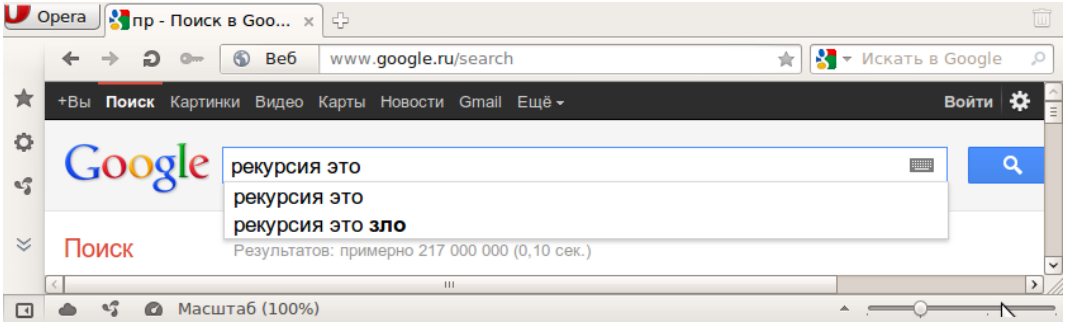
Лекция 3

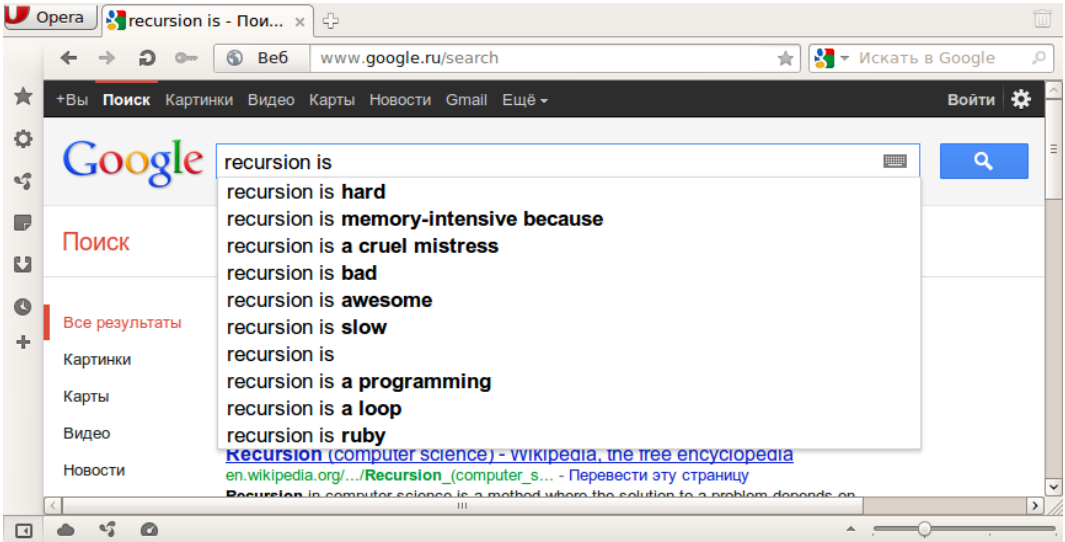
Рекурсия

Функциональное и логическое программирование

КамчатГТУ, 2013 г.

1. Почему рекурсия — это зло?
2. Почему рекурсия?
3. Где ещё бывает рекурсия?
4. Какая ещё бывает рекурсия?
5. Когда же она закончится?
6. См. пункт 1.

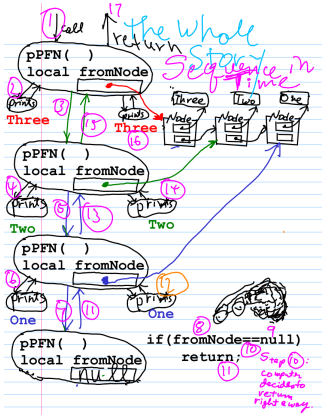

Opera
пр - Поиск в Goo...
Веб
www.google.ru/search
Искать в Google
+Вы
Поиск
Картинки
Видео
Карты
Новости
Gmail
Ещё
Войти
Google
рекурсия это
рекурсия это
рекурсия это зло
Поиск
Результатов: примерно 217 000 000 (0,10 сек.)
Масштаб (100%)


Opera
recursion is - Пои...
Веб
www.google.ru/search
Искать в Google
+Вы Поиск Картинки Видео Карты Новости Gmail Ещё
Войти
Google
recursion is
recursion is hard
recursion is memory-intensive because
recursion is a cruel mistress
recursion is bad
recursion is awesome
recursion is slow
recursion is
recursion is a programming
recursion is a loop
recursion is ruby
Поиск
Все результаты
Картинки
Карты
Видео
Новости
Recursion (computer science) - Wikipedia, the free encyclopedia
en.wikipedia.org/.../Recursion_(computer_s... - Перевести эту страницу

1. Почему рекурсия — это зло?
2. Почему рекурсия?
3. Где ещё бывает рекурсия?
4. Какая ещё бывает рекурсия?
5. Когда же она закончится?
6. См. пункт 1.

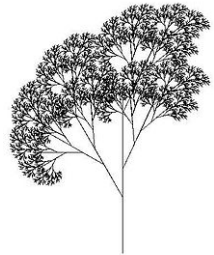
За что рекурсию не любят?

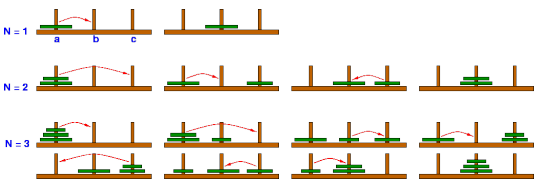
- Рекурсия – это непонятно: рекурсивные решения неочевидны, программы трудно отлаживать.



За что рекурсию не любят?

- Рекурсия – это непонятно: рекурсивные решения неочевидны, программы трудно отлаживать.
- Рекурсия хороша только для олимпиадных задач, в реальных задачах она не используется.





За что рекурсию не любят?

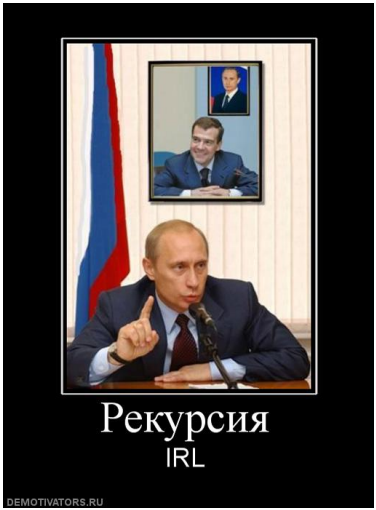
- Рекурсия – это непонятно: рекурсивные решения неочевидны, программы трудно отлаживать.
- Рекурсия хороша только для олимпиадных задач, в реальных задачах она не используется.
- Рекурсия опасна: она может никогда не закончиться.

За что рекурсию не любят?

- Рекурсия – это непонятно: рекурсивные решения неочевидны, программы трудно отлаживать.
- Рекурсия хороша только для олимпиадных задач, в реальных задачах она не используется.
- Рекурсия опасна: она может никогда не закончиться.
- Рекурсия опасна: неизвестно, сколько она потребует памяти и времени.

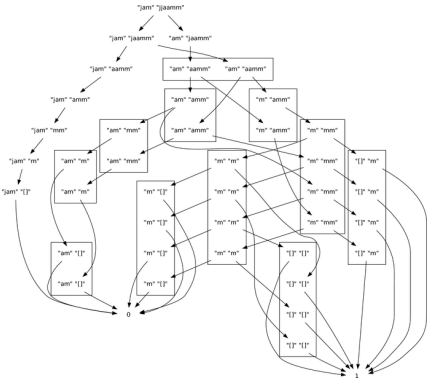
За что рекурсию не любят?

- Рекурсия – это непонятно: рекурсивные решения неочевидны, программы трудно отлаживать.
- Рекурсия хороша только для олимпиадных задач, в реальных задачах она не используется.
- Рекурсия опасна: она может никогда не закончиться.
- Рекурсия опасна: неизвестно, сколько она потребует памяти и времени.
- Рекурсивные программы ресурсоёмки и неэффективны.



За что рекурсию не любят?

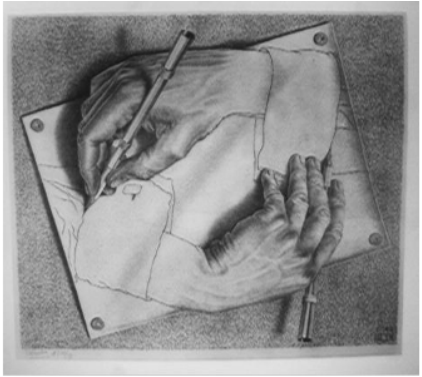
- Всё, что можно сделать с помощью рекурсии, можно сделать на циклах. Так зачем всё усложнять?



За что рекурсию не любят?

- Всё, что можно сделать с помощью рекурсии, можно сделать на циклах. Так зачем всё усложнять?
- Во многих классических учебниках по программированию настоятельно рекомендуют избегать рекурсии

  (*«...если бы программист, работающий у меня, применял рекурсию для вычисления факториала, я бы нанял кого-то другого...»*
  С. Макконнелл «Совершенный код»)



За что рекурсию не любят?

- Всё, что можно сделать с помощью рекурсии, можно сделать на циклах. Так зачем всё усложнять?
- Во многих классических учебниках по программированию настоятельно рекомендуют избегать рекурсии

  (*«...если бы программист, работающий у меня, применял рекурсию для вычисления факториала, я бы нанял кого-то другого...»*
  С. Макконнелл «Совершенный код»)
- Рекурсия противоестественна для человеческого мышления.

1. Почему рекурсия — это зло?
2. Почему рекурсия?
3. Где ещё бывает рекурсия?
4. Какая ещё бывает рекурсия?
5. Когда же она закончится?
6. См. пункт 1.

Циклический процесс

Рассмотрим задачу:

Вычислить сумму квадратов натуральных чисел от 1 до n:

$$S_n = 1 + 2^2 + 3^2 + ... + n^2$$

Циклический процесс

Рассмотрим задачу:

Вычислить сумму квадратов натуральных чисел от 1 до n:

$$S_n = 1 + 2^2 + 3^2 + ... + n^2$$

Итератор:

$sum := 0$
for i=1 **to** n **do**
  $sum := sum + i^2$
return sum

Циклический процесс

Рассмотрим задачу:

Вычислить сумму квадратов натуральных чисел от 1 до n:

$$S_n = 1 + 2^2 + 3^2 + ... + n^2$$

Итератор:

$sum := 0$
for i=1 **to** n **do**
  $sum := sum + i^2$
return sum

Циклический процесс

Рассмотрим задачу:

Вычислить сумму квадратов натуральных чисел от 1 до n:

$$S_n = 1 + 2^2 + 3^2 + ... + n^2$$

Итератор:

$sum := 0$
for i=1 **to** n **do**
  $sum := sum + i^2$
return sum

Условный цикл:

$sum := 0$
$i := n$
while $i > 0$ **do**
  $sum := sum + i^2$
  $i := i - 1$
return sum

Циклический процесс

Рассмотрим задачу:

Вычислить сумму квадратов натуральных чисел от 1 до n:

$$S_n = 1 + 2^2 + 3^2 + ... + n^2$$

Итератор:

$sum := 0$
for i=1 **to** n **do**
  $sum := sum + i^2$
return sum

Условный цикл:

$sum := 0$
$i := n$
while $i > 0$ **do**
  $sum := sum + i^2$
  $i := i - 1$
return sum

Явное зацикливание:

$sum := 0$
$i := n$
1: **if** $i = 0$ **goto** 2
$sum := sum + i^2$
$i := i - 1$
goto 1
2: **return** sum

Базовые принципы ФП

В рамках функциональной парадигмы:

Цикл:

$sum := 0$
$i := n$
while $i > 0$ **do**
  $sum := sum + i^2$
  $i := i - 1$
return sum

Базовые принципы ФП

В рамках функциональной парадигмы:

- нет переменных и присваивания;

Цикл:

$sum := 0$
$i := n$
while $i > 0$ **do**
  $sum := sum + i^2$
  $i := i - 1$
return sum

Базовые принципы ФП

В рамках функциональной парадигмы:

- нет переменных и присваивания;
- нет заданного порядка вычисления.

