Министерство образования и науки Российской Федерации

Северо-Кавказский горно-металлургический институт (государственный технологический университет)

Кафедра информатики и вычислительной техники

**ОТЧЕТ**

**по лабораторной работе № 4**

**по дисциплине:**

**«лингвистическое и программное обеспечение САПР»**

**Выполнил ст.гр. ИВб-22-2**:

Степанян Константин Александрович

**Проверил**:

Преподаватель: Фомин Андрей Николаевич

Владикавказ 2025

ВАРИАНТ 1

Задание 2:

Для рассмотренного примера реализуйте приложение на языке C#. Убедитесь в работоспособности примера.

Код:

**using** System**;**

**using** System**.**Collections**.**Generic**;**

class Program

**{**

// Класс для представления процесса

class Process

**{**

public int Start **{** get**;** set**;** **}**

public int Finish **{** get**;** set**;** **}**

public int Index **{** get**;** set**;** **}**

public Process**(**int index**,** int start**,** int finish**)**

**{**

Index **=** index**;**

Start **=** start**;**

Finish **=** finish**;**

**}**

**}**

static void Main**()**

**{**

Console**.**WriteLine**(**"Задача выбора процессов:\n"**);**

Console**.**WriteLine**(**"Цель: выбрать максимально возможное количество процессов, которые не пересекаются по времени."**);**

Console**.**WriteLine**(**"Введите количество процессов: "**);**

int processCount **=** int**.**Parse**(**Console**.**ReadLine**());**

List**<**Process**>** processes **=** **new** List**<**Process**>();**

// Ввод данных от пользователя

**for** **(**int i **=** 0**;** i **<** processCount**;** i**++)**

**{**

Console**.**WriteLine**(**"Введите начальное и конечное время процесса {0} (через пробел):"**,** i **+** 1**);**

string**[]** input **=** Console**.**ReadLine**().**Split**(**' '**);**

int start **=** int**.**Parse**(**input**[**0**]);**

int finish **=** int**.**Parse**(**input**[**1**]);**

processes**.**Add**(new** Process**(**i **+** 1**,** start**,** finish**));**

**}**

// Выводим все процессы

Console**.**WriteLine**(**"\nВсе процессы:"**);**

foreach **(**Process p in processes**)**

**{**

Console**.**WriteLine**(**"Процесс {0}: Начало: {1}, Окончание: {2}"**,** p**.**Index**,** p**.**Start**,** p**.**Finish**);**

**}**

Console**.**WriteLine**(**"\nСортируем процессы по времени окончания для выбора..."**);**

processes**.**Sort**((**x**,** y**)** **=>** x**.**Finish**.**CompareTo**(**y**.**Finish**));**

// Список для хранения выбранных процессов

List**<**Process**>** selectedProcesses **=** **new** List**<**Process**>();**

// Выбираем первый процесс

selectedProcesses**.**Add**(**processes**[**0**]);**

int lastFinishTime **=** processes**[**0**].**Finish**;**

// Проходим по оставшимся процессам

**for** **(**int i **=** 1**;** i **<** processes**.**Count**;** i**++)**

**{**

// Если текущий процесс не перекрывается с последним выбранным

**if** **(**processes**[**i**].**Start **>=** lastFinishTime**)**

**{**

// Добавляем процесс в список выбранных

selectedProcesses**.**Add**(**processes**[**i**]);**

lastFinishTime **=** processes**[**i**].**Finish**;**

**}**

**}**

// Выводим выбранные процессы с пояснением

Console**.**WriteLine**(**"\nВыбранные процессы:"**);**

foreach **(**Process p in selectedProcesses**)**

**{**

Console**.**WriteLine**(**"Процесс {0}: Начало: {1}, Окончание: {2}"**,** p**.**Index**,** p**.**Start**,** p**.**Finish**);**

**}**

Console**.**WriteLine**(**"\nПочему выбраны именно эти процессы?"**);**

Console**.**WriteLine**(**"Сначала отсортировали все процессы по времени окончания. Затем последовательно выбирали процессы, которые начинаются после окончания последнего выбранного процесса. Так мы минимизируем пересечения и максимизируем количество выбранных процессов."**);**

Console**.**WriteLine**(**"Пример: Если Процесс 1 заканчивается раньше других, он выбирается первым. Затем выбирается следующий, который начинается после его окончания и так далее."**);**

// Ожидание ввода для предотвращения закрытия приложения

Console**.**WriteLine**(**"\nНажмите любую клавишу для выхода..."**);**

Console**.**ReadKey**();**

**}**

**}**

Результат:



Задание 3: Разработайте приложение на языке C# в соответствии со своим вариантом задания.

