Краткий конспект

Лекция 1. Поиск мотивов

версия 0.1(незавершенная)

Д. Ищенко* Б. Коварский* И. Алтухов* Д. Алексеев*

11 февраля, 2016

*МФТИ

1 Зачем искать мотивы?

2 Несколько слов о сложности алгоритмов

При разработке алгоритма важно представлять и оценивать кол-во времени необходимое для его исполнения при конкретных входных данных, а также объем компьютерных ресурсов, задействованных при исполнении алгоритма. Когда мы говорим про «конкретные данные», мы подразумеваем, определенную величину, например, размер памяти выделяемой под входные данные в битах. Будем называть эту величину n. Для простоты будем считать, что время необходимое для исполнения алгоритма *пропорционально* кол-ву элементарных операций (которые в свою очередь определены архитектурой процессора). Будем считать элементарными операциями: сложение, вычитание, умножение, деление, вычисление корня. Тогда оценка времени сводится к определению $f(n)$, функции количества элементарных операций от размера входных данных. Нас не будет интересовать точное значение $f(n)$ (это и не всегда возможно определить), а только лишь его оценка (чаще всего оценка сверху). Определяется она с помощью термина «O большое» для ассимптотического поведения функций. $f(n) = O(g(n))$ означает, что кол-во операций $f(n)$ при увеличении n будет возрастать не быстрее, чем $g(n)$ умноженная на некоторую константу.

$$\exists(C > 0), n_0 : \forall(n > n_0)\ f(n) \leq Cg(n)$$

Напрмер, сложность алгоритма который вычисляет простую сумму k входных чисел a_k оценивается, как $O(k)$. Грубо говоря, мы выполняем $(k-1)$ операций сложения. Таким образом кол-во операций $f(n)$ будет возрастать пропорционально кол-ву входных чисел k. Объем входных данных в битах можем оценить, как некоторую константу (например кол-во бит, зарезервированных под одно входящее число) умноженную на их кол-во $n = c \cdot k$, тогда $f(n) = f(c \cdot k) = k - 1 = O(k)$. Рассмотрим другой пример, алгоритм находящий ... алгоритм $O(n^2)$ ...

Аналогичные рассуждения в оценке применимы и к вычислению необходимой памяти (чаще оперативной) выделяемой при исполнении алгоритма. Оценивается кол-во выделяемых бит $m(n)$ от размера входных данных и оценивается с помощью $O(p(n))$. Возвращаясь к алгоритму вычисления суммы k чисел a_k. Допустим, алгоритм построен следующим образом: (i) прочесть все k чисел и записать в массив, (ii) просуммировать все a_k и результат записать в s, (iii) выдать результат s. Задействованная оперативная память – величина $m(n) = m(k \cdot c) = k \cdot c + c = c(k+1) = O(k)$ (нам необходимо записать в массив все k чисел, каждое из которых занимает c бит, а также выделить память для перемнной s размера c бит). Если же изменить

алгоритм следующим оразом: (i) объявляется переменная $s = 0$ для хранения суммы, (ii) по-очередно читается одно число из a_k и добавляется к s, $s = s + a_k$, после чего a_k удаляется из памяти (iii) выводится s. То в такой реализации нам необходимо хранить всего два значения s и текующее a_k, а значит всего $2c$ бит. Другими словами, кол-во необходимой памяти *не зависит* от размера входных данных (колва входных чисел), такой вариант оценивается, как $m(n) = 2c = O(1)$. При этом в обоих реализациях алгоритма оценка времени одинакова $f_1(n) = f_2(n) = O(k)$.

Идеальным случаем ... написать про $O(n)$ ...

3 Простой подход к поиску мотива

Вернемся к задаче о поиске мотива. Есть строка (геном) S и паттерн (мотив) M:

```
 S :  TATGCATGCATGA
 M :  ATGCTGA
```

Необходимо определить все позиции вхождения M в S. Рассмотрим самый простой алгоритм, заключающийся в полном переборе всех позиций в S, подставления в них M и проверки попарных совпадений символов (нуклеотидов).

```
 S :  TATGCATGCATGA
M1 :  ATGCATGA
      *
M2     ATGCATGA
       +++++++*
M3      ATGCATGA
        *
M4       ATGCATGA
         *
M5        ATGCATGA
          *
M6         ATGCATGA
           ++++++++
```

Обозначаем символом «*» проверку на совпадение, которая вернула значения $FALSE$ (символы отличаются), а символом «+» проверку, вернувшую $TRUE$ (символы совпадают). Алгоритм можно описать следующим образом: двигаемся

4 Усовершенствование простого подхода

```
 S :  TATGCATGCATGA
M1 :  ATGCATGA
      *
M2     ATGCATGA
       +++++++*
M3         ATGCATGA
           ++++++++
```

```
S  :  TATGCATGCATGA
M1 :  ATGCATGA
      *
M2     ATGCATGA
       +++++++*
M3         ATGCATGA
              +++++
```

Вышеописанные подходы позволоили нам уменьшить количество сравнений. Но это был частный пример и рассуждали мы в «свободной» форме, а хочется это формализовать в виде алгоритма.

5 Z-алгоритм

6 Ссылки