Цикл:

$sum := 0$
$i := n$
while $i > 0$ do
  $sum := sum + i^2$
  $i := i - 1$
return sum

Базовые принципы ФП

В рамках функциональной парадигмы:

- нет переменных и присваивания;
- нет заданного порядка вычисления.

Цикл:

$sum := 0$
$i := n$
while $i > 0$ do
  $sum := sum + i^2$
  $i := i - 1$
return sum

Программа

Функциональная программа представляет собой композицию чистых функций.

Базовые принципы ФП

В рамках функциональной парадигмы:

- нет переменных и присваивания;
- нет заданного порядка вычисления.

Цикл:

$sum := 0$
$i := n$
while $i > 0$ do
  $sum := sum + i^2$
  $i := i - 1$
return sum

функциональный цикл:

sum (n) = ...

Программа

Функциональная программа представляет собой композицию чистых функций.

Процесс

Вычисление функционального выражения состоит в подстановке фактических аргументов вместо формальных.

Базовые принципы ФП

В рамках функциональной парадигмы:

- нет переменных и присваивания;
- нет заданного порядка вычисления.

Цикл:

$sum := 0$
$i := n$
while $i > 0$ do
  $sum := sum + i^2$
  $i := i - 1$
return sum

функциональный цикл:

sum (n) = ... sum (...) ...

Программа

Функциональная программа представляет собой композицию чистых функций.

Процесс

Вычисление функционального выражения состоит в подстановке фактических аргументов вместо формальных.

Таким образом, единственный способ повторить вычисления — вызвать функцию повторно.

Функциональная реализация

Задача вычисления суммы из n слагаемых сводится к задаче вычисления суммы $n-1$ слагаемого:

$$S_n = 1 + 2^2 + 3^2 + ... + n^2 = \left(1 + 2^2 + 3^2 + ... + (n-1)^2\right) + n^2 = S_{n-1} + n^2,$$

при этом $S_0 = 0$.

Функциональная реализация

Задача вычисления суммы из n слагаемых сводится к задаче вычисления суммы $n-1$ слагаемого:

$$S_n = 1 + 2^2 + 3^2 + ... + n^2 = \left(1 + 2^2 + 3^2 + ... + (n-1)^2\right) + n^2 = S_{n-1} + n^2,$$

при этом $S_0 = 0$.

Функциональная программа:

$S(0) = 0$
$S(n) = S(n-1) + n^2$

Функциональная реализация

Задача вычисления суммы из n слагаемых сводится к задаче вычисления суммы $n-1$ слагаемого:

$$S_n = 1 + 2^2 + 3^2 + ... + n^2 = \left(1 + 2^2 + 3^2 + ... + (n-1)^2\right) + n^2 = S_{n-1} + n^2,$$

при этом $S_0 = 0$.

Функциональная программа:

$S(0) = 0$
$S(n) = S(n-1) + n^2$

Определение

Рекурсивным является определение объекта, содержащее

1. базовые (нерекурсивные) определения для частных случаев;
2. набор правил приводящих все прочие случаи к базовым и использующих определяемый объект.

1. Почему рекурсия — это зло?
2. Почему рекурсия?
3. Где ещё бывает рекурсия?
4. Какая ещё бывает рекурсия?
5. Когда же она закончится?
6. См. пункт 1.

Индуктивные типы данных

Определение многих объектов в математике рекурсивно.

1. Формулируется базис, как нерекурсивное определение исключительных случаев при построении типа или множества.
2. Строится шаг, как рекурсивное правило построения того же объекта.

Индуктивные типы данных

Определение многих объектов в математике рекурсивно.

1. Формулируется базис, как нерекурсивное определение исключительных случаев при построении типа или множества.
2. Строится шаг, как рекурсивное правило построения того же объекта.

Множество натуральных чисел

К натуральным числам относятся 0 и любое число, следующее за натуральным.

$$\mathbf{N} = \{0, n+1 \mid n \in \mathbf{N}\}$$

Индуктивные типы данных

Определение многих объектов в математике рекурсивно.

1. Формулируется базис, как нерекурсивное определение исключительных случаев при построении типа или множества.
2. Строится шаг, как рекурсивное правило построения того же объекта.

Множество натуральных чисел

К натуральным числам относятся 0 и любое число, следующее за натуральным.

$$\mathbf{N} = \{0, n+1 \mid n \in \mathbf{N}\}$$

Базис: $0 \in \mathbf{N}$
Шаг: $n+1 \in \mathbf{N}$, если $n \in \mathbf{N}$.

Индуктивные типы данных

Определение многих объектов в математике рекурсивно.

1. Формулируется базис, как нерекурсивное определение исключительных случаев при построении типа или множества.
2. Строится шаг, как рекурсивное правило построения того же объекта.

Множество натуральных чисел

К натуральным числам относятся 0 и любое число, следующее за натуральным.

$$\mathbf{N} = \{0, n+1 \mid n \in \mathbf{N}\}$$

Базис: $0 \in \mathbf{N}$
Шаг: $n+1 \in \mathbf{N}$, если $n \in \mathbf{N}$.

Сложение:

$$n + 0 = n$$
$$n + (m+1) = (n+m) + 1$$

Индуктивные типы данных

Определение многих объектов в математике рекурсивно.

1. Формулируется базис, как нерекурсивное определение исключительных случаев при построении типа или множества.
2. Строится шаг, как рекурсивное правило построения того же объекта.

Множество натуральных чисел

К натуральным числам относятся 0 и любое число, следующее за натуральным.

$$\mathbf{N} = \{0, n+1 \mid n \in \mathbf{N}\}$$

Базис: $0 \in \mathbf{N}$
Шаг: $n+1 \in \mathbf{N}$, если $n \in \mathbf{N}$.

Сложение:

$$n + 0 = n$$
$$n + (m+1) = (n+m) + 1$$

Умножение:

$$n \times 0 = 0$$
$$n \times (m+1) = (n \times m) + n$$

Индуктивные типы данных

Определение многих объектов в математике рекурсивно.

1. Формулируется базис, как нерекурсивное определение исключительных случаев при построении типа или множества.
2. Строится шаг, как рекурсивное правило построения того же объекта.

Множество натуральных чисел

К натуральным числам относятся 0 и любое число, следующее за натуральным.

$$\mathbf{N} = \{0, n+1 \mid n \in \mathbf{N}\}$$

Базис: $0 \in \mathbf{N}$
Шаг: $n+1 \in \mathbf{N}$, если $n \in \mathbf{N}$.

Сложение:

$$n + 0 = n$$
$$n + (m+1) = (n+m) + 1$$

Умножение:

$$n \times 0 = 0$$
$$n \times (m+1) = (n \times m) + n$$

Возведение в степень:

$$n^0 = 1$$
$$n^{m+1} = (n^m) \times n$$

Индуктивные типы данных

К индуктивным множествам и типам данных относятся

- Натуральные, чётные, простые числа и т.д.;

Индуктивные типы данных

К индуктивным множествам и типам данных относятся

- Натуральные, чётные, простые числа и т.д.;
- Потоки данных (*динамические списки*, *файлы*, *строки*, *сериализованные данные*, ...);

Индуктивные типы данных

К индуктивным множествам и типам данных относятся

- Натуральные, чётные, простые числа и т.д.;
- Потоки данных (*динамические списки*, *файлы*, *строки*, *сериализованные данные*, ...);
- Древообразные структуры (*словари*, *хэш-таблицы*, *грамматики*, ...)

Индуктивные типы данных

К индуктивным множествам и типам данных относятся

- Натуральные, чётные, простые числа и т.д.;
- Потоки данных (*динамические списки*, *файлы*, *строки*, *сериализованные данные*, ...);
- Древообразные структуры (*словари*, *хэш-таблицы*, *грамматики*, ...)
- Ацикличные графы.

Индуктивные типы данных

- Рекурсивные программы наиболее просто и естественно описывают порождение и обработку индуктивных типов данных.

Индуктивные типы данных

- Рекурсивные программы наиболее просто и естественно описывают порождение и обработку индуктивных типов данных.
- При рекурсивной обработке данных нет необходимости знать их «размеры» (количество термов) на этапе компиляции.

Индуктивные типы данных

- Рекурсивные программы наиболее просто и естественно описывают порождение и обработку индуктивных типов данных.
- При рекурсивной обработке данных нет необходимости знать их «размеры» (количество термов) на этапе компиляции.
- Конечные рекурсивные программы позволяют порождать или обрабатывать бесконечное разнообразие объектов.

Индуктивные типы данных

- Рекурсивные программы наиболее просто и естественно описывают порождение и обработку индуктивных типов данных.
- При рекурсивной обработке данных нет необходимости знать их «размеры» (количество термов) на этапе компиляции.
- Конечные рекурсивные программы позволяют порождать или обрабатывать бесконечное разнообразие объектов.
- Для индуктивных множеств и рекурсивных функций возможно доказательство свойств методом математической индукции.

Индуктивные типы данных

- Рекурсивные программы наиболее просто и естественно описывают порождение и обработку индуктивных типов данных.
- При рекурсивной обработке данных нет необходимости знать их «размеры» (количество термов) на этапе компиляции.
- Конечные рекурсивные программы позволяют порождать или обрабатывать бесконечное разнообразие объектов.
- Для индуктивных множеств и рекурсивных функций возможно доказательство свойств методом математической индукции.

Вывод:

Рекурсия — идеальный инструмент для работы с индуктивными данными:

Индуктивные типы данных

- Рекурсивные программы наиболее просто и естественно описывают порождение и обработку индуктивных типов данных.
- При рекурсивной обработке данных нет необходимости знать их «размеры» (количество термов) на этапе компиляции.
- Конечные рекурсивные программы позволяют порождать или обрабатывать бесконечное разнообразие объектов.
- Для индуктивных множеств и рекурсивных функций возможно доказательство свойств методом математической индукции.

Вывод:

Рекурсия — идеальный инструмент для работы с индуктивными данными:

- перечислимыми множествами,

Индуктивные типы данных

- Рекурсивные программы наиболее просто и естественно описывают порождение и обработку индуктивных типов данных.
- При рекурсивной обработке данных нет необходимости знать их «размеры» (количество термов) на этапе компиляции.
- Конечные рекурсивные программы позволяют порождать или обрабатывать бесконечное разнообразие объектов.
- Для индуктивных множеств и рекурсивных функций возможно доказательство свойств методом математической индукции.

Вывод:

Рекурсия — идеальный инструмент для работы с индуктивными данными:

- перечислимыми множествами,
- списками и потоками,

Индуктивные типы данных

- Рекурсивные программы наиболее просто и естественно описывают порождение и обработку индуктивных типов данных.
- При рекурсивной обработке данных нет необходимости знать их «размеры» (количество термов) на этапе компиляции.
- Конечные рекурсивные программы позволяют порождать или обрабатывать бесконечное разнообразие объектов.
- Для индуктивных множеств и рекурсивных функций возможно доказательство свойств методом математической индукции.

Вывод:

Рекурсия — идеальный инструмент для работы с индуктивными данными:

- перечислимыми множествами,
- списками и потоками,
- иерархическими структурами,

Индуктивные типы данных

- Рекурсивные программы наиболее просто и естественно описывают порождение и обработку индуктивных типов данных.
- При рекурсивной обработке данных нет необходимости знать их «размеры» (количество термов) на этапе компиляции.
- Конечные рекурсивные программы позволяют порождать или обрабатывать бесконечное разнообразие объектов.
- Для индуктивных множеств и рекурсивных функций возможно доказательство свойств методом математической индукции.

Вывод:

Рекурсия — идеальный инструмент для работы с индуктивными данными:

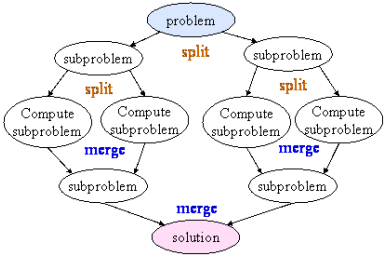
- перечислимыми множествами,
- списками и потоками,
- иерархическими структурами,
- грамматическими конструкциями и т.п.

