1. You have a bag containing 98 ordinary coins and two double-headed ones. Pick a coin at random, and toss it 11 times. Given that you see 11 heads, what is the probability that the next two tosses of that coin are also heads?

[Answer] We should strictly follow the Bayesian rule to solve it.

Let’s define N: event of choosing ordinary coins; DH: event of choosing double-headed coin. Then we could have:

1. Given two words (start and end), and a dictionary, find the length of shortest transformation sequence from start to end, such that:
2. Only one letter can be changed at a time, and it can only be changed to another letter with a distance less than ten (For example, the distance between ‘b’ and ‘i’ is 7)
3. Each intermediate word must exist in the dictionary

Example

Given:

start = "hit"

end = "cog"

dict = ["hot","dot","dog","lot","log"]

As one shortest transformation is "hit" -> "hot" -> "dot" -> "dog" -> "cog",

return its length 5.

[Answer]

1. # -\*- coding: utf-8 -\*-
2. """
3. Created on Sun Jan 27 16:27:24 2019
5. @author: Chandler
6. """
8. # To check if strings differ by
9. # exactly one character
11. **def** isadjacent(a, b):
12. count = 0
13. n = len(a)
15. # Iterate through all characters and return false
16. # if there are more than one mismatching characters
17. **for** i **in** range(n):
18. **if** a[i] != b[i]:
19. count += 1
20. **if** count > 1:
21. **break**
23. **return** True **if** count == 1 **else** False
25. # A queue item to store word and minimum chain length
26. # to reach the word.
27. **class** QItem():
29. **def** \_\_init\_\_(self, word, len):
30. self.word = word
31. self.len = len
33. # Returns length of shortest chain to reach
34. # 'target' from 'start' using minimum number
35. # of adjacent moves.  D is dictionary
36. **def** shortestChainLen(start, target, D):
38. # Create a queue for BFS and insert
39. # 'start' as source vertex
40. Q = []
41. item = QItem(start, 1)
42. Q.append(item)
44. **while**( len(Q) > 0):
46. curr = Q.pop()
48. # Go through all words of dictionary
49. **for** it **in** D:
51. # Process a dictionary word if it is
52. # adjacent to current word (or vertex) of BFS
53. temp = it
54. **if** isadjacent(curr.word, temp) == True:
56. # Add the dictionary word to Q
57. item.word = temp
58. item.len  = curr.len + 1
59. Q.append(item)
61. # Remove from dictionary so that this
62. # word is not processed again.  This is
63. # like marking visited
64. D.remove(temp)
66. # If we reached target
67. **if** temp == target:
68. **return** item.len

From <https://www.geeksforgeeks.org/word-ladder-length-of-shortest-chain-to-reach-a-target-word/>

1. Design a sparse matrix that can do basic calculation such as addition, subtraction, multiplication with constant, and multiplication with matrix (1×n, m×1 or n×m).

[Answer] *I am not very familiar with sparse matrix and I try to simply list my logic and my code structure…*

1. # -\*- coding: utf-8 -\*-
2. """
3. Created on Sun Jan 27 15:08:31 2019
5. @author: Chandler
6. """
8. **class** SparseMatrices (object):
9. """
10. Initiate the Sparse Matrices
11. """
12. **def** \_\_init\_\_(self):
13. self.row = []
14. self.column = []
15. self.value = []

18. **class** SparseMatricesOperation (object):
19. """
20. Perform Sparce Matrices Operation
22. # Example:
23. Row Col Val      Row Col Val
24. 1   2   10       1   1   2
25. 1   3   12       1   2   5
26. 2   1   1        2   2   1
27. 2   3   2        3   1   8
29. Transpose of second matrix:
31. Row Col Val      Row Col Val
32. 1   2   10       1   1   2
33. 1   3   12       1   3   8
34. 2   1   1        2   1   5
35. 2   3   2        2   2   1
37. Summation of multiplied values:
39. result[1][1] = A[1][3]\*B[1][3] = 12\*8 = 96
40. result[1][2] = A[1][2]\*B[2][2] = 10\*1 = 10
41. result[2][1] = A[2][1]\*B[1][1] + A[2][3]\*B[1][3] = 2\*1 + 2\*8 = 18
42. result[2][2] = A[2][1]\*B[2][1] = 1\*5 = 5
44. Any other element cannot be obtained
45. by any combination of row in
46. Matrix A and Row in Matrix B.
48. Hence the final resultant matrix will be:
50. Row Col Val
51. 1   1   96
52. 1   2   10
53. 2   1   18
54. 2   2   5
55. """
57. **def** SparsematricesSum(self, matricesA, matricesB):
58. """
59. Apply Matrices Sum
60. """
61. # Do sum
62. length = set(len(matricesA.value),len(matricesB.value))
63. #for i in range(length):

66. # insert remaining elements
67. **return**
69. **def** SparseMatricesTranspose (self):
70. """
71. Transpose the matrices
72. """

75. **return**
77. **def** SparseMatricesMultiply (self):
78. """
79. Take matrices A and transposed matrices B and apply multipication
80. """
81. # Take matrices B and transpose
83. # Apply multiply
85. **return**


89. **def** main():
90. # Initialize
91. matricesA = SparseMatrices()
92. matricesA.row = [1,2,3]
93. matricesA.column = [1,2,3]
94. matricesA.value = [1,1,1]
96. matricesB = SparseMatrices()
97. matricesB.row = [1,2,3]
98. matricesB.column = [1,2,3]
99. matricesB.value = [1,1,1]
101. # Calculation
102. smo = SparseMatricesOperation()
103. smo.SparseMatricesSum(matricesA,matricesB)
104. smo.SparseMatricesMultiply(matricesA,matricesB)
106. What is the time complexity of the following code?

j = 0

for i in range(0, n):

while j < n and nums[j]-nums[i] < window:

j += 1

[Answer] The boundary defined by j outside of the loop, and we have inner loop and outer loop to traverse the “nums”. Every time when entering the loop, it could either meet the while condition of j and exit or i add up to n.

So the time complexity: the best scenario is O(n), average is O(n), and worst is O(2n)->O(n)