1: Probability of k Heads

2: Count Number of Parsings

Collaboration with Maks Cegielski-Johnson.

Algorithm: Below is an algorithm for counting the number of ways a string can be parsed, given a language of valid words.

Algorithm 1 Count number of parsings

```
1:  Global Variables
2:      cache = dictionary of parsings counts for a given word
3:      language = collection of words in the language
4:      L = length of the language
5:  end Global Variables
6:
7:  procedure PARSING_COUNT(S)
8:      if len(s) == 0 then
9:          return 1
10:     end if
11:
12:     if s in cache then
13:         return cache[s]
14:     end if
15:
16:     for index in range(min(N , L) + 1) do
17:         substring = the first index characters of s
18:         if substring in language then
19:             remaining = s with substring removed from the front of s
20:             count = count + parsing_count(remaining)
21:         end if
22:     end for
23:
24:     cache[w] = count
25:     return count
26: end procedure
```

The recursive algorithm will try every possible parse of s, using the words in the language. At each call of the recursive algorithm, s is scanned left to right for matches with words in the language. If a match is found, a substring is formed (removing the match from the beginning of s) and passed to *parsing_count* and the process is repeated on the substring. The scan of s is continued to find additional parses. Scanning continues the minimum of N and L. A parse is successful if the string passed to *parsing_count* is empty. The total count of parses is returned once all possibilities are tested.

Correctness: The algorithm will terminate as the input string s gets smaller with each recursive call to *parsing_count* (moving towards the base case) or no parsing is found and no recursive call is made.

We can observe that we will find every possible parsing as we exhaustively try every possible parsing of s given the language. Thus the algorithm is always correct.

Running time: There are 2^{N-1} possible parsings in the worst-case. This worst-case is when s is a string consisting of all the same letter (ex. aaaaa) and the language is a growing sequence of length N (ex. a, aa, aaa, aaaa, aaaaa).

We combat this exponential runtime by caching the parse counts of substrings of s. There will be N entries in the language, each of which will need to be

3: The Ill-prepared Burglar

4: Central Node in Trees

5: Faster LIS

6: Maximizing Happiness