Homework 2
COSE212, Fall 2019

Hakjoo Oh

Due: 10/14, 23:59

Academic Integrity / Assignment Policy

- All assignments must be your own work.
- Discussion with fellow students is encouraged including how to approach the problem. However, your code must be your own.
  - Discussion must be limited to general discussion and must not involve details of how to write code.
  - You must write your code by yourself and must not look at someone else's code (including ones on the web).
  - Do not allow other students to copy your code.
  - Do not post your code on the public web.
- **Violating above rules gets you 0 points for the entire HW score.**

Problem 1 Write a function

```
smallest_divisor: int -> int
```

that finds the smallest integral divisor (greater than 1) of a given number n. For example,

```
smallest_divisor 15 = 3
smallest_divisor 121 =11
smallest_divisor 141 = 3
smallest_divisor 199 = 199
```

Ensure that your algorithm runs in $\Theta(\sqrt{n})$ steps.

Problem 2 Write a higher-order function

```
sigma : (int -> int) -> int -> int -> int
```

such that `sigma f a b` computes

$$\sum_{i=a}^{b} f(i).$$

For instance,

```
sigma (fun x -> x) 1 10
```

evaulates to 55 and

```
sigma (fun x -> x*x) 1 7
```

evaluates to 140.

Problem 3 Write a higher-order function

```
forall : ('a -> bool) -> 'a list -> bool
```

which decides if all elements of a list satisfy a predicate. For example,

```
forall (fun x -> x mod 2 = 0) [1;2;3]
```

evaluates to false while

```
forall (fun x -> x > 5) [7;8;9]
```

is true.

Problem 4 Write a function

```
app: 'a list ->'a list -> 'a list
```

which appends the first list to the second list while removing duplicated elements. For instance, given two lists `[4;5;6;7]` and `[1;2;3;4]`, the function should output `[1;2;3;4;5;6;7]`:

```
app [4;5;6;7] [1;2;3;4] = [1;2;3;4;5;6;7].
```

Problem 5 Write a function

```
uniq: 'a list -> 'a list
```

which removes duplicated elements from a given list so that the list contains unique elements. For instance,

```
uniq [5;6;5;4] = [5;6;4]
```

Problem 6 Write a function `reduce` of the type:

```
reduce : ('a -> 'b -> 'c -> 'c) -> 'a list -> 'b list -> 'c -> 'c
```

Given a function `f` of type `'a -> 'b -> 'c -> 'c`, the expression

```
reduce f [x1;x2;...;xn] [y1;y2;...;yn] c1
```

evaluates to `f xn yn (... (f x2 y2 (f x1 y1 c1))...)`. For example,

```
reduce (fun x y z -> x * y + z) [1;2;3] [0;1;2] 0
```

evaluates to 8.

Problem 7 A directed graph can be represented as follows:

```
type graph = (vertex * vertex) list
and vertex = int
```

For example, the following graph

$$\begin{tikzcd}
1 \arrow[r] & 2 \arrow[r] \arrow[dr] & 3 \arrow[r] & 4 \arrow[ll, bend right] \\
 & & 5 &
\end{tikzcd}$$

is represented by `[(1,2);(2,3);(3,4);(4,2);(2,5)]`. Write a function

```
reach : graph * vertex -> vertex list
```

that returns the set of vertices that are reachable from the vertex given as the second argument. For example,

```
reach ([(1,2);(2,3);(3,4);(4,2);(2,5)], 1) = [1;2;3;4;5]
reach ([(1,2);(2,3);(3,4);(4,2);(2,5)], 2) = [2;3;4;5]
reach ([(1,2);(2,3);(3,4);(4,2);(2,5)], 3) = [2;3;4;5]
reach ([(1,2);(2,3);(3,4);(4,2);(2,5)], 4) = [2;3;4;5]
reach ([(1,2);(2,3);(3,4);(4,2);(2,5)], 5) = [5]
```

Problem 8 Write a function

```
diff : aexp * string -> aexp
```

that differentiates the given algebraic expression with respect to the variable given as the second argument. The algebraic expression `aexp` is defined as follows:

```
type aexp =
 | Const of int
 | Var of string
 | Power of string * int
 | Times of aexp list
 | Sum of aexp list
```

For example, $x^2 + 2x + 1$ is represented by

```
Sum [Power ("x", 2); Times [Const 2; Var "x"]; Const 1]
```

and differentiating it (w.r.t. "x") gives $2x + 2$, which can be represented by

```
Sum [Times [Const 2; Var "x"]; Const 2]
```

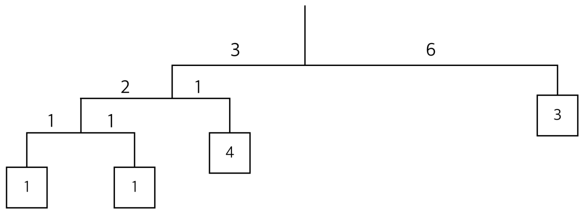
Note that the representation of $2x + 2$ in `aexp` is not unique. For instance, the following also represents $2x + 2$:

```
Sum
 [Times [Const 2; Power ("x", 1)];
  Sum
   [Times [Const 0; Var "x"];
    Times [Const 2; Sum [Times [Const 1]; Times [Var "x"; Const 0]]]];
  Const 0]
```

Problem 9 A binary mobile consists of two branches, a left branch and a right branch. Each branch is a rod of a certain length, from which hangs either a weight or another binary mobile. In OCaml datatype, a binary mobile can be defined as follows:

```
type mobile = branch * branch     (* left and rigth branches *)
and branch = SimpleBranch of length * weight
           | CompoundBranch of length * mobile
and length = int
and weight = int
```

A branch is either a simple branch, which is constructed from a length together with a weight, or a compound branch, which is constructed from a length together with another mobile. For instance, the mobile



is represented by the following:

```
(CompoundBranch (3,
  (CompoundBranch (2, (SimpleBranch (1, 1), SimpleBranch (1, 1))),
   SimpleBranch (1, 4))),
 SimpleBranch (6, 3))
```

Define the function

```
balanced : mobile -> bool
```

that tests whether a binary mobile is balanced. A mobile is said to be *balanced* if the torque applied by its top-left branch is equal to that applied by its top-right branch (that is, if the length of the left rod multiplied by the weight hanging from that rod is equal to the corresponding product for the right side) and if each of the submobiles hanging off its branches is balanced. For example, the example mobile above is balanced.

Problem 10 Consider the following expressions:

```
type exp = X
         | INT of int
         | ADD of exp * exp
         | SUB of exp * exp
         | MUL of exp * exp
         | DIV of exp * exp
         | SIGMA of exp * exp * exp
```

Implement a calculator for the expressions:

```
calculator : exp -> int
```

For instance,

$$\sum_{x=1}^{10}(x * x - 1)$$

is represented by

```
SIGMA(INT 1, INT 10, SUB(MUL(X, X), INT 1))
```

and evaluating it should give 375.