Implementation detail of c2lsh():

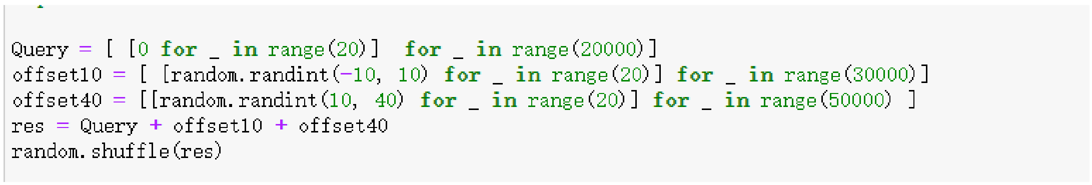
I have defined a new function count_match(), I input three variables to count_match(), the first variable is the value list in a pair of Rdd, the second is the query_hashes, and the third is the offset which is the hash value of each bit allowable difference range. I need to compare each bit in the data_hashes 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:

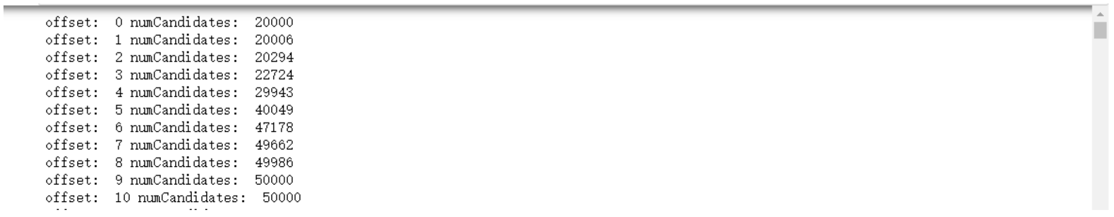
I generated two test files.

For test 1:

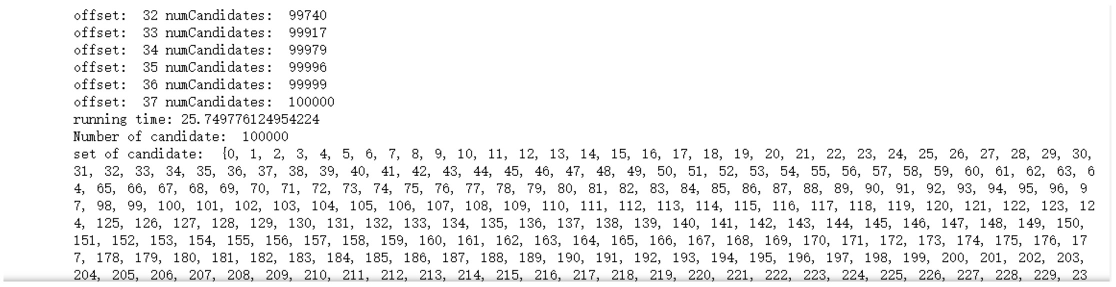


```
Query = [ [0 for _ in range(20)]  for _ in range(20000)]
offset10 = [ [random.randint(-10, 10) for _ in range(20)] for _ in range(30000)]
offset40 = [[random.randint(10, 40) for _ in range(20)] for _ in range(50000) ]
res = Query + offset10 + offset40
random.shuffle(res)
```

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:



```
offset:  0 numCandidates:  20000
offset:  1 numCandidates:  20006
offset:  2 numCandidates:  20294
offset:  3 numCandidates:  22724
offset:  4 numCandidates:  29943
offset:  5 numCandidates:  40049
offset:  6 numCandidates:  47178
offset:  7 numCandidates:  49662
offset:  8 numCandidates:  49986
offset:  9 numCandidates:  50000
offset:  10 numCandidates:  50000
```

