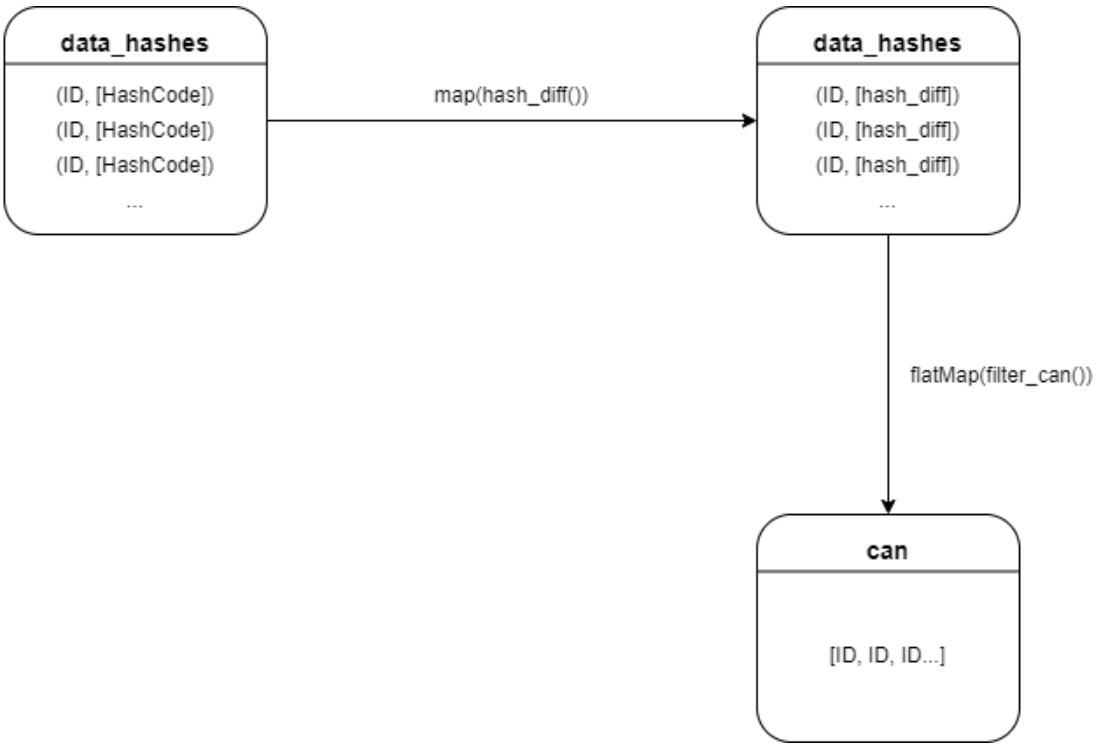
Project 1 Report

z5242692 Chenqu Zhao

1. **Implementation details of your c2lsh(). Explain how your major transform function works.**



The major transformation function is shown as the figure above.

The function hash_diff() is used to calculate the absolute differences of hash codes between data and query. The function is_candidate() checks if the hash_difference list scores enough collisions. The function filter_can() invokes is_candidate() to reformat the id in a python list if x is a candidate.

First, I map the value of data_hashes from list of hash code to list of value $abs\big(h(o) - h(q)\big)$ for each digit i. Then, I filter the candidates whose collisions are no less than alpha_m and flatMap them into an RDD of a single list.

If the count of candidates does not meet the requirement, change the value of offset and repeat the second step. Do this loop until the candidate number is equal to beta_n or find the minimum offset whose candidates number meets the requirement and is greater than beta_n.

2. **Show the evaluation result of your implementation using your own test cases.**
   The following test cases are acquired from the piazza forum:

   https://piazza.com/class/kamb1tqxe9h6np?cid=149

   https://piazza.com/class/kamb1tqxe9h6np?cid=340

   Many thanks to Rittisak Kwanda and Prof Sun! The test machine is my own laptop, Surface Laptop (1st Gen) with Intel i5-7200U CPU @ 2.50GHz, Windows 10 Pro OS. The running time varies due to computer temperature, CPU usage rate, RAM usage rate and etc.

