In [ ]:
```
from gensim.scripts.glove2word2vec import glove2word2vec
from gensim.models import Word2Vec, KeyedVectors
from nltk.corpus import wordnet as wn
import nltk
import numpy as np
from scipy.spatial.distance import cosine
import operator
```

```
/opt/anaconda3/envs/anlp/lib/python3.8/site-packages/gensim/similarities/__init__.py:15: UserWarning: The gensim.similarities.levenshtein submodule is disabled, because the optional Levenshtein package <https://pypi.org/project/python-Levenshtein/> is unavailable. Install Levenhstein (e.g. `pip install python-Levenshtein`) to suppress this warning.
  warnings.warn(msg)
```

In [ ]:
```
glove_file="../data/glove.6B.100d.100K.txt"
original_file="../data/glove.6B.100d.100K.w2v.txt"
n, dimension = glove2word2vec(glove_file, original_file)
```

```
/var/folders/xn/nc0wg5m94rs8md2f73q98w5r0000gn/T/ipykernel_9938/3320898960.py:3: DeprecationWarning: Call to deprecated `glove2word2vec` (KeyedVectors.load_word2vec_format(.., binary=False, no_header=True) loads GLoVE text vectors.).
  n, dimension = glove2word2vec(glove_file, original_file)
```

In [ ]:
```
wv = KeyedVectors.load_word2vec_format(original_file, binary=False)
```

Q1: Implement the Lesk algorithm using word vectors (Basile et al. 2014), where we measure the similarity between a gloss g = $\{g_1, \ldots, g_G\}$ and context c = $\{c_1, \ldots, c_C\}$ as the cosine similarity between the sum of distributed representations:

$$\cos\left(\sum_{i=1}^{G} g_i, \sum_{i=1}^{C} c_i\right)$$

- The gloss for a synset can be found in `synset.definition()`; be sure to tokenize it appropriately.
- You can find the cosine *distance* (not similarity) between two vectors using the `scipy.spatial.distance.cosine(vector_one, vector_two)` function.
- `wn.synsets(word, pos=part_of_speech)` gets you a list of the synsets for a word with a specific part of speech (e.g., "n" for noun)

In [ ]:
```
def lesk(word, sentence, part_of_speech):
    # look up
    synsets_list = []
    synsets_ = []
    synsets=wn.synsets(word, pos=part_of_speech)
    for synset in synsets:
        string = str(synset.definition())
        synsets_list.append(nltk.tokenize.word_tokenize(string.lower()))
        synsets_.append(string)
        #print (synset, synset.definition())

    # build context
    context = nltk.tokenize.word_tokenize(sentence.lower())
    context = context[0:len(context)-1] #words to left of bank

    # building semantic vectors
    # synsets list
    synsets_embeddings = []
    for i in range(0, len(synsets_list)):
        intermediate_synsets_list = []
        for j in range(0, len(synsets_list[i])):
            intermediate_synsets_list.append(wv.get_vector(synsets_list[i][j]
        synsets_embeddings.append(sum(intermediate_synsets_list))
        # context embeddings [0] is summed embedding of [0] definiton in syns

    context_embeddings = []
    for i in range(0, len(context)):
        context_embeddings.append(wv.get_vector(context[i]))
    context_embedding = sum(context_embeddings)

    # cos similarities
    cosine_similarities = []
    for i in range(0, len(synsets_embeddings)):
        cosine_similarities.append(1-cosine(context_embedding, synsets_embedd

    # select highest cos score
    max_index = cosine_similarities.index(max(cosine_similarities))
    return synsets_[max_index]
```

Execute the following two cells to check whether your implementation distinguishes between these two senses of "bank".

In [ ]:
```
lesk("bank", "I deposited my money into my savings account at the bank", "n")
```

Out[ ]:
```
'a financial institution that accepts deposits and channels the money into len
ding activities'
```

In [ ]:
```
lesk("bank", "I ran along the river bank", "n")
```

Out[ ]:
```
'sloping land (especially the slope beside a body of water)'
```

In [ ]:
```
# my implementation correctly distinguishes between the two types of "bank".
```