Министерство образования и науки Украины
Севастопольский национальный технический университет

МАТЕМАТИЧЕСКИЙ АНАЛИЗ АЛГОРИТМОВ
Определение среднего времени выполнения обобщенной элементарной операции методом тестовых прогонов

Методические указания
к выполнению лабораторной работы
для студентов, обучающихся по направлению
6.050101“Информационные управляющие системы и технологии”
очной и заочной форм обучения

Севастополь
2014

УДК 510.5

МАТЕМАТИЧЕСКИЙ АНАЛИЗ АЛГОРИТМОВ: методические / Сост. Ю. В. Коваленко, Е. Н. Заикина. — Севастополь: Изд-во СевНТУ, 2014— 14 с.

Целью методических указаний является экспериментальная оценка обобщенного времени выполнения элементарной операции методом тестовых прогонов программы. Получение практических навыков оценки времени выполнения элементарных операций в программе на языке высокого уровня.

Методические указания составлены в соответствии с требованиями программы дисциплины «Теория алгоритмов и математическая логика » для студентов направления 6.050101 и утверждены на заседании кафедры Информационных систем, протокол № 1 от 29 августа 2014 года.

Допущено учебно-методическим центром СевНТУ в качестве методических указаний.

Рецензент: Скороход Б.А., доктор технических наук, проф. кафедры ТК.

СОДЕРЖАНИЕ

1 ЦЕЛЬ РАБОТЫ

Экспериментальная оценка обобщенного времени выполнения элементарной операции методом тестовых прогонов программы. Получение практических навыков оценки времени выполнения элементарных операций в программе на языке высокого уровня.

2 ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ

2. 1 Временные оценки алгоритмов

Сравнение двух алгоритмов по их функции трудоемкости вносит некоторую ошибку в получаемые результаты. Основной причиной этой ошибки является различная частотная встречаемость элементарных операций, порождаемая разными алгоритмами и различие во временах выполнения элементарных операций на реальном процессоре. Таким образом, возникает задача перехода от функции трудоемкости к оценке времени работы алгоритма на конкретном процессоре:

Дано: $F_A(D_A)$ - трудоёмкость алгоритма, требуется определить время работы программной реализации алгоритма – $T_A(D_A)$.

На пути построения временных оценок мы сталкиваемся с целым набором различных проблем, пофакторный учет которых вызывает существенные трудности. Укажем основные из этих проблем:

- неадекватность формальной системы записи алгоритма и реальной системы команд процессора;
- наличие архитектурных особенностей существенно влияющих на наблюдаемое время выполнения программы, таких как конвейер, кеширование памяти, предвыборка команд и данных, и т.д.;
- различные времена выполнения реальных машинных команд;
- различие во времени выполнения одной команды, в зависимости от значений операндов
- различные времена реального выполнения однородных команд в зависимости от типов данных;
- неоднозначности компиляции исходного текста, обусловленные как самим компилятором, так и его настройками.

Попытки различного подхода к учету этих факторов привели к появлению разнообразных методик перехода к временным оценкам.

1) Пооперационный анализ

Идея пооперационного анализа состоит в получении пооперационной функции трудоемкости для каждой из используемых алгоритмом элементарных операций с учетом типов данных. Следующим шагом является экспериментальное определение среднего времени выполнения данной элементарной операции на конкретной вычислительной машине. Ожидаемое время выпол-

нения рассчитывается как сумма произведений пооперационной трудоемкости на средние времена операций:

$$T_A(N) = \sum F_{aoni}(N) * \bar{t}_{oni}$$

2) Метод Гиббсона

Метод предполагает проведение совокупного анализа по трудоемкости и переход к временным оценкам на основе принадлежности решаемой задачи к одному из следующих типов:

- задачи научно-технического характера с преобладанием операций с операндами действительного типа;
- задачи дискретной математики с преобладанием операций с операндами целого типа
- задачи баз данных с преобладанием операций с операндами строкового типа

Далее на основе анализа множества реальных программ для решения соответствующих типов задач определяется частотная встречаемость операций (рисунок 2.1), создаются соответствующие тестовые программы, и определяется среднее время на операцию в данном типе задач – $\bar{t}_{тип\ задачи}$.

