Importing modules

In [1]:

```python
import sqlite3
import pandas as pd
import networkx as nx #Find cycle
import itertools
import json
from itertools import chain
```

Importing .db file and minor data manipulation

In [2]:

```python
# Creating file path
dbfile = 'full.db'

# Create a SQL connection to our SQLite database
con = sqlite3.connect(dbfile)

# Reading all table names on .db file
model_types = pd.read_sql_query("SELECT * FROM model_types", con)
model_prerequisites = pd.read_sql_query("SELECT * FROM model_prerequisites", con)

# Close connection
con.close()
```

In [3]:

```python
# Assign 'id' column as index for easier lookup later
model_types = model_types.set_index('id')
model_types.head()
```

Out[3]:

id	name
1	xRH11
2	uoG12
3	QDD2
4	dmH13
5	OkG18

In [4]:

```python
# Switch positions of 2 columns (prereq and dependent)
model_prerequisites = model_prerequisites[['pipeline_name','prereq_model_type_id','dependent_model_type_id']]
model_prerequisites.head()
```

Out[4]:

	pipeline_name	prereq_model_type_id	dependent_model_type_id
0	test_pipeline	19	51
1	test_pipeline	35	35
2	test_pipeline	3	45
3	test_pipeline	45	3
4	test_pipeline	46	23

Segregating pipeline (Divide & Conquer)

In [5]:

```python
# Get unique values in pipeline_name
pipeline_name = model_prerequisites['pipeline_name'].unique()
pipeline_group = []
pipeline_group_ids = []

for p in pipeline_name:
    print(p)
    print('---------------------------')
    '''
    Segment pipeline by pipeline_name by pipeline_name into a list
    '''
    pipeline_subset = model_prerequisites[model_prerequisites['pipeline_name'] == p]
    pipeline_subset = pipeline_subset.drop(columns = ['pipeline_name'])
    pipeline_list = pipeline_subset.values.tolist()

    while pipeline_list != []: #Big(O) = nlogn
        test_subset = []
        group = [p for p in pipeline_list[0]]
        for t in pipeline_list:
            for e in t:
                if e in group:
                    group.extend(t)
            group = list(dict.fromkeys(group))
            if t[0] in group and t[1] in group:
                test_subset.append(t)
                pipeline_group.append(p)
                pipeline_group_ids.append(test_subset)
        print(test_subset)

        pipeline_list = [x for x in pipeline_list if x not in test_subset]

    print('')
```

```
test_pipeline
---------------------------
[[19, 51]]
[[35, 35]]
[[3, 45], [45, 3]]
[[46, 23], [46, 7], [23, 7]]
[[27, 39], [21, 27], [39, 21]]

pipeline1
---------------------------
[[8, 31], [8, 20], [31, 12], [31, 26], [20, 33], [20, 34], [12, 28], [12, 47], [26, 43], [26, 44], [33, 2], [33, 22], [34, 15], [34, 48], [28, 14], [47, 14], [43, 29], [44, 29], [2, 5], [22, 5], [14, 10], [29, 10], [5, 56], [15,
56], [48, 56], [29, 6], [5, 6], [10, 30], [56, 30], [6, 30]]

pipeline2
---------------------------
[[54, 38], [54, 53], [25, 53], [38, 40], [38, 16], [53, 41], [53, 24], [40, 55], [40, 52], [16, 49], [16, 1], [41, 11], [41, 4], [24, 32], [24, 25], [55, 50], [52, 50], [49, 42], [1, 42], [11, 17], [4, 17], [50, 13], [42, 13], [1
7, 36], [32, 36], [25, 36], [42, 37], [17, 37], [13, 18], [36, 18], [37, 18]]
```

In [6]:

```python
processed_pipeline = pd.DataFrame({
     'main_pipeline': pipeline_group,
     'pipeline_group_ids': pipeline_group_ids
    })
# Remove all duplicates
processed_pipeline = processed_pipeline.loc[processed_pipeline.astype(str).drop_duplicates().index]
processed_pipeline
```

Out[6]:

	main_pipeline	pipeline_group_ids
0	test_pipeline	[[19, 51]]
1	test_pipeline	[[35, 35]]
2	test_pipeline	[[3, 45], [45, 3]]
4	test_pipeline	[[46, 23], [46, 7], [23, 7]]
7	test_pipeline	[[27, 39], [21, 27], [39, 21]]
10	pipeline1	[[8, 31], [8, 20], [31, 12], [31, 26], [20, 33...
40	pipeline2	[[54, 38], [54, 53], [25, 53], [38, 40], [38, ...

