Министерство образования Республики Беларусь

Учреждение Образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет Компьютерных Систем и Сетей

Кафедра Информатики

Курсовой проект

по дисциплине: «Программирование»

на тему: **«Архиватор»**

|  |  |  |
| --- | --- | --- |
| Выполнил: студент гр. 853504  Специальности: ИиТП  Гончарик Г.С. |  | Проверил: Клыбик  Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (дата) |

2019

**Содержание**

1. Введение. Цели и задачи……………………………………………………..стр.3
2. Описание алгоритма Дэвида Хаффмана для кодирования последовательности байтов………………………………………………………………………….стр.4
3. Техническая реализация……………………………………………………...стр.9

3.1.Интеграция с контекстным меню файловой системы Windows……….стр.9

3.2.Пользовательский интерфейс……………………………………………стр.10

3.3.Структура программы. Описание основных классов…………………...стр.11

3.4.Описание работы программного кода…………………………………...стр.13

1. Выводы………………………………………………………………………...стр.16

4.1.Экспериментальные данные………………………………………….....стр.16

4.2.Выполненные задачи……………………………………………………..стр.16

1. Список литературы……………………………………………………………стр.17
2. **Введение. Цели и задачи**

Архиваторы - это программы позволяющие уменьшить размер файла для экономии места на диске.

Тема сжатия данных является актуальной, т.к. в настоящее время пользователи ПК часто пользуются Интернетом и оперируют большими объемами информации. Очень часто случается так, что данные не помещаются на флэш-карту или на компакт-диск, а после **архивирования данные** прекрасно поместятся.

Сжатие информации - это процесс преобразования информации, хранящийся в файле, при котором уменьшается избыточность в ее представлении и соответственно требуется меньший объем памяти для хранения.

Сжатие информации в файлах происходит за счет устранения избыточности различными способами: например, за счет упрощения кодов, исключения из них постоянных битов или представления повторяющихся символов в виде коэффициента повторения и соответствующих символов.

Применяют различные алгоритмы подобного сжатия информации. При работе над данным курсовым проектом был использован алгоритм сжатия данных Дэвида Хаффман, разработанный в 1952 году. В настоящее время данных алгоритм используется во многих программах для архивации данных.

Цель курсовой работы заключается в раскрытии понятия архивации данных и изучения алгоритма для разработки утилиты архивирования файлов.

Перед началом разработки приложения поставлены следующие задачи:

1.Обеспечить стабильную работу функционала по сжатию файла при помощи алгоритма Дэвида Хаффмана.

2.Предоставить возможность декодирования и полного восстановления сжатого файла.

3.Организовать сохранение информации о файле, которая требуется для его возвращения из архива.

4.Найти оптимальное решение касательно сжатия (восстановления) директории, содержащей файлы.

5.Разработать простой дружественный интерфейс. Предоставить пользователю возможность отменить выполняемую операцию и следить за течением процесса.

6.Произвести интеграцию приложения с контекстным меню файловой системы ОС Windows.

Результатом работы над курсовым проектом будет готовое программное решение, позволяющее производить архивацию выбранных пользователем данных.

1. **Описание алгоритма Дэвида Хаффмана для кодирования последовательности байтов**

Любой файл можно представить как последовательность байтов – единиц информации. Множество всевозможных комбинаций бит в составе байта назовем алфавитом символов.

Алгоритм Д. Хаффмана - алгоритм оптимального префиксного кодирования символов алфавита, который в настоящее время используется во многих программах для сжатия информации.

Отображение символ-код, представленное данным алгоритмом, является биекцией, что позволяет однозначно трактовать сжатую информацию и восстановить ее в полном объеме. Это свойство - следствие префиксности алгоритма: ни одна кодирующая последовательность не является префиксом другой.

Определение кодирующей последовательности бит для определенного символа включает в себя несколько этапов:

1.Подсчет количества вхождений данного символа в последовательность байтов, считанную из файла (далее приоритет символа).

2.Построение бинарного дерева на основе информации о приоритетах символов.

3.Построение однозначного отображения символ-код, используя полученное бинарное дерево.

Рассмотрим каждый из этих этапов подробно при помощи простейшего примера.

**1.Подсчет приоритетов символов**

Предположим, мы имеем считанную из файла последовательность байт, которую можно представить как последовательность целых чисел в интервале от 0 до - 1.