Разделяй и властвуй

Парадигма разработки **быстрых алгоритмов**, состоящая в разбиении задачи на подзадачи того же типа, но меньшего размера и комбинировании их результатов.

Разделяй и властвуй

Парадигма разработки **быстрых алгоритмов**, состоящая в разбиении задачи на подзадачи того же типа, но меньшего размера и комбинировании их результатов.

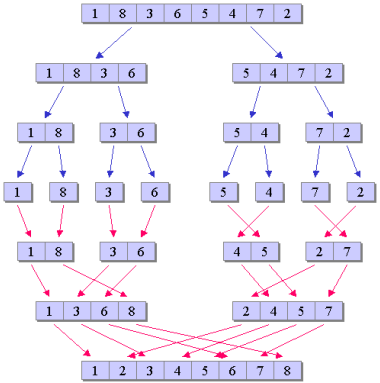


Разделяй и властвуй

Парадигма разработки **быстрых алгоритмов**, состоящая в разбиении задачи на подзадачи того же типа, но меньшего размера и комбинировании их результатов.

Примеры:

- **сортировка слиянием**

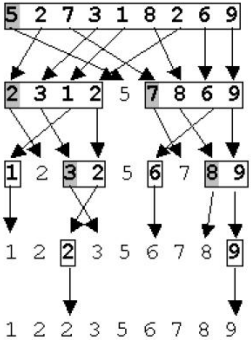


Разделяй и властвуй

Парадигма разработки **быстрых алгоритмов**, состоящая в разбиении задачи на подзадачи того же типа, но меньшего размера и комбинировании их результатов.

Примеры:

- сортировка слиянием
- **быстрая сортирова Хоара**

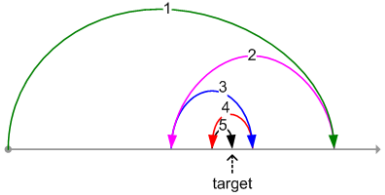


Разделяй и властвуй

Парадигма разработки **быстрых алгоритмов**, состоящая в разбиении задачи на подзадачи того же типа, но меньшего размера и комбинировании их результатов.

Примеры:

- сортировка слиянием
- быстрая сортирова Хоара
- **метод бисекции**

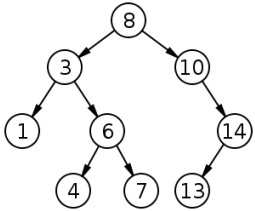


Разделяй и властвуй

Парадигма разработки **быстрых алгоритмов**, состоящая в разбиении задачи на подзадачи того же типа, но меньшего размера и комбинировании их результатов.

Примеры:

- сортировка слиянием
- быстрая сортирова Хоара
- метод бисекции
- **двоичный поиск**



Разделяй и властвуй

Парадигма разработки **быстрых алгоритмов**, состоящая в разбиении задачи на подзадачи того же типа, но меньшего размера и комбинировании их результатов.

Примеры:

- сортировка слиянием
- быстрая сортирова Хоара
- метод бисекции
- двоичный поиск
- **перемножение больших чисел**

$$12\,34 = 12 \times 10^2 + 34$$

$$56\,78 = 56 \times 10^2 + 78$$

$$z_2 = 12 \times 56 = 672$$

$$z_0 = 34 \times 78 = 2652$$

$$z_1 = (12 + 34)(56 + 78) - z_2 - z_0 = 2840$$

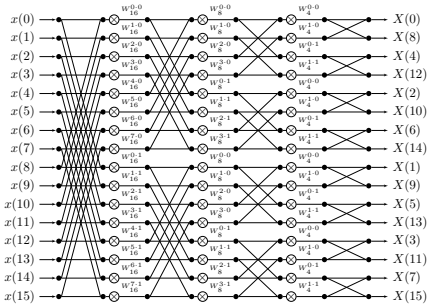
$$result = z_2 \times 10^{2\times 2} + z_1 \times 10^2 + z_0 = 7006652.$$

Разделяй и властвуй

Парадигма разработки **быстрых алгоритмов**, состоящая в разбиении задачи на подзадачи того же типа, но меньшего размера и комбинировании их результатов.

Примеры:

- сортировка слиянием
- быстрая сортирова Хоара
- метод бисекции
- двоичный поиск
- перемножение больших чисел
- **быстрое преобразование Фурье**

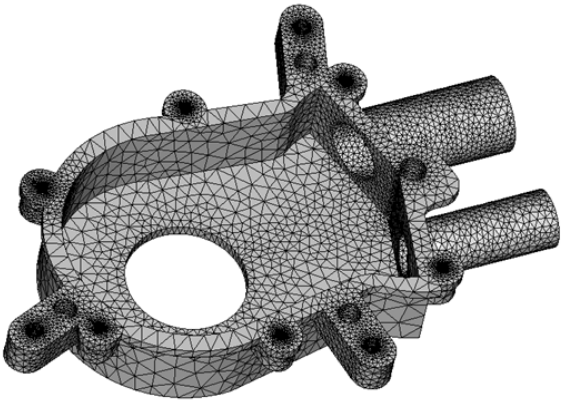


Разделяй и властвуй

Парадигма разработки **быстрых алгоритмов**, состоящая в разбиении задачи на подзадачи того же типа, но меньшего размера и комбинировании их результатов.

Примеры:

- сортировка слиянием
- быстрая сортирова Хоара
- метод бисекции
- двоичный поиск
- перемножение больших чисел
- быстрое преобразование Фурье
- **построение сеток**



Разделяй и властвуй

Парадигма разработки **быстрых алгоритмов**, состоящая в разбиении задачи на подзадачи того же типа, но меньшего размера и комбинировании их результатов.

Примеры:

- сортировка слиянием
- быстрая сортирова Хоара
- метод бисекции
- двоичный поиск
- перемножение больших чисел
- быстрое преобразование Фурье
- построение сеток

- Решение концептуально сложных задач.

Разделяй и властвуй

Парадигма разработки **быстрых алгоритмов**, состоящая в разбиении задачи на подзадачи того же типа, но меньшего размера и комбинировании их результатов.

Примеры:

- сортировка слиянием
- быстрая сортирова Хоара
- метод бисекции
- двоичный поиск
- перемножение больших чисел
- быстрое преобразование Фурье
- построение сеток

- Решение концептуально сложных задач.
- Повышение эффективности решений.

Разделяй и властвуй

Парадигма разработки **быстрых алгоритмов**, состоящая в разбиении задачи на подзадачи того же типа, но меньшего размера и комбинировании их результатов.

Примеры:

- сортировка слиянием
- быстрая сортирова Хоара
- метод бисекции
- двоичный поиск
- перемножение больших чисел
- быстрое преобразование Фурье
- построение сеток

- Решение концептуально сложных задач.
- Повышение эффективности решений.
- Пригодность к параллельной реализации.

Разделяй и властвуй

Парадигма разработки **быстрых алгоритмов**, состоящая в разбиении задачи на подзадачи того же типа, но меньшего размера и комбинировании их результатов.

Примеры:

- сортировка слиянием
- быстрая сортирова Хоара
- метод бисекции
- двоичный поиск
- перемножение больших чисел
- быстрое преобразование Фурье
- построение сеток

- Решение концептуально сложных задач.
- Повышение эффективности решений.
- Пригодность к параллельной реализации.

Разделяй и властвуй

Парадигма разработки **быстрых алгоритмов**, состоящая в разбиении задачи на подзадачи того же типа, но меньшего размера и комбинировании их результатов.

Примеры:

- сортировка слиянием
- быстрая сортирова Хоара
- метод бисекции
- двоичный поиск
- перемножение больших чисел
- быстрое преобразование Фурье
- построение сеток

- Решение концептуально сложных задач.
- Повышение эффективности решений.
- Пригодность к параллельной реализации.

Рекурсия наиболее просто и естественно реализует эту парадигму.

1. Почему рекурсия — это зло?
2. Почему рекурсия?
3. Где ещё бывает рекурсия?
4. **Какая ещё бывает рекурсия?**
5. Когда же она закончится?
6. См. пункт 1.

Рекурсивный процесс

Прислушаемся к мнению тех, кто утверждает, что рекурсия неэффективна и разберёмся, как производятся вычисления.

Рекурсивный процесс

Прислушаемся к мнению тех, кто утверждает, что рекурсия неэффективна и разберёмся, как производятся вычисления.

$$S(0) = 0$$
$$S(n) = S(n-1) + n^2$$

Рекурсивный процесс

Прислушаемся к мнению тех, кто утверждает, что рекурсия неэффективна и разберёмся, как производятся вычисления.

$$S(0) = 0$$
$$S(n) = S(n-1) + n^2$$

S(4)

Рекурсивный процесс

Прислушаемся к мнению тех, кто утверждает, что рекурсия неэффективна и разберёмся, как производятся вычисления.

$$S(0) = 0$$
$$S(n) = S(n-1) + n^2$$

S(4)

1. S(3) + 16

Рекурсивный процесс

Прислушаемся к мнению тех, кто утверждает, что рекурсия неэффективна и разберёмся, как производятся вычисления.

$$S(0) = 0$$
$$S(n) = S(n-1) + n^2$$

$\mathrm{S}(4)$

1. $\mathrm{S}(3) + 16$
2. $(\mathrm{S}(2) + 9) + 16$

Рекурсивный процесс

Прислушаемся к мнению тех, кто утверждает, что рекурсия неэффективна и разберёмся, как производятся вычисления.

$$S(0) = 0$$
$$S(n) = S(n-1) + n^2$$

$\mathrm{S}(4)$

1. $\mathrm{S}(3) + 16$
2. $(\mathrm{S}(2) + 9) + 16$
3. $((\mathrm{S}(1) + 4) + 9) + 16$

Рекурсивный процесс

Прислушаемся к мнению тех, кто утверждает, что рекурсия неэффективна и разберёмся, как производятся вычисления.

$$S(0) = 0$$
$$S(n) = S(n-1) + n^2$$

$\mathrm{S}(4)$

1. $\mathrm{S}(3) + 16$
2. $(\mathrm{S}(2) + 9) + 16$
3. $((\mathrm{S}(1) + 4) + 9) + 16$
4. $(((\mathrm{S}(0) + 1) + 4) + 9) + 16$

Рекурсивный процесс

Прислушаемся к мнению тех, кто утверждает, что рекурсия неэффективна и разберёмся, как производятся вычисления.

$$S(0) = 0$$
$$S(n) = S(n-1) + n^2$$

$\mathrm{S}(4)$

1. $\mathrm{S}(3) + 16$
2. $(\mathrm{S}(2) + 9) + 16$
3. $((\mathrm{S}(1) + 4) + 9) + 16$
4. $(((\mathrm{S}(0) + 1) + 4) + 9) + 16$
5. $(((0 + 1) + 4) + 9) + 16$

Рекурсивный процесс

Прислушаемся к мнению тех, кто утверждает, что рекурсия неэффективна и разберёмся, как производятся вычисления.

$$S(0) = 0$$
$$S(n) = S(n-1) + n^2$$

S(4)

1. S(3) + 16
2. (S(2) + 9) + 16
3. ((S(1) + 4) + 9) + 16
4. (((S(0) + 1) + 4) + 9) + 16
5. (((0 + 1) + 4) + 9) + 16
6. ((1 + 4) + 9) + 16

Рекурсивный процесс

Прислушаемся к мнению тех, кто утверждает, что рекурсия неэффективна и разберёмся, как производятся вычисления.

$$S(0) = 0$$
$$S(n) = S(n-1) + n^2$$

S(4)

1. S(3) + 16
2. (S(2) + 9) + 16
3. ((S(1) + 4) + 9) + 16
4. (((S(0) + 1) + 4) + 9) + 16
5. (((0 + 1) + 4) + 9) + 16
6. ((1 + 4) + 9) + 16
7. (5 + 9) + 16

Рекурсивный процесс

Прислушаемся к мнению тех, кто утверждает, что рекурсия неэффективна и разберёмся, как производятся вычисления.

$$S(0) = 0$$
$$S(n) = S(n-1) + n^2$$

S(4)

1. S(3) + 16
2. (S(2) + 9) + 16
3. ((S(1) + 4) + 9) + 16
4. (((S(0) + 1) + 4) + 9) + 16
5. (((0 + 1) + 4) + 9) + 16
6. ((1 + 4) + 9) + 16
7. (5 + 9) + 16
8. 14 + 16

