Анализ вычислительной сложности

Данил Браун

2023

Эффективность работы программы

- Эффективность — ???

Эффективность работы программы

- Эффективность — время + память.

Эффективность работы программы

- Эффективность — время + память.
- Как померять? (Для простоты: программа — функция в Pyret.)

Эффективность работы программы

- Эффективность — время + память.
- Как померять? (Для простоты: программа — функция в Pyret.)
- Время работы — функция $T : \mathbb{N} \to \mathbb{N}$.

Эффективность работы программы

- Эффективность — время + память.
- Как померять? (Для простоты: программа — функция в Pyret.)
- Время работы — функция $T : \mathbb{N} \to \mathbb{N}$.
- У функции T должно быть столько же параметров, сколько и у анализируемой функции!
  Для функции

```
append :: List<T>, List<T> -> List<T>
```

  имеем $T_{\text{append}} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$.

Функция len

```
1  len :: List<T> -> Number
2  fun len(lst):
3    cases (List) lst:
4      | empty => 0
5      | link(f, r) => 1 + len(r)
6    end
7  end
```

Функция len

```
1  len :: List<T> -> Number
2  fun len(lst):
3    cases (List) lst:
4      | empty => 0
5      | link(f, r) => 1 + len(r)
6    end
7  end
```

$$[k \mapsto 6k + 4]$$

Асимптотический анализ сложности

	$5k+4$	$5k$	k	k^2	k^3	2^k
$k = 10$	54 ms	50 ms	10 ms	0.1 s	1 s	1 s
$k = 100$	0.5 s	0.5 s	0.1 s	10 s	16 m	40196936841331475186 y
$k = 1000$	5 s	5 s	1 s	16.5 m	11 d	очень долго
$k = 10\,000$	50 s	50 s	10 s	27.7 h	31 y	очень долго
$k = 100\,000$	0.13 h	0.13 h	0.03 h	115 d	31709 y	очень долго
$k = 1\,000\,000$	1.3 h	1.3 h	0.28 h	31 y	очень долго	очень долго

Оценка сверху

```
1  member([list: 2, 4, -12, ..., 95, 10], 2)
2  member([list: 2, 4, -12, ..., 95, 10], 10)
```

Сравнение функций

Упражнение 1

Если `f` — функция в Pyret со временем работы $T_{\mathtt{f}}$, а `g` — функция со временем $T_{\mathtt{g}}$, то как сказать в терминах $T_{\mathtt{f}}$ и $T_{\mathtt{g}}$, что `f` работает **не медленнее** `g`?

Сравнение функций

Упражнение 1

Если $\mathtt{f}$ — функция в Pyret со временем работы T_{f}, а $\mathtt{g}$ — функция со временем T_{g}, то как сказать в терминах T_{f} и T_{g}, что $\mathtt{f}$ работает **не медленнее** $\mathtt{g}$?

$$\forall k \in \mathbb{N}\ T_{\mathrm{f}}(k) \leqslant T_{\mathrm{g}}(k)$$

О большое

Определение 1

$$O(T_{\mathrm{g}}) := \{T_{\mathrm{f}} \mid \forall k \in \mathbb{N}\ T_{\mathrm{f}}(k) \leqslant T_{\mathrm{g}}(k)\}$$

О большое

Определение 1

$$O(T_{\mathrm{g}}) := \{T_{\mathrm{f}} \mid \forall k \in \mathbb{N}\ T_{\mathrm{f}}(k) \leqslant T_{\mathrm{g}}(k)\}$$

Упражнение 2

Пусть $T_{\mathrm{g}} : k \mapsto 2k$. Тогда

$$O([k \mapsto 2k]) = \{T_{\mathrm{f}} \mid \forall k \in \mathbb{N}\ T_{\mathrm{f}}(k) \leqslant 2k\}$$

Верно ли, что

1. $[k \mapsto k] \in O([k \mapsto 2k])$;
2. $[k \mapsto \frac{k}{2}] \in O([k \mapsto 2k])$;
3. $[k \mapsto 6k] \in O([k \mapsto 2k])$.

O большое 2

Определение 2

$$O_2(T_g) := \{T_f \mid \exists c \in \mathbb{N}_{>0}\ \forall k \in \mathbb{N}\ T_f(k) \leqslant T_g(k) \cdot c\}$$

О большое 2

Определение 2

$$O_2(T_g) := \{T_f \mid \exists c \in \mathbb{N}_{>0}\ \forall k \in \mathbb{N}\ T_f(k) \leqslant T_g(k) \cdot c\}$$

Упражнение 3

Верно ли теперь утверждение $[k \mapsto 6k] \in O_2([k \mapsto 2k])$?
Почему?