Find all cycles in each pipeline

In [7]:

```python
cycle_group = []
for group in processed_pipeline['pipeline_group_ids']:
    G = nx.DiGraph(group)
    invalid_models = list(itertools.chain.from_iterable(nx.recursive_simple_cycles(G)))
    cycle_group.append(invalid_models)
for c in cycle_group:
    cycle_group[cycle_group.index(c)] = list(set(c))
processed_pipeline['cycle_group'] = cycle_group
processed_pipeline
```

Out[7]:

	main_pipeline	pipeline_group_ids	cycle_group
0	test_pipeline	[[19, 51]]	[]
1	test_pipeline	[[35, 35]]	[35]
2	test_pipeline	[[3, 45], [45, 3]]	[3, 45]
4	test_pipeline	[[46, 23], [46, 7], [23, 7]]	[]
7	test_pipeline	[[27, 39], [21, 27], [39, 21]]	[27, 21, 39]
10	pipeline1	[[8, 31], [8, 20], [31, 12], [31, 26], [20, 33...	[]
40	pipeline2	[[54, 38], [54, 53], [25, 53], [38, 40], [38, ...	[24, 25, 53]

Find all model_ids dependent on cycle

In [8]:

```python
cycle_dependent = list(zip(processed_pipeline['pipeline_group_ids'].tolist(),processed_pipeline['cycle_group'].tolist()))
valid_model_group = []
invalid_model_group = []
for cd in cycle_dependent:
    if cd[1] == []: #no cycle within pipeline
        non_cycle_id = cd[0] #list of all tuples
        non_cycle_id = [i for sub in non_cycle_id for i in sub] #merge all tuples into 1 single list
        non_cycle_id = list(set(non_cycle_id)) #eliminate duplicates in list
        valid_model_group.append(non_cycle_id)
        invalid_model_group.append([])

    else:
        cg = cd[0] #cg: cycle_group
        im = cd[1] #im: invalid_model cycle
        cycle_vertices = []
        # for c in cg:
        #     cg[cg.index(c)] = list(c)

        cg_element = list(set(list(chain(*cg))))
        while cg != []:

            #Remove vertices we know for sure fall in cycles
            for ele in cg:
                if ele[0] in im and ele[1] in im:
                    cycle_vertices.append(ele)

            cg = [i for i in cg if i not in cycle_vertices]

            #Remove vertices connected with cycles by origin
            for ele in cg:
                if ele[0] in im:
                    cycle_vertices.append(ele)
                    im.append(ele[1])

            cg = [i for i in cg if i not in cycle_vertices]

            #Remove vertices that destination is an id within a cycle
            for ele in cg:
                if ele[1] in im:
                    cg.remove(ele)

            #If there are/is still element(s) of invalid_models in all vertices of cg, the while loop continues
            for i in im:
                if i not in list(itertools.chain.from_iterable(cg)):
                    valid_model_group.append([i for i in cg_element if i not in im])
                    invalid_model_group.append(list(set(im)))
                    cg = []
                else:
                    cg = cg

# Remove duplicates
validity = pd.DataFrame({
    'valid_model_group': valid_model_group,
    'invalid_model_group':invalid_model_group
    })
validity = validity.loc[validity.astype(str).drop_duplicates().index]
validity
```

Out[8]:

	valid_model_group	invalid_model_group
0	[51, 19]	[]
1	[]	[35]
2	[]	[3, 45]
4	[7, 46, 23]	[]
5	[]	[27, 21, 39]
8	[2, 5, 6, 8, 10, 12, 14, 15, 20, 22, 26, 28, 2...	[]
9	[1, 13, 16, 38, 40, 42, 49, 50, 52, 54, 55]	[32, 4, 36, 37, 41, 11, 17, 18, 53, 24, 25]

In [9]:

```python
# Add finding to processed_pipeline dataframe and trim away unnecessary columns
processed_pipeline['valid_models'] = validity['valid_model_group'].values.tolist()
processed_pipeline['invalid_models'] = validity['invalid_model_group'].values.tolist()
processed_pipeline = processed_pipeline.drop(columns = ['cycle_group', 'pipeline_group_ids']) #Irrelevant to final output
processed_pipeline
```

Out[9]:

	main_pipeline	valid_models	invalid_models
0	test_pipeline	[51, 19]	[]
1	test_pipeline	[]	[35]
2	test_pipeline	[]	[3, 45]
4	test_pipeline	[7, 46, 23]	[]
7	test_pipeline	[]	[27, 21, 39]
10	pipeline1	[2, 5, 6, 8, 10, 12, 14, 15, 20, 22, 26, 28, 2...	[]
40	pipeline2	[1, 13, 16, 38, 40, 42, 49, 50, 52, 54, 55]	[32, 4, 36, 37, 41, 11, 17, 18, 53, 24, 25]

Lookup model_id to model_name in model_types table

In [10]:

```python
# Valid models
vmg_name_group = []
for vmg in processed_pipeline['valid_models']:
    vmg_name = []
    if vmg == []:
        vmg_name.append('')
    else:
        for v in vmg:
            v = model_types.loc[v, 'name']
            vmg_name.append(v)
    vmg_name_group.append(vmg_name)

# Invalid models
ivmg_name_group = []
for ivmg in processed_pipeline['invalid_models']:
    ivmg_name = []
    if ivmg == []:
        ivmg_name.append('')
    else:
        for iv in ivmg:
            iv = model_types.loc[iv, 'name']
            ivmg_name.append(iv)
    ivmg_name_group.append(ivmg_name)
```

