

q1

- a. a combiner class and a reducer class can be used interchangeably when the problems are commutative and associative.

$\max(0, 20, 10, 25, 15) = \max(\max(0, 20, 10), \max(25, 15)) = \max(20, 25) = 25$

Commutative:  $\max(a, b) = \max(b, a)$

Associative:  $\max(\max(a, b), c) = \max(a, \max(b, c))$

- b. it is lazy evaluated. the data is not available or transformed until an action is executed that triggers the execution. reduce is action function

q2

a.

class Mapper

```
method map(self, _, _):  
    emit("departmentID,employeeID", "salary")
```

class Reducer

```
method reduce_init(self):  
    current_departmentID = ""  
    current_dic = {}  
method reduce(key, value):  
    departmentID, employeeID = key.split(",")  
    if departmentID != current_departmentID:  
        if current_departmentID != "":  
            maximum_salary = max(current_dic.values())  
            for key1, value1 in current_dic.item():  
                if (value1 == maximum_salary):  
                    emit(current_departmentID, key1)  
            current_dic = {}  
        current_dic[employeeID] = value  
        current_departmentID = departmentID  
    else:  
        current_dic[employeeID] = value  
method reduce_final(self):  
    maximum_salary = max(current_dic.values())  
    for key1, value1 in current_dic.item():  
        if (value1 == maximum_salary):  
            emit(current_departmentID, key1)
```

in JOBCONF, configure:

```
'mapreduce.map.output.key.field.separator': ',',  
'mapreduce.partition.keypartitioner.options': '-k1,1',  
'mapreduce.partition.keycomparator.options': '-k1,1'
```

b.

```
class Mapper
```

```
    method map(self, key, list_vertex):
```

```
        for item in list_vertex:
```

```
            if key < item:
```

```
                emit("key,item", "")
```

```
            else:
```

```
                emit("item,key", "")
```

```
class Reducer
```

```
    method reduce_init(self):
```

```
        key_dic = {}
```

```
    method reduce(key, value):
```

```
        vertex1, vertex2 = key.split(",")
```

```
        if key not in key_dic:
```

```
            key_dic[key] = []
```

```
        for key1,value1 in key_dic:
```

```
            if key1.split(",")[0] == vertex1 or key1.split(",")[1] == vertex1:
```

```
                value1.append(key)
```

```
            else if key1.split(",")[0] == vertex2 or key1.split(",")[1] == vertex2:
```

```
                value1.append(key)
```

```
    method reduce_final(self):
```

```
        for key1,value1 in key_dic:
```

```
            value1 = list(set(value1))
```

```
        sorted(key_dic)
```

```
        for key1,value1 in key_dic:
```

```
            print("key1: " + " , ".join(value1))
```

q3

a.

see problemA.py

b.

see problem.py

q4

a. set of 2-shingles:

$S(A) = ['th', 'he', 'e', 's', 'sk', 'ky', 'y', 'i', 'is', 's', 'd', 'da', 'ar', 'rk', 'k', 't', 'th', 'he', 'e', 'm', 'mo', 'oo', 'on', 'n', 'i', 'is', 's', 'b', 'br', 'ri', 'ig', 'gh', 'ht']$

$S(B) = ['th', 'he', 'e', 'm', 'mo', 'oo', 'on', 'n', 'i', 'in', 'n', 't', 'th', 'he', 'e', 's', 'sk', 'ky', 'y', 'i', 'is', 's', 'b', 'br', 'ri', 'ig', 'gh', 'ht']$

number of intersecting element = 22

jaccard similarity = 0.79

Row	$C_1$	$C_2$	$5n+2 \bmod 7$	$3n+1 \bmod 7$
0	0	1	2	1
1	1	0	0	4
2	0	1	5	0
3	0	0	3	3
4	1	1	1	6
5	1	1	6	2
6	1	0	4	5

  

	$C_1$	$C_2$
$h_1$	0	1
$h_2$	2	0

b.

c.  $1 - (1 - 0.6^2)^5$

q5

a.

(i)

0	1	2	3	4	5	6
0	1	0	1	1	0	1

(ii)

"sql" is contained

b.

	0	1	2	3	4
H0	1	1	1	3	0
H1	0	1	3	1	1
H2	0	1	1	4	0

use the built CM-sketch to get the count for word "data": 3

q6