   Test A - Test 1 provided by Rittisak

```
alpha_m, beta_n = 10, 10
data2, query2 = generate(10, 20000, 0, 0, 1000)

data = data2
query_hashes = query2

data_hashes = sc.parallelize([(index, x) for index, x in enumerate(data)])
```

   Output before evaluation:

```
running time: 477.10937118530273
Number of candidate:  10
set of candidate:  {9536, 5825, 3270, 15177, 15817, 9261, 4124, 10478, 10545, 1628}
```

   Output after evaluation:

```
running time: 6.232710599899292
Number of candidate:  10
set of candidate:  {9536, 5825, 3270, 15177, 15817, 9261, 4124, 10478, 10545, 1628}
```

   Test B - Test 6 provided by Rittisak

```
alpha_m, beta_n = 10, 500
data6, query6 = generate( 13, 2_000, 100, -230_000, 50_000)

data = data6
query_hashes = query6

data_hashes = sc.parallelize([(index, x) for index, x in enumerate(data)])
```

   Output after evaluation:

```
running time: 8.449762105941772
Number of candidate:  500
set of candidate:  {1, 3, 5, 22, 27, 29, 32, 38, 40, 45, 47, 56, 59, 60, 62, 68, 69, 73, 74, 82, 90, 95, 97, 101, 102, 108,
118, 124, 126, 131, 132, 134, 137, 138, 146, 148, 149, 151, 153, 160, 162, 164, 176, 183, 185, 186, 192, 194, 203, 205, 207,
212, 213, 214, 216, 219, 220, 221, 226, 236, 248, 257, 258, 262, 266, 269, 274, 276, 278, 280, 281, 283, 293, 298, 304, 305,
307, 310, 316, 317, 322, 325, 327, 328, 330, 333, 337, 347, 348, 351, 352, 353, 357, 358, 361, 363, 374, 375, 378, 385, 391,
402, 406, 408, 414, 415, 416, 420, 421, 423, 424, 428, 430, 431, 432, 436, 441, 447, 450, 455, 458, 461, 464, 466, 473, 474,
477, 479, 480, 491, 497, 501, 512, 531, 532, 544, 546, 552, 554, 556, 558, 559, 563, 564, 566, 567, 569, 581, 585, 587, 590,
591, 600, 611, 615, 618, 623, 628, 629, 634, 635, 644, 647, 652, 657, 663, 668, 673, 677, 681, 683, 697, 698, 701, 702, 705,
707, 711, 714, 717, 728, 729, 744, 748, 752, 753, 756, 761, 770, 772, 774, 777, 781, 785, 788, 789, 790, 792, 797, 799, 802,
804, 806, 807, 808, 811, 822, 826, 829, 830, 836, 837, 839, 840, 841, 845, 850, 851, 856, 857, 861, 862, 869, 870, 872, 873,
890, 891, 892, 895, 902, 922, 931, 934, 940, 942, 943, 949, 951, 956, 957, 959, 960, 962, 968, 969, 970, 971, 973, 974, 976,
977, 979, 982, 988, 992, 996, 1004, 1011, 1022, 1023, 1025, 1026, 1039, 1041, 1043, 1047, 1053, 1059, 1060, 1061, 1072, 108
3, 1091, 1096, 1097, 1098, 1099, 1106, 1111, 1115, 1122, 1123, 1124, 1134, 1142, 1143, 1144, 1147, 1148, 1155, 1159, 1167, 1
174, 1182, 1185, 1191, 1192, 1197, 1198, 1204, 1206, 1207, 1210, 1212, 1222, 1224, 1227, 1238, 1243, 1244, 1247, 1248, 1250,
1251, 1255, 1262, 1263, 1267, 1272, 1275, 1278, 1283, 1289, 1290, 1296, 1300, 1301, 1304, 1310, 1317, 1318, 1325, 1326, 132
9, 1336, 1339, 1341, 1344, 1345, 1346, 1348, 1351, 1353, 1358, 1362, 1364, 1367, 1368, 1370, 1371, 1377, 1387, 1397, 1399, 1
400, 1406, 1407, 1410, 1422, 1426, 1442, 1444, 1448, 1451, 1453, 1456, 1461, 1464, 1474, 1482, 1488, 1499, 1500, 1513, 1517,
1522, 1523, 1524, 1525, 1526, 1530, 1538, 1548, 1557, 1559, 1561, 1562, 1563, 1564, 1565, 1566, 1567, 1573, 1574, 1575, 158
2, 1589, 1590, 1591, 1593, 1596, 1597, 1601, 1604, 1605, 1611, 1619, 1626, 1627, 1628, 1631, 1632, 1634, 1642, 1644, 1647, 1
655, 1656, 1660, 1662, 1664, 1673, 1676, 1683, 1689, 1693, 1696, 1697, 1698, 1705, 1708, 1712, 1726, 1728, 1730, 1735, 1745,
1750, 1753, 1768, 1772, 1775, 1782, 1792, 1796, 1799, 1801, 1803, 1807, 1808, 1818, 1820, 1821, 1827, 1842, 1843, 1844, 184
9, 1850, 1858, 1863, 1864, 1867, 1868, 1871, 1872, 1875, 1878, 1880, 1886, 1892, 1894, 1908, 1909, 1920, 1925, 1926, 1940, 1
941, 1943, 1945, 1948, 1950, 1951, 1952, 1953, 1956, 1958, 1959, 1963, 1968, 1971, 1972, 1974, 1981, 1983, 1985, 1990, 1998}
```

