Take Home Programming Test

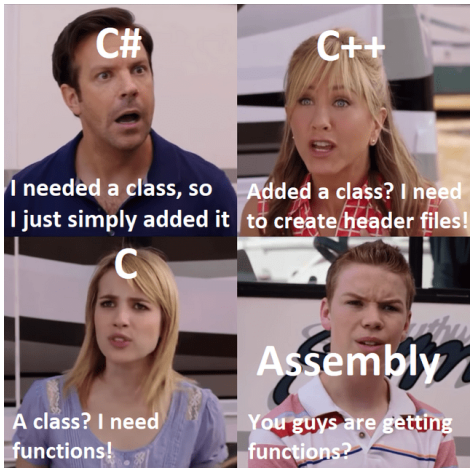
CSEE 3827: Fundamentals of Computer Systems
Prof. Martha Kim

Release Date: 11/15/22
Submission Date: 11/22/22

- All work is to be individual, with no collaboration of any sort permitted.
- You may pose questions to course staff via private post on ED. As on a test, staff will answer clarifying questions about the prompt, but *will not provide assistance writing or debugging code*.
- You may refer to all of this semester's course materials (i.e., slides, notes, problem sets and solutions, Ed Discussion board history, SPIM).
- You may also refer to external resources (e.g., slides from other courses or semesters, the Harris and Harris textbook, etc.). Such materials should serve as references and their content should never be presented as your own.
- Under no circumstances are you to solicit or consult solutions.
- As in PS4 and PS5, place all of your code above the line in the scaffolding that begins

```
#### Do not remove this separator.
```

- Your submitted .s file must contain your code followed by this separator.
- Your code will be tested for both correctness and adherence to calling conventions.
- The submission deadline is Tuesday November 22 at 11:59pm. Be sure to leave sufficient time to upload before that deadline.
- To submit, upload a single file named `mapreduce.s` to gradescope.



1 Introduction

For this assignment you are to implement a `mapreduce` function that uses the MapReduce approach to analyze a set of strings.

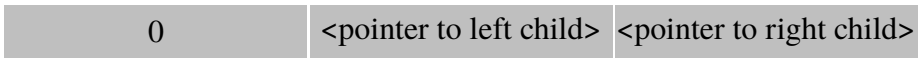
MapReduce is one way to organize a large computation to process large data sets. In a MapReduce computation a *map* function processes each element in the dataset to generate a set of intermediate values, one per element. A reduce function then reduces all of these intervalues to produce a single result. For example, to get the total wordcount on many documents, a *map* function would apply wordcount to each individual document, and then the *reducer* would sum the results.

Many real world tasks are expressible in this structure, and MapReduce is deployed in many commercial settings. One major benefit is that a user needs only write their map and reduce function, to be plugged into this framework and deployed at scale to process large datasets. For more background, Wikipedia's page is a reasonable introduction: https://en.wikipedia.org/wiki/MapReduce.

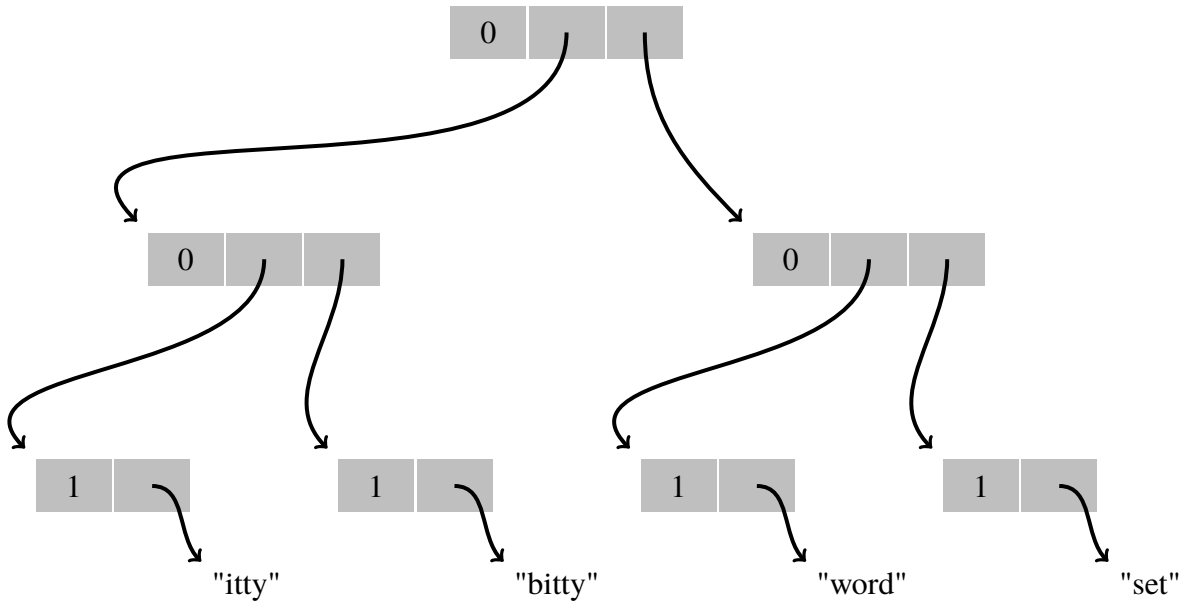
2 Data Format in Memory

The strings to be processed are stored in a full binary tree. In this tree, each leaf contains a pointer to a string. Every non-leaf node of the tree contains pointers to two children.
Leaf nodes take up two contiguous words of memory:

Non-leaf nodes use three contiguous words:



To help you visualize this data structure, the following diagram depicts one of the test trees in the scaffolding file, called `tiny`. As above, each gray cell represents one word of memory.



