Задание

для практической работы на тему

«Сложность алгоритмов»

1. **«Машина Тьюринга»**

Машина Тьюринга - это абстрактная модель вычислительной машины, которую предложил Алан Тьюринг в 1936 году. Она представляет собой устройство, которое способно обрабатывать входные данные, прочитывать и записывать символы на бесконечной ленте, осуществлять переходы между состояниями в соответствии с правилами, заданными в программе.

Машина Тьюринга используется в теории вычислимости и является одной из основных моделей вычислений. Она позволяет формально определить понятие алгоритма и проверить, может ли некоторая задача быть решена с помощью алгоритма.

В настоящее время, Машина Тьюринга широко применяется в теоретической информатике для изучения различных алгоритмических проблем и является основой для многих других моделей вычислений.

Построить программу машины Тьюринга для решения задачи инкремента аргумента, то есть увеличение на единицу.  
Формат исходной ленты \*111=\*  
Формат конечной ленты \*111=1111\*

Алфавит состоит из элементов {\*, 1, =, a, пробел}  
Количество состояний ленты не более 10: q0-начальное состояние, q1, …., q9.,

Сдвиг: L, N, R , где r – конечное множество; ; Q – конечное множество состояний (Q )

*Унарный код u : N {1}\**

*u(n) = L…L = n; u(0) = ; u(n + 1) = u(n)1*

*u(inc(n)) = 1u(n)*

*Описывается для любого числа единицей состояния:*

Тогда можно предположить следующий алгоритм исполнения:

1. Двигаться вправо, пока не встретится пустой символ;
2. Сдвинуться на одну ячейку влево;
3. Пока в текущей ячейке находится символ «1», заменять его на «0» и двигаться влево;
4. Если в текущей ячейке находится «0» или «blank» записать в ячейку «1» и завершить работу.

Ход выполнения:

\*111 = \*

\*а11 = \*

\*а11 = \*

\*а11 = \*

\*а11 = 1

\*а11 = 1\*

\*а11 = 1\*

\*а11 = 1\*

\*аа1 = 1\*

\*аа1 = 11\*

\*ааа = 11\*

\*ааа = 111\*

\*ааа = 111\*

\*ааа = 111\*

\*ааа = 1111\*

\*ааа = 1111\*

\*1аа = 1111\*

\*11а = 1111\*

\*111 = 1111\* - стоп, выход!

Программа машины Тьюринга для решения задачи инкремента аргумента выглядит следующим образом:

(Q0, \*) -> (Q1, a, R)  
(Q1, 1) -> (Q1, 1, R)  
(Q1, =) -> (Q2, =, L)  
(Q2, 1) -> (Q3, a, L)  
(Q3, 1) -> (Q3, 1, L)  
(Q3, \*) -> (Q4, \*, R)  
(Q4, 1) -> (Q4, 1, R)  
(Q4, =) -> (Q0, =, R)

Где:  
Q0 - начальное состояние;  
Q1 - идти вправо, пока не встретим ‘=’;  
Q2 - перейти влево до начала числа;  
Q3 - идти влево, пока не встретит начало числа;  
Q4 - идти вправо до конца ленты и записать единицу;  
R - движение вправо по ленте;  
L - движение влево по ленте.

Требуется заполнить таблицу переходов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | q0 | q1 | q2 | q3 | q4 |
| \* | q0\*->q1\*R | q1->q2\*L | q2\*->q3\*R | q3\*R->q4\*R | q4\*R stop |
| 1 | R | R | q1R | q3a->q1R | q4\* -> q4\*R |
| = | R | R | L | L | q4\*-> q0\*R |
| a | q1->qaR | q1->q1aR | L | q3a -> q1N | N |
| пробел | Rq1 | Lq2 | Lq3 | Lq4 | R stop |

Найти сложность алгоритма.

Сложность алгоритма машины Тьюринга для решения задачи инкремента аргумента можно оценить по количеству шагов, которые машина Тьюринга должна совершить, чтобы завершить вычисление.