Возьмем в качестве примера следующую последовательность, состоящую из 12 символов:

33 48 55 33 48 33 48 79 43 33 79 33

Далее составим таблицу приоритетов этих символов и запишем приоритеты в порядке возрастания.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Символ | 55 | 43 | 79 | 48 | 33 |
| Приоритет | 1 | 1 | 2 | 3 | 5 |

**2.Построение бинарного дерева на основе информации о приоритетах символов.**

Каждый узел бинарного дерева Хаффмана содержит целое число, назовем его приоритетом узла. Причем, дерево организовано так, что:

1)Приоритетом определенного узла является сумма приоритетов его «наследника».

2)«Листья» дерева содержат символы, входящие в состав нашей последовательности ( у любых двух «листьев» символы не совпадают ).

3)Приоритетом «листа» дерева является приоритет символа, который этот «лист» представляет.

Построение дерева производится при помощи очереди приоритетов узлов. Изначально в качестве очереди используется таблица приоритетов символов.

Каждый шаг построения дерева включает следующие действия:

1.Создать новый узел дерева, который станет «предком» для двух первых элементов очереди ( тех «узлов», которые имеют наименьший приоритет на данный момент).

2.Два первых «узла» очереди удалить .

3.Вновь созданный «узел» поставить на свое место в очереди так, чтобы она оставалась отсортированной по возрастанию.

4.Если длина очереди > 1, вернуться к пункту 1.

Таким образом, длина очереди с каждым шагом уменьшается на 1, до тех пор, пока не будет содержать только один узел – корень дерева.

Вернемся к примеру и продемонстрируем создание дерева Хаффмана наглядно, на каждом шаге создавая новый узел бинарного дерева, выполняя действия 1-4.

Изначально имеем следующую очередь «листьев» дерева.

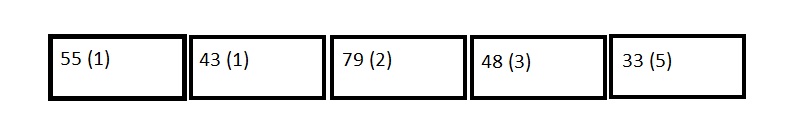


рис.1

Шаг 1.

Выполним действия 1-4, получим очередь приоритетов в виде, представленном на рис.2.

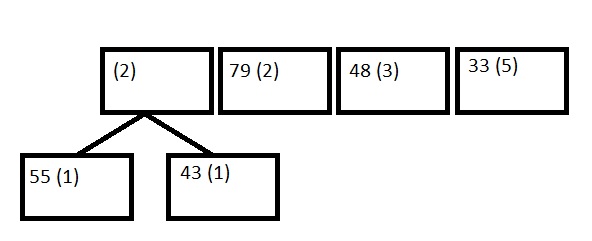


рис.2

Шаг 2.

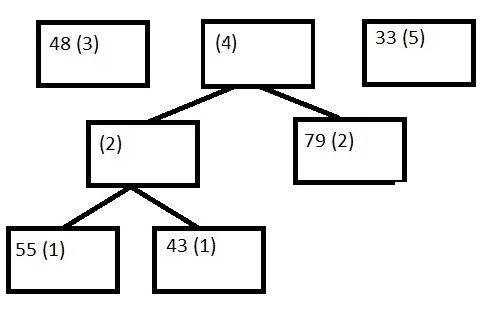


рис.3

Шаг 3.

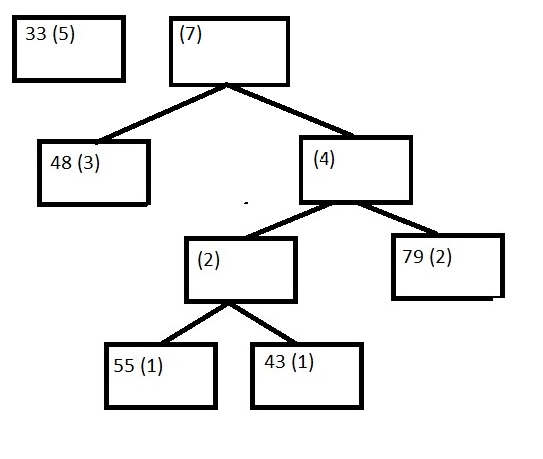


рис.4

Шаг 4.

