ИТМО

Функциональное программирование

Отчет по лабораторной работе №1

Выполнил:
Ефремов Марк Андреевич

Группа:
P3334

Преподаватель:
Пенской Александр Владимирович

СПб – 2024

Проблема 4

https://projecteuler.net/problem=4

Найти наибольшее число-палиндром, являющееся произведением двух трёхзначных чисел.

Реализация:

Перебор пар трёхзначных чисел I и J со следующими оптимизациями:
- Перебор от наибольших значений к наименьшим
- Умножение коммутативно, можно рассматривать J < I
- Если найдена пара I_1 и J, такая, что $I_1 * J_1$ – палиндром, то, переходя к дальнейшим итерациям, где I_2, можно рассматривать только $J < J_1$.

Обычная рекурсия и хвостовая

```
let largest_palindrome lower upper =
  let rec helper i j border =
    if i < lower then -1
    else if j <= border then helper (i - 1) (i - 1) j
    else
      let product = i * j in
      if is_palindrome product then max product (helper (i - 1) (i - 1) j)
      else helper i (j - 1) border
  in
  helper upper upper lower
```

```
let largest_palindrome lower upper =
  let rec helper i j border acc =
    if i < lower then acc
    else if j <= border then helper (i - 1) (i - 1) j acc
    else
      let product = i * j in
      if is_palindrome product then helper (i - 1) (i - 1) j (max product acc)
      else helper i (j - 1) border acc
  in
  helper upper upper lower (-1)
```

Модульная реализация, генерация с помощью рекурсии, ленивые коллекции

```
let generate_products lower upper =
  let rec helper i j lst =
    if i < lower then lst
    else if j < lower then helper (i - 1) (i - 1) lst
    else helper i (j - 1) ((i * j) :: lst)
  in
  helper upper upper []

let largest_palindrome upper lower =
  generate_products lower upper
  |> List.filter is_palindrome |> List.fold_left max (-1)
```

```
let generate_products diff lower =
  let range = List.init diff (fun i -> lower + i) in
  let multiply i = List.map (fun j -> i * j) range in
  List.flatten (List.map multiply range)

let largest_palindrome upper lower =
  generate_products (upper - lower + 1) lower
  |> List.filter is_palindrome |> List.fold_left max (-1)
```

```
let generate_products dif lower =
  let range = Seq.init dif (fun i -> lower + i) in
  let multiply i = Seq.map (fun j -> i * j) range in
  Seq.flat_map multiply range

let largest_palindrome upper lower =
  generate_products (upper - lower + 1) lower
  |> Seq.filter is_palindrome |> Seq.fold_left max (-1)
```

Реализации в императивном стиле

```
let largest_palindrome lower upper =
  assert (upper > lower && lower > 0);
  let champ = ref (-1) in
  for i = upper downto lower do
    for j = i downto lower do
      let product = i * j in
      if is_palindrome product then champ := max !champ product
    done
  done;
  assert (!champ > 0);
  !champ
```

```
def largest_palindrome(lower, upper):
    assert (upper > lower and lower > 0)
    champ = 0
    for i in range(upper, lower, -1):
        for j in range(i, lower, -1):
            product = i * j
            if is_palindrome(product):
                champ = max(champ, product)
    assert (champ > 0)
    return champ
```

Выводы:

Во время работы с обычной рекурсией получал переполнение стека, в то время, как хвостовая рекурсия оптимизируется компилятором. Для проверки того, является ли рекурсия хвостовой, была использована аннотация @tailcall, но убрана из кода ради читабельности.

Модульная реализация с использованием последовательностей становится очень компактной за счёт оператора конвеерной обработки. Однако он требует написания большого количества лямбда-функций, которые засоряют код. Также, сложнее было применить вышеописанные оптимизации, поэтому модульная реализация оказалась очень медленной.

Реализация с ленивой коллекцией (Seq) полностью совпадает с “map-generated”, и также получилась медленной.

```
First problem. 100, 999

Solution 1 (regular recursion):
906609

Solution 2 (tail recursion):
906609

Solution 3 (modules):
906609

Solution 4 (map-generated):
906609

Solution 5 (loop syntax):
906609

Solution 6 (lazy Seq):
906609



Latencies for 5000 iterations of "Solution 1 (regular recursion)":
Solution 1 (regular recursion):  0.96 WALL ( 0.96 usr +  0.00 sys =  0.96 CPU) @ 5182.21/s (n=5000)

Latencies for 5000 iterations of "Solution 2 (tail recursion)":
Solution 2 (tail recursion):  0.98 WALL ( 0.98 usr +  0.00 sys =  0.98 CPU) @ 5101.36/s (n=5000)

Latencies for 100 iterations of "Solution 3 (modules)":
Solution 3 (modules):  3.03 WALL ( 2.87 usr +  0.16 sys =  3.03 CPU) @ 32.97/s (n=100)

Latencies for 50 iterations of "Solution 4 (map-generated)":
Solution 4 (map-generated):  6.89 WALL ( 6.85 usr +  0.04 sys =  6.89 CPU) @  7.26/s (n=50)

Latencies for 100 iterations of "Solution 5 (loop syntax)":
Solution 5 (loop syntax):  1.28 WALL ( 1.28 usr +  0.00 sys =  1.28 CPU) @ 78.41/s (n=100)

Latencies for 100 iterations of "Solution 6 (lazy Seq)":
Solution 6 (lazy Seq):  5.24 WALL ( 5.23 usr +  0.00 sys =  5.23 CPU) @ 19.14/s (n=100)
```

Проблема 27

https://projecteuler.net/problem=27

Найти произведение чисел a и b, таких, что при подстановке в формулу n^2+an+b последовательных значений n от 0, она даёт наибольшее количество простых чисел.