Рекурсивный процесс

Прислушаемся к мнению тех, кто утверждает, что рекурсия неэффективна и разберёмся, как производятся вычисления.

$$S(0) = 0$$
$$S(n) = S(n-1) + n^2$$

S(4)

1. S(3) + 16
2. (S(2) + 9) + 16
3. ((S(1) + 4) + 9) + 16
4. (((S(0) + 1) + 4) + 9) + 16
5. (((0 + 1) + 4) + 9) + 16
6. ((1 + 4) + 9) + 16
7. (5 + 9) + 16
8. 14 + 16
9. 30

Рекурсивный процесс

Прислушаемся к мнению тех, кто утверждает, что рекурсия неэффективна и разберёмся, как производятся вычисления.

$$S(0) = 0$$
$$S(n) = S(n-1) + n^2$$

Характеристики рекурсивного процесса

- Вычисление имеет две стадии:

S(4)

1. S(3) + 16
2. (S(2) + 9) + 16
3. ((S(1) + 4) + 9) + 16
4. (((S(0) + 1) + 4) + 9) + 16
5. (((0 + 1) + 4) + 9) + 16
6. ((1 + 4) + 9) + 16
7. (5 + 9) + 16
8. 14 + 16
9. 30

Рекурсивный процесс

Прислушаемся к мнению тех, кто утверждает, что рекурсия неэффективна и разберёмся, как производятся вычисления.

$$S(0) = 0$$
$$S(n) = S(n-1) + n^2$$

Характеристики рекурсивного процесса

- Вычисление имеет две стадии:
  - разворачивание цепочки отложенных операций;

S(4)

1. S(3) + 16
2. (S(2) + 9) + 16
3. ((S(1) + 4) + 9) + 16
4. (((S(0) + 1) + 4) + 9) + 16
5. (((0 + 1) + 4) + 9) + 16
6. ((1 + 4) + 9) + 16
7. (5 + 9) + 16
8. 14 + 16
9. 30

Рекурсивный процесс

Прислушаемся к мнению тех, кто утверждает, что рекурсия неэффективна и разберёмся, как производятся вычисления.

$$S(0) = 0$$
$$S(n) = S(n-1) + n^2$$

Характеристики рекурсивного процесса

- Вычисление имеет две стадии:
  - разворачивание цепочки отложенных операций;
  - сворачивание этой цепочки.

S(4)

1. S(3) + 16
2. (S(2) + 9) + 16
3. ((S(1) + 4) + 9) + 16
4. (((S(0) + 1) + 4) + 9) + 16
5. (((0 + 1) + 4) + 9) + 16
6. ((1 + 4) + 9) + 16
7. (5 + 9) + 16
8. 14 + 16
9. 30

Рекурсивный процесс

Прислушаемся к мнению тех, кто утверждает, что рекурсия неэффективна и разберёмся, как производятся вычисления.

$$S(0) = 0$$
$$S(n) = S(n-1) + n^2$$

Характеристики рекурсивного процесса

- Вычисление имеет две стадии:
  - разворачивание цепочки отложенных операций;
  - сворачивание этой цепочки.
- Для запоминания отложенных операций используется стек.

S(4)

1. S(3) + 16
2. (S(2) + 9) + 16
3. ((S(1) + 4) + 9) + 16
4. (((S(0) + 1) + 4) + 9) + 16
5. (((0 + 1) + 4) + 9) + 16
6. ((1 + 4) + 9) + 16
7. (5 + 9) + 16
8. 14 + 16
9. 30

Рекурсивный процесс

Прислушаемся к мнению тех, кто утверждает, что рекурсия неэффективна и разберёмся, как производятся вычисления.

$$S(0) = 0$$
$$S(n) = S(n-1) + n^2$$

Характеристики рекурсивного процесса

- Вычисление имеет две стадии:
  - разворачивание цепочки отложенных операций;
  - сворачивание этой цепочки.
- Для запоминания отложенных операций используется стек.
- Процесс имеет порядок роста $\Theta(n)$ по времени и памяти.

S(4)

1. S(3) + 16
2. (S(2) + 9) + 16
3. ((S(1) + 4) + 9) + 16
4. (((S(0) + 1) + 4) + 9) + 16
5. (((0 + 1) + 4) + 9) + 16
6. ((1 + 4) + 9) + 16
7. (5 + 9) + 16
8. 14 + 16
9. 30

Итеративный процесс

Процесс можно организовать по-другому:

$$S(n) = S'(0, n)$$
$$S'(sum, 0) = sum$$
$$S'(sum, i) = S'(sum + i^2, i - 1)$$

Итеративный процесс

Процесс можно организовать по-другому:

$$S(n) = S'(0, n)$$
$$S'(sum, 0) = sum$$
$$S'(sum, i) = S'(sum + i^2, i - 1)$$

Используются итератор i и накопитель sum, так же, как и в императивных программах.

$sum := 0$
$i := n$
until $i = 0$ **do**
  $sum := sum + i^2$
  $i := i - 1$
return sum

Итеративный процесс

Процесс можно организовать по-другому:

$$S(n) = S'(0, n)$$
$$S'(sum, 0) = sum$$
$$S'(sum, i) = S'(sum + i^2, i - 1)$$

Используются итератор i и накопитель sum, так же, как и в императивных программах.

$sum := 0$
$i := n$
until $i = 0$ **do**
  $sum := sum + i^2$
  $i := i - 1$
return sum

Процесс вычисления этой функции:

Рекурсия	Цикл
S(4)	Вызов программы

Итеративный процесс

Процесс можно организовать по-другому:

$$S(n) = S'(0, n)$$
$$S'(sum, 0) = sum$$
$$S'(sum, i) = S'(sum + i^2, i - 1)$$

Используются итератор i и накопитель sum, так же, как и в императивных программах.

$sum := 0$
$i := n$
until $i = 0$ **do**
  $sum := sum + i^2$
  $i := i - 1$
return sum

Процесс вычисления этой функции:

Рекурсия	Цикл
S(4)	Вызов программы
1. S′(4, 0)	1. i = 4; sum = 0

Итеративный процесс

Процесс можно организовать по-другому:

$$S(n) = S'(0, n)$$
$$S'(sum, 0) = sum$$
$$S'(sum, i) = S'(sum + i^2, i - 1)$$

Используются итератор i и накопитель sum, так же, как и в императивных программах.

$sum := 0$
$i := n$
until $i = 0$ **do**
  $sum := sum + i^2$
  $i := i - 1$
return sum

Процесс вычисления этой функции:

Рекурсия	Цикл
S(4)	Вызов программы
1. S′(4, 0)	1. i = 4; sum = 0
2. S′(3, 16)	2. i = 3; sum = 16

Итеративный процесс

Процесс можно организовать по-другому:

$$S(n) = S'(0, n)$$
$$S'(sum, 0) = sum$$
$$S'(sum, i) = S'(sum + i^2, i - 1)$$

Используются итератор i и накопитель sum, так же, как и в императивных программах.

$sum := 0$
$i := n$
until $i = 0$ **do**
  $sum := sum + i^2$
  $i := i - 1$
return sum

Процесс вычисления этой функции:

Рекурсия	Цикл
S(4)	Вызов программы
1. S′(4, 0)	1. i = 4; sum = 0
2. S′(3, 16)	2. i = 3; sum = 16
3. S′(2, 25)	3. i = 2; sum = 25

Итеративный процесс

Процесс можно организовать по-другому:

$$S(n) = S'(0, n)$$
$$S'(sum, 0) = sum$$
$$S'(sum, i) = S'(sum + i^2, i - 1)$$

Используются итератор i и накопитель sum, так же, как и в императивных программах.

$sum := 0$
$i := n$
until $i = 0$ **do**
  $sum := sum + i^2$
  $i := i - 1$
return sum

Процесс вычисления этой функции:

Рекурсия	Цикл
S(4)	Вызов программы
1. S′(4, 0)	1. i = 4; sum = 0
2. S′(3, 16)	2. i = 3; sum = 16
3. S′(2, 25)	3. i = 2; sum = 25
4. S′(1, 29)	4. i = 1; sum = 29

Итеративный процесс

Процесс можно организовать по-другому:

$$S(n) = S'(0, n)$$
$$S'(sum, 0) = sum$$
$$S'(sum, i) = S'(sum + i^2, i - 1)$$

Используются итератор i и накопитель sum, так же, как и в императивных программах.

$sum := 0$
$i := n$
until $i = 0$ **do**
  $sum := sum + i^2$
  $i := i - 1$
return sum

Процесс вычисления этой функции:

Рекурсия	Цикл
S(4)	Вызов программы
1. S′(4, 0)	1. i = 4; sum = 0
2. S′(3, 16)	2. i = 3; sum = 16
3. S′(2, 25)	3. i = 2; sum = 25
4. S′(1, 29)	4. i = 1; sum = 29
5. S′(0, 30)	5. i = 0; sum = 30

Итеративный процесс

Процесс можно организовать по-другому:

$$S(n) = S'(0, n)$$
$$S'(sum, 0) = sum$$
$$S'(sum, i) = S'(sum + i^2, i - 1)$$

Используются итератор i и накопитель sum, так же, как и в императивных программах.

$sum := 0$
$i := n$
until $i = 0$ **do**
  $sum := sum + i^2$
  $i := i - 1$
return sum

Процесс вычисления этой функции:

Рекурсия	Цикл
S(4)	Вызов программы
1. S′(4, 0)	1. i = 4; sum = 0
2. S′(3, 16)	2. i = 3; sum = 16
3. S′(2, 25)	3. i = 2; sum = 25
4. S′(1, 29)	4. i = 1; sum = 29
5. S′(0, 30)	5. i = 0; sum = 30
6. 30	6. return 30

Итеративный процесс

Процесс можно организовать по-другому:

$$S(n) = S'(0, n)$$
$$S'(sum, 0) = sum$$
$$S'(sum, i) = S'(sum + i^2, i - 1)$$

Используются итератор i и накопитель sum, так же, как и в императивных программах.

$sum := 0$
$i := n$
until $i = 0$ **do**
  $sum := sum + i^2$
  $i := i - 1$
return sum

Процесс вычисления этой функции:

Рекурсия	Цикл
S(4)	Вызов программы
1. S′(4, 0)	1. i = 4; sum = 0
2. S′(3, 16)	2. i = 3; sum = 16
3. S′(2, 25)	3. i = 2; sum = 25
4. S′(1, 29)	4. i = 1; sum = 29
5. S′(0, 30)	5. i = 0; sum = 30
6. 30	6. return 30

Итеративный процесс

Процесс можно организовать по-другому:

$$S(n) = S'(0, n)$$
$$S'(sum, 0) = sum$$
$$S'(sum, i) = S'(sum + i^2, i - 1)$$

Используются итератор i и накопитель sum, так же, как и в императивных программах.

$sum := 0$
$i := n$
until $i = 0$ **do**
  $sum := sum + i^2$
  $i := i - 1$
return sum

Процесс вычисления этой функции:

Рекурсия	Цикл
S(4)	Вызов программы
1. S′(4, 0)	1. i = 4; sum = 0
2. S′(3, 16)	2. i = 3; sum = 16
3. S′(2, 25)	3. i = 2; sum = 25
4. S′(1, 29)	4. i = 1; sum = 29
5. S′(0, 30)	5. i = 0; sum = 30
6. 30	6. return 30

Характеристики итеративного процесса

Итеративный процесс

Процесс можно организовать по-другому:

$$S(n) = S'(0, n)$$
$$S'(sum, 0) = sum$$
$$S'(sum, i) = S'(sum + i^2, i - 1)$$

Используются итератор i и накопитель sum, так же, как и в императивных программах.

$sum := 0$
$i := n$
until $i = 0$ **do**
  $sum := sum + i^2$
  $i := i - 1$
return sum

Процесс вычисления этой функции:

Рекурсия	Цикл
S(4)	Вызов программы
1. S′(4, 0)	1. i = 4; sum = 0
2. S′(3, 16)	2. i = 3; sum = 16
3. S′(2, 25)	3. i = 2; sum = 25
4. S′(1, 29)	4. i = 1; sum = 29
5. S′(0, 30)	5. i = 0; sum = 30
6. 30	6. return 30

Характеристики итеративного процесса

- Вычисление не разбивается на две стадии.

