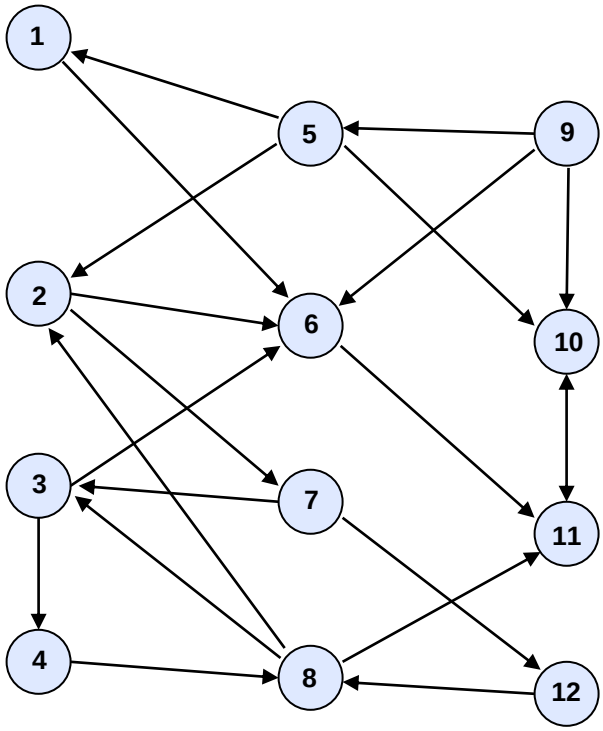
Task 1: Hubs, Authorities und PageRank (theoretical)

The following sub-graph of the Internet is given:



In this task, we order the nodes by their hub, authority, and PageRank values

a) We have defined matrices $\mathbf{M}$ and $\mathbf{A}$ for the iterations. In this sub task we use the original HITS algorithm:

$$\boldsymbol{r}^{(t+1)} = \frac{1-\alpha}{N} \cdot \mathbf{1} + \alpha \cdot \mathbf{M} \cdot \boldsymbol{r}^{(t)}$$

$$h^{(t+1)} = \mathbf{A} \cdot a^{(t)}$$

$$a^{(t+1)} = \mathbf{A}^{\mathsf{T}} \cdot h^{(t)}$$

Compute the matrices for the example graph.

b) Write a small program (e.g., with MATLAB, but also works with Excel) that evaluates the fix-point iteration to obtain all results.

c) For the example graph, determine the best hubs, authorities, and the documents with high PageRanks.

d) Apply the SALSA algorithm to the example graph. Does the order change compared to the original HITS algorithm?

Task 2: NLTK and Python (practical)

In this task, we use the NLTK library for Python to run a number of interesting text analytics. The next page contains a few hints how to setup NLTK (takes at most 5 minutes) and how to get started. You can either create your own classes and methods in Python, or simply collect the commands in a text file and copy paste to the interpreter.

a) **[easy]** Use NLTK to guess the language of an input text. Download an Italian, German, and English (or any other Language, preferably all in the same encoding to simplify matters). Use the stop word lists in NLTK to identify the language of the text. Try some harder text examples with mixed languages and return probabilities for the languages.
Hint: limit your analysis to a few fixed languages only

b) **[intermediate]** Assume we are using a service like https://www.clarifai.com to annotate images. Given a picture, we obtain a list of trained keywords associated with that picture. In order to broaden the keyword list, we want to extend each term with a set of related terms using WordNet. Use the online version of WordNet (http://wordnetweb.princeton.edu/perl/webwn) to get an idea how to do this cleverly. Then implement your idea with the NLTK corpus `nltk.corpus.wordnet`. See online documentation: Chapter 5 in http://www.nltk.org/book/ch02.html

c) **[difficult]** When translating from one language to another, a common problem is the wrong usage of words in the target language due to overlapping word semantics. For instance, the German word “stark” can have a number of English counterparts, namely: strong, intense, powerful, massive, potent, robust, vigorous, severe, heavy, thick, deep, and so on. Obviously, not all English counterparts are correct in a given context. Consider the following example:

- es regnet stark → it is raining hard / heavily (maybe: intensely / thickly)
  But not: it is raining strongly / powerfully / deeply / robustly / hardly
- der Mann ist stark → the man is strong / powerful
  But not: the man is robust / potent / heavy / thick / deep / hard
  (some combinations are possible but have different meaning)

n-grams (within windows) provide a simple way to identify the right word combinations (and also the right inflection, e.g., is it thick or thickly?). If we analyze an entire corpora of English books, we may find that the combination “rain, powerful” is less frequent than “rain, hard”. From that, we may infer the right word in the context. To simulate that process, write a Python script that takes the beginning of a sentence and completes it with the most frequent n-grams it finds in an example text (e.g., the Sherlock Holmes book referred to below). Use 3-grams and 4-grams, and match all but the last terms with the end of the sentence and extend the sentence with the most frequent matching n-gram. Repeat until you run into a punctuation (don’t eliminate punctuations). Look at the results!

Task 2: NLTK and Python (practical)

Getting started with NLTK (see also: http://www.nltk.org/install.html)

1. Install Python (https://www.python.org)
   - Ubuntu: `sudo apt-get install -y python3-pip python3-dev`
   - Windows: install python version (including pip)
2. Install Python packages with `pip` (or `pip3`) – use PowerShell on Windows
   - `pip install --upgrade pip`
   - `pip install -U numpy`
   - `pip install -U nltk`
   - `python -m nltk.downloader all` (alternative data download below)
3. Run `python` (or `python3`) – use PowerShell on Windows
   - `import nltk`
   - Alternative data download: `nltk.download()` [select all in dialog]
   - …write your commands (see below)

References:

- NLTK: http://www.nltk.org
- NLTK Book: http://www.nltk.org/book/
  - best source to find snippets of Python code for NLTK
- Python: https://www.python.org/doc/

How to get started with the exercise: (don't forget to `import nltk`)

- Read file from local folder (e.g., http://www.gutenberg.org/files/244/244-0.txt)
  - `f=open('stud.txt')`
  - `text=f.read()`
  - `f.close()`
- A few lines from the demo during the course:

```
import nltk
sentences=nltk.sent_tokenize(text)
tokens=nltk.word_tokenize(text)
words=[word.lower() for word in tokens if word.isalpha()]
bigram_measures=nltk.collocations.BigramAssocMeasures()
finder=nltk.collocations.BigramCollocationFinder.from_words(words)
finder.nbest(bigram_measures.pmi, 20)
finder.score_ngrams(bigram_measures.pmi)
nltk.pos_tag(tokens)
nltk.FreqDist(tag for (word, tag) in nltk.pos_tag(tokens)).most_common()
porter=nltk.PorterStemmer()
porter.stem("house")
nltk.corpus.wordnet.synsets("dog")
nltk.corpus.stopwords.words("English")
nltk.corpus.stopwords.words("german")
nltk.corpus.stopwords.words("Italian")
```

Task 3: LSI & Naïve Bayes (practical)

Latent Semantic Indexing and Naives Bayes are simple yet powerful methods in this exercise, we try to guess the language and the author of a text. For that purpose, download a few texts from project Gutenberg (or any other source). Tokenize the texts into sentences and work on the sentence level.

a) Train an LSI and Naïve Bayes model to classify the language. Unlike in the previous task, do not use the stop word list but label the sentences and apply it on test set. What is the advantage compared to the stop list approach in the previous task?

b) Train an LSI and Naïve Bayes model to classify authorship. Can you identify authorship? Does it work at the sentence level? Try with bigger chunks of text and compare quality.