**Implementation detail of c2lsh():**

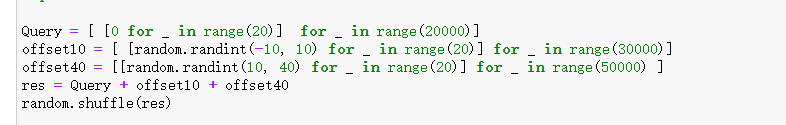
I have defined a new function count\_match(), I input three variables to count\_match(), the first variable is the value list in a pair of Rdd, the second is the query\_hashes, and the third is the offset which is the hash value of each bit allowable difference range. I need to compare each bit in the data\_hashes with each bit in the query\_hashes, if their difference is within the allowable range, the counter to accumulate. That function returns the value of counter.

In the c2lsh() function, I only need a function of rdd transformation which is flagMap(), in flatMap() I need to set a lambda x, x[1] is the data value list, input x[1], query hashes, offset into count() function, the value returned by count\_match() compare with alpha\_m(the minimum number of collide hash values between data and query). The if (count\_match(x[1],query\_hashes,offset) >= alpha\_m like a filter(), I will get the rdd keys that satisfy the condition else will get []. Then I will need the function of action that is rdd.count(), the result of rdd.count() is assigned to numcandidate. Until the number of candidate greater or equal than beta\_n(the minimum number of candidates to be returned) end the loop, return the Rdd.

**Evaluation result of own test case:**

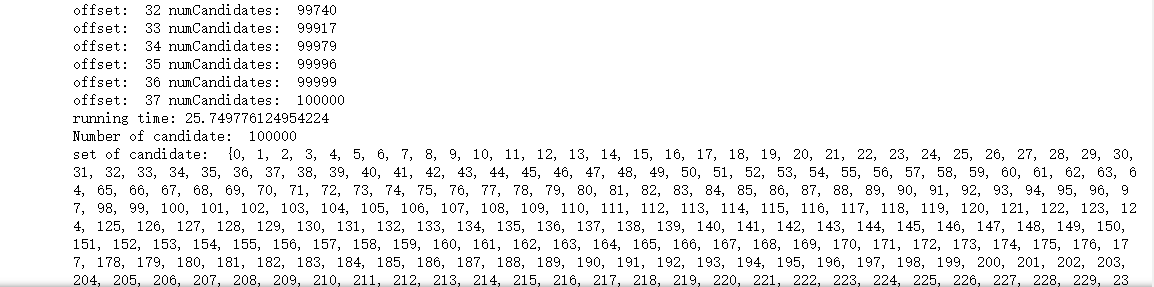
I generated two test files.

**For test 1:**

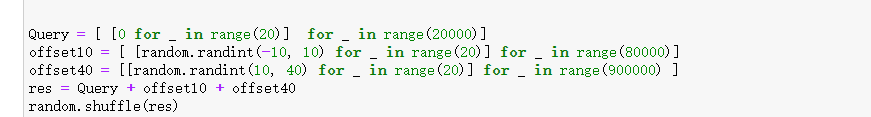


I generated a 20-bit query\_hashes dataset where each bit is 0. The data\_hashes dataset has 20,000 data same to query\_hashes, 30000 random numbers between -10 and 10, and 50000 random numbers between 10 and 40. I set the alpha\_m = 10 and beta\_n = 100000. In order to verify the correctness of my algorithm, I need to print out the numcandidates corresponding to each offset. When offset = 0 should be get numcandidates = 20000,then I will get numcandidates = 50000 when offset <= 10,finally I can get the result for numcandidates = 100000 when offset <= 40,because the data\_hashes max value is 40. The result show that:





**For test 2:**



The query\_hashes is same to test1 query\_hashes. The data\_hashes dataset has 20,000 data same to query\_hashes, 80000 random numbers between -10 and 10, and 900000 random numbers between 10 and 40. This test is to compare whether the final version of the c2lsh() algorithm is more efficient than the previous version when the data set is relatively large. The result show that:



**Improve the efficiency:**

Version 1:

I needed to use three transformation functions and one action function. mapValues(), filter(), map(), count(). Firstly, I need to use mapValues() to calculate the number of collisions between data\_hashes list and query\_hashes list corresponding to each key in Rdd (tmp=data\_hashes.mapValues(lambda x: count\_match(x,query\_hashes,offset))). Secondly, tmp=tmp.filter(lambda x: x[1] >= alpha\_m) will get rdd pairs with the number of collisions greater than or equal to alpha m. thirdly, tmp.count() compare to beta\_n, if less than beta\_n, offset will increase 1. Until the tmp.count() greater than or equal beta\_n. Finally, tmp.map(lambda x: x[0]) or tmp.keys() will get the rdd keys.

Version 2:

I combine the filter() and mapValues() to only filter(). Like data\_hashes.filter(lambda x: count\_match(x[1],query\_hashes,offset) >= alpha\_m). That will decrease 1 transformation function.

Version 3:

I found that Mappartitions() can replace Map(), Mappartitions() is more efficient than map(). But Mappartitions() in a larger data set may cause memory overflow. In order to ensure that there is no memory overflow, the version of Mapparttions() is finally abandoned.

Version 4:

The final version is to merge Map() and filter() into a flatMap(). Therefore, I only use 1 transformation and 1 action in c2lsh().