Итеративный процесс

Процесс можно организовать по-другому:

$$S(n) = S'(0, n)$$
$$S'(sum, 0) = sum$$
$$S'(sum, i) = S'(sum + i^2, i - 1)$$

Используются итератор i и накопитель sum, так же, как и в императивных программах.

$sum := 0$
$i := n$
until $i = 0$ **do**
  $sum := sum + i^2$
  $i := i - 1$
return sum

Процесс вычисления этой функции:

Рекурсия	Цикл
S(4)	Вызов программы
1. S′(4, 0)	1. i = 4; sum = 0
2. S′(3, 16)	2. i = 3; sum = 16
3. S′(2, 25)	3. i = 2; sum = 25
4. S′(1, 29)	4. i = 1; sum = 29
5. S′(0, 30)	5. i = 0; sum = 30
6. 30	6. return 30

Характеристики итеративного процесса

- Вычисление не разбивается на две стадии.
- Процесс имеет порядок роста $\Theta(n)$ по времени и $\Theta(1)$ по памяти.

Итеративный процесс

$$S(n) = S'(0, n)$$
$$S'(sum, 0) = sum$$
$$S'(sum, i) = S'(sum + i^2, i - 1)$$

sum := 0
i := n
until $i = 0$ do
  sum := sum + i^2
  i := i - 1
return sum

Итеративный цикл состоит из

1. инициализации;
2. условия выхода;
3. тела цикла;

Рекурсивный и итеративный процессы

Определение

Рекурсивный процесс — вычислительный процесс, состоящий из двух стадий: а) создания рекурсивной последовательности отложенных вычислений и б) вычисления этой последовательности.

Рекурсивный и итеративный процессы

Определение

Рекурсивный процесс — вычислительный процесс, состоящий из двух стадий: а) создания рекурсивной последовательности отложенных вычислений и б) вычисления этой последовательности.

Определение

Итеративный процесс — вычислительный процесс, сводящийся к многократному повторению вычислений, использующих результат предыдущих аналогичных вычислений.

Рекурсивный и итеративный процессы

Определение

Рекурсивный процесс — вычислительный процесс, состоящий из двух стадий: а) создания рекурсивной последовательности отложенных вычислений и б) вычисления этой последовательности.

Определение

Итеративный процесс — вычислительный процесс, сводящийся к многократному повторению вычислений, использующих результат предыдущих аналогичных вычислений.

Универсальность рекурсии

С помощью рекурсивных определений можно описывать как рекурсивные процессы, так и итеративные.

Рекурсивный и итеративный процессы

Рекурсивный процесс

$$S(n) = n^2 + S(n-1)$$
$$S(0) = 0$$

Итеративный процесс

$$S'(sum, i) = S'(sum + i^2, i - 1)$$
$$S'(sum, 0) = sum$$

Рекурсивный и итеративный процессы

Рекурсивный процесс

$$S(n) = n^2 + S(n-1)$$
$$S(0) = 0$$

Итеративный процесс

$$S'(sum, i) = S'(sum + i^2, i - 1)$$
$$S'(sum, 0) = sum$$

Обобщение

$$F(...) = f(..., F(...), ...)$$
$$F(base) = ...$$

Рекурсивный и итеративный процессы

Рекурсивный процесс

$$S(n) = n^2 + S(n-1)$$
$$S(0) = 0$$

Итеративный процесс

$$S'(sum, i) = S'(sum + i^2, i - 1)$$
$$S'(sum, 0) = sum$$

Обобщение

$$F(...) = f(..., F(...), ...)$$
$$F(base) = ...$$

Решение задачи **сводится** к комбинации решений других задач. Ответ задачи **выражается** через ответы других задач.

Рекурсивный и итеративный процессы

Рекурсивный процесс

$$S(n) = n^2 + S(n-1)$$
$$S(0) = 0$$

Итеративный процесс

$$S'(sum, i) = S'(sum + i^2, i - 1)$$
$$S'(sum, 0) = sum$$

Обобщение

$$F(...) = f(..., F(...), ...)$$
$$F(base) = ...$$

$$F(...) = F(..., f(...), ...)$$
$$F(base) = ...$$

Решение задачи **сводится** к комбинации решений других задач. Ответ задачи **выражается** через ответы других задач.

Рекурсивный и итеративный процессы

Рекурсивный процесс

$$S(n) = n^2 + S(n-1)$$
$$S(0) = 0$$

Итеративный процесс

$$S'(sum, i) = S'(sum + i^2, i - 1)$$
$$S'(sum, 0) = sum$$

Обобщение

$$F(...) = f(..., F(...), ...)$$
$$F(base) = ...$$

Решение задачи **сводится** к комбинации решений других задач.
Ответ задачи **выражается** через ответы других задач.

$$F(...) = F(..., f(...), ...)$$
$$F(base) = ...$$

Решение задачи **заменяется** решением другой задачи.
Ответ к задаче **совпадает** с ответом другой задачи.

Хвостовая рекурсия

Определение

Хвостовым называется вызов функции из тела другой функции, при котором результат вызываемой функции является результатом функции её вызывающей.

Хвостовая рекурсия

Определение

Хвостовым называется вызов функции из тела другой функции, при котором результат вызываемой функции является результатом функции её вызывающей.

$$f(...) = g(..., h(...), ...)$$

здесь g вызывается в хвостовой позиции, а h — нет.

Хвостовая рекурсия

Определение

Хвостовым называется вызов функции из тела другой функции, при котором результат вызываемой функции является результатом функции её вызывающей.

$$f(...) = g(..., h(...), ...)$$

здесь g вызывается в хвостовой позиции, а h — нет.

Определение

Рекурсия, в которой все рекурсивные вызовы являются хвостовыми, называется **хвостовой рекурсией**.

Хвостовая рекурсия

Определение

Хвостовым называется вызов функции из тела другой функции, при котором результат вызываемой функции является результатом функции её вызывающей.

$$f(...) = g(..., h(...), ...)$$

здесь g вызывается в хвостовой позиции, а h — нет.

Определение

Рекурсия, в которой все рекурсивные вызовы являются хвостовыми, называется **хвостовой рекурсией**.

Хвостовая рекурсия

- Реализует итеративный процесс:

Хвостовая рекурсия

Определение

Хвостовым называется вызов функции из тела другой функции, при котором результат вызываемой функции является результатом функции её вызывающей.

$$f(...) = g(..., h(...), ...)$$

здесь g вызывается в хвостовой позиции, а h — нет.

Определение

Рекурсия, в которой все рекурсивные вызовы являются хвостовыми, называется **хвостовой рекурсией**.

Хвостовая рекурсия

- Реализует итеративный процесс:
  - не образует цепочки отложенных вычислений;

Хвостовая рекурсия

Определение

Хвостовым называется вызов функции из тела другой функции, при котором результат вызываемой функции является результатом функции её вызывающей.

$$f(...) = g(..., h(...), ...)$$

здесь g вызывается в хвостовой позиции, а h — нет.

Определение

Рекурсия, в которой все рекурсивные вызовы являются хвостовыми, называется **хвостовой рекурсией**.

Хвостовая рекурсия

- Реализует итеративный процесс:
  - не образует цепочки отложенных вычислений;
  - не имеет стадий разворачивания и сворачивания.

Хвостовая рекурсия

Определение

Хвостовым называется вызов функции из тела другой функции, при котором результат вызываемой функции является результатом функции её вызывающей.

$$f(...) = g(..., h(...), ...)$$

здесь g вызывается в хвостовой позиции, а h — нет.

Определение

Рекурсия, в которой все рекурсивные вызовы являются хвостовыми, называется **хвостовой рекурсией**.

Хвостовая рекурсия

- Реализует итеративный процесс:
  - не образует цепочки отложенных вычислений;
  - не имеет стадий разворачивания и сворачивания.
- Однозначно транслируется в условный цикл, не использующий стек и рекурсивных вызовов.

Хвостовая рекурсия

Определение

Хвостовым называется вызов функции из тела другой функции, при котором результат вызываемой функции является результатом функции её вызывающей.

$$f(...) = g(..., h(...), ...)$$

здесь g вызывается в хвостовой позиции, а h — нет.

Определение

Рекурсия, в которой все рекурсивные вызовы являются хвостовыми, называется **хвостовой рекурсией**.

Хвостовая рекурсия

- Реализует итеративный процесс:
  - не образует цепочки отложенных вычислений;
  - не имеет стадий разворачивания и сворачивания.
- Однозначно транслируется в условный цикл, не использующий стек и рекурсивных вызовов.

Хвостовая рекурсия

Определение

Хвостовым называется вызов функции из тела другой функции, при котором результат вызываемой функции является результатом функции её вызывающей.

$$f(...) = g(..., h(...), ...)$$

здесь g вызывается в хвостовой позиции, а h — нет.

Определение

Рекурсия, в которой все рекурсивные вызовы являются хвостовыми, называется **хвостовой рекурсией**.

Хвостовая рекурсия

- Реализует итеративный процесс:
  - не образует цепочки отложенных вычислений;
  - не имеет стадий разворачивания и сворачивания.
- Однозначно транслируется в условный цикл, не использующий стек и рекурсивных вызовов.

В SCHEME она входит в стандарт языка и является обязательной.
Оптимизация хвостовой рекурсии используется в языках OCAML, HASKELL.

Хвостовая рекурсия

Плюсы и минусы

Хвостовая рекурсия

Плюсы и минусы

- Любая программа может быть переписана так, чтобы в ней содержались только хвостовые вызовы.

Хвостовая рекурсия

Плюсы и минусы

- Любая программа может быть переписана так, чтобы в ней содержались только хвостовые вызовы.
- Существует формальный способ трансляции рекурсивных программ в итеративные.

Хвостовая рекурсия

Плюсы и минусы

- Любая программа может быть переписана так, чтобы в ней содержались только хвостовые вызовы.
- Существует формальный способ трансляции рекурсивных программ в итеративные.
- Хвостовая рекурсия часто (но не всегда!) формулируется более громоздким кодом.

Хвостовая рекурсия

Плюсы и минусы

- Любая программа может быть переписана так, чтобы в ней содержались только хвостовые вызовы.
- Существует формальный способ трансляции рекурсивных программ в итеративные.
- Хвостовая рекурсия часто (но не всегда!) формулируется более громоздким кодом.
- При оптимизации хвостовой рекурсии трудно сохранить отладочную информацию о стеке вызовов.

Мемоизация

Иногда рекурсия приводит к многократному вычислению одних и тех же выражений.

Рассмотрим рекурсивное определение для последовательности чисел Фибоначчи:

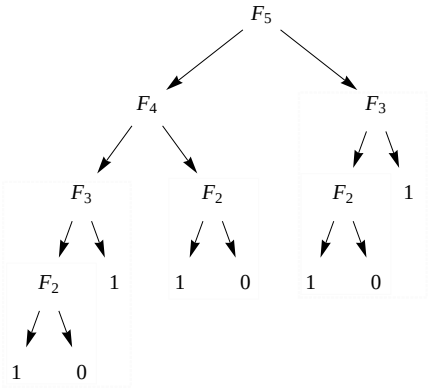
$$F_0 = 0$$
$$F_1 = 1$$
$$F_n = F_{n-1} + F_{n-2}$$

Мемоизация

Иногда рекурсия приводит к многократному вычислению одних и тех же выражений.

Рассмотрим рекурсивное определение для последовательности чисел Фибоначчи:

$$F_0 = 0$$
$$F_1 = 1$$
$$F_n = F_{n-1} + F_{n-2}$$

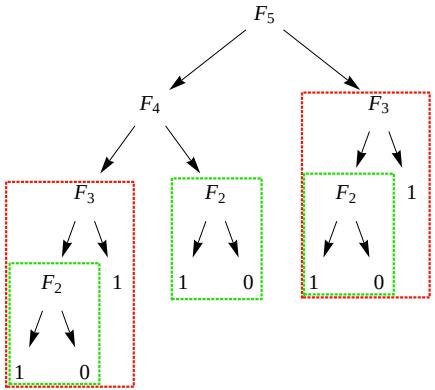


Мемоизация

Иногда рекурсия приводит к многократному вычислению одних и тех же выражений.

