int value)

{

if (value == \_currentEl.Value)

{

if (\_currentEl == \_currentEl.Next) \_currentEl = null; // Если список длины один, то очищаем его

else

{

\_currentEl.Value = \_currentEl.Next.Value;

\_currentEl.Next = \_currentEl.Next.Next;

}

Length--;

return index;

}

if (index + 1 == Length && \_currentEl.Next.Value != value) // Если следующий элемент последний для проверки и его значение не равно искомому, следовательно его там нет

{

\_currentEl = \_currentEl.Next.Next; // Делаем изначальный порядок списка

return -1;

}

\_currentEl = \_currentEl.Next;

return RemoveEl(index+1,value);

} // Вспомогательная рекурсивная функция для удаления элемента по его значению

1. Листинг программы задачи №10

private void Destruct(BinTree tree)

{

if (tree.Left == null && tree.Right == null)

{

tree = null;

return;

}

if (tree.Left != null) Destruct(tree.Left);

if (tree.Right != null) Destruct(tree.Right);

} // Обходит дерево в глубину и присваивает листьям null

1. Листинг программы задачи №11

static char[,] Encrypt(string text, bool[,] keyMatr, char[,] encryptedMatr)

{ // Текст для шифровки, матрица-ключ, матрица с зашифрованным текстом

if (text == "") return encryptedMatr; // Если текст кончился, то шифрованная матрица зашифрована

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

if (keyMatr[i, j]) // Если в ключе-матрице "дыра", значит букву при наложение требуется переписать в зашифрованную матрицу

{

encryptedMatr[i,j] = text[0]; // Берём всегда первый элемент, т.к. после записи он будет удалён

text=text.Remove(0, 1); // Удаляем первый элемент

if (text.Length%(n\*n/4) == 0) // Если длина текста кратна четверти длины матрицы, значит обход закончен

return Encrypt(text, TurnMatrixBy90Clockwise(keyMatr), encryptedMatr); // Вызов рекурсии с поворотом матрицы-ключа на 90\* по часовой стрелке

}

return new char[0,0];

}

static string Decrypt(char[,] encryptedMatr, bool[,] keyMatr, string text)

{ // Зашифрованная матрица, матрица-ключ и расшифрованный текст

if (text.Length == n \* n) return text; // Если длина текста равна длине матрицы, значит дешифровка окончена

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

if (keyMatr[i, j]) // Если в ключе-матрице "дыра", значит букву при наложение требуется переписать в текст

{

text += encryptedMatr[i, j];

if (text.Length % (n \* n / 4) == 0) // Если длина текста равна четверти длины матрицы, то нужное количество букв на этоп этапе дешифровано

return Decrypt(encryptedMatr, TurnMatrixBy90Clockwise(keyMatr), text); // Вызов рекурсии, матрица-ключ поворачивается по часовой

}

return ""; // Недостижимый возврат в случае корректных данных

} // Рекурсивная дешифровка зашифрованной матрицы

1. Листинг программы задачи №12

static void InsertionSort(int[] arr, ref int movedAmount, ref int comparedAmount)

{ // Массив для сортировки, счётчик перессылок и счётчик сравнений

for (int i = 1; i < arr.Length; i++) // С 1, т.к. не требуется сравнивать 0-ой элемент

{

int j; // Начинаем с i, т.к. предыдущие элементы уже отсортированы

for (j=i;j > 0 && arr[j - 1] > arr[j]; j--) // Пока j>0 и элемент j-1 > j

{

comparedAmount++;

int temp = arr[j];

arr[j] = arr[j - 1];

arr[j - 1] = temp;

movedAmount++;

}

if (j!=0) comparedAmount++; // Если цикл кончился обнулением j, значит сравнение не прошло (IndexOutOfRange)

}

} // Сортировка вставками с счётчиками пересылок и сравнений

static BinTree Insert(BinTree tree, int el, ref int comparedAmount)

{ // Само дерево, элемент для вставки и кол-во сравнений (кол-во пересылок = кол-ву элементу)

if (tree == null) // Если дошли до пустого листка, то создаём его

{

tree = new BinTree(el);

return tree;

}

comparedAmount++; // Сравнение с левой веткой

if (el <= tree.Data) // Слева элементы меньше или равны

{

tree.Left = Insert(tree.Left, el,ref comparedAmount);

return tree;

}

comparedAmount++; // Сравнение с правой веткой

tree.Right = Insert(tree.Right, el, ref comparedAmount);

return tree;

} // Вставка элемента в двоичное дерево

static void SortAscendedArray(int arrLength)

