Программирование

Лекция 2. Анализ алгоритмов

- Алгоритмы являются принципиально важным компонентом информатики, т. к. их изучение не требует использования языка программирования или компьютера. Это означает необходимость в методах, позволяющих сравнивать эффективность алгоритмов, не прибегая к их реализации. Самыми значимыми из этих инструментов являются ***модель вычислений RAM*** и ***асимптотический анализ сложности*** наихудших случаев.

Модель вычислений RAM

- Разработка машинно-независимых алгоритмов основывается на гипотетическом компьютере, называющемся машиной с произвольным доступом к памяти (Random Access Machine) или **RAM-машиной**. Согласно этой модели наш компьютер работает таким образом:
- для исполнения любой простой операции (+, *, -, =, if) требуется ровно один временной шаг;
- циклы и подпрограммы не считаются простыми операциями, а состоят из нескольких простых операций. Нет смысла считать подпрограмму сортировки одношаговой операцией, т. к. для сортировки 1 000 000 элементов потребуется определенно намного больше времени, чем для сортировки десяти элементов. Время исполнения цикла или подпрограммы зависит от количества итераций или специфического характера подпрограммы;
- каждое обращение к памяти занимает один временной шаг. Кроме этого, наш компьютер обладает неограниченным объемом оперативной памяти. Кэш и

Анализ сложности наилучшего, наихудшего и среднего случая

- С помощью RAM-модели можно подсчитать количество шагов, требуемых алгоритму для исполнения любого экземпляра задачи. Но чтобы получить общее представление о том, насколько хорошим или плохим является алгоритм, нам нужно знать, как он работает со всеми экземплярами задачи.
- Чтобы понять, что означает наилучший, наихудший и средний случай сложности алгоритма (т. е. время его исполнения в соответствующем случае), нужно рассмотреть исполнение алгоритма на всех возможных экземплярах входных данных. В случае задачи сортировки множество входных экземпляров состоит из всех возможных компоновок ключей *n* по всем возможным значениям *n*.

Анализ сложности наилучшего, наихудшего и среднего случая

На практике наиболее важной является оценка сложности алгоритма в наихудшем случае!

Асимптотические обозначения

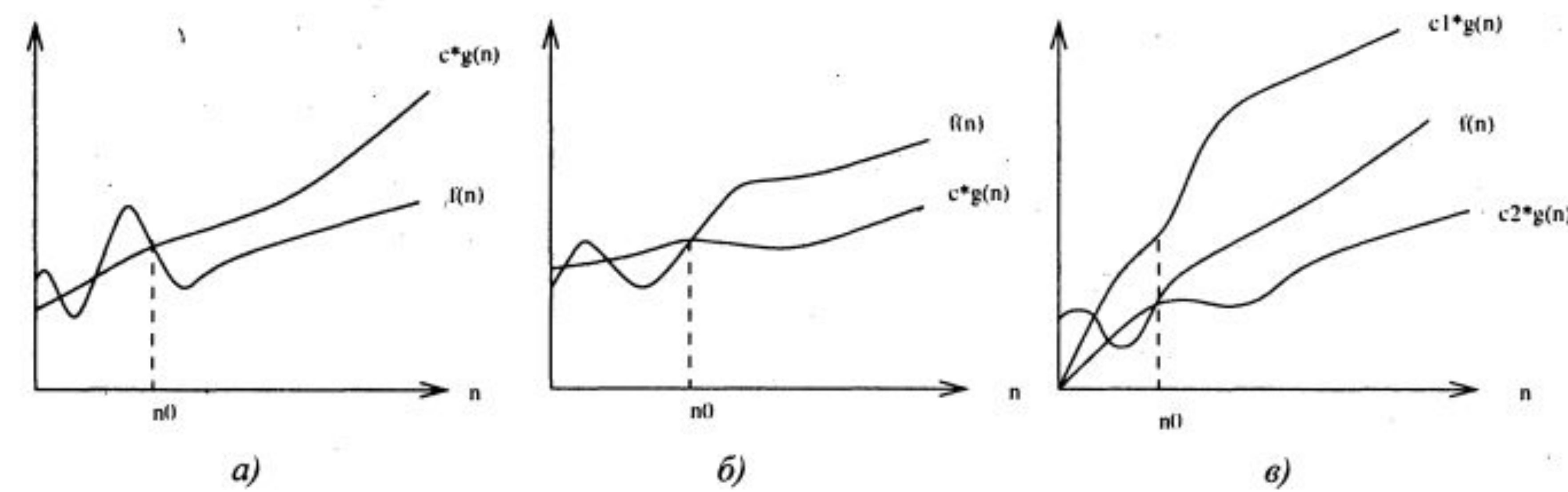
$$T(n) = 12754n^2 + 4353n + 834\lg_2 n + 13546$$

