15-150 Spring 2012
Lab 15

May 2, 2012

1 Introduction

1.1 Getting Started

Update your clone of the `git` repository to get the files for this weeks lab as usual by running

```
git pull
```

from the top level directory (probably named `15150`).

2 Google MapReduce™

In this problem, you will define Google MapReduce, which is a useful higher-order function that computes a dictionary from a sequence. Google MapReduce is inspired by the `map` and `reduce` functions on sequences that we have studied but, as you will see, it's not just `Seq.mapreduce`.

Google MapReduce is defined by the following signature:

```
signature GOOGLE_MAPREDUCE =
sig
    structure D : DICT
    val gmapred : ('a -> (D.Key.t * 'v) Seq.seq)
                -> ('v * 'v -> 'v)
                -> 'a Seq.seq
                -> 'v D.dict
end
```

The module `D` specifies an ordered type `D.Key.t` of keys, as well as a dictionary on such keys.

Think of `s` as a sequence of documents. `gmapred extract combine s` takes

1. a function `extract` that extracts key-value pairs from each document (the keys in the sequence it returns need not be unique) and

2. a function `combine` that combines two values into a single value.

`gmapred extract combine s` extracts key-value pairs from each document in `s`, and then builds a dictionary mapping each key to the value produced by applying `combine` pairwise to the values extracted with that key. `combine` is assumed to be associative, so that the order in which it is applied to values does not matter.

For example, if `MR : GOOGLE_MAPREDUCE` where `D.Key.t` is `string`, then we can count how many times each word occurs in a collection of documents:

```
val words : string -> string Seq.seq
fun wordCounts (d : string Seq.seq) : int MR.D.dict =
   MR.gmapred (fn s => Seq.map (fn w => (w, 1)) (words s))
              (fn (x,y) => x + y)
              d
```

Specifically,

```
     wordCounts <"this is is document 1", "this is document 2">
==   ("1" ~ 1,"2" ~ 1,"document" ~ 2,"is" ~ 3,"this" ~ 2)
```

This is because the `extract` function pairs each word in the document with `1`:

```
     Seq.map (fn w => (w, 1)) (words "this is is document 1")
==   <("this",1),("is",1),("is",1),("document",1),("1",1)>
```

and the `combine` function sums the counts.

You will implement the following functor:

```
functor GoogleMapReduce (Key : ORDERED) : GOOGLE_MAPREDUCE =
struct
  structure D = TreeDict(Key)
  fun gmapred extract combine s = ...
end
```

Task 2.1 Implement a helper function

```
val collect : ('v * 'v -> 'v) -> (Key.t * 'v) Seq.seq -> 'v D.dict
```

that takes a sequence of key-value pairs, and produces a dictionary mapping each key in the sequence to the value resulting from applying `combine` pairwise to all of the values associated with it.

For example,

```
     collect (fn (x,y) => x + y) <("this",1),("is",1),("is",1)>
==>  ("this" ~ 1, "is" ~ 2)
```

Hint: see `datastructures.sig` for the `DICT` signature. This signature contains an operation `merge`:

```
(* merge combine (d1,d2) == d where
   - k in d if and only if k is in d1 or k is in d2
   - If k~v in d1 and k is not in d2, then k ~ v in d
   - If k~v in d2 and k is not in d1, then k ~ v in d
   - If k~v1 in d1 and k~v2 in d2, then k ~ combine (v1, v2) in d
   *)
val merge  : ('v * 'v -> 'v) -> 'v dict * 'v dict -> 'v dict
```

that will be helpful.

Task 2.2 Implement the function

```
val gmapred : ('a -> (D.Key.t * 'v) Seq.seq) -> ('v * 'v -> 'v)
            -> 'a Seq.seq -> 'v D.dict
```

as described above. **Try to make the span of your implementation as small as possible.** Hint: use `collect`. You may also wish to use `Seq.flatten`.

Task 2.3 Test `gmapred` by running the frequency counting code in `WordFreq`.

Have the TAs check your work before proceeding!

3 Anagrams

Suppose we have a sequence of strings:

```
<"Ethers are a class of organic compounds that contain an ether group...",
 "Elvis began his career there in 1954 when Sun Records owner Sam Phillips...",
 "hI thEres do you livEs at three main street?">
```

In this problem, you will write a function that computes all of the sets of words that are anagrams of each other. For the above strings, this computes

```
<<"Ethers","thEres">,
 <"ether","there","three">,
 <"Elvis","livEs">>
```

Note that:

- A "word" is a case-sensitive space-delimited sequence of characters. `SeqUtils.words` divides a string into a sequence of words.
- Each set contains no duplicates (e.g., `<"ether","there","three">` not `<"ether","there","three","three">`).
- 1-word anagram sets should be excluded (e.g. `<are>` doesn't appear).

Task 3.1 Write the function

```
anagrams : string Seq.seq -> (string Seq.seq) Seq.seq
```

Hint: `datastructures.sig` contains some data structures and algorithms that we have taked about in the course. If you use `gmapred` with the right notion of key and value, your code will be quite short.