{ // Кол-во элементов в массиве

int comparedAmount = 0, movedAmount = 0; // Счётчики сравнений и пересылок

int[] arr = GenerateAscendingArray(arrLength); // Создаём и заполняем массив данными, упорядоченными по возрастанию

InsertionSort(arr, ref movedAmount, ref comparedAmount); // Сперва сортировка вставками (требуется клонирование, т.к. элементы в массиве - сслыки => массив отсортируется и в этой функции), затем вывод полученных данных с теорией и объяснение несовпадений

Console.WriteLine("\n\nСортировка простыми вставками массива упорядоченного по возрастанию (лучшее время: O(n) сравнений, 0 обменов):\nКоличество пересылок = " +

movedAmount + ", количество сравнений равно = " + comparedAmount +

" (не совпадает с формулой т.к. O(n-1)=O(n), 1-ый эл. не сравнивается)");

comparedAmount = 0;

BinTreeSort(arr, ref comparedAmount); // Затем сортировка бинарным деревом. После, вывод полученных данных с теорией и объяснение несовпадений

Console.WriteLine("\nСортировка двоичным деревом массива упорядоченного по возрастанию (худшее время (т.к. не сбалансированное и сравнение '>=' идёт после '<'): O(n^2) сравнений, O(n) обменов):\nКоличество пересылок = " +

arrLength + ", количество сравнений равно = " + comparedAmount +

" (не совпадает с формулой т.к. O() округляется до O(n^2))");

} // Сортирует массив, упорядоченный по возрастанию, вставками и бинарным деревом с выводом кол-во пересылок, сравнений, теории и объяснений несовпадений

static void SortDescendedArray(int arrLength)

{ // Кол-во элементов в массиве

int comparedAmount = 0, movedAmount = 0; // Счётчики сравнений и пересылок

int[] arr = GenerateDescendingArray(arrLength); // Создаём и заполняем массив данными, упорядоченными по убыванию

InsertionSort((int[])arr.Clone(), ref movedAmount, ref comparedAmount); // Сперва сортировка вставками (требуется клонирование, т.к. элементы в массиве - сслыки => массив отсортируется и в этой функции), затем вывод полученных данных с теорией и объяснение несовпадений

Console.WriteLine("\n\n\nСортировка простыми вставками массива упорядоченного по убыванию (худшее время: O(n^2) сравнений и обменов):\nКоличество пересылок = " +

movedAmount + ", количество сравнений равно = " + comparedAmount +

" (не совпадает с формулой т.к. вложенный цикл начинается с i, но O() округляется до O(n^2))");

comparedAmount = 0;

BinTreeSort(arr, ref comparedAmount); // Затем сортировка бинарным деревом. После, вывод полученных данных с теорией и объяснение несовпадений

Console.WriteLine("\nСортировка двоичным деревом массива упорядоченного по убыванию (не лучшее, а среднее время, т.к. дерево не сбалансированное и '<' перед '>=', O(n^2) сравнений, O(n) обменов):\nКоличество пересылок = " +

arrLength + ", количество сравнений равно = " + comparedAmount +

" (не совпадает с формулой т.к. O() округляется до O(n^2))");

} // Сортирует массив, упорядоченный по убыванию, вставками и бинарным деревом с выводом кол-во пересылок, сравнений, теории и объяснений несовпадений

static void SortRandomArray(int arrLength)

{ // Кол-во элементов в массиве

int comparedAmount = 0, movedAmount = 0; // Счётчики сравнений и пересылок

int[] arr = GenerateRandomArray(arrLength); // Создаём и заполняем массив неупорядоченными данными

InsertionSort((int[])arr.Clone(), ref movedAmount, ref comparedAmount); // Сперва сортировка вставками (требуется клонирование, т.к. элементы в массиве - сслыки => массив отсортируется и в этой функции), затем вывод полученных данных с теорией и объяснение несовпадений

Console.WriteLine("\n\n\nСортировка простыми вставками неупорядоченного массива (среднее время: O(n^2) сравнений и обменов):\nКоличество пересылок = " +

movedAmount + ", количество сравнений равно = " + comparedAmount +

" (не совпадает с формулой т.к. вложенный цикл начинается с i, но O() округляется до O(n^2))");

comparedAmount = 0;

BinTreeSort(arr, ref comparedAmount); // Затем сортировка бинарным деревом. После, вывод полученных данных с теорией и объяснение несовпадений

Console.WriteLine("\nСортировка двоичным деревом неупорядоченного массива (также среднее время):\nКоличество пересылок = " +

arrLength + ", количество сравнений равно = " + comparedAmount);

} // Сортирует неупорядоченный массив вставками и бинарным деревом с выводом кол-во пересылок, сравнений, теории и объяснений несовпадений