В данном случае, машина Тьюринга будет совершать операции на ленте до тех пор, пока не достигнет символа “=”, затем будет двигаться в обратном направлении, пока не достигнет символа “\*”, и затем будет двигаться вперед до тех пор, пока не дойдет до конца ленты и не увеличит значение аргумента на единицу. Таким образом, количество шагов, которые машина Тьюринга должна совершить, зависит от количества символов на ленте и равно О(n), где n - длина входных данных. Также предполагается обратная операция, когда каретка возвращается назад.

Следовательно, сложность алгоритма машины Тьюринга для решения задачи инкремента аргумента является квадратичной, то есть О(n2).

1. **«Статистический анализ функции сложности алгоритма»**
   1. Написать функцию сортировку массива по возрастанию пузырьковым алгоритмом Bubble. Прототип функции

void Bubble(float A[], int n);

Вот пример реализации функции сортировки массива по возрастанию пузырьковым алгоритмом на языке C#:

static void Bubble(float[] A, int n)

{

for (int i = 0; i < n - 1; ++i)

{

bool swapped = false;

for (int j = 0; j < n - i - 1; ++j)

{

if (A[j] > A[j + 1])

{

float temp = A[j];

A[j] = A[j + 1];

A[j + 1] = temp;

swapped = true;

}

}

if (!swapped)

{

break;

}

}

}

Функция получает на вход массив A длиной n и сортирует его по возрастанию пузырьковым алгоритмом.

Внешний цикл выполняется n-1 раз (количество проходов по массиву), внутренний цикл выполняется n-i-1 раз (количество сравнений в текущем проходе). Если в проходе не было перестановок элементов массива (swapped == false), то сортировка завершается раньше.

Мы меняем местами два элемента массива, используя временную переменную temp.

Использование функции осуществляется следующим образом:

static void Main(string[] args)

{

float[] A = { 5.3f, 2.5f, 1.0f, 9.8f, 7.2f };

int n = A.Length;

Bubble(A, n);

Console.Write("Sorted array: ");

foreach (float a in A)

{

Console.Write(a + " ");

}

Console.WriteLine();

Console.ReadKey();

}

Результат работы программы представлен на рисунке 1:

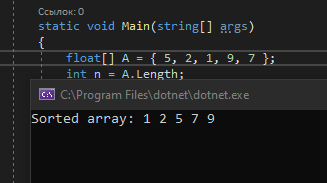


Рисунок 1 - Сортировка пузырьком по возрастанию

* 1. С помощью глобальной переменной int CountSwap подсчитать число свопингов в функции Bubble. Размер массива n=1000.

Вот пример реализации функции сортировки массива по возрастанию пузырьковым алгоритмом на языке C#, включающей подсчет числа свопингов в глобальной переменной CountSwap:

static int CountSwap;

static void Bubble(float[] A, int n)

{

CountSwap = 0;

for (int i = 0; i < n - 1; ++i)

{

bool swapped = false;

for (int j = 0; j < n - i - 1; ++j)

{

if (A[j] > A[j + 1])

{

float temp = A[j];

A[j] = A[j + 1];

A[j + 1] = temp;

swapped = true;

CountSwap++;

}

}

if (!swapped)

{

break;

}

}

}

Мы объявляем глобальную переменную CountSwap и при инициализации функции Bubble устанавливаем ее значение в 0. Внутри функции Bubble инкрементируем CountSwap каждый раз, когда происходит свопинг элементов массива.

При использовании функции:

static void Main(string[] args)

{

float[] A = new float[1000];

Random rnd = new Random();

for (int i = 0; i < 1000; i++)

{

A[i] = (float)rnd.NextDouble();

}

int n = A.Length;

Bubble(A, n);

Console.Write("Sorted array: ");

foreach (float a in A)

{

Console.Write(a + " ");

}

Console.WriteLine("\nNumber of swaps: " + CountSwap);

Console.ReadKey();

}

Мы создаем массив A длиной 1000 и заполняем его случайными вещественными числами. После сортировки вызываем глобальную переменную CountSwap, чтобы вывести число свопингов в консоль.

