Дисциплина

«РАЗРАБОТКА ЭФФЕКТИВНЫХ АЛГОРИТМОВ»

Лекция 7

МЕТОД КЛАССОВ ЭКВИВАЛЕНТНОСТИ И АЛГОРИТМЫ СОРТИРОВКИ ТРЕХ ЧИСЕЛ ПО МЕСТУ

Лектор: Михаил Васильевич Ульянов,

muljanov@mail.ru, **8 916 589 94 04**

Математическое введение

1. Принцип Дирихле.

Ио́ганн Пе́тер Гу́став Лежён Дирихле́ (нем. Johann Peter Gustav Lejeune Dirichlet)

«Если голуби рассажены в клетки, причём число голубей больше числа клеток, то хотя бы в одной из клеток находится более одного голубя».

Невозможно построить биективное отображение на двух конечных множествах неравной мощности.

2. Отношение эквивалентности

Бинарное отношение — любое подмножество декартова произведения множеств или множества на себя. Если отношение: рефлексивно, симметрично и транзитивно, то оно называется отношением эквивалентности. Отношение эквивалентности разбивает множество на непересекающиеся подмножества — классы эквивалентности.

Анализ алгоритмов методом классов эквивалентности

Основная идея: рассмотрим множество входов фиксированной длины D_n и множество значений трудоемкости алгоритма на входах из D_n — множество F_n.

Очевидно, что мощность D_n существенно больше, чем F_n. В силу принципа Дирихле некоторое значение трудоемкости порождается не менее, чем двумя входами. Таким образом, мы можем ввести на D_n отношение эквивалентности — «иметь равную трудоемкость». Это отношение разбивает D_n на подмножества — классы эквивалентности. Все входы из одного класса порождают одинаковую трудоемкость для исследуемого алгоритма.

Если мы можем определить мощность этих классов, то очевидно просто получить трудоемкость в среднем. В реальности это достаточно трудная задача. Вариант решения — разбить D_n на равномощные подмножества (опираясь на особенности алгоритма) и затем сгруппировать их в классы по трудоемкости.

Задача сортировки трех чисел по месту

Рассмотрим следующую, на первый взгляд простую задачу. Исходными данными являются три произвольных числа, реально — адреса этих чисел. Необходимо отсортировать эти числа по возрастанию и возвратить результат «по месту», т. е. результат должен быть расположен в тех же ячейках, что и исходные числа. Может быть предложено несколько различных алгоритмов решения с различными ресурсными характеристиками. Цель этого небольшого исследования — получить алгоритм с оптимальной трудоемкостью в среднем случае на основе результатов детального анализа «простого» алгоритма.

Рассмотрим вначале алгоритм, решающий данную задачу с помощью сравнений пар чисел и соответствующих перестановок. Фактически это реализация метода «пузырька» (камешка) для трех чисел. Запись алгоритма **`SortABC_A1`**, реализующего этот метод, выглядит следующим образом:

Алгоритм SortABC_A1

```
SortABC_A1(a, b, c)
  If a>b                          1
    then (обмен a и b)            3
      t ← a
      a ← b
      b ← t
  If b>c                          1
    then (обмен b и c)            3
      t ← b
      b ← c
      c ← t
  If a>b                          1
    then (обмен a и b)            3
      t ← a
      a ← b
      b ← t
Return (a, b, c)
End
```

Метод классов эквивалентности для анализа алгоритмов

Для анализа трудоемкости этого алгоритма воспользуемся методом классов. Действительно, множество всех допустимых входов достаточно велико, и определяется всеми возможными тройками чисел, представляемых в выбранном формате данных. Даже для целых чисел без знака длиной два байта (16 бит)

$$|D_A| = (2^{16})^3 = 2^{48},$$

но трудоемкость алгоритма зависит не от значений чисел, а от соотношений между ними в смысле сравнений. Введем обозначение «1» для минимального значения, «3» для максимального, и «2» для промежуточного. Очевидно, что все множество входов разбивается при этом на $3! = 6$ классов входных данных, и трудоемкость алгоритма для любого входа из каждого класса одинакова. Введем следующие обозначения классов входных данных, которые соответствуют шести различным соотношениям по сравнению между тремя числами.

Таблица обозначений классов

	a	b	c
D_1	1	2	3
D_2	1	3	2
D_3	2	1	3
D_4	2	3	1
D_5	3	1	2
D_6	3	2	1

Трудоемкости для различных классов могут быть получены на основе анализа выполнения тех ветвей алгоритма, которые задействованы при обработке входов, соответствующих данному классу. В результате получаем:

$$f_{A1}(D_1) = 3,$$

$$f_{A1}(D_2) = 6,$$

$$f_{A1}(D_3) = 6,$$

$$f_{A1}(D_4) = 9,$$

$$f_{A1}(D_5) = 9,$$

$$f_{A1}(D_6) = 12,$$

таким образом, лучшим случаем является любой вход из класса D_1, при этом $f_{A1}^{\vee}(D) = 3$, а худшим — любой вход из класса D_6, для которого $f_{A1}^{\wedge}(D) = 12$. Трудоемкость в среднем может быть получена, если задана вероятность появления входов, относящихся к различным классам. В предположении, что эти вероятности одинаковы, получаем

$$\overline{f_{A1}}(D) = \frac{1}{6} \cdot (3 + 6 + 6 + 9 + 9 + 12) = 7\frac{1}{2}.$$

Обсуждение результатов и идеи модификации

Полученные результаты позволяют сделать вывод о том, что алгоритм **`SortABC_A1`** относится к типу L_C и классу PR в этом типе, при этом параметрическая зависимость определяется взаимными соотношениями между числами по операции сравнения.

