Урок 3. Трудоемкость алгоритмов и временные оценки

Элементарные операции в языке записи алгоритмов

Для получения функции трудоемкости алгоритма, представленного в формальной системе введенной абстрактной машины необходимо уточнить понятия «элементарных» операций, соотнесенных с языком высокого уровня.

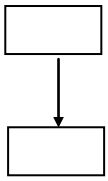
В качестве таких «элементарных» операций предлагается использовать следующие:

1) Простое присваивание: $a \leftarrow b$;
2) Одномерная индексация *a[i]* : (адрес *(a)+i**длинна элемента);
3) Арифметические операции: *(*, /, -, +);*
4) Операции сравнения: $a < b$;
5) Логические операции *(l1) {or, and, not} (l2);*

Опираясь на идеи структурного программирования, исключим команду перехода по адресу, считая ее связанной с операцией сравнения в конструкции ветвления.

После введения элементарных операций анализ трудоемкости основных алгоритмических конструкций в общем виде сводится к следующим положениям:

A) Конструкция «Следование»

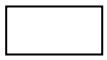


Трудоемкость конструкции есть сумма трудоемкостей блоков, следующих друг за другом.

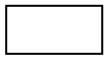
$F_{«следование»} = f_1 + \ldots + f_k$, *где* k – количество блоков.

B) Конструкция «Ветвление»

if (l) then

 f_{then} с вероятностью p

else

 f_{else} с вероятностью *(1-p)*

Общая трудоемкость конструкции «Ветвление» требует анализа вероятности выполнения переходов на блоки «Then» и «Else» и определяется как:

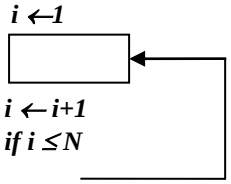
$$F_{«ветвление»} = f_{then} * p + f_{else} * (1-p).$$

C) Конструкция «Цикл»

for i ← 1 to N

end

⇔



После сведения конструкции к элементарным операциям ее трудоемкость определяется как:

$$F_{«цикл»} = 1+3*N+N*f_{«тела\ цикла»}$$

Примеры анализа простых алгоритмов

Пример 1. Задача суммирования элементов квадратной матрицы

```
SumM (A, n; Sum)
Sum ← 0
For i ← 1 to n
        For j ← 1 to n
                Sum ← Sum + A[i,j]
end for
Return (Sum)
End
```

Алгоритм выполняет одинаковое количество операций при фиксированном значении n, и следовательно является количественно-зависимым. Применение методики анализа конструкции «Цикл » дает:

$$F_A(n)=1+1+ n *(3+1+ n *(3+4))=7 n^2+4* n +2 = \Theta(n^2),$$

заметим, что под *n* понимается линейная размерность матрицы, в то время как на вход алгоритма подается n^2 значений.

Пример 2. Задача поиска максимума в массиве

```
MaxS (S,n; Max)
Max ← S[1]
For i ← 2 to n
        if Max < S[i]
                then   Max ← S[i]
end for
```

return Max
End

Данный алгоритм является количественно-параметрическим, поэтому для фиксированной размерности исходных данных необходимо проводить анализ для худшего, лучшего и среднего случая.

А) Худший случай

Максимальное количество переприсваиваний максимума (на каждом проходе цикла) будет в том случае, если элементы массива отсортированы по возрастанию. Трудоемкость алгоритма в этом случае равна:

$$F_A^{\wedge}(n)=1+1+1+ (n-1) (3+2+2)=7\ n - 4 = \Theta(n).$$

Б) Лучший случай

Минимальное количество переприсваивания максимума (ни одного на каждом проходе цикла) будет в том случае, если максимальный элемент расположен на первом месте в массиве. Трудоемкость алгоритма в этом случае равна:

$$F_A^{\vee}(n)=1+1+1+ (n-1) (3+2)=5\ n - 2 = \Theta(n).$$

В) Средний случай

Алгоритм поиска максимума последовательно перебирает элементы массива, сравнивая текущий элемент массива с текущим значением максимума. На очередном шаге, когда просматривается *к-ый* элемент массива, переприсваивание максимума произойдет, если в подмассиве из первых *к* элементов максимальным элементом является последний. Очевидно, что в случае равномерного распределения исходных данных, вероятность того, что максимальный из *к* элементов расположен в определенной (последней) позиции равна *1/к*. Тогда в массиве из *n* элементов общее количество операций переприсваивания максимума определяется как:

$$\sum_{i=1}^{N} 1/i = Hn \approx Ln(N) + \gamma, \quad \gamma \approx 0{,}57$$

Величина *Hn* называется *n-ым* гармоническим числом. Таким образом, точное значение (математическое ожидание) среднего количества операций присваивания в алгоритме поиска максимума в массиве из *n* элементов определяется величиной *Hn* (на бесконечности количества испытаний), тогда:

$$\bar{F}_A(n)=1 + (n-1) (3+2) + 2 (Ln(n) + \gamma)=5\ n +2\ Ln(n) - 4 +2\ \gamma = \Theta(n).$$

Переход к временным оценкам

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

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

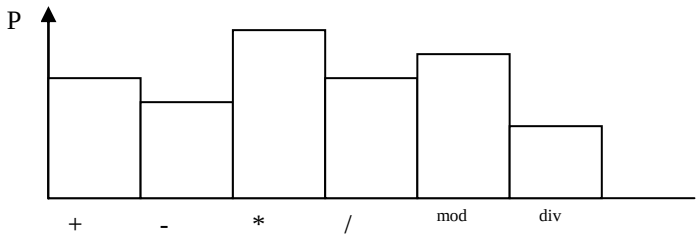
2) Метод Гиббсона

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 (рис. 1), создаются соответствующие тестовые программы, и определяется среднее время на операцию в данном типе задач – $\bar{t}_{тип\ задачи}$.