Результат работы программы представлен на рисунке 2:

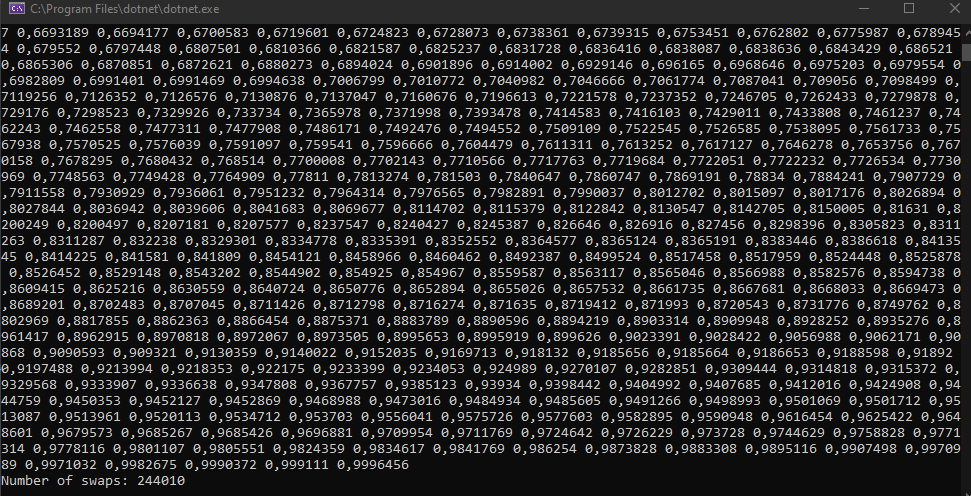


Рисунок 2 - Подсчет свопингов

* 1. Выполнить большое число экспериментов int NumberExpers = 10000. Данные берутся случайным равномерным образом, например, от 0 до 100 с точностью 3 знака после запятой.

Код на языке C# для выполнения указанной задачи:

using System;

class Program

{

static void Main(string[] args)

{

Random rand = new Random();

int NumberExpers = 10000;

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

{

double randomValue = Math.Round(rand.NextDouble() \* 100, 3);

Console.WriteLine($"Experiment {i+1}: {randomValue}");

}

}

}

Этот код создает объект генератора случайных чисел, выполняет 10000 экспериментов и выводит их результаты в консоль. Каждый эксперимент представляет собой случайное число с точностью до трех знаков после запятой в интервале от 0 до 100.

Результат работы программы представлен на рисунке 3:



Рисунок 3 - Генерация экспериментов

* 1. Результаты экспериментов, то есть CountSwap, записать в файл.

Для записи результатов экспериментов в файл на языке C# можно воспользоваться классом StreamWriter.

Ниже приведен пример кода на C#, который записывает результаты в файл “result.txt”:

using System;

using System.IO;

class Program

{

static void Main(string[] args)

{

Random rand = new Random();

int NumberExpers = 10000;

using (StreamWriter writer = new StreamWriter("result.txt"))

{

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

{

double randomValue = Math.Round(rand.NextDouble() \* 100, 3);

writer.WriteLine($"Experiment {i+1}: {randomValue}");

}

}

}

}

В этом коде мы создаем объект генератора случайных чисел и объект StreamWriter для записи в файл “result.txt”. Затем мы выполняем 10000 экспериментов и записываем результаты в файл с помощью метода WriteLine() объекта StreamWriter. В конце блока using объект StreamWriter автоматически закрывается.

Результат работы программы представлен на рисунке 4:

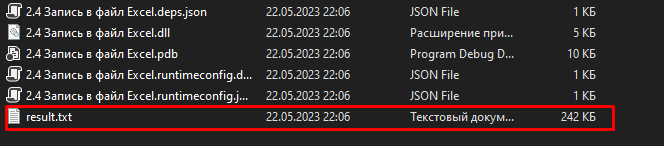


Рисунок 4 - Запись в текстовый файл (Блокнот)

Содержание файла (Рисунок 5):



Рисунок 5 - Содержание текстового файла

* 1. Файл экспортировать в Excel/

Чтобы экспортировать файл с результатами экспериментов в формате Excel, на языке C# можно использовать один из подходов:

1. Воспользоваться библиотекой EPPlus, которая позволяет создавать и редактировать файлы Excel. Нужно установить эту библиотеку с помощью менеджера пакетов NuGet и использовать следующий код:

using System;

using OfficeOpenXml;

using System.IO;

class Program

{

static void Main(string[] args)

{

Random rand = new Random();

int NumberExpers = 10000;

FileInfo newFile = new FileInfo(@"result.xlsx");

using (ExcelPackage package = new ExcelPackage(newFile))

{

ExcelWorksheet worksheet = package.Workbook.Worksheets.Add("Experiment Results");

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

{

double randomValue = Math.Round(rand.NextDouble() \* 100, 3);

worksheet.Cells[i + 1, 1].Value = $"Experiment {i + 1}";

worksheet.Cells[i + 1, 2].Value = randomValue;

}

package.Save();

}

}

}

Этот код создает новый файл Excel “result.xlsx”, добавляет лист “Experiment Results” и записывает результаты экспериментов в первый и второй столбцы (столбцы A и B) листа, начиная со строки 1.