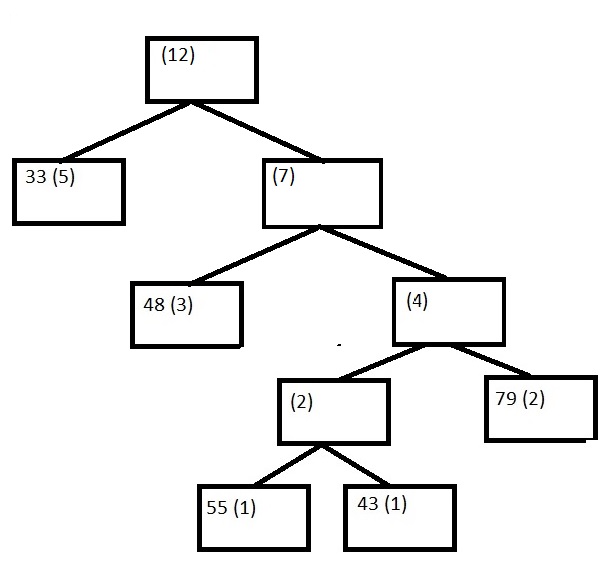


рис.5

На последнем шаге получаем искомое бинарное дерево.

**3.Построение однозначного отображения символ-код.**

Имея бинарное дерево Хаффмана, где всякий «лист» содержит один символ из считанной последовательности байтов, построим для каждого символа кодирующую последовательность бит по следующему принципу.

Всякий раз проходя путь от «корня» дерева до «листа» по «узлам» дерева, необходимо произвести выбор: выбрать следующим «узлом» либо правый, либо левый «наследник» текущего «узла». В том случае, если выбирается правый узел, в кодирующую последовательность листа справа необходимо добавить бит «1», иначе – бит «0».

Полученное закодированное бинарное дерево Хаффмана отображено на рис.6.

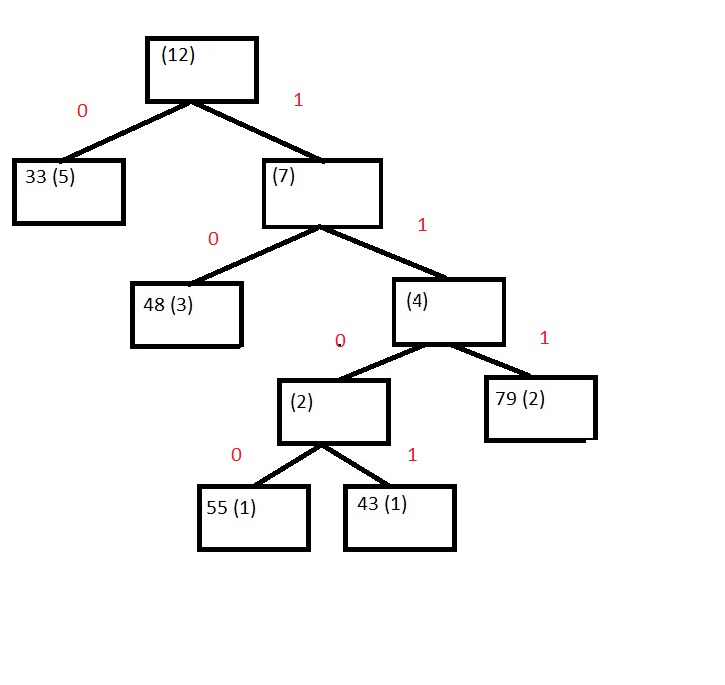


рис.6

Таким образом, получим таблицу символ-код для данной последовательности байтов.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Символ(приоритет) | 55(1) | 43(1) | 79(2) | 48(3) | 33(5) |
| Код | 1100 | 1101 | 111 | 10 | 0 |

Из таблицы видно, что длина кодирующей последовательности символа тем меньше, чем больше его приоритет. Это свойство является основополагающим свойством алгоритма Дэвида Хаффмана и позволяет существенно сократить объем сжимаемого файла.

В примере мы использовали исходную последовательность, состоящую из 12 байт. В исходном файле она занимала 12 \* 8 = 96 бит.

Подсчитаем, какой объем будет занимать та же информация в закодированном виде по общей формуле :

Где n – количество встречаемых символов, – длина кодирующей последовательности для i-ого символа, - приоритет i-ого символа.

