CS 5001 – Applied Social Network Analysis
Case Study: Mining Wikipedia

What's Wikipedia?

- It's a free **online encyclopedia**, created and edited by volunteers around the world

  What does it have to do with social networks???

  - **Users are encouraged to make personal connections** to develop the encyclopedia
  - It's a web-based platform that **facilitates collaboration among people** with specialized interests and promote those interests with the rest of the world
  - It unites people who have the goal of **making human knowledge available to everyone** on the planet

- It's been around since 2001; first proposal for an online dictionary was in 1993

- Worldwide monthly readership is ~495M; there are ~50M pages in ~309 languages

- Kind of **information you can "mine"** from it:
  - You can grab whatever information is on a page; some of that is common to every page (i.e., **title, summary, etc.**), and the rest you just can get via the **page's html and scrub** for particular things you're looking for
  - You can also get **links** (i.e., URLs) on the page that take you to related pages, and on and on…

Python Networkx for Wikipedia

- Our first example will just show some basic operations: **how to search for and retrieve a page, get meta-data from the page (including links to other pages)**

  Ex: **# Perform some basic operations on a Wikipedia page**

  import wikipedia

  # You can ask wikipedia for matches to a phrase
  print(**wikipedia.suggest**("KC Cheifs"))

```
# Get a short summary; can limit the number of sentences
print(wikipedia.summary("KC Chiefs"))
print(wikipedia.summary("KC Chiefs", sentences=1))

# Can do a search and limit number of matches
# There's also wikipedia.geosearch(lat, long) that will return
# articles based on coordinates
result = wikipedia.search("KC Chiefs", results=5)
print(result)

# This will return the "most relevant" match
page_object=wikipedia.page("KC Chiefs")

# These should be fairly obvious
print(page_object.original_title)
print(page_object.content)
print(page_object.html)          # there are web-scrapers to parse this

# The following doesn't work; just returns [ ]
print(page_object.sections)

# Categories this page has been associated with
print(page_object.categories)

# URLs to external links on the page (alphabetized)
print(page_object.references[0:10])      # we'll just show 10

# List of titles of pages whose links are on the page
# These don't map (index-wise) to references
print(page_object.links[0:10])           # we'll just show 10

# Get the URLs of images displayed on the page
print(page_object.images[0])             # this is just 1 image

# Save the image as jpg file on your computer
import urllib.request
urllib.request.urlretrieve(page_object.images[0], "chiefs.jpg")
```

- The next example retrieves a page on the topic “Missouri S&T”, then **builds a directed graph out of the links from that page** (i.e., those links plus what those pages link to)

  The program could go on to add edges from the links to those links, and so on, but that would get huge! We’ll just triage only going 1 level deep, and making a subgraph from our result based on links with degree ≥ 2.

Ex: **# Build a directed graph of linked Wikipedia pages**

```
import networkx as nx
import matplotlib.pyplot as plt
from operator import itemgetter
import wikipedia

# Specify name of the starting page
SEED = "Missouri S&T"

# We want to ignore any links to these
STOPS = ("International Standard Serial Number",
"Issn (Identifier)",
"International Standard Book Number", "Isbn (Identifier)",
"International Standard Name Identifier", "Isni (Identifier)",
"International Standard Book Number (Identifier)",
"Pubmed Identifier", "Pubmed Central", "Viaf (Identifier)",
"Ndl (Identifier)", "Gnd (Identifier)", "S2Cid (Identifier)",
"Geographic Coordinate System", "Bibcode (Identifier)",
"Digital Object Identifer","Doi (Identifier)", "Arxiv",
"Wayback Machine", "Citeseerx (Identifier)",
"Proc Natl Acad Sci Usa", "Bibcode (Identifier)",
"Worldcat Identities (Identifier)",
"Library of Congress Control Number", "Lccn (Identifier)", "Jstor")

# Keep track of pages to process; we'll only do 2 layers,
# process links as a BFS
toDo_list = [(0, SEED)]
toDo_set = set(SEED)
done_set = set()

# Create a directed graph
F = nx.DiGraph()

layer, page = toDo_list[0]        # initially, layer = 0 and page = SEED
```

```
while layer < 2:
 del toDo_list[0]
 done_set.add(page)
 print(layer, page)
 if (layer > 0) and (page > "C"):
  break  # enough already!
 try:
  # Download the selected page
  wiki = wikipedia.page(page)
 except:
  layer, page = toDo_list[0]
  print("Could not load", page)
  continue
 for link in wiki.links:    # go thru the links on that page
  link = link.title()
  if link not in STOPS and not link.startswith("List Of"):
   if link not in toDo_set and link not in done_set:
    toDo_list.append((layer+1, link))
    toDo_set.add(link)
   F.add_edge(page, link)
   print("Added link", link, "for page", page)
 if len(toDo_list) == 0:
   print("toDo_list is empty")
   break
 layer, page = toDo_list[0]

# Here's the graph we made (6k nodes, 8k edges)
print("{} nodes, {} edges". format(len(F),nx.number_of_edges(F)))

# Many Wikipedia pages exist under > 1 name (e.g., "KC Chief"
# and "KC Chiefs") and one may redirect to the former;
# the following will get rid of some "self loops"
F.remove_edges_from(nx.selfloop_edges(F))

# F is a pretty big graph!
# We can make a smaller subgraph by only including nodes
# that have degree >= 2
core = [node for node, deg in dict(F.degree()).items() if deg >= 2]
G = nx.subgraph(F, core)
# G is a lot smaller (1k nodes, 2k edges)
print("{} nodes, {} edges".format(len(G), nx.number_of_edges(G)))
```

```
# Display the smaller graph
pos = nx.spring_layout(G)
plt.figure(figsize=(50,50))
nx.draw_networkx(G, pos=pos, with_labels=True)
plt.axis('off')
plt.show()

# Display a list of subjects sorted by in-degree
top_indegree = sorted(dict(G.in_degree()).items(),
                      reverse=True, key=itemgetter(1))[:100]
print("\n".join(map(lambda t: "{} {}".format(*reversed(t)), top_indegree)))
```