Arya Asgari
Michael Elliott

Implementation 2 Write-Up

Pseudocode:

1. Read in Cost Matrix, and create 2d dictionary with insertion values for each alphabet
2. Read in the input file
3. For line in input file:
    a. Split the line into two sequences (sequence 1, sequence 2)
    b. Send those sequences to a function to calculate edit distance
4. To calculate edit distance:
    a. M = length of sequence1 + 1
    b. N = length of sequence2 + 1
    c. D = [], Total Directions = []
    d. For i to M (row = [] and directions = [])
        i. For j to N
            1. If i == 0 && j == 0:
                a. row.append(0) & directions.append(diagonal)
            2. Else If i == 0:
                a. row.append(cost of deletion) & directions.append(down)
            3. Else If j == 0:
                a. row.append(cost of insertion) & directions.append(left)
            4. Else:
                a. row.append(minimum(cost of deletion, cost of insertion, cost of alignment))
                b. If cost of deletion == minimum : directions.append(left)
                c. If cost of alignment == minimum : directions.append(diag)
                d. If cost of insertion == minimum : directions.append(down)
        ii. D.append(row) & Total_directions.append(directions)
    e. Total Distance = D[i][j]
    f. BackTrace:
        i. Start from Total_directions [i][j]
        ii. Work your through the Total_directions matrix, inserting, deleting, and aligning when necessary (appending to respective sequence outputs)
        iii. Return outputs and total distance

Analysis of runtime:

M is the length of the first sequence, N is the length of the second sequence.

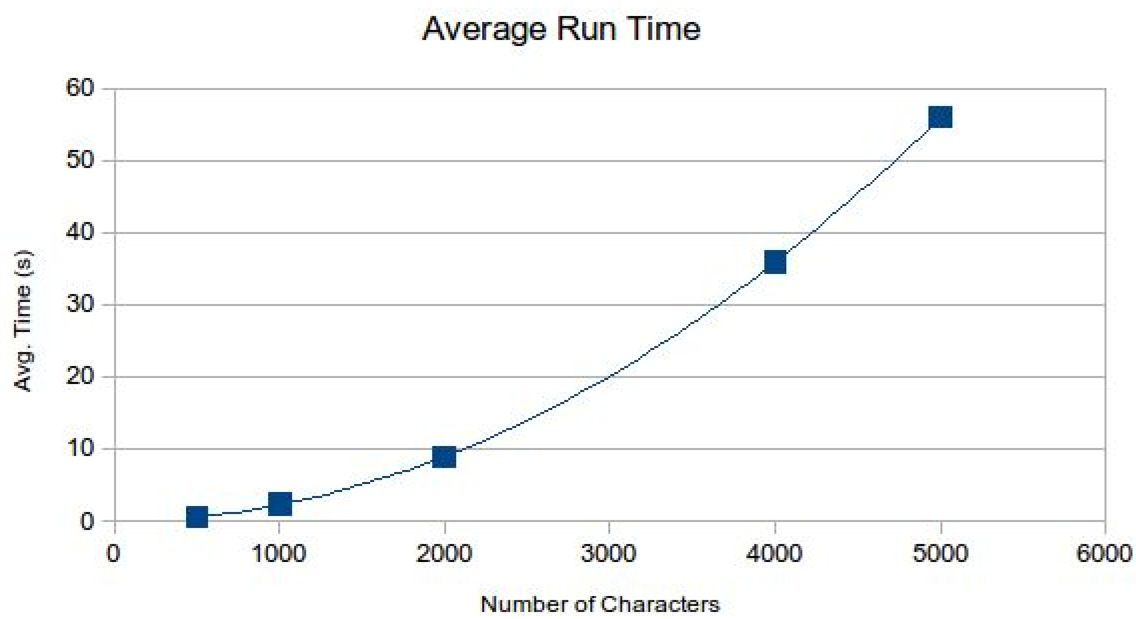
In this algorithm, you have to calculate all the edit distance possibilities of the first sequence turning into the second. In order to do this, you must go through for each alphabet in the first sequence (which is of length M), you must go through N amount of the second sequence. The outer loop runs M times and the inner loop runs N times for each M. This ends up being O(nm) time.