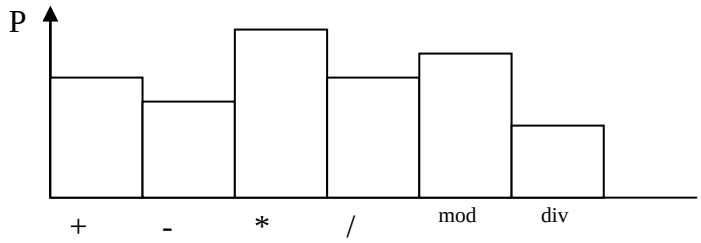

Рисунок 2.1 – Возможный вид частотной встречаемости операций

На основе полученной информации оценивается общее время работы алгоритма в виде:

$$T_A(N) = F_A(N) * \bar{t}_{тип\ задачи}$$

3) Метод прямого определения среднего времени

В этом методе так же проводится совокупный анализ по трудоемкости – определяется $F_A(N)$, после чего на основе прямого эксперимента для различных значений $N_э$ определяется среднее время работы данной программы $T_э$ и на основе известной функции трудоемкости рассчитывается среднее время на обобщенную элементарную операцию, порождаемое данным алгоритмом, компилятором и компьютером – $\bar{t}_a$. Эти данные могут быть (в предположении об устойчивости среднего времени по N) интерполированы или экстраполированы на другие значения размерности задачи следующим образом:

$$\bar{t}_a = T_э(N_э) / F_A(N_э),\ T(N) = \bar{t}_a * F_A(N).$$

2.2 Программная реализация временных оценок алгоритма

В приведённой ниже программе Time вычисляется время выполнения некоторых операций и/или операторов (фрагментов) программы путём использования системных часов компьютера. Показания системных часов хранятся в оперативной памяти по адресу P [$0040:$006C] в формате LongInt, причем каждые 55 миллисекунд значение системных часов увеличивается на единицу.

Время выполнения некоторого фрагмента исследуемого алгоритма определяется вычислением числа его исполнения за определённый отрезок времени (55 мс или кратный ему). В программе Time подсчитывается количество циклов выполнения операции сравнения n и количество циклов выполнения операторов присваивания (исследуемый алгоритм - сортировка простым включением) за 18*55 = 990 миллисекунд системных часов.

2.3 Пример программы

```
Program Test;
uses crt;
var t: longint;
    n1, n2: longint;
    Tc, Tm: real;
    a: array[0..2] of integer;
    x, i, j: integer;

begin

       n1 := 0;
       j := 1;
       a[j] := 4;
       x := 3;

       t := MemL[$0040:$006C]; {Фиксация начала отсчета}
       while MemL[$0040:$006C] = t do; {Ожидание начала нового
цикла}

       while MemL[$0040:$006C] < t + 19 do {Подсчет количества
операций сравнения за 18 циклов}
       begin
            inc(n1);
            if x < a[j] then;
       end;

       n2 := 0;
       i := 1;
       a[i] := 2;

       t := MemL[$0040:$006C]; {Фиксация начала отсчета}
```

```
        while MemL[$0040:$006C] = t do; {Ожидание начала нового
    цикла}


        while MemL[$0040:$006C] < t + 19 do {Количество четырех
    операций присваивания за 18 циклов}

        begin
             inc(n2);
             x := a[i];
             a[0] := x;
             a[j+1] := a[j];
             a[j] := x;
        end;

        writeln('Tc = ', 55*18 / n1 : 10 : 7); {Время выполнения
    первой части алгоритма}
        writeln('Tm = ', 55*15 / n2 : 10 : 7); {Время выполнения
    второй части алгоритма}
    end.
```

Пример программы на C#:

