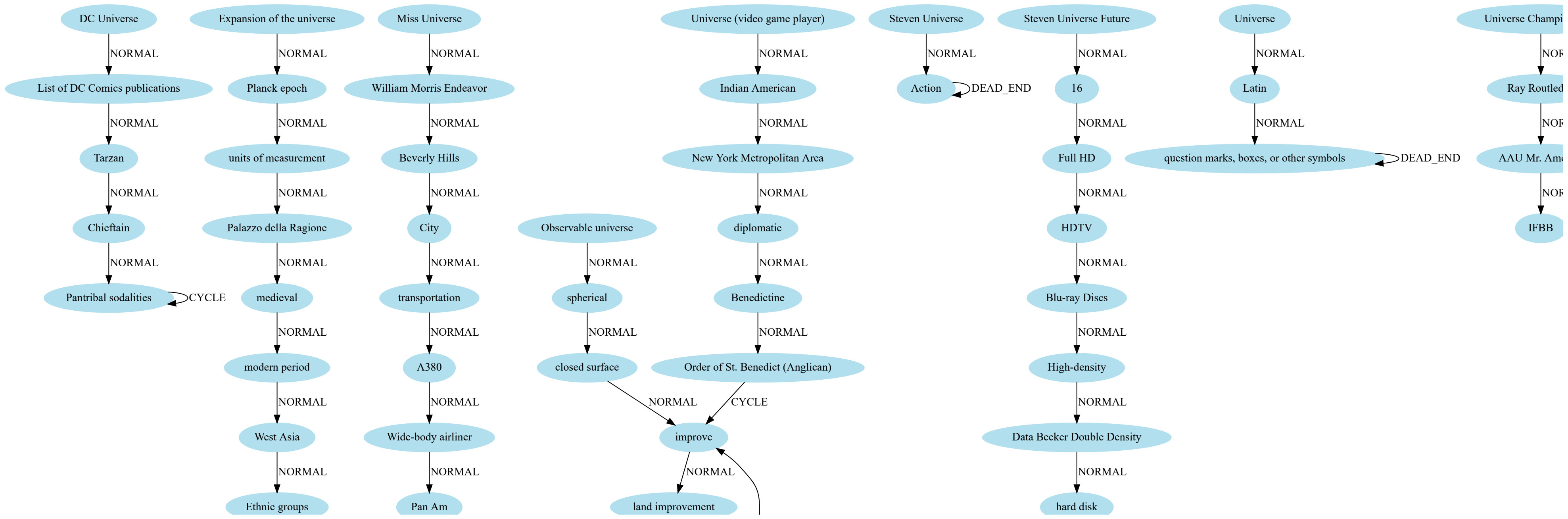
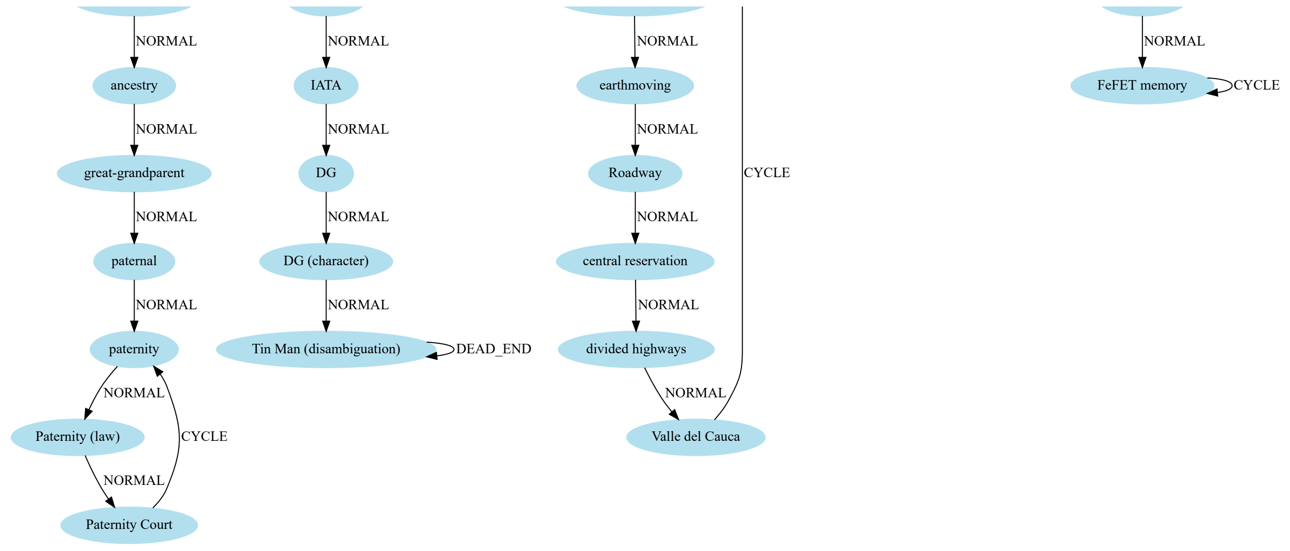
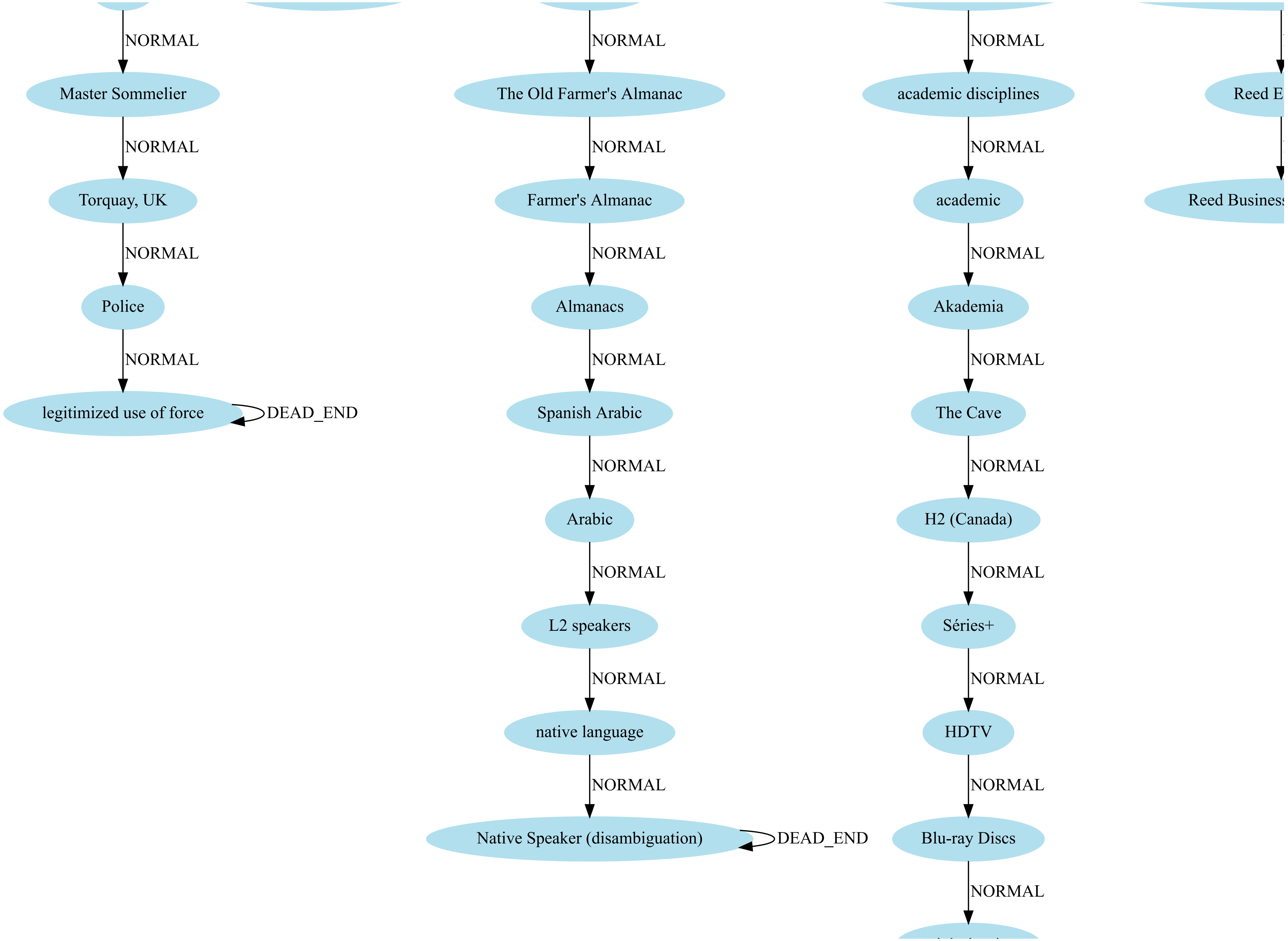
```
from operator import itemgetter
from typing import List

import requests


def get_titles(gsrsearch: str, limit: int = 5) -> List[str]:
    url = "https://en.wikipedia.org/w/api.php"
    params = {
        'action': 'query',
        'generator': 'search',
        'origin': '*',
        'gsrsearch': {gsrsearch},
        'format': 'json',
        'gsrlimit': f'{limit}'
    }
    session = requests.Session()
    req = session.get(url=url, params=params)
    data = req.json()
    titles_list = list(map(
        itemgetter("title"),
        data["query"]["pages"].values()
    ))
    return titles_list


import re
from typing import Optional

import bs4


def get_redirect(title) -> Optional[str]:
    session = requests.Session()
    url = f"https://wikipedia.org/wiki/{title}"
    req = session.get(url=url)
    if not req.ok:
        return None
    soup = bs4.BeautifulSoup(req.text, features="lxml")
    body_content = soup.find("div", id="bodyContent")
    tag_a = body_content.find('a', {
        "class": "mw-redirect",
        "href": re.compile("^/wiki/*")
    })
    return tag_a.string if tag_a else None



from collections import namedtuple


RedirectEdge = namedtuple(
    'RedirectEdge', [
        'from_vertex',
        'to_vertex',
        'type',
        'depth'
    ]
)


class RedirectGraph:

    def __init__(self):
        self.__history = set()
        self.__redirects = []

