プログラミング B 課題 2

諏訪 凌太

2018 年 5 月 28 日

1 整数のリストの要素の最大値を返す関数 max

1.1 ソースプログラム

```
let rec range a b =
  if a > b then []
  else a :: range (a+1) b;;

let rec max_good a = match a with
  | [] -> min_int
  | h::[] -> h
  | h::t ->
    let v = max_good t in
    if h > v then h else v;;

let rec max_bad a = match a with
  | [] -> min_int
  | h::[] -> h
  | h::t -> if h > max_bad t then h else max_bad t;;

let ans1 = max_good([3;6;3;1;4;6;10;13;4;5;2;3;5;6;4;15;2;4]);;
let ans2 = max_good
    ([1;3;6;10;4;1;2;3;4;5;6;2;1;4;5;6;1;3;4;5;5;6;6;100;1;3;34;1;100;1002;1;3;5;6;3;2;4;1001;6;5;4;3])
    ;;
let ans3 = max_good([10;1;1;1;3;5;6;20]);;
let ans4= max_bad(range 1 100000);;
let () = Printf.printf "%d\n%d\n%d\n%d" ans1 ans2 ans3 ans4;;
```

1.2 実行結果

```
% ocaml max.ml
1002
20
100000
```

1.3 考察

trace max_bad を実行した結果, max は最小 n 回, 最大 n^2 回呼ばれることがわかった. 改良した max_good は $O(n)$ の手間で最大値が求まる.

2 fold_right を使ったリストの要素の最大値を求める関数

2.1 ソースプログラム

```
let max lst = List.fold_right (fun x y -> if x > y then x else y) lst (List.hd lst);;
let ans1 = max
    ([1;3;6;10;4;1;2;3;4;5;6;2;1;4;5;6;1;3;4;5;5;6;6;100;1;3;34;1;100;1002;1;3;5;6;3;2;4;1001;6;5;4;3])
    ;;
let ans2 = max ([1;3;5;7;3;2;2;4;5;6;9;10;4;3;2;2;4]);;
let ans3 = max ([1;2;3;4;5;6;7]);;

let () = Printf.printf "%d\n%d\n%d\n" ans1 ans2 ans3;;
```

2.2 実行結果

```
    % ocaml max_without_recursive.ml
    1002
    10
    7
```

3 fold_right を使った append 関数

3.1 ソースプログラム

```
let append lst1 lst2 = List.fold_right (fun e a -> e :: a) lst1 lst2;;

let ans1 = append ([1;3;6;10;4])([1;3]);;
let ans2 = append ([2;4])([]);;
let ans3 = append ([])([1;3;6;7]);;
```

3.2 実行結果

```
    # #use "append.ml";;
    # val append : 'a list -> 'a list -> 'a list = <fun>
    # val ans1 : int list = [1; 3; 6; 10; 4; 1; 3]
    # val ans2 : int list = [2; 4]
    # val ans3 : int list = [1; 3; 6; 7]
    #
```

4 fold_right を使った map 関数

4.1 ソースプログラム

```
let map f lst = List.fold_right (fun e a -> f e :: a) lst [];;

let ans1 = map (fun x -> x * x)([1;2;3;4;5;6;7;8;9;10]);;
let ans2 = map (fun x -> [x])([1;2;3;4;5;6;7;8;9;10]);;
let ans3 = map (fun x -> if x mod 2 = 0 then x else 0)([1;2;3;4;5;6;7;8;9;10]);;
```

4.2 実行結果

```
# #use "map.ml";;
# val map : ('a -> 'b) -> 'a list -> 'b list = <fun>
# val ans1 : int list = [1; 4; 9; 16; 25; 36; 49; 64; 81; 100]
# val ans2 : int list list = [[1]; [2]; [3]; [4]; [5]; [6]; [7]; [8]; [9]; [10]]
# val ans3 : int list = [0; 2; 0; 4; 0; 6; 0; 8; 0; 10]
#
```

5 fold_left を使ったリストの長さを求める関数

5.1 ソースプログラム

Listing 1: length.ml

```
open Printf

let length lst = List.fold_left (fun p a -> p + 1) 0 lst;;

let ans1 = length([1;100;189;156;554]);;
let ans2 = length([]);;
let ans3 = length([5;5;5;5;5;5;5;5;5;5;5;5;5;5]);;
print_int ans1;;
printf "\n";;
print_int ans2;;
printf "\n";;
print_int ans3;;
```

5.2 実行結果

```
% ocaml length.ml
5
0
14
```

6 fold_left を使った reverse 関数

6.1 ソースプログラム

```
let reverse lst = List.fold_left (fun p a -> a :: p) [] lst;;

let ans1 = reverse([1;100;189;156;554]);;
let ans2 = reverse([2;3;4;1;4]);;
let ans3 = reverse(["a";"b";"c"]);;
```

6.2 実行結果

```
# #use "reverse.ml";;
# val reverse : 'a list -> 'a list = <fun>
```

```
    # val ans1 : int list = [554; 156; 189; 100; 1]
    # val ans2 : int list = [4; 1; 4; 3; 2]
    # val ans3 : string list = ["c"; "b"; "a"]
    #
```