О большое 2

Определение 2

$$O_2(T_g) := \{T_f \mid \exists c \in \mathbb{N}_{>0}\ \forall k \in \mathbb{N}\ T_f(k) \leqslant T_g(k) \cdot c\}$$

Упражнение 3

Верно ли теперь утверждение $[k \mapsto 6k] \in O_2([k \mapsto 2k])$? Почему?

Пусть $c = 4$. Тогда $\forall k \in \mathbb{N}\ 6k \leqslant 2k \cdot 4 = 8k$.

О большое 2

Определение 2

$$O_2(T_g) := \{T_f \mid \exists c \in \mathbb{N}_{>0}\ \forall k \in \mathbb{N}\ T_f(k) \leqslant T_g(k) \cdot c\}$$

Упражнение 3

Верно ли теперь утверждение $[k \mapsto 6k] \in O_2([k \mapsto 2k])$? Почему?

Пусть $c = 4$. Тогда $\forall k \in \mathbb{N}\ 6k \leqslant 2k \cdot 4 = 8k$.

Упражнение 4

$[k \mapsto 6k + 4] \in O_2([k \mapsto 2k])$?

O большое 3

Определение 3

$$O_3(T_g) := \{T_f \mid \exists c \in \mathbb{N}_{>0}\ \exists N \in \mathbb{N}\ \forall k \geqslant N\ T_f(k) \leqslant T_g(k) \cdot c\}$$

О большое 3

Определение 3

$$O_3(T_g) := \{T_f \mid \exists c \in \mathbb{N}_{>0}\ \exists N \in \mathbb{N}\ \forall k \geqslant N\ T_f(k) \leqslant T_g(k) \cdot c\}$$

Упражнение 5

Верно ли теперь $[k \mapsto 6k + 4] \in O_3([k \mapsto 2k])$? Почему?

О большое 3

Определение 3

$$O_3(T_g) := \{T_f \mid \exists c \in \mathbb{N}_{>0}\ \exists N \in \mathbb{N}\ \forall k \geqslant N\ T_f(k) \leqslant T_g(k) \cdot c\}$$

Упражнение 5

Верно ли теперь $[k \mapsto 6k + 4] \in O_3([k \mapsto 2k])$? Почему?

Пусть $c = 6$, $N = 1$. Тогда для всех $k \geqslant 1$ верно $6k + 4 \leqslant 2k \cdot 6$.

Замечания

Замечание 1

Пользуясь O-нотацией, мы будем всегда записывать функции в «наиболее простой» форме. Например, вместо того, чтобы писать $O([k \mapsto 3k^3 + 2k^2 + k + 7])$, мы пишем $O([k \mapsto k^3])$, т. к. $[k \mapsto 3k^3 + 2k^2 + k + 7] \in O([k \mapsto k^3])$ (младшие члены и коэффициент при старшем члене отбрасываются).

Замечания

Замечание 1

Пользуясь O-нотацией, мы будем всегда записывать функции в «наиболее простой» форме. Например, вместо того, чтобы писать $O([k \mapsto 3k^3 + 2k^2 + k + 7])$, мы пишем $O([k \mapsto k^3])$, т. к. $[k \mapsto 3k^3 + 2k^2 + k + 7] \in O([k \mapsto k^3])$ (младшие члены и коэффициент при старшем члене отбрасываются).

Замечание 2

Ясно, что если $T_{\mathrm{f}} \in O([k \mapsto k])$, то и $T_{\mathrm{f}} \in O([k \mapsto k^2])$, и $T_{\mathrm{f}} \in O([l \mapsto k^{100}])$, и $T_{\mathrm{f}} \in O([k \mapsto 2^k])$, и т. д. Но когда мы говорим об оценке T_{f} сверху, то имеем в виду именно $T_{\mathrm{f}} \in O([k \mapsto k])$, т. е. наименьшую из них.

Замечания

Замечание 3

Многие авторы пишут $O(g(n)) = \{f(n) \mid \ldots\}$, имея в виду $O(g) = \{f \mid\}$. Мы используем второй вариант, т. к. мы знаем, что $f(n)$ и $g(n)$ это **значения функций** f и g в n, а не **сами функции** f и g.

Замечания

Замечание 3

Многие авторы пишут $O(g(n)) = \{f(n) \mid \ldots\}$, имея в виду $O(g) = \{f \mid\}$. Мы используем второй вариант, т. к. мы знаем, что $f(n)$ и $g(n)$ это **значения функций** f и g в n, а не **сами функции** f и g.

Замечание 4

Вместо, например, $[k \mapsto 2k] \in O([k \mapsto k^2])$ можно часто видеть запись вида $2k = O(k^2)$. Это, что называется, «abuse of notation» (злоупотребление обозначениями). Если считать такую запись осмысленной, то тогда можно получить

$$2k = O(k^2) \wedge 2k = O(k^3) \implies O(k^2) = O(k^3)$$

что очевидно неверно.