Разработка оптимального по трудоемкости алгоритма должна базироваться на очевидном предположении, что вход из каждого класса обрабатывается за минимально возможное число операций. Таким образом, необходимо используя сравнения, определить принадлежность входа к одному из шести классов, и построить оптимальный фрагмент обмена ячеек, сортирующий данный вход.

Основная идея состоит в использовании кольцевого сдвига для входов из классов D_4 и D_5, что позволяет выполнить сортировку за 4 операции. Заметим, что такой подход приводит только к одному обмену ячеек для класса D_6.

Алгоритм SortABC_A2

```
SortABC_A2(a, b, c)
  If a>b
    then

      If b>c
        then (класс «321» - обмен 3 и 1 местами f=2+3=5)
          t ← a
          a ← c
          c ← t
        else

          If a>c
            then (класс «312» - кольцевой обмен f=3+4=7)
              t ← a
              a ← b
              b ← c
              c ← t
```

```
        else (класс «213» - обмен 2 и 1 местами f=3+3=6)
          t ← a
          a ← b
          b ← t
      end If a>c

  end If b>c

else a<b

  If b>c
    then

      If a>c
        then (класс «231» - кольцевой обмен f=3+4=7)
          t ← c
          c ← b
          b ← a
          a ← t
```

```
          else (класс «132» - обмен 3 и 2 f=3+3=6)
            t ← b
            b ← c
            c ← t
        end If a>c

      else b<c (класс «123» - без обменов f=2+0=2)

    end If b>c

  end If a>b

Return (a, b, c)

End
```

В приведенных внутри текста алгоритма комментариях в записи типа `f=3+4=7` первое слагаемое есть трудоемкость идентификации класса входных данных, а второе — трудоемкость собственно сортировки.

Трудоемкость алгоритма **SortABC_A2** для различных классов определяется на основании анализа текста алгоритма, и составляет

$$f_{A2}(D_1) = 2,$$
$$f_{A2}(D_2) = 6,$$
$$f_{A2}(D_3) = 6,$$
$$f_{A2}(D_4) = 7,$$
$$f_{A2}(D_5) = 7,$$
$$f_{A2}(D_6) = 5,$$

таким образом, лучшим случаем для алгоритма **SortABC_A2** является любой вход из класса D_1, при этом $f_{A2}^{\vee}(D) = 2$, а худшим — любые входы из классов D_4 и D_5, для которых $f_{A2}^{\wedge}(D) = 7$.

Трудоемкость в среднем, в предположении о равной вероятности входов из любого класса составляет:

$$\overline{f_{A2}}(D) = \frac{1}{6} \cdot (2 + 6 + 6 + 7 + 7 + 5) = 5\frac{1}{2}.$$

Полученный результат не означает, что при каждом запуске данного алгоритма экспериментально определенная трудоемкость будет равна 5,5.

Боле того, эта трудоемкость вообще никогда не будет равна указанному значению, поскольку функция трудоемкости для конкретного входа есть целое положительное число. Формула для $\overline{f_{A2}}(D)$ есть теоретически полученное математическое ожидание функции трудоемкости этого алгоритма, как дискретной ограниченной случайной величины.

Этот результат был получен методом вероятностного анализа для классов входных данных, в предположении о равной вероятности появления входов, принадлежащих любому из шести введенных классов.

Обсуждение результатов

Полученные результаты позволяют отнести алгоритм **SortABC_A2** к типу L_C и классу PR для этого типа. Кроме того отметим, что алгоритм **SortABC_A2** обладает меньшим размахом варьирования (7 - 2=5) значений трудоемкости.

С точки зрения теории ресурсной эффективности возникает резонный вопрос — за счет чего в алгоритме **SortABC_A2** удалось снизить трудоемкость в среднем?

Ответ — более длинный текст записи **SortABC_A2** порождает более длинный программный код. Таким образом, платой за снижение трудоемкости является увеличение затрат памяти в области программного кода.

Еще один вопрос, оставшийся пока без ответа — это доказательство оптимальности алгоритма **SortABC_A2** по трудоемкости.

Может ли быть предложен иной алгоритм, имеющий трудоемкость в среднем, лучшую, чем $5\frac{1}{2}$ базовых операций? Обоснуем отрицательный ответ на этот вопрос, при условии, что принятая модель вычислений и введенные базовые операции остаются неизменными. Можно рассуждать следующим образом — определение принадлежности входа к одному из шести классов невозможно сделать за два сравнения, это следует из рассмотрения бинарного дерева сравнений. Полное бинарное дерево глубиной три содержит восемь листьев, а дерево глубиной два — только четыре, поэтому бинарное дерево с шестью листьями имеет глубину три.

Наличие только шести листьев позволяет не строить полное бинарное дерево глубиной три, однако при этом не более чем два класса могут быть идентифицированы только за два сравнения. Но именно такое бинарное дерево сравнений и реализует алгоритм **`SortABC_A2`**.

В принятой модели вычислений обмен содержимого ячеек требует минимум трех операций, а кольцевой сдвиг содержимого трех ячеек не может быть реализован быстрее, чем за четыре присваивания. Таким образом, алгоритм выполняет минимальное число базовых операций для любого входа из каждого класса входных данных, и, следовательно, он оптимален по трудоемкости для любого входа. Сказанное относится к общему случаю задачи сортировки трех чисел — т. е. к случаю, когда эти числа различны.