Реализация:

Перебор пар трёхзначных чисел a и b со следующими оптимизациями:

- при n=0, $n^2+an+b = b$, должно быть простым числом. Вывод – b должно быть простым.
- при n=1, $n^2+an+b = 1 + a + b$. А значит, чтобы число получилось простым, у a и b должна быть разная чётность.

Обычная и хвостовая рекурсия:

```
let max_product limitA limitB =
  let rec helper a b =
    let parity_check = b mod 2 <> a mod 2 in
    let suitable = is_prime b && a <= limitA && parity_check in
    if b > limitB then (0, 0)
    else if suitable then
      let count = count_primes a b in
      let further = helper (a + 1) b in
      if count > fst further then (count, a * b) else further
    else helper (-limitA) (b + 1)
  in
  let count, result = helper (-limitA) (-limitB) in
  assert (count > 0);
  result
```

```
let max_product limitA limitB =
  let rec helper count product a b =
    let parity_check = b mod 2 <> a mod 2 in
    let suitable = is_prime b && a <= limitA && parity_check in
    if b > limitB then product
    else if suitable then
      let new_count = count_primes a b in
      if new_count > count then helper new_count (a * b) (a + 1) b
      else helper count product (a + 1) b
    else helper count product (-limitA) (b + 1)
  in
  helper 0 0 (-limitA) (-limitB)
```

Модульные реализации:

```
let generate_products limitA limitB =
  let rec helper a b acc =
    if a > limitA then acc
    else if b > limitB then helper (a + 1) (-limitB) acc
    else
      let count = count_primes a b in
      helper a (b + 1) ((count, a * b) :: acc)
  in
  helper (-limitA) (-limitB) []

let max_product lower upper =
  let select_champ x y = if fst x > fst y then x else y in
  generate_products lower upper |> List.fold_left select_champ (0, 0) |> snd
```

```
let generate_products limitA limitB =
  let modulo_range lim = List.init ((lim * 2) + 1) (fun i -> i - lim) in
  let parity_check b a = b mod 2 <> a mod 2 in
  let combine b =
    modulo_range limitA
    |> List.filter (parity_check b)
    |> List.map (fun a -> (count_primes a b, a * b))
  in
  modulo_range limitB |> List.filter is_prime |> List.map combine
  |> List.flatten

let max_product lower upper =
  let select_champ x y = if fst x > fst y then x else y in
  generate_products lower upper |> List.fold_left select_champ (0, 0) |> snd
```

```
let generate_products limitA limitB =
  let modulo_range lim = Seq.init ((lim * 2) + 1) (fun i -> i - lim) in
  let parity_check b a = b mod 2 <> a mod 2 in
  let combine b =
    modulo_range limitA
    |> Seq.filter (parity_check b)
    |> Seq.map (fun a -> (count_primes a b, a * b))
  in
  modulo_range limitB |> Seq.filter is_prime |> Seq.flat_map combine

let max_product lower upper =
  let aux x y = if fst x > fst y then x else y in
  generate_products lower upper |> Seq.fold_left aux (0, 0) |> snd
```

Реализации в императивном стиле:

```
let find_max_product limitA limitB =
  let max_product = ref 0 in
  let max_count = ref 0 in
  for b = -limitB to limitB do
    if is_prime b then
      for a = -limitA to limitA do
        if b mod 2 <> a mod 2 then
          let count = count_primes a b in
          if count > !max_count then (
            max_product := a * b;
            max_count := count)
      done
  done;
  assert (!max_count > 0);
  !max_product
```

```
def solution(limit_a, limit_b):
    max_count = 0
    max_product = 0
    for b in range(2, limit_b + 1):
        if is_prime(b):
            for a in range(-limit_a, limit_a + 1):
                if a % 2 == b % 2:
                    count = count_primes(a, b)
                    if (count > max_count):
                        max_count = count
                        max_product = a * b
    return max_product
```

```
Latencies for 30 iterations of "Solution 1 (regular recursion)":
Solution 1 (regular recursion):  2.26 WALL ( 2.26 usr +  0.00 sys =  2.26 CPU) @ 13.26/s (n=30)

Latencies for 30 iterations of "Solution 2 (tail recursion)":
Solution 2 (tail recursion):  1.86 WALL ( 1.86 usr +  0.00 sys =  1.86 CPU) @ 16.11/s (n=30)

Latencies for 10 iterations of "Solution 3 (modules)":
Solution 3 (modules):
 8.26 WALL ( 7.96 usr +  0.30 sys =  8.26 CPU) @  1.21/s (n=10)

Latencies for 10 iterations of "Solution 4 (map-generated)":
Solution 4 (map-generated):  1.15 WALL ( 1.15 usr +  0.00 sys =  1.15 CPU) @  8.67/s (n=10)

Latencies for 30 iterations of "Solution 5 (loop syntax)":
Solution 5 (loop syntax):  1.47 WALL ( 1.46 usr +  0.00 sys =  1.46 CPU) @ 20.54/s (n=30)

Latencies for 30 iterations of "Solution 6 (lazy Seq)":
Solution 6 (lazy Seq):  2.02 WALL ( 2.02 usr +  0.00 sys =  2.02 CPU) @ 14.85/s (n=30)
```

Выводы:

В данном случае обычную рекурсию было сделать сложнее, чем хвостовую.

Благодаря мутабельным полям record, а также синтаксису для циклов, на Ocaml можно написать решение, полностью аналогичное решению на python. Однако этот язык не предоставляет аналогов для “break” и “continue”, что значительно ограничивает возможности работы с циклами.

Генерация последовательности с помощью map - решение, которое для первой проблемы было самым медленным, для второй оказалось самым быстрым. Я предполагаю, что это связано с тем, что я применил фильтрацию на всех этапах формирования последовательности.