Для рассматриваемого примера, объем закодированной информации равен

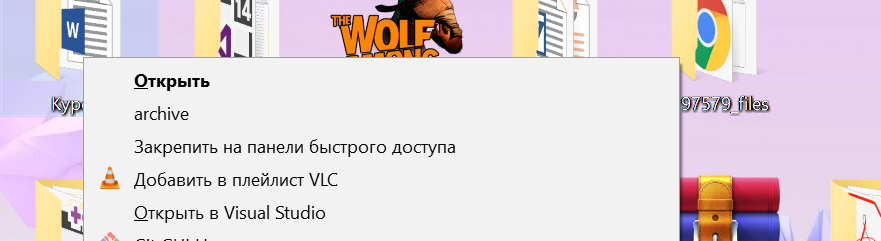
1 \* 4 + 1 \* 4 + 2 \* 3 + 3 \* 2 + 5 \* 1 = 25 бит.

Как можно заметить, данный алгоритм является довольно эффективным средством сокращения объема информации и экономии памяти компьютера.

**3.Техническая реализация**

**3.1.Интеграция с контекстным меню файловой системы Windows**

Для того, чтобы запустить приложение, пользователю необходимо нажать правой кнопкой мыши по нужному файлу или папке и выбрать пункт “Archive”. В данном случае программа запускается, и в качестве аргумента ей передается адрес файла или папки.



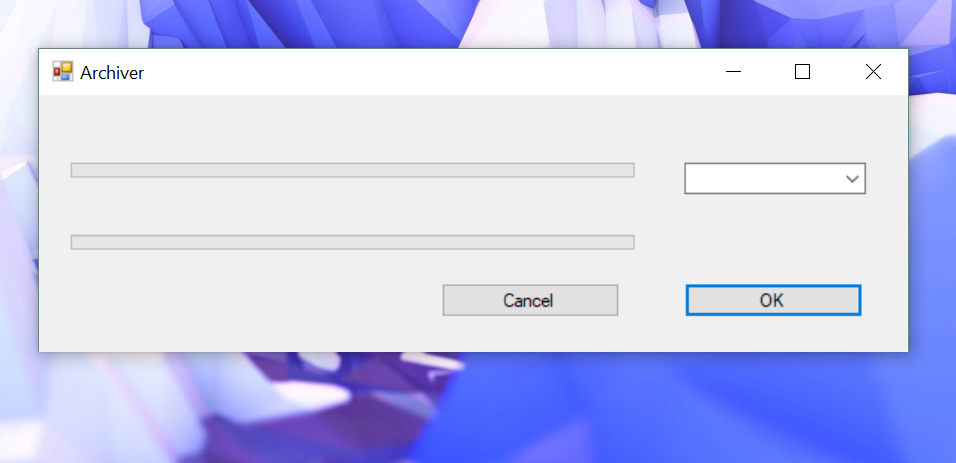
Данная возможность была достигнута при помощи использования программного обеспечения AddToMenu.exe, которое производит обращение к регистру компьютера, добавляя данный пункт в контекстное меню файловой системы.

**3.2.Пользовательский интерфейс**

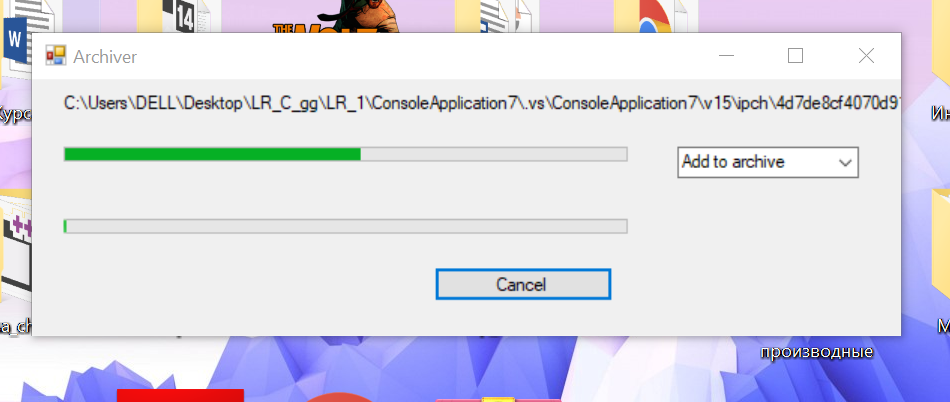
В процессе выполнения курсового проекта, одной из целей было разработать простой и интуитивно понятный интерфейс приложения.