ВАРИАНТ 1

Разработайте программу, реализующую алгоритм Хаффмана –адаптивный алгоритм оптимального префиксного кодирования алфавита с минимальной избыточностью

Код:

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**using** System**.**Text**;**

// Основной класс программы

class Program

**{**

// Точка входа в программу

static void Main**()**

**{**

Console**.**WriteLine**(**"Введите строку для кодирования алгоритмом Хаффмана:"**);**

string input **=** Console**.**ReadLine**();**

// Проверяем, что строка не пустая

**if** **(!**string**.**IsNullOrEmpty**(**input**))**

**{**

// Создаем объект для кодирования

HuffmanCoding huffman **=** **new** HuffmanCoding**();**

string encoded **=** huffman**.**Encode**(**input**);**

Console**.**WriteLine**(**"Закодированная строка: " **+** encoded**);**

**}**

**else**

**{**

Console**.**WriteLine**(**"Строка не должна быть пустой!"**);**

**}**

// Программа не закрывается, ожидая нажатия клавиши

Console**.**WriteLine**(**"Нажмите любую клавишу для выхода..."**);**

Console**.**ReadKey**();**

**}**

**}**

// Класс для узла дерева Хаффмана

class HuffmanNode

**{**

public char Symbol**;**

public int Frequency**;**

public HuffmanNode Left**;**

public HuffmanNode Right**;**

**}**

// Класс для реализации кодирования по Хаффману

class HuffmanCoding

**{**

// Метод для кодирования строки

public string Encode**(**string input**)**

**{**

// Создаем словарь для подсчета частоты символов

Dictionary**<**char**,** int**>** frequencyTable **=** **new** Dictionary**<**char**,** int**>();**

// Подсчитываем частоту символов

foreach **(**char c in input**)**

**{**

**if** **(**frequencyTable**.**ContainsKey**(**c**))**

frequencyTable**[**c**]++;**

**else**

frequencyTable**[**c**]** **=** 1**;**

**}**

// Формируем очередь с приоритетом для построения дерева

List**<**HuffmanNode**>** nodes **=** **new** List**<**HuffmanNode**>();**

foreach **(**KeyValuePair**<**char**,** int**>** pair in frequencyTable**)**

**{**

nodes**.**Add**(new** HuffmanNode **{** Symbol **=** pair**.**Key**,** Frequency **=** pair**.**Value **});**

**}**

// Построение дерева

**while** **(**nodes**.**Count **>** 1**)**

**{**

// Сортируем узлы по частоте

nodes**.**Sort**((**a**,** b**)** **=>** a**.**Frequency **-** b**.**Frequency**);**

// Извлекаем два узла с наименьшей частотой

HuffmanNode left **=** nodes**[**0**];**

HuffmanNode right **=** nodes**[**1**];**

// Создаем новый родительский узел

HuffmanNode parent **=** **new** HuffmanNode

**{**

Symbol **=** '\*'**,**

Frequency **=** left**.**Frequency **+** right**.**Frequency**,**

Left **=** left**,**

Right **=** right

**};**

// Удаляем использованные узлы и добавляем родительский

nodes**.**Remove**(**left**);**

nodes**.**Remove**(**right**);**

nodes**.**Add**(**parent**);**

**}**

// Получаем корень дерева

HuffmanNode root **=** nodes**[**0**];**

// Генерируем кодовые слова для каждого символа

Dictionary**<**char**,** string**>** codes **=** **new** Dictionary**<**char**,** string**>();**

GenerateCodes**(**root**,** ""**,** codes**);**

// Кодируем строку

StringBuilder encodedString **=** **new** StringBuilder**();**

foreach **(**char c in input**)**

**{**

encodedString**.**Append**(**codes**[**c**]);**

**}**

**return** encodedString**.**ToString**();**

**}**

// Рекурсивный метод для генерации кодов символов

private void GenerateCodes**(**HuffmanNode node**,** string code**,** Dictionary**<**char**,** string**>** codes**)** **{**

// Если узел не пустой

**if** **(**node **!=** null**)**

**{**

// Если это листовой узел, сохраняем код

**if** **(**node**.**Left **==** null **&&** node**.**Right **==** null**)**

**{**

codes**[**node**.**Symbol**]** **=** code**;**

**}**

**else**

**{**

// Рекурсивно обходим левую и правую ветви

GenerateCodes**(**node**.**Left**,** code **+** "0"**,** codes**);**

GenerateCodes**(**node**.**Right**,** code **+** "1"**,** codes**);**

**}** **}** **} }**

Результат:

****