Калужский филиал федерального государственного бюджетного
образовательного учреждения высшего профессионального образования
«Московский государственный технический университет имени Н.Э. Баумана»

О.Ю. Жарова

Изучение понятия рекурсии. Анализ эффективности рекурсивного и итеративного метода реализации

Методические указания по выполнению лабораторной работы
по дисциплине «Алгоритмы и структуры данных»

Калуга 2013

УДК 681.3-7
ББК 32.973

Данные методические указания разработаны в соответствии с учебным планом специальности 090903.65 «Информационная безопасность автоматизированных систем».

Указания рассмотрены и одобрены:
кафедрой ЭИУ6-КФ «Информационная безопасность автоматизированных систем»
протокол № _____ от _____ 20__ г.
Зав. кафедрой ЭИУ6-КФ _____к.т.н., доц. Мазин А.В.

методической комиссией факультета ЭИУК
протокол № _____ от _____ 20__ г.
Председатель методической комиссии
факультета ЭИУК _____к.т.н., доц. Адкин М.Ю.

методической комиссией Калужского филиала МГТУ им. Н.Э. Баумана
протокол № _____ от _____ 20__г.
Председатель методической комиссии Зам. директора по учебной
КФ МГТУ им. Н.Э. Баумана _____ работе Перерва О.Л.

Рецензент:
к.т.н., доц. кафедры ЭИУ2-КФ _____

Автор ассистент кафедры ЭИУ6-КФ
_____ Жарова Ольга Юрьевна

В методических указаниях изложены принципы анализа эффективности рекурсивны и итеративных алгоритмов. Предназначено для студентов специальности 090903.65 «Информационная безопасность автоматизированных систем» и может быть рекомендовано к применению при проведении лабораторных работ по курсу «Алгоритмы и структуры данных».

СОДЕРЖАНИЕ

Введение

Тема работы: Изучение понятия рекурсии. Анализ эффективности рекурсивного и итеративного метода реализации

Требования к оборудованию:

- Компьютер уровня не ниже, чем Pentium CoreDuo/RAM 1024 MB/HDD 1 GB;
- Операционная система Windows версии XP и выше с типовым вариантом установки;
- среда программирования Visual Studio8 и выше;
- MS Excel.

Цель работы: выработать практические навыки составления алгоритмов с использованием псевдокода, научиться оценивать эффективность составленных алгоритмов

Содержание работы: Изучение понятий рекурсии и итерации, составление псевдокода реализующего индивидуальное задание как с использованием рекурсии, так и итерации. Произведение оценки эффективности алгоритмов. Сравнение эффективности алгоритмов с использованием графиков.

Требования к отчету. Индивидуальный отчет студента должен быть представлен в электронной форме с типовым титульным листом и должен содержать:

1. Постановку задачи.
2. Описание выполнения работы.
3. Экранные формы хода работы.

Длительность работы: 6 академических часа.

Защита работы: собеседование с преподавателем по контрольным вопросам.

1 Теоретические сведения

1.1. Рекурсия и итерация

Рекурсия — это способ организации обработки данных, при котором программа вызывает сама себя непосредственно, либо с помощью других программ.

Итерация — способ организации обработки данных, при котором определенные действия повторяются многократно, не приводя при этом к рекурсивным вызовам программ.

Математическая модель рекурсии заключается в вычислении рекурсивно определенной функции на множестве программных переменных. Примерами таких функций могут служить факториал числа и числа Фибоначчи. В каждом из этих случаев значение функции для всех значений аргумента, начиная с некоторого, определяется через предыдущие значения.

Теорема: *Любой алгоритм, реализованный в рекурсивной форме, может быть переписан в итерационном виде, и наоборот.*

Данная теорема не означает, что временная и емкостная эффективности получающихся программ обязаны совпадать. Однако **наилучшие** рекурсивный и итерационный алгоритмы имеют совпадающую с точностью до постоянного множителя временную сложность.

1.2. Анализ эффективности алгоритма.

Одним из важнейших критериев оценки алгоритма является эффективность, характеризующая прежде всего время выполнения программы $T(n)$ для различных входных данных (параметра n).

Нахождение точной зависимости $T(n)$ для конкретной программы — задача достаточно сложная. По этой причине обычно ограничиваются **асимптотическими оценками** этой функции, то есть описанием ее примерного поведения при больших значениях параметра n .

Скорость роста сложности алгоритма играет важную роль, и определяется старшим членом формулы описывающей поведение алгоритма.

Поэтому можно пренебречь младшими членами, которые растут медленнее. Отбросив все младшие члены, *порядок* функции или алгоритма, скоростью роста сложности которого она является. Алгоритмы можно сгруппировать по скорости роста их сложности. Можно выделить три категории:

- алгоритмы, сложность которых растет по крайней мере так же быстро, как данная функция;
- алгоритмы, сложность которых растет с той же скоростью;
- алгоритмы, сложность которых растет медленнее, чем эта функция.

Омега большое

Класс функций, растущих по крайней мере так же быстро, как *f*, мы обозначаем через $\Omega(f)$. Функция *g* принадлежит этому классу, если при всех значениях аргумента n, больших некоторого порога n_0, значение $g(n) > cf(n)$ для некоторого положительного числа *c*. Можно считать, что класс $\Omega(f)$ задается указанием свой нижней границы: все функции из него растут по крайней мере так же быстро, как *f*.

О большое

На другом конце спектра находится класс *O(f)*. Этот класс состоит из функций, растущих не быстрее *f*. Функция *f* образует верхнюю границу для класса *O(f)*. С формальной точки зрения функция *g* принадлежит классу *O(f),* если $g(n) \le cf(n)$ для всех *n,* больших некоторого порога n_0, и для некоторой положительной константы *c*.

