"Company Logo Consolidation using Recursion"

Problem Statement:

A company is interested in aggregating and making known a list of customer who have initiated purchases with a given company.

Issue 1: The source system of record which have suffered from a history of non-standaradized data entry procedures causing unobvious name matches. (ex: "F5 Networks" vs "F5 Networks Incorporated")

Issue 2: Merge/Acquisitions, legal name changes, company divestitures, legal hierarchies, and a partner-distributor model sometimes obfuscate relationhips between accounts that are related to eachother but whose names differ widely. 3rd-party enrichment data is intended to expose this relationship. (ex: "F5 Networks" vs "F5 Inc" vs "NGINX" which was acquired)

Goal:

Given an account name, compile a list of the related accounts

Stretch Goal:

Given a series of account names, compile a list of the related accounts grouped seperately.

Input fields:

Account Keys

Account Names

(Primary Key) Account Site Keys

Account Site Local Unique Identifier

Account Site Global Unique Identifier

Output fields:

Same

Approach 1 (depicted here):

- 1) Read data into memory
- 2) Build capability to search for relations via account names
- 3) Build capability to search for relations via 3rd-party enrichment
- 4) Perform both searches
- 5) Recursively perform both searches using the account names and 3rd-party enrichment gathered from the newly found accounts
- 6) When no new accounts are found, return the aggregated list of accounts related to the original account of interest

In [14]:

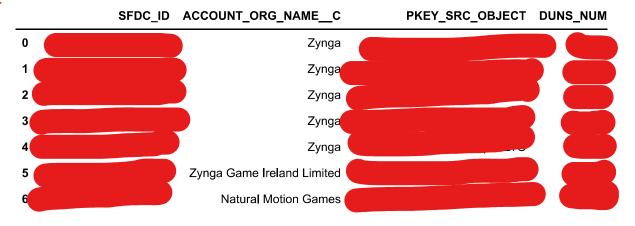
```
import pandas as pd
import pyodbc
import sys
import cmd
import numpy as np
from openpyxl import load_workbook
filepath = 'sfdc_account_w_gult.csv'
```

```
In [15]:
             df = pd.read_csv(filepath, encoding = 'ISO-8859-1', dtype = str)
             df.dropna()
             #0G_org_names = df['ACCOUNT_ORG_NAME__C'].unique().tolist()
             #0G_org_names = ['City of Seattle','Zynga']
             result = pd.DataFrame(columns=df.columns)
             result['Group_Flag'] = ''
In [16]:
             "1st function to recursively search on = account org name"
             "takes a name and returns all rows with similar names"
             def account_org_name_search (account_org_name):
                 group_name_list = pd.DataFrame()
                 group_name_list = group_name_list.append(df[df['ACCOUNT_ORG_NAME__C'] ==
                 group_name_list = group_name_list.append(df[df['ACCOUNT_ORG_NAME__C'].str
                 group_name_list = group_name_list.append(df[df['ACCOUNT_ORG_NAME__C'].str
                 return group_name_list
             "2nd function to recursively search on = GULT duns"
In [17]:
             "takes a gult and returns all other rows with that same gult"
             def gult_search (gult_duns_number):
                 group_gult_list = df[df['DNB_GULT_DUNS'] == gult_duns_number]
                 return group_gult_list
```

```
In [18]:
             "higher level function that calls upon the recursive search variables, repeat
              "takes an account name and returns a company logo group"
             "if the account name is in already gathered account logo group, this function
             "returned group will adopt the flag of the existing group "
             def group_flag_func(account_org_name, group_name_list = [], group_gult_list =
                 iteration = iteration + 1
                  print('Iteration ')
                  print(iteration )
                 if group is None:
                      group = account_org_name_search(account_org_name)
                      group_name_list = group['ACCOUNT_ORG_NAME__C'].dropna().unique().toli
                 group_name_list_end = group_name_list[:]
                 group_gult_list_end = group_gult_list[:]
                 for name in group_name_list:
                      print(name)
                      group = group.append(account_org_name_search(name))
                      group_gult_list.extend(account_org_name_search(name)['DNB_GULT_DUNS']
                 for gult in group_gult_list:
                      print(gult)
                      group = group.append(gult_search(gult))
                      group_name_list.extend(gult_search(gult)['ACCOUNT_ORG_NAME__C'].dropr
                 group_gult_list_end = list( dict.fromkeys(group_gult_list_end) )
                 group_gult_list = list( dict.fromkeys(group_gult_list) )
                 group = group.dropna().drop_duplicates(subset = 'SFDC_ID').reset_index(dr
             #If the new list equals the inputted list - end recursion"
             #If new accounts were added to the inputted list - assume there may be more \mathfrak q
                 if sorted(group_gult_list_end) == sorted(group_gult_list):
                      return group
                 else:
                      group_flag_func(account_org_name, group_name_list = group_name_list,
```



Out[24]:



Notes on results:

Desired results met! Fuzzy matching for the account names and some basic casenormalizing would aggregate results more comprehensively reducing any manual review.

Notes on performance:

Performing operations all in memory on a laptop is potentially slow. Consider indexing, using correct datatypes, utilizing pandas dataframes which can potentially reduce expensive looping, adding cores if using an MPP or cloud cluster, more?

Next Steps:

To be written - Data Science utilizes graph algorithms via networkio which is a form of non-relational database management. I have yet to build out this method, but a side-by-side comparison with a related dataset between this method and the graph method indicated that results were EXACTLY the same.

Other Next Steps:

Cryptography?? (So I can share results, kind of)