# COMP9313 Report

By z5244467(Chen Wu)

Ideas

First create a new list of candidates, save the candidate ID into this list, and determine the parameter βn to determine how many elements are in the list of candidates. Then use query\_hashes to traverse from the first ID in data\_hashes, for example, when offset=0, αm=10, if query\_hashes and data\_hashes meet the conditions, the current ID will be added to the list of candidates, otherwise it will not meet the conditions Will continue to execute. When the traversal is over, compare the candidate list length with the parameter βn, if the list length is less than the parameter βn, then relax the condition, offset+=1, continue to traverse data\_hashes

data\_hashes

|  |  |  |
| --- | --- | --- |
| ID | HashCode | |
| 0 | [1,2,3,4,5,…,32] | |
| 1 | [2,1,34,43,123,…,32] | |
| 2 | | [1,2,3,4,5,6,2,3,…,32] |
| .. | | .. |

Query\_hashes

|  |
| --- |
| [1,2,3,4,5,…,32] |

Offset = 0

αm=10

Look up

candidates

|  |
| --- |
| [0,2] |

If match,

add to list

compare

βn

<βn, offset+=1

data\_hashes

|  |  |  |
| --- | --- | --- |
| ID | HashCode | |
| 0 | [1,2,3,4,5,…,32] | |
| 1 | [2,1,34,43,123,…,32] | |
| 2 | | [1,2,3,4,5,6,2,3,…,32] |
| .. | | .. |

candidates

|  |
| --- |
| [0,2] |

C2lsh()

RDD

Query\_hashes

|  |
| --- |
| [1,2,3,4,5,…,32] |

Python list

transformation

RDD

## Details

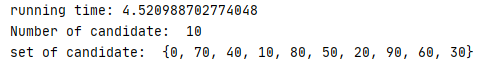
The diff function in my submission is to traverse data\_hashCode and query\_hashCode, (the two digits are the same), compare their sizes and return the difference. After traversing using the map function, the data\_hashes is obtained after processing. Enter the while loop. When the elements in the Candidates list are less than beta\_n, offset+=1, continue to traverse the data\_hashes. Use the mapPartitions function to traverse the data\_hashess,and use the flatmap function to get the final result. The data\_hases are flattened and stored in RDDcandidates. The number of eligible elements in data\_hashes is stored in the lists of candidatesNum until the end of the loop. In the Ifmatched function, compare the difference of the matched digits with the offset. When differenc meets the requirements of the offset, then compare with the alpha\_m. Whether it meets the alpha\_m, then use Boolean to return.

## Evaluation result

Env: win10 cpu: i7-7700

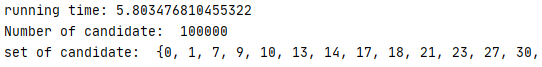
Testcase1: toy\_hashed\_data and toy\_hashed\_query

Result:



Testcase2: 20w data 32bit hashCode and 5w data having the same hash with query and other 5w data is matched with query with offset = 10

Result:



Testcase3: use 100w data 32bit hashCode and 5w data having the same hash with query and other 5w data is matched with query with offset = 10

Result:



## Improvement of the Efficiency

### Effective

1. First, directly implement the result according to the pseudo code in the handout, and compare the difference between dathashCode and queryhashCode each time with the offset, and relax the conditions to see if it reaches alpha\_m. After analyzing the algorithm, it is found that this difference is fixed every time, so it can be directly stored in the memory, so that there is no need to calculate each time the code is run.

2. Change the map function to mapPartitions function. As the name suggests, the data is partitioned and the code is slightly optimized, but the effect is not good. Analyze the code and consider that after the first run is completed, some data results have met the offset, then there is no need to continue calculations in the subsequent. So add flag to the data that has been judged to meet the conditions before to iterate.

3. Considering that the offset will take a very large value, so each time the offset is not increased by 1, but consider adding a larger number to reduce the number of runs or use binary search to improve efficiency

### Invalid (only refers to this machine)

1. Use cache function to store data\_hashes in memory each time and run it. The result is not ideal, taking up memory space leads to slower efficiency

2. Use groupbykey which fail to reduce efficiency.

3. Use multiple mapPartitions functions to replace the map functions, which fail to reduce efficiency