Рис. 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:

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

Пример пооперационного временного анализа

В ряде случаев именно пооперационный анализ позволяет выявить тонкие аспекты рационального применения того или иного алгоритма решения задачи. В качестве примера рассмотрим задачу умножения двух комплексных чисел:

$$(a+bi)*(c+di)=(ac - bd) + i(ad + bc)=e + if$$

1. Алгоритм A1 (прямое вычисление e, f – четыре умножения)

```
MultComplex1 (a, b, c, d; e, f)
        e←a*c - b*d          f_A1 =8 операций
        f←a*d + b*c          f_*  =4 операций
        Return (e, f)        f_±  =2 операций
End.                         f_←  =2 операций
```

2. Алгоритм A2 (вычисление e, f за три умножения)

```
MultComplex2 (a, b, c, d; e, f)
        z1←c*(a + b)
        z2←b*(d + c)         f_A2 =13 операций
        z3←a*(d - c)         f_*  =3 операций
        e←z1 - z2            f_±  =5 операций
        f←z1 + z3            f_←  =5 операций
        Return (e, f)
End.
```

Пооперационный анализ этих двух алгоритмов не представляет труда, и его результаты приведены справа от записи соответствующих алгоритмов.

По совокупному количеству элементарных операций алгоритм *A2* уступает алгоритму *A1*, однако в реальных компьютерах операция умножения требует большего времени, чем операция сложения, и можно путем пооперационного анализа ответить на вопрос: при каких условиях алгоритм *A2* предпочтительнее алгоритма *A1*?

Введем параметры q и r, устанавливающие соотношения между временами выполнения операции умножения, сложения и присваивания для операндов действительного типа.

Тогда мы можем привести временные оценки двух алгоритмов к времени выполнения операции сложения/вычитания – t_+:

$t_* = q*t_+$, $q>1$;

$t_←=r*t_+$, $r<1$, тогда приведенные к t_+ временные оценки имеют вид:

$T_{A1} = 4*q*t_+ + 2*t_+ + 2*r*t_+ = t_+*(4*q+2+2*r)$;

$T_{A2} = 3*q*t_+ + 5*t_+ + 5*r*t_+ = t_+*(3*q+5+5*r)$.

Равенство времен будет достигнуто при условии:

$4*q+2+2*r = 3*q+5+5*r$, откуда:

$q = 3 + 3r$ и следовательно при $q > 3 + 3r$ алгоритм *A2* будет работать более эффективно.

Таким образом, если среда реализации алгоритмов *A1* и *A2* – язык программирования, обслуживающий его компилятор и компьютер на котором реализуется задача – такова, что время выполнения операции умножения двух действительных чисел более чем втрое превышает время сложения двух действительных чисел, в предположении, что $r << 1$, а это реальное соотношение, то для реализации более предпочтителен алгоритм *A2*.

Конечно, выигрыш во времени пренебрежимо мал, если мы перемножаем только два комплексных числа, однако, если этот алгоритм является частью сложной вычислительной задачи с комплексными числами, требующей существенно значимого по времени количества умножений, то выигрыш во времени может быть ощутим. Оценка такого выигрыша на одно умножение комплексных чисел следует из только что проведенного анализа:

$$\Delta T = (q - 3 - 3*r)* t_+$$

Литературные источники

1. Ахо А. Ульман Дж. Теория синтаксического анализа, перевода и компиляции. Том 1 – Синтаксический анализ. – М.: Мир, 1978 г. – 612 с., ил.
2. Ахо А., Хопкрофт Дж., Ульман Дж. Структуры данных и алгоритмы: Пер. с англ.: – М.: Издательский дом «Вильямс», 2001 г. –384 с., ил.
3. Вирт Н. Алгоритмы и структуры данных: Пер. с англ. – 2-ое изд., испр. – СПб.: Невский диалект, 2001 г. – 352 с., ил.
4. Карпов Ю.Г. Теория автоматов – СПб.: Питер, 2002 г. – 224с., ил.
5. Кнут Д. Искусство программирования. Тома 1, 2, 3. 3-е изд. Пер. с англ.: Уч. пос. – М.: Изд. дом "Вильямс", 2001 г.
6. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. – М.: МЦНМО, 2001 г. – 960 с., 263 ил.
7. Макконнел Дж. Анализ алгоритмов. Вводный курс. – М.: Техносфера, 2002 г. –304 с.
8. Новиков Ф. А. Дискретная математика для программистов. – СПб.: Питер, 2001 г. – 304 с., ил.

9. Романовский И.В. Дискретный анализ. Учебное пособие для студентов, специализирующихся по прикладной математике. – Издание 2-ое, исправленное. – СПб.; Невский диалект, 2000 г. – 240 с., ил.
10. Успенский В.А. Машина Поста. – М.: Наука, 1979 г. – 96 с. – (Популярные лекции по математике).
11. Ульянов М.В., Шептунов М.В. Математическая логика и теория алгоритмов, часть 2: Теория алгоритмов. – М.: МГАПИ, 2003. – 80 с.

Контрольные вопросы

1) Элементарные операции в псевдоязыке высокого уровня;
2) Анализ трудоемкости основных алгоритмических конструкций;
3) Построение функции трудоемкости для суммирования матрицы;
4) Построение функции трудоемкости для задачи поиска максимума ;
5) Проблемы при переходе от трудоемкости к временным оценкам;
6) Методики перехода от функции трудоемкости к временным оценкам;
7) Возможности пооперационного анализа алгоритмов на примере задачи умножения комплексных чисел;

Задание: ответить на контрольные вопросы и прислать ответы на проверку (файл должен быть в формате pdf).