CpSc 418 **PIKA 2** Due: Jan. 12, 2018, 1:00pm

34 points.

Please submit your solution using the handin program. Submit your solution as
cs418 pika2
Your submission should consist of one file:

- `pika2.erl`: Erlang source code for your solution.

The template for `pika2.erl` is available at
`http://www.ugrad.cs.ubc.ca/~cs418/2017-2/pika/2/code.html`.
We are also providing `simple_reduce.erl` that provides a simpler interface for reduce operation than the one in wtree.erl. Using `simple_reduce` makes this PIKA simpler – we'll use the greater generality provided by wtree.erl in subsequent examples and assignments.

Please submit code that compiles without errors or warnings. If your code does not compile, you will get zero for this PIKA. If your code generates compiler warnings, we will take off points for that as well, but not as many as for code that doesn't compile successfully.

This PIKA consist of three parts:

`q0`: an example. You don't need to do anythig here. `q0` provides you with an example that you can modify to get your solutions to `q1` and `q2`.

`q1` (26 points): use reduce to find the largest element of a list.

`q2` (16 points): use reduce to find the number of "adjacent duplicates" in a list.

My intention is that `q1` should be "easy" and `q2` is provides more of a challenge. I'm viewing `q2` as going a bit beyond "reasonable. Therefore:

- This PIKA will be graded on a scale of 34 points. If you complete all questions correctly, you can score 42 points. You will get credit in your PIKA total for scores over 34.
- For the purpose of the waitlist, I will take the 70% threshold on the points available for `q1`. In other words, anyone scoring 18.2 or higher will be eligible to move from the waitlist into the course.

1. `q0`: Figure 1 shows a reduce tree. The list

   ```
   [9,97,46,46,76,67,23,23,66,87,17,0,0,0,0,9,81,51,27,27]
   ```

   is distributed across eight worker processes as shown. For this example, we use reduce to compute the sum of the elements in the list. Each vertex of the tree is labeled. The function `q0` in `pika2.erl` indicates the values that are computed at each vertex of the tree. Don't change these values. `q0` is an example (zero points), but we will deduct points if you modify it to give incorrect answers.

   In the figure, blue vertices indicate leaf-node operations, green-vertices indicate combine operations, and the magenta vertex indicates the final operation at the root. For computing the sum of the full list:

   - The leaf function computes the sum of the elements of its segment of the list. My implementation uses lists:sum.
   - The combine function simply adds the totals from its left and right subtrees to compute that total for the vertex.
   - The root function is the identity function – the total from the final combine is the sum of the full list.

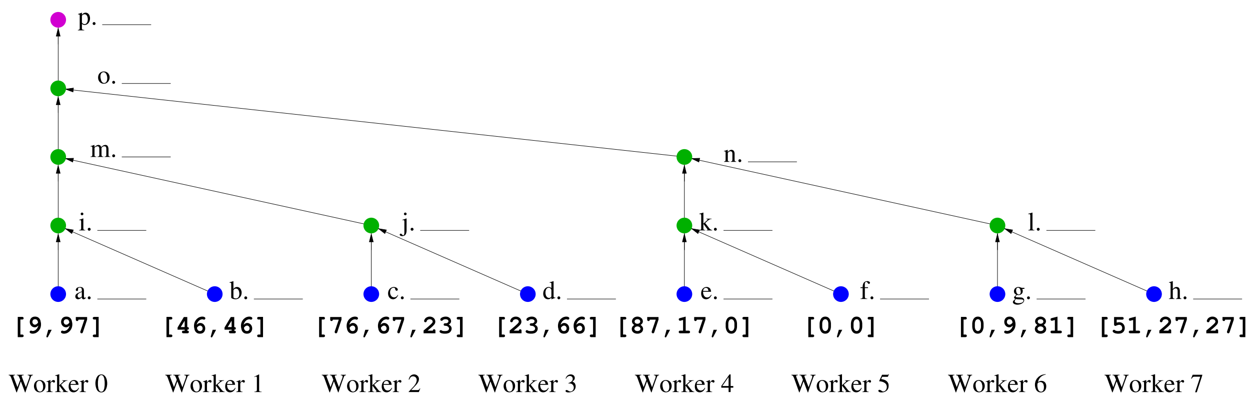

Figure 1: A reduce tree

The function `sum_test/0` runs the test case corresponding figure 1. If you want to try other tests, you can use the function `sum/1`. For example:

```
1>:  c(pika2).
{ok,pika2}
2> Data = [N*N || N <- lists:seq(0,100)].  % list of [0*0, 1*1, 2*2, ..., 100*100]
[0,1,4,9,16,25,36,49,64,81,100,121,144,169,196,225,256,289|...]
3> W = simple_reduce:create(4, Data).  % create 4 worker, distribute Data across them
[<0.73.0>,<0.74.0>,<0.75.0>,<0.76.0>]
4> pika2:sum(W).
338350
5> lists:sum(Data).  % just checking
338350 % Yay – They match!
6> simple_reduce:reap(W). % terminate the worker processes
```

That's all there is to `q0`. You don't need to do anything. Understanding this example should make solving `q1` and `q2` easier.

2. `q1` (26 points): Now, we want to find the largest element in the distributed list using reduce.
   (a) (16 points) Replace each `your_answer(...)` in the body of `q1` with the value that is computed at that node for the example shown in figure 1.
   (b) (4 points) Complete the implementations of the function `max_leaf`.
   (c) (4 points) Complete the implementations of the function `max_combine`.
   (d) (2 points) Complete the implementations of the function `max_root`.

   If you understand the corresponding functions from `q0`, then `max_leaf`, `max_combine`, and `max_root` should be easy to complete. In fact, `max_root` is the identity function, just like `sum_root`.

   The template file `pika2.erl` includes functions `max_test` and `maxr`. These are like `sum_test` and `sum`. You can use them to test your solution. We will test your code on lists other than just the one depicted in figure 1.

3. `q2` (16 points): In this problem, we count the number of "adjacent duplicates" in a list. Two elements of a list are an adjacent duplicate if they have the same value and they are consecutive elements of the

list. For example, the list `[42, 42, bananas]` has one adjacent duplicate, and the list `[42, bananas, 42]` has no adjacent duplicates. Here are a few more examples:

`[0,1,1,2,3,5,8]`: one adjacent duplicate.

`[0,1,0,0,0,1,1,0,1,0]`: three adjacent duplicates – note that `[0,0,0]` counts as *two* adjacent duplicates. Likewise,

```
[ marlin, monkey, monkey, monkey, monkey, millipede, manatee,
  manatee, mongoose, millipede]
```

has four adjacent duplicates.

This problem is more challenging than `q1` and `q2` because we need to include information in addition to the adjacent duplicate count in the summaries for leaves and subtrees. In particular, we need to know the first and last element of the segment for each leaf of subtree so we can determine whether the last element of one segment matches the first element of the next segment. For our summary, we use a segment of the form:

```
{FirstElement, DuplicateCount, LastElement}
```

This pattern of needing to record or summarize what is at each end of a segment is a common one.

(a) (11 points) Replace each `your_answer(...)` in the body of `q2` with the value that is computed at that node for the example shown in figure 1. Note that template already has completed several of the entries to provide examples.

(b) (4 points) Complete the implementation of `dup_combine`.

(c) (1 point) Complete the implementation of `dup_root`.