Рассмотрим рекурсивное определение для последовательности чисел Фибоначчи:

$$F_0 = 0$$
$$F_1 = 1$$
$$F_n = F_{n-1} + F_{n-2}$$



Процесс имеет экспоненциальный порядок роста $\Theta(1.6^n)$.

Мемоизация

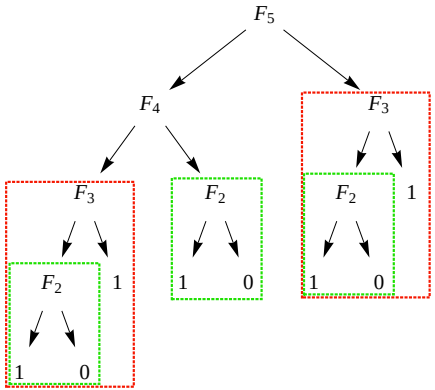
Иногда рекурсия приводит к многократному вычислению одних и тех же выражений.

Рассмотрим рекурсивное определение для последовательности чисел Фибоначчи:

$$F_0 = 0$$
$$F_1 = 1$$
$$F_n = F_{n-1} + F_{n-2}$$

Определение

Мемоизация – замена повторного вызова функции от одних и тех же аргументов результатом, однажды полученным для этих аргументов.



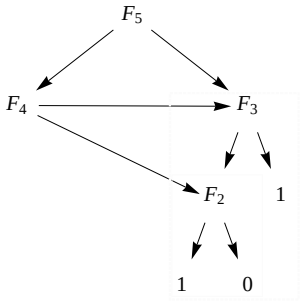
Процесс имеет экспоненциальный порядок роста $\Theta(1.6^n)$.

Мемоизация

Иногда рекурсия приводит к многократному вычислению одних и тех же выражений.

Рассмотрим рекурсивное определение для последовательности чисел Фибоначчи:

$$F_0 = 0$$
$$F_1 = 1$$
$$F_n = F_{n-1} + F_{n-2}$$



Определение

Мемоизация – замена повторного вызова функции от одних и тех же аргументов результатом, однажды полученным для этих аргументов.

Процесс имеет линейный порядок роста $\Theta(n)$.

Мемоизация

- Мемоизированная функция, произведя вычисления для конкретных значений аргументов, запоминает в таблице полученный результат.

Мемоизация

- Мемоизированная функция, произведя вычисления для конкретных значений аргументов, запоминает в таблице полученный результат.
- В дальнейшем, при вызове этой функции от этих же аргументов, результат возвращается сразу вместо вычисления.

Мемоизация

- Мемоизированная функция, произведя вычисления для конкретных значений аргументов, запоминает в таблице полученный результат.
- В дальнейшем, при вызове этой функции от этих же аргументов, результат возвращается сразу вместо вычисления.
- Для каждого набора аргументов функция вычисляется лишь однажды.

Мемоизация

- Мемоизированная функция, произведя вычисления для конкретных значений аргументов, запоминает в таблице полученный результат.
- В дальнейшем, при вызове этой функции от этих же аргументов, результат возвращается сразу вместо вычисления.
- Для каждого набора аргументов функция вычисляется лишь однажды.
- Поиск в таблице может быть организован в виде быстрого поиска (поиск в двоичном дереве).

Мемоизация

- Мемоизированная функция, произведя вычисления для конкретных значений аргументов, запоминает в таблице полученный результат.
- В дальнейшем, при вызове этой функции от этих же аргументов, результат возвращается сразу вместо вычисления.
- Для каждого набора аргументов функция вычисляется лишь однажды.
- Поиск в таблице может быть организован в виде быстрого поиска (поиск в двоичном дереве).

- Мемоизация оптимизирует время вычисления и использование стека за счёт памяти.

Мемоизация

- Мемоизированная функция, произведя вычисления для конкретных значений аргументов, запоминает в таблице полученный результат.
- В дальнейшем, при вызове этой функции от этих же аргументов, результат возвращается сразу вместо вычисления.
- Для каждого набора аргументов функция вычисляется лишь однажды.
- Поиск в таблице может быть организован в виде быстрого поиска (поиск в двоичном дереве).

- Мемоизация оптимизирует время вычисления и использование стека за счёт памяти.
- Мемоизация возможна только при использовании функций, прозрачных по ссылкам.

Мемоизация

- Мемоизированная функция, произведя вычисления для конкретных значений аргументов, запоминает в таблице полученный результат.
- В дальнейшем, при вызове этой функции от этих же аргументов, результат возвращается сразу вместо вычисления.
- Для каждого набора аргументов функция вычисляется лишь однажды.
- Поиск в таблице может быть организован в виде быстрого поиска (поиск в двоичном дереве).

- Мемоизация оптимизирует время вычисления и использование стека за счёт памяти.
- Мемоизация возможна только при использовании функций, прозрачных по ссылкам.
- Реализована не во всех языках программирования.

Мемоизация

- Мемоизированная функция, произведя вычисления для конкретных значений аргументов, запоминает в таблице полученный результат.
- В дальнейшем, при вызове этой функции от этих же аргументов, результат возвращается сразу вместо вычисления.
- Для каждого набора аргументов функция вычисляется лишь однажды.
- Поиск в таблице может быть организован в виде быстрого поиска (поиск в двоичном дереве).

- Мемоизация оптимизирует время вычисления и использование стека за счёт памяти.
- Мемоизация возможна только при использовании функций, прозрачных по ссылкам.
- Реализована не во всех языках программирования.

  Есть в Haskell, Python, Mathematica, Formica.

Мемоизация

- Мемоизированная функция, произведя вычисления для конкретных значений аргументов, запоминает в таблице полученный результат.
- В дальнейшем, при вызове этой функции от этих же аргументов, результат возвращается сразу вместо вычисления.
- Для каждого набора аргументов функция вычисляется лишь однажды.
- Поиск в таблице может быть организован в виде быстрого поиска (поиск в двоичном дереве).

- Мемоизация оптимизирует время вычисления и использование стека за счёт памяти.
- Мемоизация возможна только при использовании функций, прозрачных по ссылкам.
- Реализована не во всех языках программирования.

  Есть в Haskell, Python, Mathematica, Formica.

  Доступна в виде библиотек в в Scheme, Common Lisp, OCaml, Java, C++ и др.

1. Почему рекурсия — это зло?
2. Почему рекурсия?
3. Где ещё бывает рекурсия?
4. Какая ещё бывает рекурсия?
5. Когда же она закончится?
6. См. пункт 1.

Обоснованность рекурсии

Как определить, завершится ли рекурсия?

Обоснованность рекурсии

Как определить, завершится ли рекурсия?

1. Необходимо, чтобы подзадача к которой сводится решение, была в каком-то смысле «меньше» основной задачи.

Обоснованность рекурсии

Как определить, завершится ли рекурсия?

1. Необходимо, чтобы подзадача к которой сводится решение, была в каком-то смысле «меньше» основной задачи.
2. Необходимо существование «минимальной» задачи, являющейся базой рекурсии.

Обоснованность рекурсии

Как определить, завершится ли рекурсия?

1. Необходимо, чтобы подзадача к которой сводится решение, была в каком-то смысле «меньше» основной задачи.
2. Необходимо существование «минимальной» задачи, являющейся базой рекурсии.

Определение

Отношение $\prec$ на множестве S называется **вполне обоснованным**, если для любого подмножества S существует элемент, минимальный относительно $\prec$.

Обоснованность рекурсии

Как определить, завершится ли рекурсия?

1. Необходимо, чтобы подзадача к которой сводится решение, была в каком-то смысле «меньше» основной задачи.
2. Необходимо существование «минимальной» задачи, являющейся базой рекурсии.

Определение

Отношение $\prec$ на множестве S называется **вполне обоснованным**, если для любого подмножества S существует элемент, минимальный относительно $\prec$.

Такое отношение не образует бесконечной последовательности упорядоченных элементов $\ldots \prec a_1 \prec a_2 \prec \ldots$.

Обоснованность рекурсии

Примеры вполне обоснованных отношений:

Обоснованность рекурсии

Примеры вполне обоснованных отношений:

- отношение $<$ на множестве натуральных чисел;

$$0 < 1 < 2 < 3 < 4 < 5 < \ldots$$

$$3 < 64 < 123 < 19293$$

Обоснованность рекурсии

Примеры вполне обоснованных отношений:

- отношение $<$ на множестве натуральных чисел;
- отношение над конечными строками конечного алфавита: $s_1 \prec s_2$ если s_1 является подстрокой s_2;

$$0 < 1 < 2 < 3 < 4 < 5 < \ldots$$

$$3 < 64 < 123 < 19293$$

$$\varnothing \prec \mathrm{n} \prec \mathrm{unc} \prec \mathrm{func} \prec \mathrm{function}$$

Обоснованность рекурсии

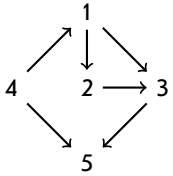
Примеры вполне обоснованных отношений:

- отношение $<$ на множестве натуральных чисел;
- отношение над конечными строками конечного алфавита: $s_1 \prec s_2$ если s_1 является подстрокой s_2;
- отношение связности над множеством узлов любого **направленного нецикличного графа**: $n_1 \prec n_2$, если существует путь из n_1 в n_2.

$$0 < 1 < 2 < 3 < 4 < 5 < \ldots$$

$$3 < 64 < 123 < 19293$$

$$\varnothing \prec \text{n} \prec \text{unc} \prec \text{func} \prec \text{function}$$



$$4 \prec 1 \prec 2 \prec 3 \prec 5$$

Обоснованность рекурсии

Примеры отношений, не являющихся вполне обоснованными:

Обоснованность рекурсии

Примеры отношений, не являющихся вполне обоснованными:

- отношение $>$ на множестве натуральных чисел;

$$0 < 1 < 2 < 3 < 4 < 5 < \ldots$$

Обоснованность рекурсии

Примеры отношений, не являющихся вполне обоснованными:

- отношение $>$ на множестве натуральных чисел;
- отношение $<$ на множестве целых чисел;

$$0 < 1 < 2 < 3 < 4 < 5 < \ldots$$

$$\ldots < -2 < -1 < -0 < 1 < 2 \ldots$$

Обоснованность рекурсии

Примеры отношений, не являющихся вполне обоснованными:

- отношение $>$ на множестве натуральных чисел;
- отношение $<$ на множестве целых чисел;
- отношение $<$ на множестве рациональных чисел;

$$0 < 1 < 2 < 3 < 4 < 5 < ...$$

$$... < -2 < -1 < -0 < 1 < 2...$$

$$0 < ... < \frac{1}{3} < \frac{1}{2} < ... < \frac{999}{1000} < ... < 1$$

Обоснованность рекурсии

Примеры отношений, не являющихся вполне обоснованными:

- отношение $>$ на множестве натуральных чисел;
- отношение $<$ на множестве целых чисел;
- отношение $<$ на множестве рациональных чисел;
- отношение лексикографического порядка над строками конечного алфавита.

$$0 < 1 < 2 < 3 < 4 < 5 < ...$$

$$... < -2 < -1 < -0 < 1 < 2...$$

$$0 < ... < \frac{1}{3} < \frac{1}{2} < ... < \frac{999}{1000} < ... < 1$$

$$... \prec aaab \prec aab \prec ab \prec b \prec bab \prec bb \prec ...$$

Обоснованность рекурсии

Примеры отношений, не являющихся вполне обоснованными:

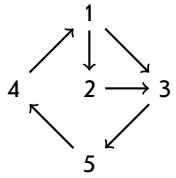
- отношение $>$ на множестве натуральных чисел;
- отношение $<$ на множестве целых чисел;
- отношение $<$ на множестве рациональных чисел;
- отношение лексикографического порядка над строками конечного алфавита.
- отношение связности над множеством узлов любого *цикличного графа*: $n_1 \prec n_2$, если существует путь из n_1 в n_2.

$$0 < 1 < 2 < 3 < 4 < 5 < ...$$

$$... < -2 < -1 < -0 < 1 < 2...$$

$$0 < ... < \frac{1}{3} < \frac{1}{2} < ... < \frac{999}{1000} < ... < 1$$

$$... \prec aaab \prec aab \prec ab \prec b \prec bab \prec bb \prec ...$$



Структурно обоснованная рекурсия

Обработка любых **индуктивных данных**
удовлетворяет условиям обоснованности:

Структурно обоснованная рекурсия

Обработка любых **индуктивных данных** удовлетворяет условиям обоснованности:

1. Для этих данных существует понятие следующего элемента, определяемого рекурсивным конструктором, следовательно для них можно определить отношение порядка.