3 Provided Map and Reduce Functions

In the scaffolding you will find a selection of map and reduce functions. For the purpose of this project, all of the map functions take a pointer to a null-terminated ASCII string and return an integer. The scaffolding contains two such functions:

- `unit` always returns 1, regardless of the argument
- `strlen` returns the length of the argument string

As for reduce functions, all of ours take two integer arguments and return a single integer. The scaffolding contains three:

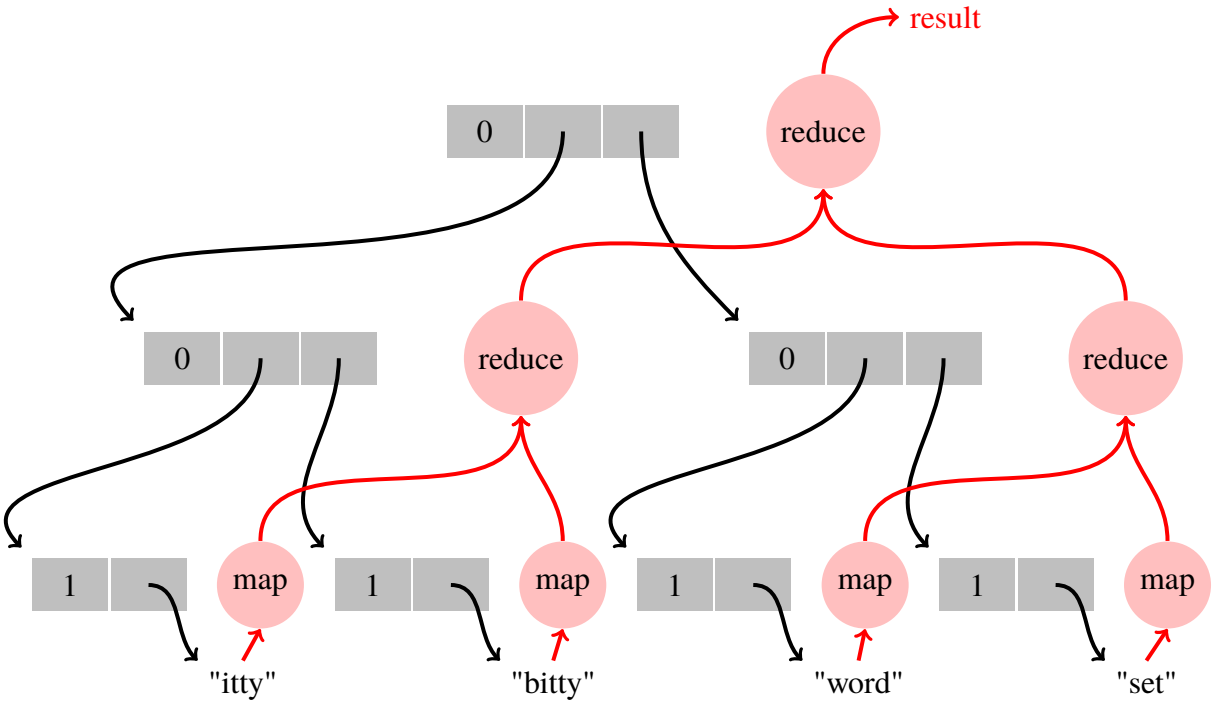
- `sum` returns the sum of the two arguments
- `min` returns the smaller of the two arguments
- `max` returns the larger of the two arguments

4 Your `mapreduce` Function

Your assignment is to implement a function, called `mapreduce`, that takes three arguments:

- A pointer to the root of the binary tree containing the strings to be processed. This pointer will never be null.
- A pointer to a map function
- A pointer to a reduce function

`mapreduce` should return an integer whose value is the result of applying the map function and then the reduce function to the given tree. The information flow between map and reduce functions on the `tiny` tree is illustrated in red here:



As always, your `mapreduce` function should adhere to MIPS calling conventions. Further, it *should not* modify the contents of the tree.

5 Calling Map and Reduce Functions

Since you are given pointers to the map and reduce functions, you will need to call them by pointer. So far we have been calling functions using their label:

```
jal myfunction
```

But when you have a pointer to a function, and not its label name, you can call it with `jalr` ("jump and link register") on whatever register holds the pointer to the function. For example, if a function pointer were in `$t0` you would call it with:

```
jalr $t0
```

All of the arguments and return values are handled in the normal fashion.

6 Scaffolding

As mentioned above, there are two mappers (`unit` and `strlen`) and three reducers (`sum`, `min`, and `max`) provided in the scaffolding. You will also find two test trees under `.data`. `tiny` is the tree depicted above with four leaves. `lorem` is a larger tree that has 64 leaves.

In the provided `main`, you will find twelve invocations of `mapreduce`, calling all six mapper/reducer pairs on each of the two test trees.

7 Advice and Hints

- You will almost certainly prefer a recursive implemention
- Helper functions are allowed if you wish, but they are probably not necessary; if you write them, they should also adhere to conventions
- We strongly advising building your code and testing incrementally, for example
  - Comment out all but one mapreduce invocation from main.
  - Consider testing mapreduce on subtrees if you want something even smaller (e.g., tinyRR is a leaf node of tiny, tinyL is a tree with two leaves).
  - Implement the tree traversal only to start,
  - then actually invoke a mapper,
  - then invoke a reducer.
- You may assume the reduce function will always be commutative, and therefore can be applied in any order.