    def __contains__(self, item):
        return item in self.__history

    def append(self, redirect_edge: RedirectEdge):
        self.__redirects.append(redirect_edge)

    def register(self, vertex):
        self.__history.add(vertex)
    @property
    def edges(self):
      return self.__redirects


from collections import deque

import time


def traverse(entry_title: str, limit: int = 1, stop_word: str = "philosophy"):
    queue = deque(list(map(
        lambda title_nm: (0, title_nm),
        get_titles(entry_title, limit)
    )))
    redirect_graph = RedirectGraph()
    while queue:
        step, title = queue.popleft()
        print(f"{step}: {title}")
        redirect_title = get_redirect(title)
        if step == 0 and redirect_title is None:
            queue.append((0, get_titles(entry_title, 1)[0]))
        elif redirect_title is None:
            print(f"[NO REDIRECT]: {title}")
            redirect_graph.append(RedirectEdge(
                from_vertex=title,
                to_vertex=title,
                type='DEAD_END',
                depth=step
            ))
            continue
        elif re.search(stop_word, redirect_title.strip().lower()):
            print(f"[FINISH]: {redirect_title}")
            redirect_graph.register(redirect_title)
            redirect_graph.append(RedirectEdge(
                from_vertex=title,
                to_vertex=redirect_title,
                type='FINISH',
                depth=step+1
            ))
            return step + 1, redirect_graph