Структурно обоснованная рекурсия

Обработка любых **индуктивных данных** удовлетворяет условиям обоснованности:

1. Для этих данных существует понятие следующего элемента, определяемого рекурсивным конструктором, следовательно для них можно определить отношение порядка.
2. Существуют нерекурсивные базовые элементы (один или несколько), «минимальные» для отношения порядка.

Структурно обоснованная рекурсия

Обработка любых **индуктивных данных** удовлетворяет условиям обоснованности:

1. Для этих данных существует понятие следующего элемента, определяемого рекурсивным конструктором, следовательно для них можно определить отношение порядка.
2. Существуют нерекурсивные базовые элементы (один или несколько), «минимальные» для отношения порядка.

Определение

Процедура называется структурно рекурсивной по аргументу индуктивного типа T, если она

1. нерекурсивно определена для всех базовых элементов T,
2. для каждого из рекурсивных конструкторов T всякий рекурсивный вызов производятся от непосредственных аргументов этого конструктора.

Структурно обоснованная рекурсия

Обработка любых **индуктивных данных** удовлетворяет условиям обоснованности:

1. Для этих данных существует понятие следующего элемента, определяемого рекурсивным конструктором, следовательно для них можно определить отношение порядка.
2. Существуют нерекурсивные базовые элементы (один или несколько), «минимальные» для отношения порядка.

Определение

Процедура называется структурно рекурсивной по аргументу индуктивного типа T, если она

1. нерекурсивно определена для всех базовых элементов T,
2. для каждого из рекурсивных конструкторов T всякий рекурсивный вызов производятся от непосредственных аргументов этого конструктора.

Структурно рекурсивная процедура обязательно завершается на всех конечных аргументах.

Структурно обоснованная рекурсия

Вычисление суммы квадратов

```
ℕ = 0
  | 1 + ℕ

S 0 = 0
S (n + 1) = (S n) + n²
```

Функция **структурно обоснована** по аргументу – натуральному числу.

Структурно обоснованная рекурсия

Вычисление суммы квадратов

```
ℕ = 0
  | 1 + ℕ

S 0 = 0
S (n + 1) = (S n) + n²
```

Функция **структурно обоснована** по аргументу – натуральному числу.

Вычисление суммы листьев бинарного дерева

```
𝔹 = ∅
  | Leaf x
  | Node 𝔹 𝔹

S ∅ = 0
S (Leaf x) = x
S (Node l r) = (S l) + (S r)
```

Функция **структурно обоснована**: рекурсивный вызов производится от аргумента рекурсивного конструктора.

Вполне обоснованная рекурсия

Вычисление НОД

```
gcd 0 b = b
gcd a 0 = a
gcd a b = if a ⩽ b
          then  gcd a (b − a)
          else  gcd (a − b) b
```

Функция **не является структурно рекурсивной**, но она **вполне обосновано рекурсивна** по отношению между аргументами рекурсивного (a', b') и исходного (a, b) вызовов: $a' + b' < a + b$, поэтому она гарантированно завершается.

Вполне обоснованная рекурсия

Вычисление НОД

```
gcd 0 b = b
gcd a 0 = a
gcd a b = if a ⩽ b
          then  gcd a (b − a)
          else  gcd (a − b) b
```

Функция **не является структурно рекурсивной**, но она **вполне обосновано рекурсивна** по отношению между аргументами рекурсивного (a', b') и исходного (a, b) вызовов: $a' + b' < a + b$, поэтому она гарантированно завершается.

Вычисление синуса

$$\sin x = 3 \sin \frac{x}{3} - 4 \sin^3 \frac{x}{3}$$

$$\sin x = x, \quad \text{при} \quad x \to 0$$

Рекурсия **не является вполне обоснованной**.

Вполне обоснованная рекурсия

Вычисление НОД

```
gcd 0 b = b
gcd a 0 = a
gcd a b = if a ⩽ b
          then  gcd a (b − a)
          else  gcd (a − b) b
```

Функция **не является структурно рекурсивной**, но она **вполне обосновано рекурсивна** по отношению между аргументами рекурсивного (a', b') и исходного (a, b) вызовов: $a' + b' < a + b$, поэтому она гарантированно завершается.

Вычисление синуса

$$\sin x = 3 \sin \frac{x}{3} - 4 \sin^3 \frac{x}{3}$$

$$\sin x = x, \quad \text{при} \quad x < \varepsilon$$

Рекурсия **не является вполне обоснованной**.

Введение малого числа ε определяющего точность вычислений делает рекурсию **вполне обоснованной**.

Вполне обоснованная рекурсия

Вычисление НОД

```
gcd 0 b = b
gcd a 0 = a
gcd a b = if a ⩽ b
          then  gcd a (b − a)
          else  gcd (a − b) b
```

Функция **не является структурно рекурсивной**, но она **вполне обосновано рекурсивна** по отношению между аргументами рекурсивного (a', b') и исходного (a, b) вызовов: $a' + b' < a + b$, поэтому она гарантированно завершается.

Вычисление синуса

$$\sin x = 3 \sin \frac{x}{3} - 4 \sin^3 \frac{x}{3}$$

$$\sin x = x, \quad \text{при} \quad x < \varepsilon$$

Рекурсия **не является вполне обоснованной.**

Введение малого числа ε определяющего точность вычислений делает рекурсию **вполне обоснованной.**

Введение счётчика максимального числа итераций делает рекурсию **структурно обоснованной.**

Завершаемость циклов

Условный цикл

$sum := 0$
$i := n$
while $i \neq 0$ **do**
  $sum := sum + i^2$
  $i := i - 1$
return sum

Обоснованность обрабатываемого множества требуется для условного цикла в той же мере, что и для рекурсии.

Завершаемость циклов

Условный цикл

$sum := 0$
$i := n$
while $i \neq 0$ **do**
  $sum := sum + i^2$
  $i := i - 1$
return sum

Обоснованность обрабатываемого множества требуется для условного цикла в той же мере, что и для рекурсии.

Условие $i \neq 0$ не гарантирует завершаемости.

Завершаемость циклов

Условный цикл

$sum := 0$
$i := n$
while $i \neq 0$ **do**
  $sum := sum + i^2$
  $i := i - 1$
return sum

Обоснованность обрабатываемого множества требуется для условного цикла в той же мере, что и для рекурсии.

Условие $i \neq 0$ не гарантирует завершаемости.

Условие $i > 0$ – гарантирует.

Завершаемость циклов

Условный цикл

$sum := 0$
$i := n$
while $i \neq 0$ **do**
  $sum := sum + i^2$
  $i := i - 1$
return sum

Обоснованность обрабатываемого множества требуется для условного цикла в той же мере, что и для рекурсии.

Условие $i \neq 0$ не гарантирует завершаемости.

Условие $i > 0$ – гарантирует.

Завершаемость цикла также зависит от типа счётчика i.

Завершаемость циклов

Условный цикл

$sum := 0$
$i := n$
while $i \neq 0$ **do**
  $sum := sum + i^2$
  $i := i - 1$
return sum

Итератор

$sum := 0$
for i=1 **to** n **do**
  $sum := sum + i^2$
return sum

Обоснованность обрабатываемого множества требуется для условного цикла в той же мере, что и для рекурсии.

Условие $i \neq 0$ не гарантирует завершаемости.

Условие $i > 0$ – гарантирует.

Завершаемость цикла также зависит от типа счётчика i.

Итерационный цикл эквивалентен структурно-обоснованной рекурсии и завершается гарантированно.

1. Почему рекурсия — это зло?
2. Почему рекурсия?
3. Где ещё бывает рекурсия?
4. Какая ещё бывает рекурсия?
5. Когда же она закончится?
6. См. пункт 1.

Здравый взгляд на рекурсию

- Рекурсия – это непонятно: рекурсивные решения неочевидны, программы трудно отлаживать.

Здравый взгляд на рекурсию

- Рекурсия – это непонятно: рекурсивные решения неочевидны, программы трудно отлаживать.
- Рекурсия может быть непривычной, но она столь же естественна, как и циклические конструкции.

Haskell

```
qsort []     = []
qsort (p:xs) = (qsort (filter (< p) xs)) ++ [p] ++ (qsort (filter (>= p) xs))
```

Haskell

```
qsort []     = []
qsort (p:xs) = (qsort (filter (< p) xs)) ++ [p] ++ (qsort (filter (>= p) xs))
```

C++

```
int quickSort(int *arr, int elements) {
#define  MAX_LEVELS  1000
int  piv, beg[MAX_LEVELS], end[MAX_LEVELS], i=0, L, R ;
beg[0]=0; end[0]=elements;
while (i>=0) {
  L=beg[i]; R=end[i]-1;
  if (L<R)
  { piv=arr[L]; if (i==MAX_LEVELS-1) return 0;
    while (L<R)
    { while (arr[R]>=piv && L<R) R--; if (L<R) arr[L++]=arr[R];
      while (arr[L]<=piv && L<R) L++; if (L<R) arr[R--]=arr[L]; }
    arr[L]=piv; beg[i+1]=L+1; end[i+1]=end[i]; end[i++]=L; }
  else { i--; }}
return 1; }
```

Здравый взгляд на рекурсию

- Рекурсия – это непонятно: рекурсивные решения неочевидны, программы трудно отлаживать.
- Рекурсия может быть непривычной, но она столь же естественна, как и циклические конструкции.

Здравый взгляд на рекурсию

- Рекурсия – это непонятно: рекурсивные решения неочевидны, программы трудно отлаживать.

- Рекурсия может быть непривычной, но она столь же естественна, как и циклические конструкции.
- Сложные задачи требуют сложных решений.

Здравый взгляд на рекурсию

- Рекурсия – это непонятно: рекурсивные решения неочевидны, программы трудно отлаживать.

- Рекурсия может быть непривычной, но она столь же естественна, как и циклические конструкции.
- Сложные задачи требуют сложных решений.
- Отладку программ, во многих случаях, необходимо заменять на юнит-тестирование в сочетании с повышением модульности.

Здравый взгляд на рекурсию

- Рекурсия – это непонятно: рекурсивные решения неочевидны, программы трудно отлаживать.

- Рекурсия может быть непривычной, но она столь же естественна, как и циклические конструкции.
- Сложные задачи требуют сложных решений.
- Отладку программ, во многих случаях, необходимо заменять на юнит-тестирование в сочетании с повышением модульности.
- Отладку следует заменять доказательством корректности. Для рекурсивных решений корректность решения малой подзадачи подразумевает корректность решения всей задачи.

Здравый взгляд на рекурсию

- Рекурсия – это непонятно: рекурсивные решения неочевидны, программы трудно отлаживать.

- Рекурсия может быть непривычной, но она столь же естественна, как и циклические конструкции.
- Сложные задачи требуют сложных решений.
- Отладку программ, во многих случаях, необходимо заменять на юнит-тестирование в сочетании с повышением модульности.
- Отладку следует заменять доказательством корректности. Для рекурсивных решений корректность решения малой подзадачи подразумевает корректность решения всей задачи.
- Корректность рекурсивных программ можно доказать по индукции.

Здравый взгляд на рекурсию

- Рекурсия хороша только для олимпиадных задач и для теоретиков. В реальных задачах она практически не используется.

Здравый взгляд на рекурсию

- Рекурсия хороша только для олимпиадных задач и для теоретиков. В реальных задачах она практически не используется.
- Для решения каждодневных задач используются простые (линейные) комбинации библиотечных модулей. Рекурсия лежит в основе очень многих быстрых и надёжных алгоритмов, инкапсулированных в библиотеки.

Здравый взгляд на рекурсию

- Рекурсия хороша только для олимпиадных задач и для теоретиков. В реальных задачах она практически не используется.

- Для решения каждодневных задач используются простые (линейные) комбинации библиотечных модулей. Рекурсия лежит в основе очень многих быстрых и надёжных алгоритмов, инкапсулированных в библиотеки.
- Рекурсия — инструмент для работы с рекурсивными (индуктивными) данными: динамическими списками, очередями, потоками, бинарными и размеченными деревьями, синтаксическими и грамматическими структурами и т.д. Если вы каждый день работаете с такими данными, рекурсия – инструмент на каждый день.