- Неудобно пользоваться

- Намного легче работать с верхней и нижней границами функций временной сложности, используя для этого асимптотические обозначения (O -большое и Ω- большое соответственно).

Асимптотические обозначения

- $f(n) = O(g(n))$ означает, что функция $f(n)$ *ограничена сверху* функцией $c \cdot g(n)$. Иными словами, существует такая константа c, для которой $f(n) \le c \cdot g(n)$ при достаточно большом значении n (т. е. $n \ge n_0$ для некоторой константы n_0);
- $f(n) = \Omega(g(n))$ означает, что функция $f(n)$ *ограничена снизу* функцией $c \cdot g(n)$. Иными словами, существует такая константа c, для которой $f(n) \ge c \cdot g(n)$ для всех $n \ge n_0$;
- $f(n) = \Theta(g(n))$ означает, что функция $f(n)$ *ограничена сверху* функцией $c_1 \cdot g(n)$, а *снизу* функцией $c_2 \cdot g(n)$ для всех $n \ge n_0$. Иными словами, существуют константы c_1 и c_2, для которых $f(n) \le c_1 \cdot g(n)$ и $f(n) \ge c_2 \cdot g(n)$. Следовательно, функция $g(n)$ дает нам хорошие ограничения для функции $f(n)$.



Анализ наихудшего случая и асимптотические обозначения являются инструментами, которые существенно упрощают задачу сравнения эффективности алгоритмов.

$3n^2 - 100n + 6 = O(n^2)$, т. к. выбрано $c = 3$ и $3n^2 > 3n^2 - 100n + 6$;

$3n^2 - 100n + 6 = O(n^3)$, т. к. выбрано $c = 1$ и $n^3 > 3n^2 - 100n + 6$ при $n > 3$;

$3n^2 - 100n + 6 \neq O(n)$, т. к. для любого значения c выбрано $cn < 3n^2$ при $n > c$;

$3n^2 - 100n + 6 = \Omega(n^2)$, т. к. выбрано $c = 2$ и $2n^2 < 3n^2 - 100n + 6$ при $n > 100$;

$3n^2 - 100n + 6 \neq \Omega(n^3)$, т. к. выбрано $c = 3$ и $3n^2 - 100n + 6 < n^3$ при $n > 3$;

$3n^2 - 100n + 6 = \Omega(n)$, т. к. для любого значения c выбрано $cn < 3n^2 - 100n + 6$ при $n > 100c$;

$3n^2 - 100n + 6 = \Theta(n^2)$, т. к. применимо как O, так и Ω;

$3n^2 - 100n + 6 \neq \Theta(n^3)$, т. к. применимо только O;

$3n^2 - 100n + 6 \neq \Theta(n)$, т. к. применимо только Ω.

ЗАДАЧА. Верно ли равенство $2^{n+1} = \Theta(2^n)$?

Решение. Для разработки оригинальных алгоритмов требуется умение и вдохновение. Но при использовании асимптотических обозначений лучше всего подавить все свои творческие инстинкты. Все задачи по асимптотическим обозначениям можно правильно решить, работая с первоначальным определением.

