ИТМО

Функциональное программирование

Отчёт по лабораторной работе №1

Решение задач проекта Эйлер на Elixir

Группа *P3331*
Вариант *9,21*

Выполнил: *Дворкин Борис Александрович*

Дата защиты: 05.10.2024

Количество баллов:

СПб – 2024

Задача 9

Условие:

Существуют такие натуральные числа a, b и c, что:

$$a^2 + b^2 = c^2, \text{при } a + b + c = 1000$$

Нужно найти произведение этих чисел abc.

Ключевые элементы реализации

Решение с использованием потоков (Stream):

```
defmodule Euler9Stream do
  def find_triplet(sum) do
    Stream.iterate(1, &(&1 + 1))
    |> Stream.take_while(&(&1 < sum / 3))
    |> Stream.flat_map(fn a ->
      Stream.iterate(a + 1, &(&1 + 1))
      |> Stream.take_while(&(&1 < sum / 2))
      |> Stream.map(fn b ->
        c = sum - a - b
        {a, b, c}
      end)
    end)
    |> Stream.filter(fn {a, b, c} -> a * a + b * b ==
       c * c end)
    |> Enum.map(fn {a, b, c} -> a * b * c end)
    |> Enum.at(0)
  end
end
```

Листинг 1: Генерация чисел с использованием Stream

Этот код использует потоки для генерации чисел a, b, c и фильтрации только тех, которые удовлетворяют условию Пифагора.

Решение с использованием модульного подхода:

```
defmodule Euler9Modular do
  def find_triplet(sum) do
    1..(sum - 2)
    |> Enum.flat_map(fn a ->
      (a + 1)..(sum - a - 1)
      |> Enum.map(fn b ->
        c = sum - a - b
```

```
8           {a, b, c}
9         end)
10      end)
11      |> Enum.filter(fn {a, b, c} -> a * a + b * b == c
            * c end)
12      |> Enum.map(fn {a, b, c} -> a * b * c end)
13      |> Enum.at(0)
14    end
15  end
```

Листинг 2: Генерация чисел с использованием диапазонов

Модульный подход использует диапазоны для генерации возможных значений a, b, c.

Задача 21

Условие:

Определим $d(n)$ как сумму всех собственных делителей числа n (чисел меньше n, на которые n делится нацело). Если $d(a) = b$ и $d(b) = a$, где $a \neq b$, то a и b — дружественная пара, а a и b называются дружественными числами.

Например, собственные делители 220 — $1, 2, 4, 5, 10, 11, 20, 22, 44, 55$ и 110, поэтому $d(220) = 284$. Собственные делители 284 — $1, 2, 4, 71$ и 142, так что $d(284) = 220$.

Необходимо найти сумму всех дружественных чисел меньше 10000.

Ключевые элементы реализации

Решение с использованием потоков (Stream):

```
1  defmodule Euler21Stream do
2    def sum_amicable_numbers(limit) do
3      Stream.iterate(2, &(&1 + 1))
4      |> Stream.take_while(&(&1 < limit))
5      |> Stream.filter(&amicable?/1)
6      |> Enum.sum()
7    end
8
9    defp amicable?(n) do
10     sum_div = sum_of_divisors(n)
11     sum_div != n and sum_div < limit() and
         sum_of_divisors(sum_div) == n
12   end
13
```

```
  defp sum_of_divisors(n) do
    if n > 1 do
      1..div(n, 2)
      |> Enum.filter(&(rem(n, &1) == 0))
      |> Enum.sum()
    else
      0
    end
  end

  defp limit, do: 10_000
end
```

Листинг 3: Использование потоков для нахождения дружественных чисел

Использование потоков позволяет эффективно фильтровать дружественные числа.

Модульное решение:

```
defmodule Euler21Modular do
  def sum_amicable_numbers(limit) do
    2..(limit - 1)
    |> Enum.filter(&amicable?/1)
    |> Enum.sum()
  end

  defp amicable?(n) do
    sum_div = sum_of_divisors(n)
    sum_div != n and sum_div < limit() and
      sum_of_divisors(sum_div) == n
  end

  defp sum_of_divisors(n) do
    if n > 1 do
      1..div(n, 2)
      |> Enum.filter(&(rem(n, &1) == 0))
      |> Enum.sum()
    else
      0
    end
  end

  defp limit, do: 10_000
end
```

Листинг 4: Модульный подход к поиску дружественных чисел

Модульный подход использует диапазоны и фильтрацию для нахождения дружественных чисел.

Рекурсивное решение:

```elixir
defmodule Euler21Recursion do
  def sum_amicable_numbers(limit) do
    do_sum(2, limit, [])
  end

  defp do_sum(n, limit, acc) when n < limit do
    sum_div = sum_of_divisors(n)

    if sum_div != n and sum_of_divisors(sum_div) == n do
      do_sum(n + 1, limit, [n | acc])
    else
      do_sum(n + 1, limit, acc)
    end
  end

  defp do_sum(_, _, acc), do: Enum.sum(acc)

  defp sum_of_divisors(n), do: sum_of_divisors(n, div(n, 2), 0)

  defp sum_of_divisors(_, i, acc) when i <= 0, do: acc

  defp sum_of_divisors(n, i, acc) do
    if rem(n, i) == 0 do
      sum_of_divisors(n, i - 1, acc + i)
    else
      sum_of_divisors(n, i - 1, acc)
    end
  end
end
```

Листинг 5: Рекурсивный подход к поиску дружественных чисел

Использование рекурсии для нахождения суммы дружественных чисел.

Решение на основе Map:

```elixir
defmodule Euler21Map do
  def sum_amicable_numbers(limit) do
    2..(limit - 1)
    |> Enum.map(&{&1, sum_of_divisors(&1)})
    |> Enum.filter(fn {n, sum_div} ->
      sum_div != n and sum_div < limit and
        sum_of_divisors(sum_div) == n
    end)
    |> Enum.map(fn {n, _} -> n end)
    |> Enum.sum()
  end

  defp sum_of_divisors(n) do
    if n > 1 do
      1..div(n, 2)
      |> Enum.filter(&(rem(n, &1) == 0))
      |> Enum.sum()
    else
      0
    end
  end
end
```

Листинг 6: Использование Map для нахождения дружественных чисел

Использование функций отображения (Map) для нахождения дружественных чисел.

Решение с использованием списков (List Comprehensions):

```elixir
defmodule Euler21ListComp do
  def sum_amicable_numbers(limit) do
    for(
      n <- 2..(limit - 1),
      amicable?(n),
      do: n
    )
    |> Enum.sum()
  end

  defp amicable?(n) do
    sum_div = sum_of_divisors(n)
```

```
13      sum_div != n and sum_div < limit() and
          sum_of_divisors(sum_div) == n
14    end
15
16    defp sum_of_divisors(n) do
17      if n > 1 do
18        for(
19          i <- 1..div(n, 2),
20          rem(n, i) == 0,
21          do: i
22        )
23        |> Enum.sum()
24      else
25        0
26      end
27    end
28
29    defp limit, do: 10_000
30  end
```

Листинг 7: Использование списковых выражений для нахождения дружественных чисел

Использование списковых выражений для поиска дружественных чисел.

Выводы

В ходе решения задач были применены различные техники: потоки, рекурсия, модульный подход, отображения и списковые выражения. Каждая из них предоставляет свои преимущества в определённых условиях. Потоки и рекурсия обеспечивают элегантные решения для большого количества данных, в то время как отображения и списковые выражения упрощают код и делают его более читаемым.