Для его создания была использована технология Windows Forms.

При запуске программы открывается небольшое окно, где при помощи элемента ComboBox, кнопок “OK”, “Cancel”(“Отмена”) пользователь выбирает нужную для него операцию.



Использование двух элементов ProgressBar позволяет пользователю следить как за количеством обработанных файлов, так и процентом обработанного объема текущего файла.



**3.3.Структура программы. Описание основных классы**

1)ArchiveForm.cs

а) Класс ArchiveForm, который служит для:

-считывания пути файла(директории), который подлежит обработке, и создание папки для результатов

- обработки выбора операции( добавить или извлечь из архива )

-обработки нажатий на кнопки “Start” и “Cancel”

-отображения прогресса процесса на экране

-обработки ситуации внезапного закрытия приложения

б)класс PathFunctions, содержащий статические методы для создания папки, куда сохранится конечный результат сжатия(восстановления).

2)FileNotFoundForm.cs

Файл содержит класс, описывающий форму диалога с пользователем при ошибке в прочтении информации из файла.

3)Huffman.cs

а)Класс DecodingInfo, объект которого представляет собой информацию о закодированном единичном файле:

-частичный путь, по которому будет сохраняться восстановленный файл

- объект встроенного типа Dictionary, содержащий таблицу код-символ

-количество символов, которые содержал исходный файл

-минимальная длина кодирующей последовательности

б)Класс Huffman, который отвечает за:

-считывание информации из файлов;

- вычисление путей для результата обработки каждого из файлов и создание соответствующих директорий

- подсчет приоритетов символов - для сжатия файлов

-заполнение полей объекта класса DecodingInfo и последующую его сериализацию в файл “dictionary.bin” - для сжатия файлов

-процесс восстановления исходного файла, используя десериализованный объект класса DecodingInfo

- заполнение конечного массива байтов, который будет содержать сжатый (возвращаемый) файл;

3)HuffmanBinaryTree.cs

Данный класс отвечает бинарному дереву, описанному ранее в пункте 2.

Класс содержит:

-конструктор, воссоздающий структуру дерева в соответствии с таблицей приоритетов символов, поступающей в качестве аргумента.

-методы для кодирования листьев дерева( т.е. формирования отображения символ-код ).

-методы, которые возвращают кодирующую последовательность для определенного символа.

4)Node.cs

Класс Node, представляющий узел бинарного дерева

5)Code.cs

Класс Code , который служит для представления кодирующей последовательности. Содержит количество бит кодирующей последовательности и число, которое получается переведением ее в десятичную систему счисления.

6)BitOperations.cs

Cодержит статические классы BitForUint и BitForByte, которые служат для получения бита, стоящего на определенной позиции.

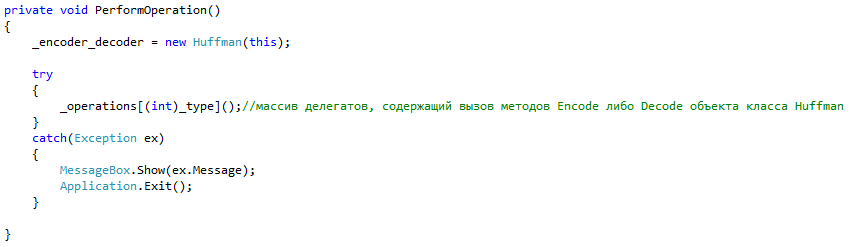
7)Program.cs

Содержит функцию Main, которая инициализирует объект класса ArchiveForm и запускает главное окно приложения.

**3.2.Описание работы программного кода**

Процесс обработки файлов занимает немалое количество времени и ресурсов. В таком случае приложение может перестать реагировать на действия пользователя, что, безусловно, не является ожидаемым поведением утилиты такого типа. Для предотвращения данной проблемы было решено проводить обработку файлов в фоновом потоке. Для этого использовался класс Task пространства имен System.Threading.Tasks.