Общие правила

- Если функция $\mathtt{f}$ со временем $T_{\mathtt{f}} \in O(T)$ выполняется некоторое (постоянное) количество раз, то общее время выполнения всех этих вызовов остаётся в $O(T)$.

Общие правила

- Если функция $\mathtt{f}$ со временем $T_{\mathrm{f}} \in O(T)$ выполняется некоторое (постоянное) количество раз, то общее время выполнения всех этих вызовов остаётся в $O(T)$.
- Если f и g (со временем $T_{\mathrm{f}} \in O(T)$ и $T_{\mathrm{g}} \in O(T')$ соответственно) — функции выполняющиеся друг за другом, то общее время выполнения лежит в $O(T + T')$. Например, если $T_{\mathrm{f}} \in O([k \mapsto k^2])$, а $T_{\mathrm{g}} \in O([k \mapsto k])$, то T_{h}, которая определяется как время выполнения $\mathtt{f}$ и $\mathtt{g}$ вместе взятых, содержится в $O([k \mapsto k^2 + k])$, что эквивалентно $O([k \mapsto k^2])$.

Общие правила

- Если функция `f` со временем $T_{\mathrm{f}} \in O(T)$ при каждом своём вызове вызывает другую функцию `g` со временем $T_{\mathrm{g}} \in O(T')$, то общее время работы есть функция из $O(T \cdot T')$.
  Допустим, у нас есть два списка: `a` и `b`. Мы пишем функцию `is-sublist`, которая будет смотреть, содержится ли каждый элемент `a` также и в `b`. Для каждого из k элементов списка `a` мы вызываем функцию `member` на списке `b` (есть ли данный элемент `a` в списке `b`), А т. к. `member` выполняется за $O([k \mapsto k])$, и мы вызываем её k раз, то общее время выполнения лежит в $O([k \mapsto k \cdot k]) = O([k \mapsto k^2])$.

Общие правила

- Вызов k раз одной функции — умножение на константу
- Последовательное выполнение функций — сумма
- Вызов одной функции из другой — произведение

Время работы

- Константное — $[k \mapsto C]$ для некоторой константы $C > 0$.

Время работы

- Константное — $[k \mapsto C]$ для некоторой константы $C > 0$. Взять первый элемент у списка.

Время работы

- Константное — $[k \mapsto C]$ для некоторой константы $C > 0$. Взять первый элемент у списка.
- Логарифмическое — $[k \mapsto \log k]$.

Время работы

- Константное — $[k \mapsto C]$ для некоторой константы $C > 0$.
  Взять первый элемент у списка.
- Логарифмическое — $[k \mapsto \log k]$.
  Поиск элемента в бинарном дереве поиска.

Время работы

- Константное — $[k \mapsto C]$ для некоторой константы $C > 0$.
  Взять первый элемент у списка.
- Логарифмическое — $[k \mapsto \log k]$.
  Поиск элемента в бинарном дереве поиска.
- Линейное — $[k \mapsto k]$.

Время работы

- Константное — $[k \mapsto C]$ для некоторой константы $C > 0$.
  Взять первый элемент у списка.
- Логарифмическое — $[k \mapsto \log k]$.
  Поиск элемента в бинарном дереве поиска.
- Линейное — $[k \mapsto k]$.
  Вычисление длины списка.

Время работы

- Константное — $[k \mapsto C]$ для некоторой константы $C > 0$.
  Взять первый элемент у списка.
- Логарифмическое — $[k \mapsto \log k]$.
  Поиск элемента в бинарном дереве поиска.
- Линейное — $[k \mapsto k]$.
  Вычисление длины списка.
- Квазилинейное — $[k \mapsto k \cdot \log k]$.

Время работы

- Константное — $[k \mapsto C]$ для некоторой константы $C > 0$.
  Взять первый элемент у списка.
- Логарифмическое — $[k \mapsto \log k]$.
  Поиск элемента в бинарном дереве поиска.
- Линейное — $[k \mapsto k]$.
  Вычисление длины списка.
- Квазилинейное — $[k \mapsto k \cdot \log k]$.
  Сортировка слиянием (merge sort), лучшее время для сортировок.

Время работы

- Константное — $[k \mapsto C]$ для некоторой константы $C > 0$.
  Взять первый элемент у списка.
- Логарифмическое — $[k \mapsto \log k]$.
  Поиск элемента в бинарном дереве поиска.
- Линейное — $[k \mapsto k]$.
  Вычисление длины списка.
- Квазилинейное — $[k \mapsto k \cdot \log k]$.
  Сортировка слиянием (merge sort), лучшее время для сортировок.
- Квадратичное — $[k \mapsto k^2]$.