- Верно ли равенство $2^{n+1} = O(2^n)$? Очевидно, что $f(n) = O(g(n))$ тогда и только тогда, когда существует такая константа c, при которой для всех достаточно больших значений n функция $f(n) \le c \cdot g(n)$. Существует ли такая константа? Заметим, что $2^{n+1} = 2 \cdot 2^n$, откуда следует, что $2 \cdot 2^n \le c \cdot 2^n$ для всех $c \ge 2$.

- Верно ли равенство $2^{n+1} = \Omega(2^n)$? Согласно определению, $f(n) = \Omega(g(n))$ тогда и только тогда, когда существует такая константа $c > 0$, при которой для всех достаточно больших значений n функция $f(n) \ge c \cdot g(n)$. Это условие удовлетворяется для любой константы $0 < c \le 2$. Границы O-большое и Ω-большое совместно подразумевают $2^{n+1} = \Theta(2^n)$. ■

ЗАДАЧА. Верно ли равенство $(x + y)^2 = O(x^2 + y^2)$?

Решение. При малейших трудностях в работе с асимптотическим обозначением немедленно возвращаемся к его определению, согласно которому это выражение действительно тогда и только тогда, когда мы можем найти такое значение c, при котором $(x + y)^2 \leq c(x^2 + y^2)$.

Лично я первым делом раскрыл бы скобки в левой части уравнения, т. е. $(x + y)^2 = x^2 + 2xy + y^2$. Если бы в развернутом выражении отсутствовал член $2xy$, то вполне очевидно, что неравенство соблюдалось бы для любого значения $c > 1$. Но т. к. этот член имеется, то нам нужно рассмотреть его связь с выражением $x^2 + y^2$. Если $x \leq y$, то $2xy \leq 2y^2 \leq 2(x^2 + y^2)$. А если $x \geq y$, то $2xy \leq 2x^2 \leq 2(x^2 + y^2)$. В любом случае теперь мы можем ограничить это выражение двойным значением функции с правой стороны. Это означает, что $(x + y)^2 \leq 3(x^2 + y^2)$, поэтому результат остается в силе. ■

Скорость роста и отношения доминирования

- Используя асимптотические обозначения, мы пренебрегаем постоянными множителями, не учитывая их при вычислении функций.
- При таком подходе функции $f(n) = 0{,}001n^2$ и $g(n) = 1000n^2$ для нас одинаковы, несмотря на то, что значение функции $g(n)$ в миллион раз больше значения функции $f(n)$ для любого n.

Скорость роста основных функций

$n/f(n)$	$\lg n$	n	$n \lg n$	n^2	2^n	$n!$
10	0,003 мкс	0,01 мкс	0,033 мкс	0,1 мкс	1 мкс	3,63 мс
20	0,004 мкс	0,02 мкс	0,086 мкс	0,4 мкс	1 мс	77,1 лет
30	0,005 мкс	0,03 мкс	0,147 мкс	0,9 мкс	1 с	$8{,}4\times10^{15}$ лет
40	0,005 мкс	0,04 мкс	0,213 мкс	1,6 мкс	18,3 мин	
50	0,006 мкс	0,05 мкс	0,282 мкс	2,5 мкс	13 дней	
100	0,007 мкс	0,1 мкс	0,644 мкс	10 мкс	4×10^{13} лет	
1 000	0,010 мкс	1,00 мкс	9,966 мкс	1 мс		
10 000	0,013 мкс	10 мкс	130 мкс	100 мс		
100 000	0,017 мкс	0,10 мс	1,67 мс	10 с		
1 000 000	0,020 мкс	1 мс	19,93 мс	16,7 мин		
10 000 000	0,023 мкс	0,01 с	0,23 с	1,16 дней		
100 000 000	0,027 мкс	0,10 с	2,66 с	115,7 дней		
1 000 000 000	0,030 мкс	1 с	29,90 с	31,7 лет		

Время исполнения f(n) операций алгоритмов на быстродействующем компьютере, исполняющем каждую операцию за одну наносекунду (10^{-9} секунд).