Когда пользователь нажимает кнопку “OK”, начинается обработка файлов в фоновом потоке.поток.png

Код функции PerformOperation: 

В зависимости от того, какая операция была выбрана, запускается метод Encode либо Decode объекта \_encoder\_decoder.Они имеют следующие сигнатуры

сигнатура encode.png

сигнатура decode.png

Оба данных метода являются рекурсивными и производят предварительные операции перед непосредственным кодированием или декодированием последовательностей байтов, полученных из файлов. Среди этих операций присутствуют такие, как:

- проверка на существование обрабатываемого файла или папки

-настраивание элементов ProgressBar для главного окна

-рекурсивный вызов, если в качестве первого аргумента выступает путь к директории

-чтения информации из файла в массив байтов

Помимо этого, метод Encode отвечает за создание объекта типа DecodingInfo для каждого из файлов директории и заполнение его свойства PathForDecoded. Также метод Encode отвечает за определение путей для каждого из сжатых файлов. Метод Decode вызывает десериализацию объекта типа DecodingInfo, из файла “dictionary.bin”, который содержится в каждой папке, представляющей сжатый файл.

Для каждого из файлов из этих методов вызываются методы EncodeBytes и DecodeBytes соответственно.

Метод EncodeBytes имеет следующую сигнатуру:

сигнатура encodeBytes.png

Из данного метода вызывается подсчет приоритетов символов в source; внутри него создается объект класса HuffmanBinaryTree, а также вызывается заполнение массива байтов для сжатого файла, используя бинарное дерево. Далее проводится заполнение оставшихся полей объекта info типа DecodingInfo и его сериализация по пути dictionary\_file\_path; происходит запись полученных байтов в файл по пути destination\_file\_path.

Метод DecodeBytes имеет следующую сигнатуру:

сигнатура decodeBytes.png

Данный метод восстанавливает массив байтов, который был считан при кодировании первоначального файла, используя элемент Dictionary, который содержится в объекте типа DecodingInfo.

При закрытии приложения происходит обязательная проверка завершения обработки файлов, и если она не завершена, все вычисления отменяются, результаты - получаемые файлы - удаляются. В том случае, если все операции прошли успешно, приложение немедленно закрывается, а папка с результатами становится видимой для пользователя.

**4.Выводы**

**4.1.Экспериментальные данные**

Для проверки работы и тестирования разработанного программного обеспечения используются следующие тестовые данные, которые приведены в таблице

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Файл или папка | Размер файла или папки до сжатия | Размер файла или папка после сжатия | Время создания архива | Время извлечения файлов из архива |
|  |  |  |  |  |
| Папка c лабораторными работами по программированию | 938МБ | 560МБ | 4 мин 52с | 8 мин 35 с |

В таблице предоставлены результаты сжатия и восстановления тестовых данных и время, за которое эти операции были выполнены. Как можно видеть, разработанное программное обеспечение по архивации файлов дает результат сжатия до 54% от первоначального размера данных.

**4.2.Заключение**

В данной курсовом проекте рассматривалась тема разработки утилиты архивирования файлов для OC Windows, которая включает в себя базовые знания в области информатики и программирования.

Можно сделать вывод, что поставленная цель курсового проекта : раскрытие понятия архивации данных и изучения алгоритма для разработки утилиты архивирования файлов, достигнута.

Тема данного курсового проекта обладает большим потенциалом, так как в ходе работы происходит обобщение знаний, связанных с формированием целостной картины о роли и месте программ - архиваторов, а также изучение различных алгоритмов сжатия данных.

**5.Список литературы**

1.https://ru.wikipedia.org/wiki/Код\_Хаффмана

Википедия. Свободная энциклопедия

2.https://habrahabr.ru/post/144200

Хабрахабр

3.https://msdn.microsoft.com/ru-ru/library/system.threading.tasks.task(v=vs.110).aspx

Microsoft Developer Network

4.http://stackoverflow.com/questions/1838856/integrating-into-windows-explorer-context-menu

Stack Overflow

5.https://msdn.microsoft.com/en-us/library/system.io.directory(v=vs.110).aspx

Microsoft Developer Network

6.https://msdn.microsoft.com/ru-ru/library/system.text.regularexpressions.

regex(v=vs.110).aspx

Microsoft Developer Network

7.https://msdn.microsoft.com/ru-ru/library/system.runtime.serialization.

iserializable(v=vs.110).aspx

Microsoft Developer Network