In [11]:

```python
# Add those model_names to process_pipeline table
processed_pipeline['vmg_name_group'] = vmg_name_group
processed_pipeline['ivmg_name_group'] = ivmg_name_group

# Drop valid_models and invalid_models (id) columns, keep their model_names
processed_pipeline = processed_pipeline.drop(columns = ['valid_models', 'invalid_models'])
processed_pipeline = processed_pipeline.rename(columns = {'vmg_name_group': 'valid_models',
                                                           'ivmg_name_group': 'invalid_models'})

processed_pipeline
```

Out[11]:

	main_pipeline	valid_models	invalid_models
0	test_pipeline	[kcB2, zvB1]	[]
1	test_pipeline	[]	[CGC1]
2	test_pipeline	[]	[QDD2, nBD1]
4	test_pipeline	[NXE3, NPE1, OfE2]	[]
7	test_pipeline	[]	[PoF3, guF2, SzF1]
10	pipeline1	[uoG12, OkG18, bYG21, SSG1, fkG19, RhG4, egG16...	[]
40	pipeline2	[xRH11, AnH19, PtH5, PVH2, aDH4, VZH17, PQH10,...	[yYH14, dmH13, qVH20, aMH21, wbH6, wFH12, XLH1...

JSON Output

In [15]:

```python
# Convert process_pipeline dataframe into json
process_pipeline_json = processed_pipeline.to_json(orient="records")
process_pipeline_parsed = json.loads(process_pipeline_json)

# Review user-friendly json output
process_pipeline_parsed # Look leaner on simple pipelines

'''
Some minor edits to the original assignments
since I find it much easier to read top to bottom.
I hope you don't mind.
'''

process_pipeline_parsed
```

Out[15]:

```
[{'main_pipeline': 'test_pipeline',
  'valid_models': ['kcB2', 'zvB1'],
  'invalid_models': ['']},
 {'main_pipeline': 'test_pipeline',
  'valid_models': [''],
  'invalid_models': ['CGC1']},
 {'main_pipeline': 'test_pipeline',
  'valid_models': [''],
  'invalid_models': ['QDD2', 'nBD1']},
 {'main_pipeline': 'test_pipeline',
  'valid_models': ['NXE3', 'NPE1', 'OfE2'],
  'invalid_models': ['']},
 {'main_pipeline': 'test_pipeline',
  'valid_models': [''],
  'invalid_models': ['PoF3', 'guF2', 'SzF1']},
 {'main_pipeline': 'pipeline1',
  'valid_models': ['uoG12',
   'OkG18',
   'bYG21',
   'SSG1',
   'fkG19',
   'RhG4',
   'egG16',
   'wAG14',
   'CYG3',
   'PPG13',
   'yRG5',
   'AwG8',
   'EEG17',
   'WWG22',
   'mmG2',
   'LRG6',
   'tKG7',
   'TnG10',
   'XLG11',
   'lpG9',
   'xqG15',
   'FfG20'],
  'invalid_models': ['']},
 {'main_pipeline': 'pipeline2',
  'valid_models': ['xRH11',
   'AnH19',
   'PtH5',
   'PVH2',
   'aDH4',
   'VZH17',
   'PQH10',
   'HWH16',
   'FTH9',
   'trH1',
   'hiH8'],
  'invalid_models': ['yYH14',
   'dmH13',
   'qVH20',
   'aMH21',
   'wbH6',
   'wFH12',
   'XLH18',
   'SnH22',
   'TaH3',
   'UqH7',
   'gMH15']}]
```

In [13]:

```python
#Write to JSON file
with open('pipeline_json.txt', 'w') as jsonout:
    jsonout.write(process_pipeline_json)
    jsonout.close()

# Review JSON Output
with open('pipeline_json.txt') as jsonout:
    contents = jsonout.read()
    print(contents)
```


```
[{"main_pipeline":"test_pipeline","valid_models":["kcB2","zvB1"],"invalid_models":[""]},{"main_pipeline":"test_pipeline","valid_models":[""],"invalid_models":["CGC1"]},{"main_pipeline":"test_pipeline","valid_models":[""],"invalid
_models":["QDD2","nBD1"]},{"main_pipeline":"test_pipeline","valid_models":["NXE3","NPE1","OfE2"],"invalid_models":[""]},{"main_pipeline":"test_pipeline","valid_models":[""],"invalid_models":["PoF3","guF2","SzF1"]},{"main_pipelin
e":"pipeline1","valid_models":["uoG12","OkG18","bYG21","SSG1","fkG19","RhG4","egG16","wAG14","CYG3","PPG13","yRG5","AwG8","EEG17","WWG22","mmG2","LRG6","tKG7","TnG10","XLG11","lpG9","xqG15","FfG20"],"invalid_models":[""]},{"main_
pipeline":"pipeline2","valid_models":["xRH11","AnH19","PtH5","PVH2","aDH4","VZH17","PQH10","HWH16","FTH9","trH1","hiH8"],"invalid_models":["yYH14","dmH13","qVH20","aMH21","wbH6","wFH12","XLH18","SnH22","TaH3","UqH7","gMH15"]}]
```