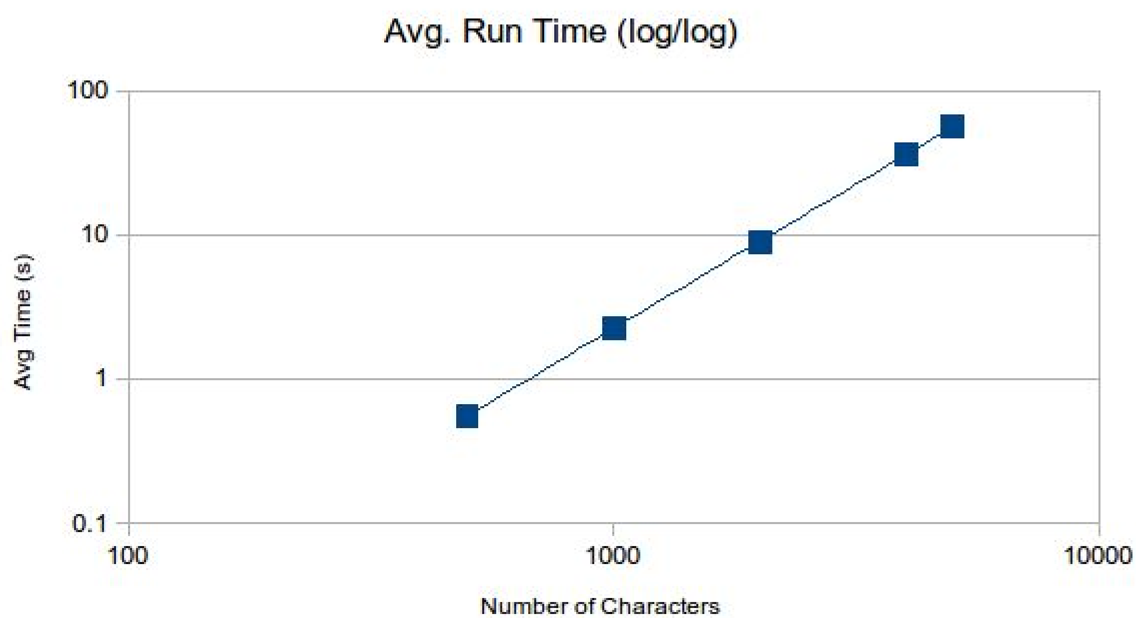
Time of computing Edit Distance : O(nm)
Space necessary for computation: O(nm)
Performance time of backtracing: O(n + m)

Reporting Runtime:

These tests were run on a Lenovo Thinkpad T440p with a dual core 3rd generation Core-i series Intel processor. For each test, a timer was started right before the edit distance began computing, and ended right after the backtracing finished. The setup, including reading in the files, as well as the resolution, including writing output, are ignored. A table of all the trials follows.

	500 Characters	1000 Characters	2000 Characters	4000 Characters	5000 Characters
Trial 1	0.592027902	2.3983418941	9.5086920261	37.660459041	60.712857008
Trial 2	0.548804998	2.1980559826	8.7970499992	37.444994926	56.304822921
Trial 3	0.554628133	2.2022109032	8.7784948349	37.291842937	56.347357034
Trial 4	0.548182964	2.2061300278	9.0260939598	37.308770895	56.287292957
Trial 5	0.545863866	2.2477970123	8.8315529823	34.678545951	55.312494039
Trial 6	0.546308994	2.2463991642	8.8047049046	34.680281162	55.259968042
Trial 7	0.543909788	2.2596359253	8.7765629292	34.751081943	54.377645015
Trial 8	0.544835090	2.27287817	8.828166008	35.421052932	55.405184030
Trial 9	0.545866966	2.2724339962	8.7934331894	34.999752044	55.225342989
Trial 10	0.543765068	2.2626547813	8.7977290154	34.870760917	55.079543113
Average	0.551419377	2.2566537857	8.8942479849	35.910754275	56.031250715





The slope of the log/log graph is 2.00469599, meaning the experimental runtime for our algorithm is approximately $O(n^2)$.

Discussion:

As far as the runtime plot, the results matched our expectations. We assumed that as the input size increased, the average time would increase exponentially due to the O(nm) runtime. The slope of the log/log graph was approximately 2, which means that we have an experimental runtime of $O(n^2)$. Based on the theoretical bounds, our growth curve matches our expectations.