

Homework 2: Distributed Programming with MapReduce

Name Jinsu Lim
Division School of Computing
Student number 20223558

1 Implementation (75 Points)

1.1 Subtask 1: Shingling (25 points)

SubTask1 Mapper:

```
def ngrams(words, n=3):
    return [tuple(words[i: i + n]) for i in range(len(words) - n + 1)]

def _map(key, value):
    for title in value:
        words = list(filter(lambda e: len(e), title.split(" "))) # To remove blank token
        for ngram in ngrams(words):
            _write(key, ",".join(ngram)) # directly print for set?
```

SubTask1 Reducer:

```
def _reduce(key, values):
    for value in values:
        _write(key, value[0]) # If # of value is over 1, only the first one is used.
```

Figure 1: Subtask1 Mapper & Reducer

1.2 Subtask 2: Min-hashing (25 points)

SubTask2 Mapper:

```
s = 10

def hashFunction(func_id, first_word, second_word, third_word):
    # hashFunction(2, 'i', 'have', 'a') -> 2nd hash function value of 3-shingle ('i', 'have', 'a')
    # 1 <= func_id <= 10
    def wordHash(s):
        ret = 0
        for c in s[:-1]:
            ret = ((ret * 31) + (ord(c) - ord('a')) + 1) % 1234567891
        return ret
    return ((41 ** func_id) * wordHash(first_word) * (2 ** 64) + (506 ** func_id) * wordHash(second_word) * (
        2 ** 32) + wordHash(third_word)) % 1234567891

def _write(key, value):
    sys.stdout.write(f"{key}\t{value}\n")

# paper_id, 3-shingle
def _map(key, value):
    # _write(key, "\t".join(value)) # example
    first, second, third = value[0].split(",")
    for i in range(1, s + 1):
        hashed_value = hashFunction(i, first, second, third)

        # key : paper_id, vale : hashidx-hashed_value
        _write(key, ",".join([str(i), str(hashed_value)]))
    # signature = tuple([hashFunction(i, first, second, third) for i in range(1, s + 1)])
```

SubTask2 Reducer:

```
# key : paper_id, vale : hashidx-hashed_value
def _reduce(key, values):
    collector = defaultdict(list)

    for value in values:
        for vv in value:
            tkn = vv.split(",")
            hash_idx, hashed_value = int(tkn[0]), int(tkn[1])
            collector[hash_idx].append(hashed_value)

    _write(key, ",".join([str(min(collector[i])) for i in range(1, 11)]))
```

Figure 2: Subtask2 Mapper & Reducer

1.3 Subtask 3: Bucketing (25 points)

SubTask3 Mapper:

```
b = 5
r = 2

def _write(key, value):
    sys.stdout.write(f"{key}\t{value}\n")

# key = each bands vector, value = paper id
def _map(key, value):
    for vv in value:
        signatures = vv.split(",")
        for i in range(b):
            pair_key = [signatures[i * r + j] for j in range(r)]
            _write(".".join(pair_key), key)
```

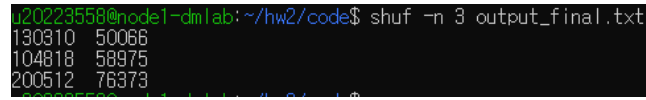
SubTask3 Reducer:

```
# key = " ".join(each bands vector), value = paper id
def _reduce(key, values):
    if len(values) > 1:
        docs = sorted([doc_id[0] for doc_id in values])
        for paper1, paper2 in itertools.combinations(docs, 2):
            _write(paper1, paper2)
```

Figure 3: Subtask3 Mapper & Reducer

2 Analysis (25 points)

2.1 Q1: Provide any three candidate pairs with their titles.



```
u20223558@node1-dmlab:~/hw2/code$ shuf -n 3 output_final.txt
130310 50066
104818 58975
200512 76373
```

Figure 4: random select from output_final.txt using shuf

	Doc ID	Raw Doc Title
Pair 1)	130310	Low-Rank Training of Deep Neural Networks for Emerging Memory Technology
	50066	Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning
Pair 2)	104818	Stochastic Optimization of Plain Convolutional Neural Networks with Simple methods
	58975	3D Convolutional Neural Networks for Classification of Functional Connectomes
Pair 3)	200512	Precipitaion Nowcasting using Deep Neural Network
	76373	Uncertainty Propagation in Deep Neural Network Using Active Subspace

Table 1: random selected result from output_final.txt

2.2 Q2: Describe your design of map/reduce functions for solving each subtask.

2.2.1 SubTask 1 : Shingling

In Figure 1, you can see the mapper and reducer code.

Mapper: For each line in the input, the mappers split the line into document id and title by splitting it with tab separator, and then generate 3-grams for title.
For each 3-gram, emit **<document id, 3-gram >**.

Reducer: Emit the result of the mapper as is. emit **<document id, 3-gram >**..

2.2.2 SubTask 2 : Min-hashing

In Figure 2, you can see the mapper and reducer code.

Mapper: For each 3-gram in each line, the mapper computes the hash function $h(3\text{-gram}, i)$. It uses i as the index of the order of the hash function, from 1 to 10.
Then it outputs i and the hashed value, which is the output of the i -th hash function, together.
Emit in the form **<doc.id, (i, hashed.value) >**.

Reducer: For each doc_id, the reducers take a list of (index, hashed.value) pairs.
Aggregate the hashed values based on the index of the pair and output the min value from the each i -th hashed value list L_i , $1 \leq i \leq 10$.
Emit the list of minimum values for each i -th hashed values as a signature, **<doc.id, ",".join([min(L_i) for i in range(1, 11)]) >**

2.2.3 SubTask 3 : Bucketing

In Figure 3, you can see the mapper and reducer code.

Mapper: In each line, split the signature of doc_id into sub-signatures with band size b. Then emit the sub-signature as a new key and doc_id as value. `<sub-signature, doc_id >`

Reducer: For each the sub-signature, the reducer takes a list of matched documents. It then emits a combination of document pairs from the matched documents.

Emit `<doc_A ID, doc_B ID >` from matched document list.

In the code, I use the sorted() method to print the sorted document id pair.