Отношения доминирования

$$n! \gg 2^n \gg n^3 \gg n^2 \gg n\log n \gg n \gg \log n \gg 1$$

- факториальные
- показательные
- кубические
- квадратичные
- суперлинейные
- линейные
- логарифмические
- функции-константы

Сложение и умножение функций

Сумма двух функций определяется доминантной функцией, а именно:

$$O(f(n)) + O(g(n)) = O(\max(f(n),(g(n)))$$

$$\Omega(f(n)) + \Omega(g(n)) = \Omega(\max(f(n),(g(n)))$$

$$\Theta(f(n)) + \Theta(g(n)) = \Theta(\max(f(n),(g(n)))$$

- $n^3+n^2+n+1 = O(n^3)$

$$O(cf(n)) \to O(f(n))$$

$$\Omega(cf(n)) \to \Omega(f(n))$$

$$\Theta(cf(n)) \to \Theta(f(n))$$

$$O(f(n))*O(g(n)) \to O(f(n)*g(n))$$

$$\Omega(f(n))*\Omega(g(n)) \to \Omega(f(n)*g(n))$$

$$\Theta(f(n))*\Theta(g(n)) \to \Theta(f(n)*g(n))$$

ЗАДАЧА. Доказать, что отношение O-большое обладает транзитивностью, т. е. если $f(n) = O(g(n))$ и $g(n) = O(h(n))$, тогда $f(n) = O(h(n))$.

Решение. Работая с асимптотическими обозначениями, мы всегда обращаемся к определению. Нам нужно показать, что $f(n) \le c_3h(n)$ при $n > n_3$ при условии, что $f(n) \le c_1g(n)$ и $g(n) \le c_2h(n)$ при $n > n_1$ и $n > n_2$ соответственно. Из этих неравенств получаем: $f(n) \le c_1g(n) \le c_1c_2h(n)$ при $n > n_3 = \max(n_1,n_2)$. ■

Оценка эффективности

- Сортировка методом выбора: При сортировке этим способом определяется наименьший неотсортированный элемент и помещается в конец отсортированной части массива. Процедура повторяется до тех пор, пока все элементы массива не будут отсортированы.

```
selection_sort(int s[], int n)
{
  int i,j;                /* Счетчики */
  int min;                /* Указатель наименьшего элемента */

  for (i=0; i<n; i++) {
    min=i;
    for (j=i+1; j<n; j++)
      if (s[j] < s[min]) min=j;
    swap(&s[i],&s[min]);
  }
}
```

```
S E L E C T I O N S O R T
C E L E S T I O N S O R T
C E L E S T I O N S O R T
C E E L S T I O N S O R T
C E E I S T L O N S O R T
C E E I L T S O N S O R T
C E E I L N S O T S O R T
C E E I L N O S T S O R T
C E E I L N O O T S S R T
C E E I L N O O R S S T T
C E E I L N O O R S S T T
C E E I L N O O R S S T T
C E E I L N O O R S S T T
C E E I L N O O R S S T T
```

Оценка эффективности

Внешний цикл исполняется n раз. Внутренний цикл исполняется $n - i - 1$ раз, где i — счетчик внешнего цикла. Точное количество исполнений оператора `if` определяется следующей формулой:

$$S(n) = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-1} n - i - 1$$

Это формула сложения целых чисел в убывающем порядке, начиная с $n - 1$, т. е.:

$$S(n) = (n - 1) + (n - 2) + (n - 3) + \ldots + 2 + 1$$

Таким образом мы получаем $S(n) \approx n(n - 1)/2$.

$S(n) \le n(n - 1) = O(n^2)$.

Умножение матриц

ЗАДАЧА. Умножить матрицы.

Вход. Матрица A (размером $x \times y$) и матрица B (размером $y \times z$).