Здравый взгляд на рекурсию

- Рекурсия опасна: она может никогда не закончиться.

Здравый взгляд на рекурсию

- Рекурсия опасна: она может никогда не закончиться.
- Цикл **while** опасен: он может никогда не закончиться,

Здравый взгляд на рекурсию

- Рекурсия опасна: она может никогда не закончиться.

- Цикл **while** опасен: он может никогда не закончиться,

  а **goto** ещё опаснее.

Здравый взгляд на рекурсию

- Рекурсия опасна: она может никогда не закончиться.

- Цикл **while** опасен: он может никогда не закончиться,

  а **goto** ещё опаснее.

- Обоснованность гарантирует завершимость рекурсии при условии использования чистых функций.

Здравый взгляд на рекурсию

- Рекурсия опасна: она может никогда не закончиться.

- Цикл **while** опасен: он может никогда не закончиться,

  а **goto** ещё опаснее.
- Обоснованность гарантирует завершимость рекурсии при условии использования чистых функций.
  - Для цикла **for** завершаемость гарантирована.

Здравый взгляд на рекурсию

- Рекурсия опасна: она может никогда не закончиться.

- Цикл **while** опасен: он может никогда не закончиться,

  а **goto** ещё опаснее.

- Обоснованность гарантирует завершимость рекурсии при условии использования чистых функций.
  - Для цикла **for** завершаемость гарантирована.
  - Для условного цикла **while** — нет.

Здравый взгляд на рекурсию

- Рекурсия опасна: она может никогда не закончиться.

- Цикл **while** опасен: он может никогда не закончиться,

  а **goto** ещё опаснее.

- Обоснованность гарантирует завершимость рекурсии при условии использования чистых функций.
  - Для цикла **for** завершаемость гарантирована.
  - Для условного цикла **while** — нет.
  - Проблема завершаемости условного цикла с побочными эффектами неразрешима!

Здравый взгляд на рекурсию

- Рекурсия опасна: она может никогда не закончиться.

- Цикл **while** опасен: он может никогда не закончиться,

  а **goto** ещё опаснее.

- Обоснованность гарантирует завершимость рекурсии при условии использования чистых функций.
  - Для цикла **for** завершаемость гарантирована.
  - Для условного цикла **while** — нет.
  - Проблема завершаемости условного цикла с побочными эффектами неразрешима!

Здравый взгляд на рекурсию

- Рекурсия опасна: она может никогда не закончиться.
- Рекурсия опасна: неизвестно, сколько она потребует памяти и времени.
- Цикл **while** опасен: он может никогда не закончиться,

  а **goto** ещё опаснее.
- Обоснованность гарантирует завершимость рекурсии при условии использования чистых функций.
  - Для цикла **for** завершаемость гарантирована.
  - Для условного цикла **while** — нет.
  - Проблема завершаемости условного цикла с побочными эффектами неразрешима!

Здравый взгляд на рекурсию

- Рекурсия опасна: она может никогда не закончиться.
- Рекурсия опасна: неизвестно, сколько она потребует памяти и времени.
- Цикл **while** опасен: он может никогда не закончиться,

  а **goto** ещё опаснее.
- Обоснованность гарантирует завершимость рекурсии при условии использования чистых функций.
  - Для цикла **for** завершаемость гарантирована.
  - Для условного цикла **while** — нет.
  - Проблема завершаемости условного цикла с побочными эффектами неразрешима!
- Рекурсия универсальна: неважно насколько велики обрабатываемые данные.

Здравый взгляд на рекурсию

- Рекурсия опасна: она может никогда не закончиться.
- Рекурсия опасна: неизвестно, сколько она потребует памяти и времени.

- Цикл **while** опасен: он может никогда не закончиться,

  а **goto** ещё опаснее.
- Обоснованность гарантирует завершимость рекурсии при условии использования чистых функций.
  - Для цикла **for** завершаемость гарантирована.
  - Для условного цикла **while** — нет.
  - Проблема завершаемости условного цикла с побочными эффектами неразрешима!
- Рекурсия универсальна: неважно насколько велики обрабатываемые данные.

  Это свойство совершенно необходимо для обработки динамических данных.

Здравый взгляд на рекурсию

- Рекурсивные программы ресурсоёмки и неэффективны.

Здравый взгляд на рекурсию

- Рекурсивные программы ресурсоёмки и неэффективны.
- Используйте хвостовую рекурсию, мемоизацию и подходящие языки программирования.

Здравый взгляд на рекурсию

- Рекурсивные программы ресурсоёмки и неэффективны.

- Используйте хвостовую рекурсию, мемоизацию и подходящие языки программирования.
- Рекурсивные решения часто лежат в основе быстрых алгоритмов («разделяй и властвуй»).

Здравый взгляд на рекурсию

- Рекурсивные программы ресурсоёмки и неэффективны.

- Используйте хвостовую рекурсию, мемоизацию и подходящие языки программирования.
- Рекурсивные решения часто лежат в основе быстрых алгоритмов («разделяй и властвуй»).
- Оптимизировать следует не только машинные ресурсы (они дешевеют), но и человеческие (они дорожают).

Здравый взгляд на рекурсию

- Рекурсивные программы ресурсоёмки и неэффективны.

- Используйте хвостовую рекурсию, мемоизацию и подходящие языки программирования.
- Рекурсивные решения часто лежат в основе быстрых алгоритмов («разделяй и властвуй»).
- Оптимизировать следует не только машинные ресурсы (они дешевеют), но и человеческие (они дорожают).

  Рекурсия позволяет эффективно находить надёжные и естественные для предметной области решения.

Здравый взгляд на рекурсию

- Всё, что можно сделать с помощью рекурсии, можно сделать на циклах. Так зачем всё усложнять?

Здравый взгляд на рекурсию

- Всё, что можно сделать с помощью рекурсии, можно сделать на циклах. Так зачем всё усложнять?

- См. пример с быстрой сортировкой.

Здравый взгляд на рекурсию

- Всё, что можно сделать с помощью рекурсии, можно сделать на циклах. Так зачем всё усложнять?

- См. пример с быстрой сортировкой.
- Рекурсии – рекурсивное, итерациям – итеративное.

Здравый взгляд на рекурсию

- Всё, что можно сделать с помощью рекурсии, можно сделать на циклах. Так зачем всё усложнять?

- См. пример с быстрой сортировкой.
- Рекурсии – рекурсивное, итерациям – итеративное.
- Не стоит использовать рекурсивные процессы для обработки статических массивов и для выполнения явных итерационных операций.

Здравый взгляд на рекурсию

- Всё, что можно сделать с помощью рекурсии, можно сделать на циклах. Так зачем всё усложнять?

- См. пример с быстрой сортировкой.
- Рекурсии – рекурсивное, итерациям – итеративное.
- Не стоит использовать рекурсивные процессы для обработки статических массивов и для выполнения явных итерационных операций.
- Не стоит усложнять и запутывать программу разворачивая древообразные структуры в линейные статические структуры.

Здравый взгляд на рекурсию

- Во многих классических учебниках по программированию настоятельно рекомендуют избегать рекурсии

  (*«...если бы программист, работающий у меня, применял рекурсию для вычисления факториала, я бы нанял кого-то другого...» С. Макконнелл «Совершенный код»*)

Здравый взгляд на рекурсию

- Во многих классических учебниках по программированию настоятельно рекомендуют избегать рекурсии

  (*«...если бы программист, работающий у меня, применял рекурсию для вычисления факториала, я бы нанял кого-то другого...» С. Макконнелл «Совершенный код»*)

- Правильно — не надо вычислять факториалы рекурсией,

Здравый взгляд на рекурсию

- Во многих классических учебниках по программированию настоятельно рекомендуют избегать рекурсии

  (*«...если бы программист, работающий у меня, применял рекурсию для вычисления факториала, я бы нанял кого-то другого...» С. Макконнелл «Совершенный код»*)

- Правильно — не надо вычислять факториалы рекурсией,

  и числа Фибоначчи не надо.

Здравый взгляд на рекурсию

- Во многих классических учебниках по программированию настоятельно рекомендуют избегать рекурсии

  (*«...если бы программист, работающий у меня, применял рекурсию для вычисления факториала, я бы нанял кого-то другого...» С. Макконнелл «Совершенный код»*)

- Правильно — не надо вычислять факториалы рекурсией,

  и числа Фибоначчи не надо.

  А функцию Аккермана, сопоставление с образцом, синтаксический разбор текстов или символьное интегрирование с упрощением выражений — надо.

Здравый взгляд на рекурсию

- Во многих классических учебниках по программированию настоятельно рекомендуют избегать рекурсии

  (*«...если бы программист, работающий у меня, применял рекурсию для вычисления факториала, я бы нанял кого-то другого...» С. Макконнелл «Совершенный код»*)

- Правильно — не надо вычислять факториалы рекурсией,

  и числа Фибоначчи не надо.

  А функцию Аккермана, сопоставление с образцом, синтаксический разбор текстов или символьное интегрирование с упрощением выражений — надо.

- Полный вариант цитаты: «Одна из проблем с учебниками по выч. технике в том, что они предлагают глупые примеры рекурсии. Типичными примерами рекурсии являются вычисление факториала или последовательности Фибоначчи. Рекурсия — мощный инструмент, и очень глупо использовать ее в этих 2-х случаях. Если бы программист, работающий у меня, применял рекурсию для вычисления факториала, я бы нанял кого-то другого.»

А на самом деле

А на самом деле

А на самом деле

хорошим стилем ФП считается, по возможности, избегать явного использования рекурсии, заменяя её комбинаторами общего назначения, таких как

`map`, `accumulate`, `fold`, `filter`, `fixed-point` и т. п.

А на самом деле

А на самом деле

хорошим стилем ФП считается, по возможности, избегать явного использования рекурсии, заменяя её комбинаторами общего назначения, таких как

`map`, `accumulate`, `fold`, `filter`, `fixed-point` и т. п.

$$S_n = \sum_{i=1}^{n} i^2$$

А на самом деле

А на самом деле

хорошим стилем ФП считается, по возможности, избегать явного использования рекурсии, заменяя её комбинаторами общего назначения, таких как

`map`, `accumulate`, `fold`, `filter`, `fixed-point` и т. п.

$$S_n = \sum_{i=1}^{n} i^2$$

Явный цикл:

$sum := 0$
for i=1 **to** n **do**
  $sum := sum + i^2$
return sum

А на самом деле

А на самом деле

хорошим стилем ФП считается, по возможности, избегать явного использования рекурсии, заменяя её комбинаторами общего назначения, таких как

`map`, `accumulate`, `fold`, `filter`, `fixed-point` и т. п.

$$S_n = \sum_{i=1}^{n} i^2$$

Явный цикл:

$sum := 0$
for i=1 **to** n **do**
  $sum := sum + i^2$
return sum

Рекурсия:

```
S 0 = 0
S (n + 1) = (S n) + n²
```

А на самом деле

А на самом деле

хорошим стилем ФП считается, по возможности, избегать явного использования рекурсии, заменяя её комбинаторами общего назначения, таких как

`map`, `accumulate`, `fold`, `filter`, `fixed-point` и т. п.

$$S_n = \sum_{i=1}^{n} i^2$$

Явный цикл:

$sum := 0$
for i=1 **to** n **do**
  $sum := sum + i^2$
return sum

Рекурсия:

```
S 0 = 0
S (n + 1) = (S n) + n²
```

Комбинаторное решение:

```
S = sum (x ↦ x²)
```

А на самом деле

А на самом деле

хорошим стилем ФП считается, по возможности, избегать явного использования рекурсии, заменяя её комбинаторами общего назначения, таких как

`map`, `accumulate`, `fold`, `filter`, `fixed-point` и т. п.

$$S_n = \sum_{i=1}^{n} i^2$$

Явный цикл:

$sum := 0$
for i=1 **to** n **do**
  $sum := sum + i^2$
return sum

Рекурсия:

```
S 0 = 0
S (n + 1) = (S n) + n²
```

Комбинаторное решение:

```
S = sum (x ↦ x²)

sum = accumulate + 0
```