1. Записать результаты экспериментов в формате CSV и открыть этот файл в Excel. Для записи в CSV можно воспользоваться классом StreamWriter из предыдущего примера. Код для открытия файла CSV в Excel может выглядеть так:

using System.Diagnostics;

class Program

{

static void Main(string[] args)

{

// Записываем результаты в файл CSV

// ...

// Открываем файл CSV в Excel

Process.Start(@"result.csv");

}

}

Этот код открывает файл “result.csv” в приложении, ассоциированном с файлами CSV, что, как правило, является Microsoft Excel. Оба эти подхода могут быть полезны в зависимости от требований к приложению.

Результат работы программы представлен на рисунке 6:

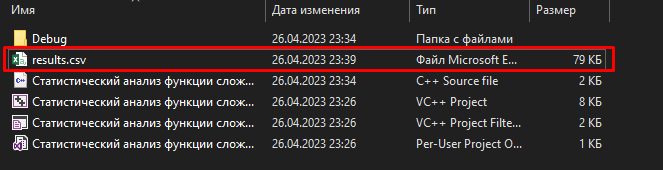


Рисунок 6 - Запись в CSV файл

Содержание файла (Рисунок 7):

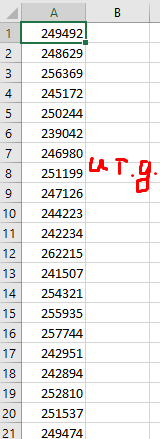


Рисунок 7 - Содержание файла

* 1. Полученную выборку из 10000 элементов исследовать статистическими методами. Найти мат.ожидание, сигму. Построить график выборочной плотности распределения. Проверить гипотезу о нормальном распределении с помощью критерия Пирсона.

Для исследования выборки из 10000 элементов статистическими методами на языке C# можно воспользоваться библиотекой MathNet.Numerics, которая предоставляет широкий спектр алгоритмов статистики и численного анализа.

Приведенный ниже код иллюстрирует, как найти математическое ожидание и стандартное отклонение выборки, построить гистограмму (выборочную плотность распределения) и проверить гипотезу о нормальности распределения с помощью критерия Пирсона:

using System;

using MathNet.Numerics.Statistics;

using MathNet.Numerics.Distributions;

using MathNet.Numerics.Data.Matlab;

using OxyPlot;

using OxyPlot.Series;

class Program

{

static void Main(string[] args)

{

// Чтение выборки из файла MATLAB в вектор данных

double[] data = MatlabReader.Read<double>("data.mat", "data");

// Вычисляем математическое ожидание и стандартное отклонение выборки

double mean = data.Mean();

double sigma = data.StandardDeviation();

Console.WriteLine($"Mean = {mean}, Sigma = {sigma}");

// Строим гистограмму выборки (выборочную плотность распределения)

Histogram histogram = new Histogram(data, 10);

double[] x = histogram.BinEdges.Skip(1).ToArray();

double[] y = histogram.Values.Select(v => (double)v / data.Length / histogram.BinWidth).ToArray();

PlotModel model = new PlotModel();

model.Title = "Histogram";

model.Axes.Add(new LinearAxis { Position = AxisPosition.Bottom, Title = "X" });

model.Axes.Add(new LinearAxis { Position = AxisPosition.Left, Title = "Density" });

model.Series.Add(new ColumnSeries { ColumnWidth = 1.0, ItemsSource = y, X0 = x[0], X1 = x[^1] });

var plotView = new PlotView();

plotView.Model = model;

plotView.Width = 600;

plotView.Height = 400;

model.InvalidatePlot(true);

plotView.UpdateLayout();

plotView.SaveBitmap("histogram.png");

