Daily Coding Problem #264

Problem

This problem was asked by LinkedIn.

Given a set of characters `C` and an integer `k`, a De Bruijn sequence is a cyclic sequence in which every possible `k`-length string of characters in `C` occurs exactly once.

For example, suppose `C = {0, 1}` and `k = 3`. Then our sequence should contain the substrings `{'000', '001', '010', '011', '100', '101', '110', '111'}`, and one possible solution would be `00010111`.

Create an algorithm that finds a De Bruijn sequence.

Solution

There is a neat way of solving this problem using a graph representation.

Let every possible string of length `k - 1` of characters in `C` be vertices. For the example above, these would be `00`, `01`, `10`, and `11`. Then each string can be represented as an edge between two vertices that overlap to form it. For example, `000` would be a loop from `00` to itself, whereas `001` would be an edge between `00` and `01`.

We can construct this graph like so:

```
from itertools import product

def make_graph(C, k):
    vertices = product(C, repeat=k-1)

    edges = {}
    for v in vertices:
        edges[v] = [v[1:] + (char,) for char in C]

    return edges
```

The edges created would be as follows:

```
{'00': ['00', '01']
 '01': ['10', '11']
 '10': ['00', '01']
 '11': ['10', '11']}
```

In order to find the De Bruijn sequence, we must traverse each edge exactly once, or in other words, we must find an Eulerian cycle. One method to accomplish this is known as Hierholzer's algorithm, which works as follows.

Starting with a given vertex, we move along edges randomly, adding the vertices seen to our path, until we come back to where we started. If this path uses up every edge in our graph, we are done. Otherwise, we can take one of the vertices in our path that has an unused edge, perform the same process on that vertex, and substitute the new path back into the original path to replace the vertex.

We can continue inserting new cycles in this manner until every edge of the graph is used.

For example, suppose for we traversed the graph above starting with `01`, and found the following path: `01 -> 11 --> 11 --> 10 --> 00 --> 00 --> 01`. Since there is still an unused edge attached to `01`, we would next find the path `01 --> 10 --> 01` and substitute it at the beginning of our original path, resulting in a valid De Bruijn sequence.

```
def find_eulerian_cycle(graph):
    cycle = []
    start = list(graph)[0]
    before = after = []
    while graph:
        if cycle:
            start = next(vertex for vertex in cycle if vertex in graph)
            index = cycle.index(start)
            before = cycle[:index]; after = cycle[index + 1:]

        cycle = [start]
        prev = start

        while True:
            curr = graph[prev].pop()
            if not graph[prev]:
                graph.pop(prev)

            cycle.append(curr)
            if curr == start:
                break

            prev = curr


        cycle = before + cycle + after

    return cycle
```

Instead of using the vertices, it suffices to return the last element of each one, since that determines the actual path taken. Therefore, our main function should look like this:

```
def debruijn(C, k):
    graph = make_graph(C, k)

    cycle = find_eulerian_cycle(graph)

    sequence = [v[-1] for v in cycle[:-1]]

    return sequence
```

The time required to create the graph will be on the order of the number of edges, which is $|C|^k$. To find the Eulerian cycle, in the best case, we will not need to insert any new paths, so we only need to consider the time needed to pop off and append each edge, which is

`O(E)`. In the worst case, however, we might perform around `log E` substitutions, so this algorithm would be closer to `O(E log E)`.