Выход. Матрица C размером $x \times z$, где $C[i][j]$ является скалярным произведением строки i матрицы A и столбца j матрицы B.

```
for (i=1; i<=x; i++)
    for (j=1; j<=y; j++) {
        C[i][j] = 0;
        for (k=1; k<=z; k++)
            C[i][j] += A[i][k] * B[k][j];
    }
```

Оценка эффективности

$$M(x, y, z) = \sum_{i=1}^{x} \sum_{j=1}^{y} \sum_{k=1}^{z} 1$$

$$M(x, y, z) = \sum_{i=1}^{x} \sum_{j=1}^{y} z$$

$$M(x, y, z) = \sum_{i=1}^{x} yz .$$

Таким образом, время исполнения этого алгоритма для умножения матриц равно $O(xyz)$. В общем случае, когда все три измерения матриц одинаковы, это сводится к $O(n^3)$, т. е. алгоритм имеет кубическую формулу времени исполнения.

Логарифмы и их применения

- *Логарифм* – это функция, обратная показательной.

$$\log_a b = c$$

$$a^c = b$$

$$a > 0, a \neq 1, b > 0, c \in R$$

Показательные функции возрастают чрезвычайно быстро. Соответственно, функции, обратные показательным, т.е. логарифмы, возрастают довольно медленно.

Логарифмы и двоичный поиск

- *Двоичный (бинарный) поиск* (также известен как *метод деления пополам* и *дихотомия*) — классический алгоритм поиска элемента в отсортированном массиве (векторе), использующий дробление массива на половины.

Массив разбивается пополам на каждом вызове до тех пор, пока мы не достигнем единицы. Запишем количество элементов в массиве на каждом вызове:

0-я итерация: n
1-я итерация: $n / 2$
2-я итерация: $n / 4$
3-я итерация: $n / 8$
…
i-я итерация: $n / 2^i$
…
последняя итерация: 1

$1 = n / 2^i$
$2^i = n$
$i = \log(n)$

Быстрое возведение в степень

- Допустим, что нужно вычислить точное значение a^n для достаточно большого значения n. Самый простой алгоритм выполняет (n-1) операцию умножения (a×a × …a×a). Но можно указать лучший способ решения этой задачи, приняв во внимание, что $n = [n/2]+[n/2]$.

```
function power(a, n)
    if (n = 0) return(1)
    x = power (a, ⌊n/2⌋)
    if (n is even) then return(x²)
        else return(a × x²)
```

Для вычисления конечного значения будет достаточно O(log n) операций умножения.

Логарифмы и система уголовного судопроизводства

Понесенные убытки	Повышение уровня наказания
(A) 2 000 долларов или меньше	Уровень не повышается
(B) Свыше 2 000 долларов	Повысить на один уровень
(C) Свыше 5 000 долларов	Повысить на два уровня
(D) Свыше 10 000 долларов	Повысить на три уровня
(E) Свыше 20 000 долларов	Повысить на четыре уровня
(F) Свыше 40 000 долларов	Повысить на пять уровней
(G) Свыше 70 000 долларов	Повысить на шесть уровней
(H) Свыше 120 000 долларов	Повысить на семь уровней
(I) Свыше 200 000 долларов	Повысить на восемь уровней
(J) Свыше 350 000 долларов	Повысить на девять уровней
(K) Свыше 500 000 долларов	Повысить на десять уровней
(L) Свыше 800 000 долларов	Повысить на одиннадцать уровней
(M) Свыше 1 500 000 долларов	Повысить на двенадцать уровней
(N) Свыше 2 500 000 долларов	Повысить на тринадцать уровней
(O) Свыше 5 000 000 долларов	Повысить на четырнадцать уровней
(P) Свыше 10 000 000 долларов	Повысить на пятнадцать уровней
(Q) Свыше 20 000 000 долларов	Повысить на шестнадцать уровней
(R) Свыше 40 000 000 долларов	Повысить на семнадцать уровней
(S) Свыше 5 000 000 долларов	Повысить на восемнадцать уровней

Рекомендуемые наказания в федеральных судах США за преступления финансового мошенничества

Мораль логарифмического роста функций ясна: уж если воровать, так миллионы.

Логарифмические функции возникают при решении задач с повторяющимся делением или удваиванием входных данных.