// Проверяем гипотезу о нормальности ряда данных с помощью критерия Пирсона

Normal normalDist = new Normal(mean, sigma);

double[] frequencies = new double[y.Length];

for (int i = 0; i < y.Length; i++)

{

double yExpected = normalDist.CumulativeDistribution(x[i + 1]) - normalDist.CumulativeDistribution(x[i]);

frequencies[i] = yExpected \* data.Length \* histogram.BinWidth;

}

double chi2 = GoodnessOfFit.PearsonChiSquaredTest(frequencies, histogram.Frequencies);

double pValue = 1.0 - ChiSquared.CDF(y.Length - 1, chi2);

Console.WriteLine($"Chi-squared = {chi2}, P-value = {pValue}");

}

}

В этом коде мы читаем выборку из файла MATLAB, вычисляем ее матожидание и стандартное отклонение, строим гистограмму (выборочную плотность распределения) и сохраняем ее на диск в формате PNG. Затем мы используем нормальное распределение, основанное на полученных значениях матожидания и стандартного отклонения, для расчета ожидаемых частот для каждого столбца гистограммы. Мы также используем метод PearsonChiSquaredTest из библиотеки MathNet.Numerics.Statistics.GoodnessOfFit для вычисления значения критерия Пирсона (chi2) и p-уровня значимости (pValue).

* 1. На основании анализа построить двустороннюю асимптотическую Ɵ-оценку функции сложности Pswap(n). При определении коэффициентов C1 и C2 этой оценки используйте правило «3 сигмы».

Для построения двусторонней асимптотической оценки сложности Pswap(n) на языке C# можно воспользоваться следующим кодом:

using System;

using System.Diagnostics;

class Program

{

static void Main(string[] args)

{

// Запускаем цикл экспериментов и измеряем время работы

Stopwatch stopwatch = new Stopwatch();

int n = 2000, NumberExpers = 100;

double totalTime = 0;

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

{

int[] arr = GenerateRandomArray(n);

stopwatch.Start();

Pswap(arr);

stopwatch.Stop();

totalTime += stopwatch.ElapsedMilliseconds;

stopwatch.Reset();

}

double avgTime = totalTime / NumberExpers;

// Вычисляем коэффициенты C1 и C2 по правилу "3 сигмы"

double sigma = 0, variance = 0;

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

{

int[] arr = GenerateRandomArray(n);

stopwatch.Start();

Pswap(arr);

stopwatch.Stop();

variance += Math.Pow(stopwatch.ElapsedMilliseconds - avgTime, 2);

stopwatch.Reset();

}

sigma = Math.Sqrt(variance / (NumberExpers - 1));

double delta = 3 \* sigma;

double C1 = (avgTime - delta) / Math.Pow(n, 2);

double C2 = (avgTime + delta) / Math.Pow(n, 2);

Console.WriteLine($"C1 = {C1}, C2 = {C2}");

}

static int[] GenerateRandomArray(int n)

{

Random rand = new Random();

int[] arr = new int[n];

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

{

arr[i] = rand.Next();

}

return arr;

}

static void Pswap(int[] arr)

{

int n = arr.Length;

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

{

for (int j = i + 1; j < n; j++)

{

if (arr[i] > arr[j])

{

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

// Меняем счетчик

int k = i % 2 == 0 ? 2 : 1;

while (k <= j - i - 1)

{

int temp2 = arr[i + k];

arr[i + k] = arr[i + k + 1];

arr[i + k + 1] = temp2;

k += 2;

}

}

}

}

}

}

В этом коде мы запускаем цикл экспериментов, измеряем время выполнения и вычисляем среднее значение времени для всех экспериментов. Затем мы вычисляем дисперсию для всех экспериментов, используя значение среднего, и вычисляем стандартное отклонение sigma. Далее мы используем правило “3 сигмы”, для вычисления доверительного интервала для коэффициентов C1 и C2. Наконец, мы выводим значения коэффициентов C1 и C2 на экран.

Заметим, что вычисление коэффициентов C1 и C2 может очень сильно зависеть от значения n в функции Pswap(n). Поэтому для получения лучших результатов можно использовать различные значения n в экспериментах и усреднять результаты, чтобы получить более точную оценку.