A: сортировка прямым выбором;
B: среднее арифметическое положительных чисел двумерного массива размером n*n.

```
using System;
using System.Diagnostics;

namespace laba4
{
    class Program
    {
        protected static int[] gener1(int k, int n)
        {
            int[] mas = new int[n];
            Random rnd = new Random();
            for(int i = 0; i < n; i++)
            {
                switch (k)
                {
                    case 1:
                        mas[i] = i; // 123
                        break;
                    case 2:
                        mas[i] = rnd.Next(0, n); //-12-313
                        break;
```

```
            case 3:
                mas[i] = n - i; //321
                break;
        }
    }
    return mas;
}

protected static int[,] gener2(int k, int n)
{
    int[,] mas = new int[n,n];
    Random rnd = new Random();
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
        {
            switch (k)
            {
                case 1:
                    mas[i,j] = -1; // 000
                    break;
                case 2:
                    mas[i,j] = rnd.Next(-n, n); //-12-313
                    break;
                case 3:
                    mas[i,j] = rnd.Next(1,n); //1235215
                    break;
            }
        }
    return mas;
}

protected static void SelectionSort(ref int[] arr)
{
    int min, temp;
    int length = arr.Length;
    for (int i = 0; i < length - 1; i++)
    {
        min = i;
        for (int j = i + 1; j < length; j++)
        {
            if (arr[j] < arr[min])
            {
                min = j;
            }
        }
        temp = arr[i];
        arr[i] = arr[min];
        arr[min] = temp;
    }
}
```

```
protected static double sr(ref int[,] mas, int n)
{
    uint cntr = 0, sum = 0;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            if (mas[i, j] > 0)
            {
                cntr++;
                sum += (uint)mas[i, j];
            }
    return (double)sum / cntr;
}

protected static double izmer1(bool v, int n)
{
    Stopwatch stp = new Stopwatch();
    int[] a = { 1, 2 };
    int i = 0;
    if (v)
    {
        while (i < n)
        {
            stp.Start();
            if (a[0] < a[1]) ;
            if (a[0] < a[1]) ;
            if (a[0] < a[1]) ;
            if (a[0] < a[1]) ;
            if (a[0] < a[1]) ;
            if (a[0] < a[1]) ;
            if (a[0] < a[1]) ;
            if (a[0] < a[1]) ;
            if (a[0] < a[1]) ;
            stp.Stop();
            i++;
        }
        return stp.Elapsed.TotalMilliseconds / (9.0*n);
    }
    else
    {
        int b = 1, c = 1, d = 1;
        while (i < n)
        {
            stp.Start();
            b = a[c];
            a[c] = a[d];
            a[d] = b;
            b = a[c];
            a[c] = a[d];
            a[d] = b;
            b = a[c];
            a[c] = a[d];
```

```
            a[d] = b;
            stp.Stop();
            i++;
        }
        return stp.Elapsed.TotalMilliseconds / (3.0*n);
    }
}

protected static double izmer2(bool v, int n)
{
    Stopwatch stp = new Stopwatch();
    int[,] mas = { { 1, 2 }, { 3, 4 } };
    int i = 0;
    int a = 1, b = 0;
    if (v)
    {
        while (i < n)
        {
            stp.Start();
            if (mas[a, a] > 0) ;
            if (mas[a, b] > 0) ;
            if (mas[b, a] > 0) ;
            if (mas[b, b] > 0) ;
            if (mas[a, a] > 0) ;
            if (mas[a, b] > 0) ;
            if (mas[b, a] > 0) ;
            if (mas[b, b] > 0) ;
            if (mas[a, a] > 0) ;
            stp.Stop();
            i++;
        }
        return stp.Elapsed.TotalMilliseconds / (9.0*n);
    }
    else
    {
        uint sum = 1;
        while (i < n)
        {
            stp.Start();
            sum += (uint)mas[a, b];
            sum += (uint)mas[a, b];
            sum += (uint)mas[a, b];
            stp.Stop();
            i++;
        }
        return stp.Elapsed.TotalMilliseconds / (3.0*n);
    }
}

public static void Main()
{
```

```
            Console.Write("Введите кол-во проходов: ");
            int c = Int32.Parse(Console.ReadLine());
            Console.Write("Введите размер одномерного массива: ");
            int n = Int32.Parse(Console.ReadLine());
            Console.Write("Содержание массива(1 - min; 2-medium 3-max): ");
            int k = Int32.Parse(Console.ReadLine());
            int[] mas = gener1(k, n);
            Console.Write("Введите размер квадратной матрицы: ");
            int nn = Int32.Parse(Console.ReadLine());
            Console.Write("Содержание матрицы(1 - min; 2-medium 3-max): ");
            k = Int32.Parse(Console.ReadLine());
            int[,] mass = gener2(k, nn);

            Stopwatch stp = new Stopwatch();
            while (c > 0)
            {
                stp.Start();
                SelectionSort(ref mas);
                stp.Stop();
                Console.Write("A: {0} ms  Тактов: {1}",
stp.Elapsed.TotalMilliseconds, stp.ElapsedTicks);
                stp.Reset();

                stp.Start();
                sr(ref mass, nn);
                stp.Stop();
                Console.WriteLine("\tB: {0} ms  Тактов: {1}",
stp.Elapsed.TotalMilliseconds, stp.ElapsedTicks);
                stp.Reset();
                c--;
            }
            Console.WriteLine("A:\r\nОп. присв.: {0} сек",izmer1(false,10000));
            Console.WriteLine("Оп сравн.: {0} сек",izmer1(true,10000));
            Console.WriteLine("B:\r\nОп. присв.: {0} сек",izmer2(false,10000));
            Console.WriteLine("Оп. сравн.: {0} сек",izmer2(true,10000));
            Console.WriteLine("Жмите Enter!");
            Console.ReadLine();
        }
    }
}
```

