Secret Message

IF YOU CAN READ THIS THE CODE IS WORKING WELL DONE AND PARTY ON

Algorithms Explanation

This algorithm works by generating words from the inputted Morris Code and uses backtracking to find every possible work combination of words the Morris Code could represent in the English dictionary. This part of the algorithm would have an efficacy class of O(n!) . In order to speed up this algorithm, bigrams from the decoding dictionary are used to break out of branches early that do not seem very likely. Each pair of words in the sentence (first and second, second and third, ect) is tested to make sure it has a score of at least 177 in order for the branch to be continued. This score represents the number of times these words were found to be next to each of in the corpus being used. After compiling a list of each branch that was worth perusing, the list is then sorted in descending order based of the scores they were assigned. These scores or weights are based on multiplying the bigram frequency each pair of words in the phrase were given by a large number (10,000 in this case).

Help

I received help to clarify how the example backtracking algorithm worked in order to gain a better understanding on how to approach this lab

Mathematical Analysis

This algorithm should perform at na or better. N is the number of Morris Code characters that are inputted and a is the number of original branches that were perused. Therefore, a is a constant.

Lim n -> infinity !n/ na

Using number testing rather than finding the derivatives I found the algorithm to be more efficient than O(!n) . While !n is smaller a small inputs, !n will surpass na eventually.

N = 100,000

A= 10,000

1.038975936340406172579675860466701347876106 × 10^456573 / 1 × 10^50000

Therefore, !n has a much larger growth rate than na meaning this algorithm is O(!n)

Empirical Analysis

In these timed experiments I wanted to see what variables of the algorithm affected the completion time the most. Additionally, I found that the highest minimum Bigram Frequency that would return the correct result in the highest in all the provided phrases was 177. I used this as a base for the timed tests and finished by checking the bigram frequency’s effect of the completion time. I found the number of matches had the strongest effect on completion time followed by the minimum bigram frequency and then message length. These finding lead me to believe having a flotation bigram frequency based on the message length would provide the most accurate and quick results from this algorithm.

|  |  |  |  |
| --- | --- | --- | --- |
| Message Length | Minimum Bigram Frequency | Number of Possible Matches | Completion Time (ms) |
| 34 | 177 | 59 | 9 |
| 29 | 177 | 10 | 7 |
| 78 | 177 | 11 | 12 |
| 30 | 177 | 2 | 5 |
| 52 | 177 | 31 | 16 |
| 34 | 177 | 23 | 7 |
| 45 | 177 | 188 | 23 |
| 132 | 177 | 1592 | 289 |
| 132 | 200 | 786 | 159 |
| 132 | 300 | 15 | 27 |