Именно этому классу будет принадлежать большая часть исследуемых алгоритмов

Тета большое

Через $\Theta(f)$ обозначают класс функций, растущих с той же скоростью, что и *f*. С формальной точки зрения этот класс представляет собой пересечение двух предыдущих классов, $\Theta(f) = \Omega(f) \cap O(f)$.

Если алгоритм не зависит от входных данных, например вычисление результата производится по какой-то математической формуле без использования рекурсий и итераций он имеет **единичную сложность Θ(1).**

Если число операций пропорционально числу n,

характеризующему входные данные, говорят о **линейной сложности - Θ(*n*)**. Т.е. сложность алгоритма растет линейно в зависимости от роста потока входных данных.

Если алгоритм использует например функцию возведения в степень, которая не может быть реализована быстрее, чем за логарифмическое время $log_2 n$, то в таком алгоритме количество операций примерно пропорционально *log n* (в информатике принято не указывать основание двоичного логарифма), и он имеет **логарифмическую сложность Θ (*log n*)**.

Очевидно, что алгоритмы рост числа операции которых описывается квадратичной функцией имеют — **квадратичную сложность Θ(n^2)**, соответственно алгоритмы с экспоненциальным ростом операций — **экспоненциальную сложность Θ(2^n).**

Таблица 1- Сравнительная таблица времен выполнения алгоритмов

Сложность алгоритма	**n=10**	**$n=10^3$**	**$n=10^6$**
log n	0.2 сек.	0.6 сек.	1.2 сек.
n	0.6 сек.	1 мин.	16.6 час.
n^2	6 сек.	16.6 час.	1902 года
2^n	1 мин.	10^{295} лет	10^{300000} лет

Как наглядно демонстрирует таблица, самыми неэффективными являются алгоритмы характер роста операций которых, описывается экспонентой и квадратичной функцией.

2 Практическая часть

2.1. Пример составления рекурсивного алгоритма

Рекурсивное вычисление факториала числа имеет вид:

```
Factorial(N)
N       число, факториал которого нам нужен
Factorial возвращает целое число


if (N=l) then
   return 1
else
    smaller = N-l
    answer=Factorial(smaller)
    return (N*answer)
end if
```

2.2. Выполнение лабораторной работы

1. Согласно индивидуальному заданию, составить два алгоритма используя псевдокод. Первый алгоритм должен быть реализован без рекурсии (с использованием итерации или математической формулы, если это возможно), второй алгоритм должен решать поставленную задачу с использованием рекуррентного метода.
2. Проанализировать получившиеся алгоритмы, найти набор входных данных для наихудшего и наилучшего случая.
3. Посчитать эффективность для полученных алгоритмов на различных наборах данных (минимальное число наборов 10) и построить графики в MS Excel.
4. Реализовать алгоритмы на любом выбранном языке программирования.
5. Составить отчет в электронном виде.
6. Ответить на дополнительные вопросы преподавателя.

2.3. Варианты индивидуальных заданий

1. Дано натуральное число n. Вычислить сумму его цифр.
2. Дано натуральное число n>1. Выведите все простые множители этого числа в порядке убывания. Например: ввод — 18, вывод — 3, 3, 2.
3. Дано натуральное число n. Вычислить его факториал.
4. Дано натуральное число n. Вывести последовательность ряда Фибоначи до n-ого члена.
5. Дано натуральное число n. Написать алгоритм, выводящий все возможные перестановки для целых чисел от 1 до n.
6. Дано натуральное число n. Написать алгоритм, выводящий все возможные подмножества множества {1,2,..,n}
7. Дано натуральное число n. Написать алгоритм, выводящий все возможные представления натурального числа n в виде суммы других натуральных чисел.
8. Дана монотонная последовательность, в которой каждое натуральное число k встречается ровно k раз: 1, 2, 2, 3, 3, 3, 4, 4, 4, 4, ... По данному натуральному n вывести первые n членов этой последовательности. Например, ввод — 2, вывод — 1, 2; ввод — 5, вывод — 1,2,2,3,3.
9. Дано натуральное числа n и q. Вывести геометрическую прогрессию для n-го члена, с постоянным множителем q.
10. Дано натуральное число n. Вывести алгебраическую прогрессию для n-го члена.
11. Найти наибольший общий делитель, для двух чисел M и N НОД(M,N). Например: ввод — 9,15, вывод — 3; ввод — 51,34, вывод — 17.
12. Найти наименьшее общее кратное для двух числе M и N НОК(M,N). Например: ввод —4,16, вывод — 16; ввод — 16, 20, вывод — 80.

2.4. Требования к оформлению отчета

Отчет по лабораторной работе должен содержать:

1. Титульный лист установленного образца.
2. Постановку задачи.
3. Описание хода выполнения:
   - два алгоритма;
   - анализ наихудшего и наилучшего случая для каждого из них;
   - таблицу, демонстрирующую зависимость количества операций от набора данных, два графика построенных по данным таблицы;
   - Код программы, скриншоты демонстрирующие ее работу.
4. Вывод, содержащий заключение, сделанное на основе анализа полученного алгоритма.

3 Контрольные вопросы

1) Что такое рекурсия?
2) Отличие итеративного и рекурсивного способа реализации программ?
3) Что такое наилучший случай?
4) В чем преимущество рекурсивной реализации?
5) Что такое О большое?
6) Дать определение термину Тетта большое.

Литература

1. Дж. Макконелл. Основы современных алгоритмов. 2-е дополнительное издание. – М.:Техносфера, 2004. – 368 с.