Test C – Test 7 provided by Rittisak

```
alpha_m, beta_n = 10, 500
data7, query7 = generate( 15, 70_000, 140, -500_000, 500_000)

data = data7
query_hashes = query7

data_hashes = sc.parallelize([(index, x) for index, x in enumerate(data)])
```

Output after evaluation:

```
running time: 16.485996961593628
Number of candidate:  500
set of candidate:  {63494, 26634, 55308, 43024, 53264, 57360, 18461, 43039, 32801, 34850, 38948, 53294, 18489, 67652, 38984,
43082, 8270, 12367, 63568, 55378, 36947, 67670, 55394, 10344, 28779, 14444, 116, 63604, 20600, 32890, 34939, 59514, 36995, 3
4949, 134, 69769, 34954, 47242, 69772, 41102, 69785, 2208, 41121, 53410, 32933, 2222, 65712, 43186, 10425, 30906, 65721, 104
32, 65733, 32966, 16598, 67798, 57560, 41177, 43230, 41187, 20709, 18664, 22762, 57581, 2289, 63739, 22784, 8456, 10507, 207
54, 43289, 47385, 20766, 51487, 16672, 55584, 22831, 31026, 39230, 39231, 53568, 18759, 53576, 59721, 49483, 332, 63821, 33
5, 53593, 6491, 14683, 20829, 39264, 22882, 2406, 67947, 69998, 372, 55674, 65923, 59781, 63877, 4489, 29065, 2452, 53658, 5
7757, 37278, 49573, 51625, 68010, 43438, 2486, 8631, 53691, 47548, 45506, 68034, 61894, 47559, 41416, 35289, 14815, 66016, 1
0725, 486, 57836, 12785, 25073, 27131, 55811, 27141, 41484, 45586, 64018, 31256, 68120, 31261, 37411, 6693, 21035, 14893, 67
02, 21040, 563, 566, 27190, 31304, 53835, 49742, 66128, 37459, 43609, 62046, 27233, 53857, 16996, 37483, 6764, 4745, 64139,
25228, 31372, 62097, 49810, 58004, 25237, 2716, 41632, 47781, 23207, 55976, 39596, 51885, 19122, 62132, 10944, 43720, 10953,
8906, 717, 25310, 8933, 49894, 45799, 2798, 19185, 39669, 35575, 66300, 33540, 49926, 27402, 2840, 793, 11035, 47899, 41761,
39716, 35657, 37711, 56152, 21338, 47968, 19302, 21353, 64361, 25453, 52082, 47988, 54149, 4998, 39816, 17289, 17293, 66455,
56218, 5020, 39850, 939, 11182, 13231, 58293, 39869, 11212, 64465, 43989, 3032, 21466, 58330, 58332, 15326, 31722, 13295, 17
393, 50162, 41971, 25590, 64506, 11271, 33799, 56327, 60435, 29717, 39957, 50203, 3105, 21547, 37933, 58424, 11323, 31804, 3
134, 50245, 44103, 64583, 27724, 13391, 29790, 58463, 64609, 3180, 11380, 19583, 23693, 15502, 64661, 27805, 31904, 60582, 6
0586, 7345, 13503, 50377, 40141, 27859, 52437, 68825, 54491, 60635, 34013, 3295, 38111, 48356, 54501, 21743, 40183, 11514, 2
9950, 3333, 64778, 25868, 36108, 5394, 66836, 3352, 11545, 62747, 52512, 5413, 27944, 44335, 54575, 5426, 56632, 1338, 1158
0, 15684, 50503, 48456, 1363, 21844, 15707, 40290, 66918, 40296, 64873, 48491, 34165, 5499, 50562, 34179, 3463, 9608, 32135,
1420, 25997, 28044, 30093, 58773, 1433, 13724, 52639, 46498, 60841, 17835, 26033, 56753, 11699, 34230, 23996, 7614, 17855, 1
7859, 7626, 36300, 26061, 52692, 26074, 56800, 69088, 52707, 48614, 5609, 48622, 38382, 38385, 1529, 1532, 28160, 46593, 117
80, 54788, 26118, 54797, 22032, 17941, 13853, 40477, 65059, 60965, 5671, 52776, 30260, 56886, 13885, 46653, 58941, 38465, 42
562, 69188, 50761, 3660, 48716, 30286, 17999, 11857, 40534, 18012, 52839, 22130, 18035, 9850, 28284, 40572, 48766, 16001, 63
112, 28301, 1682, 20116, 59042, 18089, 34474, 1708, 22195, 38579, 1722, 69309, 48832, 52932, 50890, 24270, 16080, 50900, 529
49, 38615, 20185, 67295, 9956, 34532, 26342, 63209, 67310, 44788, 32501, 59128, 65273, 12026, 24322, 7939, 48899, 32521, 202
38, 67346, 16153, 16159, 59175, 63273, 69417, 18230, 26424, 59199, 69439, 24388, 20294, 3911, 24391, 34636, 61261, 67405, 80
17, 5971, 28501, 40793, 14181, 67429, 22377, 69484, 26477, 12142, 57199, 61300, 32629, 16246, 51062, 22393, 69499, 10114, 38
788, 46982, 55174, 65414, 20366, 3983, 14243, 63405, 38837, 42934, 26559, 18368, 22468, 10182, 63439, 69586, 30676, 32724, 4
0920, 53210, 24539, 59358, 65508, 51177, 2026, 22508, 47087, 38897, 12275, 59385, 51199}
```