Время работы

- Константное — $[k \mapsto C]$ для некоторой константы $C > 0$.
  Взять первый элемент у списка.
- Логарифмическое — $[k \mapsto \log k]$.
  Поиск элемента в бинарном дереве поиска.
- Линейное — $[k \mapsto k]$.
  Вычисление длины списка.
- Квазилинейное — $[k \mapsto k \cdot \log k]$.
  Сортировка слиянием (merge sort), лучшее время для сортировок.
- Квадратичное — $[k \mapsto k^2]$.
  Сортировка вставкой (insertion sort).

Время работы

- Константное — $[k \mapsto C]$ для некоторой константы $C > 0$.
  Взять первый элемент у списка.
- Логарифмическое — $[k \mapsto \log k]$.
  Поиск элемента в бинарном дереве поиска.
- Линейное — $[k \mapsto k]$.
  Вычисление длины списка.
- Квазилинейное — $[k \mapsto k \cdot \log k]$.
  Сортировка слиянием (merge sort), лучшее время для сортировок.
- Квадратичное — $[k \mapsto k^2]$.
  Сортировка вставкой (insertion sort).
- Экспоненциальное — $[k \mapsto C^k]$, $C > 0$.

Время работы

- Константное — $[k \mapsto C]$ для некоторой константы $C > 0$.
  Взять первый элемент у списка.
- Логарифмическое — $[k \mapsto \log k]$.
  Поиск элемента в бинарном дереве поиска.
- Линейное — $[k \mapsto k]$.
  Вычисление длины списка.
- Квазилинейное — $[k \mapsto k \cdot \log k]$.
  Сортировка слиянием (merge sort), лучшее время для сортировок.
- Квадратичное — $[k \mapsto k^2]$.
  Сортировка вставкой (insertion sort).
- Экспоненциальное — $[k \mapsto C^k]$, $C > 0$.
  Поиск всех подмножеств данного множества.

Упражнения

Дайте верхнюю оценку в терминах O-большого для каждой функции.

Упражнение 6

Функция $\mathtt{f}$ вызывается с входными данными размера k. Для каждого из k элементов она вызывает другую функцию, $\mathtt{g}$, со временем работы из $O([p \mapsto \log p])$, передавая ей данные размера k.

Упражнения

Упражнение 7

Следующая функция удаляет все элементы одного списка, которые содержатся в другом списке. Пусть k — размер `to-delete`, а m — размер `lst`.

```
1  fun delete(to-delete, lst):
2    cases (List) to-delete:
3      | empty => lst
4      | link(f, r) => delete(r, lst.remove(f))
5    end
6  end
```

Упражнения

Упражнение 8

Функция принимает два списка; размер первого — q, размер второго — p. Возвращает сумму длин этих двух списков.

Упражнения

Упражнение 8

Функция принимает два списка; размер первого — q, размер второго — p. Возвращает сумму длин этих двух списков.

Упражнение 9

Функция `f` вызывает функцию `g` времени $O([n \mapsto n])$ десять раз, а затем вызывает функцию `h` времени $O([m \mapsto m])$, передавая данные размера s каждой из них.

Рекуррентные соотношения

Определение 4

Рекуррентным соотношением называется рекурсивная функция на натуральных числах, т. е. $f : \mathbb{N} \to S$. Например:

$$T(n) = T(n-1) + 4$$

Рекуррентные соотношения

Определение 4

Рекуррентным соотношением называется рекурсивная функция на натуральных числах, т. е. $f : \mathbb{N} \to S$. Например:

$$T(n) = T(n-1) + 4$$

Замечание 5

В такой форме можно выражать время работы функции (программы).

Рекуррентные соотношения

Определение 5

Решить рекуррентное соотношение значит найти его **замкнутую форму**.

Рекуррентные соотношения

Пример 1

Проанализировав функцию `length`, мы получили:

$$T_{\text{length}}(n) = \begin{cases} 4, & \text{если } n = 0 \\ T_{\text{length}}(n-1) + 6, & \text{если } n > 0 \end{cases}$$

Нахождение замкнутой формы:

$$\begin{aligned} T_{\text{length}}(n) &= T_{\text{length}}(n-1) + 6 \\ &= (T_{\text{length}}(n-2) + 6) + 6 \\ &= (\ldots(T_{\text{length}}(n-n) + 6) + \ldots + 6 \\ &= T_{\text{length}}(0) + 6n = 4 + 6n \end{aligned}$$

Замкнутая форма: $T(n) = 6n + 4$.

Рекуррентные соотношения

Упражнение 10

1. $T(k) = T(k-1) + c$
2. $T(k) = T(k-1) + k$
3. $T(k) = T(k/2) + c$
4. $T(k) = T(k/2) + k$
5. $T(k) = 2T(k/2) + k$
6. $T(k) = 2T(k-1) + c$