        elif redirect_title in redirect_graph:
            print(f"[CYCLED]: {redirect_title}")
            redirect_graph.append(RedirectEdge(
                from_vertex=title,
                to_vertex=redirect_title,
                type='CYCLE',
                depth=step+1
            ))
        else:
            if step == 0:
                redirect_graph.register(title)
            redirect_graph.register(redirect_title)
            redirect_graph.append(RedirectEdge(
                from_vertex=title,
                to_vertex=redirect_title,
                type='NORMAL',
                depth=step+1
            ))
            queue.append((step + 1, redirect_title))
        time.sleep(0.1)
    return -1, redirect_graph


try:
    import graphviz
except:
    import sys
    !{sys.executable} -m pip3 install graphviz
    import graphviz


def test(entry_title: str, limit: int = 1, stop_word: str = "philosophy"):
    depth, graph = traverse(entry_title, limit, stop_word)
    if depth == -1:
        print(f"{'>'*20}NO PHILOSOPHY{'<'*20}")
    else:
        print(f"{'>'*20}PHILOSOPHY REACHED AFTER {depth} STEPS{'<'*20}")
    graphics = graphviz.Digraph(
        'unix',
        node_attr={
            'color': 'lightblue2',
            'style': 'filled'
        },
        comment='Wiki traverse'
    )
    graphics.attr(size='12')
    for edge in graph.edges:
        graphics.edge(edge.from_vertex, edge.to_vertex, label=edge.type)
    return graphics


test("Universe", 10)