7　inner_product 関数

7.1　ソースプログラム

```
let inner_product lst1 lst2 =
  let lst = List.combine lst1 lst2 in
  List.fold_right (fun (a,b) c -> a * b + c) lst 0;;

let ans1 = inner_product([1;2;3])([4;5;6]);;
let ans2 = inner_product([1;1;1])([1;1;1]);;
let ans3 = inner_product([1;2;3])([0;0;0]);;
```

7.2　実行結果

```
    # #use "inner_product.ml";;
    # val inner_product : int list -> int list -> int = <fun>
    # val ans1 : int = 32
    # val ans2 : int = 3
    # val ans3 : int = 0
    #
```

8　product 関数

8.1　ソースプログラム

```
let product lst1 lst2 =
  List.fold_right (fun a x -> (List.map (fun y -> (a,y)) lst2) @ x) lst1 [];;

let ans1 = product ([1;2;3])(["a";"b"]);;
let ans2 = product ([1;2;3;4])(["";"a"]);;
let ans3 = product ([1;2;3])([]);;
```

8.2　実行結果

```
    # #use "product.ml";;
    # val product : 'a list -> 'b list -> ('a * 'b) list = <fun>
    val ans1 : (int * string) list = [(1, "a"); (1, "b"); (2, "a"); # (2, "b"); (3, "a");
        (3, "b")]
    val ans2 : (int * string) list = [(1, ""); (1, "a"); (2, ""); # (2, "a"); (3, ""); (3,
        "a"); (4, ""); (4, "a")]
    # val ans3 : (int * 'a) list = []
    #
```

9 map_product 関数

9.1 ソースプログラム

```
let map_product f lst1 lst2 =
  let product lst1 lst2 =
    List.fold_right (fun a x -> (List.map (fun y -> (a,y)) lst2) @ x) lst1 []
  in
  let lst = product lst1 lst2 in
  List.map f lst;;
let ans1 = map_product (fun (x,y) -> x + y)([1;2;3])([1;2]);;
let ans2 = map_product (fun (x,y) -> x * y)([1;2;3])([1;2]);;
let ans3 = map_product (fun (x,y) -> (y,x))([1;2;3])([1;2]);;
```

9.2 実行結果

```
    # #use "map_product.ml";;
    # val map_product : ('a * 'b -> 'c) -> 'a list -> 'b list -> 'c list = <fun>
    # val ans1 : int list = [2; 3; 3; 4; 4; 5]
    # val ans2 : int list = [1; 2; 2; 4; 3; 6]
    # val ans3 : (int * int) list =[(1, 1); (2, 1); (1, 2); (2, 2); (1, 3); (2, 3)]
    #
```

10 index 関数およびその改良関数

10.1 ソースプログラム

```
let index lst =
  let rec aux n = function
    | [] -> []
    | h::t -> List.map (fun x -> (x, n)) h @ aux (n + 1) t in
  List.sort compare (aux 1 lst);;

let index2 lst =
  let rec aux n = function
    | [] -> []
    | h::t -> List.map (fun x -> (x, [n])) h @ aux (n + 1) t in
  let rec aux2 (y, n) = function
    | [] -> (y, n) :: []
    | (hy, hn) :: t ->
      if hy = y && hn = n then aux2 (hy, hn) t
      else if hy = y then aux2 (hy, n @ hn) t
      else (y, n) :: aux2 (hy, hn) t in
  let lst = List.sort compare (aux 1 lst) in
  aux2 (List.hd lst) lst

let list_of_string_lists = [["red";"green";"blue";"black";"yellow"];["light-blue";"blue";"
    dark-blue"];["pink";"orange";"red";"gray"];["black";"white"];["gray";"blue";"silver";"
    yellow"]];;

let ans = index list_of_string_lists;;
let ans2 = index2 list_of_string_lists;;
```

10.2 実行結果

```
# #use "index.ml";;
# val index : 'a list list -> ('a * int) list = <fun>
# val index2 : 'a list list -> ('a * int list) list = <fun>
# val list_of_string_lists : string list list =
  [["red"; "green"; "blue"; "black"; "yellow"];
   ["light-blue"; "blue"; "dark-blue"]; ["pink"; "orange"; "red"; "gray"];
   ["black"; "white"]; ["gray"; "blue"; "silver"; "yellow"]]
# val ans : (string * int) list =
  [("black", 1); ("black", 4); ("blue", 1); ("blue", 2); ("blue", 5);
   ("dark-blue", 2); ("gray", 3); ("gray", 5); ("green", 1);
   ("light-blue", 2); ("orange", 3); ("pink", 3); ("red", 1); ("red", 3);
   ("silver", 5); ("white", 4); ("yellow", 1); ("yellow", 5)]
# val ans2 : (string * int list) list =
  [("black", [1; 4]); ("blue", [1; 2; 5]); ("dark-blue", [2]);
   ("gray", [3; 5]); ("green", [1]); ("light-blue", [2]); ("orange", [3]);
   ("pink", [3]); ("red", [1; 3]); ("silver", [5]); ("white", [4]);
   ("yellow", [1; 5])]
#
```

11 感想

問題 7, 8 は個人的に難しく感じた．もっときれいな書き方があると思われる，

今回用いたソースコードは GitHub に公開した．リンクを以下に示す．

`https://github.com/Enantiomer/OCaml_ProB`