Test D – Extra test provided by Prof Sun

```
with open("hashed_data", "rb") as file:
    data = pickle.load(file)

with open("hashed_query", "rb") as file:
    query_hashes = pickle.load(file)
```

Output after Evaluation:

```
running time: 15.366596937179565
Number of candidate:  11
set of candidate:  {161, 34, 68, 139, 492, 461, 303, 401, 307, 248, 447}
```

3. **What did you do to improve the efficiency of your implementation?**

The major evaluation is the way to define the value of offset. Instead of adding it by one from 0 which is a disaster when handling tremendous dataset, I apply binary search. Since I have the $abs(h(o) - h(q))$ value for every data, I flatMap the data_hashes and find the max value max_diff among them. According to the definition of offset, it will not be greater than max_diff. Therefore, I can apply binary search in the range [0, max_diff] to define the value of offset. The expected time complexity is changed from $O(n)$ to $O(\log n)$. I print the details in each iteration of Test C mentioned in the previous section as below to prove.

```
data_ran: 998150
offset:  499075 num_can:  60747
offset:  249537 num_can:  4358
offset:  124768 num_can:  34
offset:  187152 num_can:  658
offset:  155960 num_can:  185
offset:  171556 num_can:  364
offset:  179354 num_can:  495
offset:  183253 num_can:  575
offset:  181303 num_can:  529
offset:  180328 num_can:  516
offset:  179841 num_can:  509
offset:  179597 num_can:  505
offset:  179475 num_can:  500
running time: 15.009795188903809
Number of candidate:  500
```

Also, in function is_candidate(), instead of comparing collision with alpha_m at last, let them do comparison immediately when the collision is added. This way, the function can return True immediately once collision == alpha_m without going through further.

In addition, in function hash_diff(), instead of traversing these two lists and append the absolute value, I switched to a map function which does the same thing but saves time complexity verified by several tests.