```

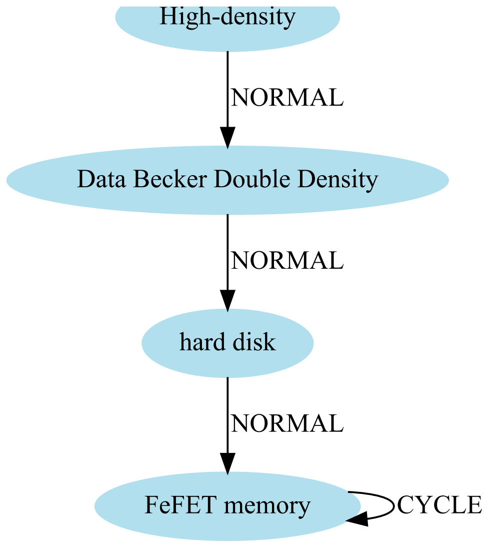
```
7: Pan Am
7: central reservation
7: hard disk
8: ancestry
8: IATA
8: divided highways
8: FeFET memory
[CYCLED]: FeFET memory
9: great-grandparent
9: DG
9: Valle del Cauca
[CYCLED]: improve
10: paternal
10: DG (character)
11: paternity
11: Tin Man (disambiguation)
[NO REDIRECT]: Tin Man (disambiguation)
12: Paternity (law)
13: Paternity Court
Warning: node 16, port 9 unrecognized
Warning: node 16, port 9 unrecognized
[CYCLED]: paternity
>>>>>>>>>>>>>>>>>>>>NO PHILOSOPHY<<<<<<<<<<<<<<<<<<<
```




NORMAL
ancestry
NORMAL
great-grandparent
NORMAL
paternal
NORMAL
paternity
NORMAL
Paternity (law)
NORMAL
Paternity Court
CYCLE
NORMAL
IATA
NORMAL
DG
NORMAL
DG (character)
NORMAL
Tin Man (disambiguation)
DEAD_END
NORMAL
earthmoving
NORMAL
Roadway
NORMAL
central reservation
NORMAL
divided highways
NORMAL
Valle del Cauca
CYCLE
NORMAL
FeFET memory
CYCLE

```
test("Science", 5)
```


NORMAL
Master Sommelier
NORMAL
Torquay, UK
NORMAL
Police
NORMAL
legitimized use of force
DEAD_END
NORMAL
The Old Farmer's Almanac
NORMAL
Farmer's Almanac
NORMAL
Almanacs
NORMAL
Spanish Arabic
NORMAL
Arabic
NORMAL
L2 speakers
NORMAL
native language
NORMAL
Native Speaker (disambiguation)
DEAD_END
NORMAL
academic disciplines
NORMAL
academic
NORMAL
Akademia
NORMAL
The Cave
NORMAL
H2 (Canada)
NORMAL
Séries+
NORMAL
HDTV
NORMAL
Blu-ray Discs
NORMAL
Reed E
Reed Business


High-density
NORMAL
Data Becker Double Density
NORMAL
hard disk
NORMAL
FeFET memory
CYCLE

```
test("Ancient", 5)
```