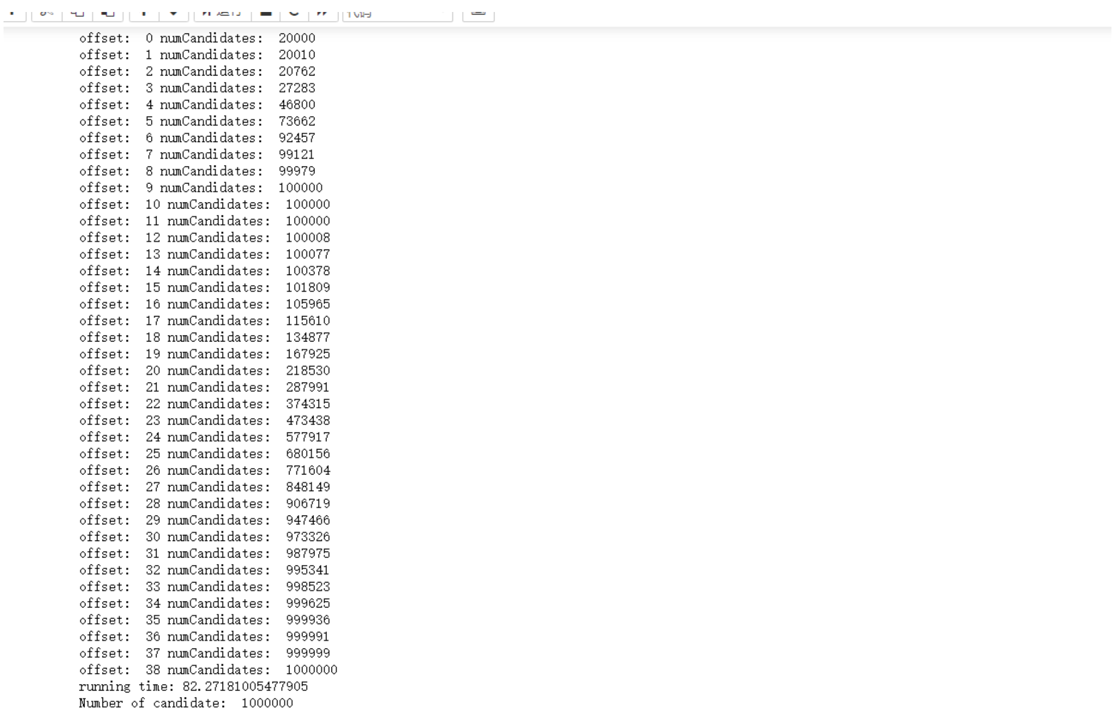


```
offset:  32 numCandidates:  99740
offset:  33 numCandidates:  99917
offset:  34 numCandidates:  99979
offset:  35 numCandidates:  99996
offset:  36 numCandidates:  99999
offset:  37 numCandidates:  100000
running time: 25.749776124954224
Number of candidate:  100000
set of candidate:  {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30,
31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 6
4, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 9
7, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 12
4, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150,
151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 17
7, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203,
204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 23
```

For test 2:

```
Query = [ [0 for _ in range(20)]  for _ in range(20000)]
offset10 = [ [random.randint(-10, 10) for _ in range(20)] for _ in range(80000)]
offset40 = [[random.randint(10, 40) for _ in range(20)] for _ in range(900000) ]
res = Query + offset10 + offset40
random.shuffle(res)
```

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:



```
offset:  0 numCandidates:  20000
offset:  1 numCandidates:  20010
offset:  2 numCandidates:  20762
offset:  3 numCandidates:  27283
offset:  4 numCandidates:  46800
offset:  5 numCandidates:  73662
offset:  6 numCandidates:  92457
offset:  7 numCandidates:  99121
offset:  8 numCandidates:  99979
offset:  9 numCandidates:  100000
offset:  10 numCandidates:  100000
offset:  11 numCandidates:  100000
offset:  12 numCandidates:  100008
offset:  13 numCandidates:  100077
offset:  14 numCandidates:  100378
offset:  15 numCandidates:  101809
offset:  16 numCandidates:  105965
offset:  17 numCandidates:  115610
offset:  18 numCandidates:  134877
offset:  19 numCandidates:  167925
offset:  20 numCandidates:  218530
offset:  21 numCandidates:  287991
offset:  22 numCandidates:  374315
offset:  23 numCandidates:  473438
offset:  24 numCandidates:  577917
offset:  25 numCandidates:  680156
offset:  26 numCandidates:  771604
offset:  27 numCandidates:  848149
offset:  28 numCandidates:  906719
offset:  29 numCandidates:  947466
offset:  30 numCandidates:  973326
offset:  31 numCandidates:  987975
offset:  32 numCandidates:  995341
offset:  33 numCandidates:  998523
offset:  34 numCandidates:  999625
offset:  35 numCandidates:  999936
offset:  36 numCandidates:  999991
offset:  37 numCandidates:  999999
offset:  38 numCandidates:  1000000
running time: 82.27181005477905
Number of candidate:  1000000
```

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().