Maximal Simplicial Set

In this code, we implement the simplicial set $\Delta[\mathbf{d}]_\bullet$, by the name `MaximalSet()`.

The `PosetΔ()` function creates objects of the category $[d]$ via list comprehension. An easier way to achieve the same is by using a command `nx.path_graph(self.d, create_using=nx.DiGraph())`. This command creates a directed graph with d vertices and $d-1$ directed edges, pointing in increasing order. Of course we don't have identity maps. To include them, we could create a list of lists of the form $X = [[i,i],[i,i+1]]$ and pass this into `PosetΔ = nx.DiGraph() PosetΔ.add_edges_from(X)`.

Building on this is the `SimplexΔ()` method, taking in argument `n`. This function creates a list of lists containing n objects taken from $\{0,1,...,d\}$, in ascending order. The method will incorporate degenerate elements, of the form $[0,0,1]$ for $d=1$ and $n=2$. This explains the need of converting the existing list `PosetΔ` containing numbers $[0,1,...,d]$ to $[[0],[1],...,[d]]$ in line. As an added advantage, we can simply use the function list + list to combine two lists, where one is treated as an element of a list itself.

One way we can weed out degenerate elements from this generated list is by simply comparing the size of the list with its corresponding set. If these sizes differ, the list is degenerate:

```
    def DegenerateOrNot(self,entryofalist):
        if len(entryofalist) == len(set(entryofalist)):
            return False
        else:
            return True
```

But we don't need this function right now. Instead, we take advantage of indexing structure of `SimplexΔ()`. We simply need to compare two lists within `SimplexΔ(n)` and `SimplexΔ(n+1)`. For list A in `SimplexΔ(n)` and list B in `SimplexΔ(n+1)`, considered as sets, if $A \subset B$, then we know that we have a degeneracy map $s : A \longrightarrow B$. If $B \subset A$, then we have a face map $d : B \longrightarrow A$. These account for $d_i : X_{n+1} \longrightarrow X_n$ and $s_j : X_n \longrightarrow X_{n+1}$. As for the index i and j, they are chosen by looking at the first object different in the list A and B. The former is in the for loop in line 41.

These maps are stored as edges in a graph. [1] Here, we're adding lists as nodes in a multigraph `Xgraph` using the `networkx` library. The domain and range for the face and degeneracy map comes from nodes in the (multi)graph. Python uses a key-value dictionary to save attribute of an edge.

One problem with this code is that the nodes are not exactly the lists we generated. These lists are converted to simple strings and these are added as vertices in the multigraph. Ditto for the values of the keys (i.e., the name of the edges). I suppose this won't be much trouble, as the values are separately passed from functions, and the functions can always be appealed to, should the need arise.

The complete code is below, and uploaded on github.

```
import networkx as nx

class MaximalSet():
    def __init__(self,d):
        self.d = d

    def PosetΔ(self):
        return [index for index in range(0, self.d+1)]

```

[1] By the way, what's interesting about Python is that we can have a graph with graphs as nodes, graphs with files as nodes, and even graphs with dictionaries as nodes.

```
10      def SimplexΔ(self,n):
11          if n == 0:
12              return(self.PosetΔ())
13          Xbulletn = [[v] for v in self.PosetΔ()]
14          for i in range(n):
15              Xbulletn = [r + [v] for r in Xbulletn for v in self.PosetΔ() if r[-1] <= v]
16          return Xbulletn
17
18      def DegenerateOrNot(self,entryofalist):
19          if len(entryofalist) == len(set(entryofalist)):
20              return False
21          else:
22              return True
23
24      def DegeneracyName(self,d,j):
25          return "s_{} {}".format(d,j)
26
27      def FaceName(self,d,j):
28          return "d_{} {}".format(d,j)
29
30      def MapValue(self):
31          Xgraph = nx.MultiDiGraph()
32          for index in range(0,self.d+1):
33             for listitem in MaximalSet(self.d).SimplexΔ(index):
34                 firstlist = MaximalSet(self.d).SimplexΔ(index)
35                 secondlist = MaximalSet(self.d).SimplexΔ(index+1)
36                 if set(firstlist).issubset(set(secondlist)):
37                     for index, (first, second) in enumerate(zip(firstlist, secondlist)):
38                         if first != second:
39                             Xgraph.add_edge(str(firstlist), str(secondlist), degeneracy=
    MaximalSet(self.d).DegeneracyName(self.d,first))
40                 if set(secondlist).issubset(set(firstlist)):
41                      for index, (first, second) in enumerate(zip(firstlist, secondlist)):
42                          if first != second:
43                              Xgraph.add_edge(str(secondlist), str(firstlist), face=MaximalSet
    (self.d).DegeneracyName(self.d,second))
44          return Xgraph
```

The following modification to line 15 will yield a simplicial set in which $\Delta[\mathbf{d}]_k$ will consist of a list containing k entries, but in which a preceding entry divides its immediate successor

```
Xbulletn = [r + [v] for r in Xbulletn for v in self.PosetΔ() if r[-1] <= v and v | r[-1]]
```