3 ХОД РАБОТЫ

1. Написать две программы (A и B) на каком-либо алгоритмическом языке соответственно номеру варианта, выданному преподавателем. Определить те операторы и/или операции программы, кото-

рые оказывают наиболее существенное влияние на время выполнения алгоритма.

2. Разработать два тестовых набора (и целых чисел) для каждой из двух программ (A и B), при которых их время выполнения окажется минимальным и максимальным.

3. Определить аналитические зависимости минимального, максимального и среднего числа выполнения выбранных операторов и/или операций от количества n исходных данных для программ A и B:

4. Определить время выполнения операторов и/или операций программ (A и B), которые существенно влияют на время выполнения исследуемого алгоритма. Для задач данной лабораторной работы это t^{A}_{c}, t^{B}_{c} , t^{A}_{m} и t^{B}_{m} (время выполнения операции сравнения и оператора присваивания в алгоритмах A и B соответственно). Для определения этих значений рекомендуется использовать программу Time.

4 ВАРИАНТЫ ЗАДАНИЙ

1) A: сортировка простыми включениями;
B: минимальный элемент двумерного массива размером n*n .

2) A: сортировка прямым выбором;
B: сумма нечётных отрицательных чисел двумерного массива размером n*n.

3) A: сортировка прямым обменом;
B: количество положительных чисел двумерного массива размером n*n.

4) A: сортировка простыми включениями;
B: сумма минимальных чисел столбцов матрицы размером n*n.

5) A: сортировка прямым выбором;
B: среднее арифметическое отрицательных чисел двумерного массива размером n*n.

6) A: сортировка прямым обменом;

B: количество чётных положительных чисел двумерного массива размером n*n.

7) A: сортировка простыми включениями;
B: количество отрицательных чисел, кратных числу 3 в двумерном массиве размером n*n.

8) A: сортировка прямым выбором;
B: среднее арифметическое положительных чисел двумерного массива размером n*n.

9) A: сортировка прямым обменом;
B: сумма чётных положительных чисел двумерного массива размером n*n.

10) A: сортировка простыми включениями;
B: количество отрицательных чисел двумерного массива размером n*n.

11) A: сортировка прямым выбором;
B: сумма максимальных чисел строк матрицы размером n*n.

12) A: сортировка прямым обменом;
B: сумма положительных чисел, кратных числу 5 в двумерном массиве размером n*n.

13) A: обменная сортировка (2-ой вариант);
B: наибольший общий делитель двух целых чисел .

14) A: сортировка простыми включениями;
B: сумма положительных чисел матрицы размером n*n.

5 СОДЕРЖАНИЕ ОТЧЁТА

1. Цель работы.
2. Вариант задания.
3. Тексты программы, реализующих расчеты по соответствующему варианту, описания функций, используемых для генерации наборов исходных данных, структурные схемы основных функций, используемых в программе.

4. Результаты работы программы.
5. Развернутый вывод по работе.

6 КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Определение трудоемкости алгоритма.
2. Как определить какие операторы более всего влияют на время выполнения алгоритма?
3. Как определить количество сравнений и перестановок в алгоритмах сортировки?
4. Как измерить время выполнения фрагмента программы?

7 БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Т. Кормен, Ч. Лейзерсон, Р. Ривест Алгоритмы: построение и анализ. М.: МЦИМО, 2001.-960 с.,263 ил.
2. Гулд Х., Тобочник Я. Компьютерное моделирование в физике: в 2-х частях. Часть 2: Пер.с англ. - М.: Мир, 1990. – 400 с., ил.
3. Ульянов М.В., Шептунов М.В. Математическая логика и теория алгоритмов, часть 2: Теория алгоритмов. – М.: МГАПИ, 2003. – 80 с.