# Assignment 1: Beam Search

The goal of the assignment is to implement ***beam search algorithm***. You need to extract a **directed weighted graph** from a corpus and perform beam search on this graph to generate the sentence with the highest score.

**Task 1: Extracting graph from a set of sentences.**

In this task, you will work on ExtractGraph.py

The sentence dataset locates in "**assign1_sentences.txt**"
Each line of file is one sentence,
    starting with "<s>",
    and end with "</s>".
Punctuations include only "," and "."
It is easy to obtain the words by
    splitting the sentence with white space.
Please keep the original cases (lowercase and uppercase).

Your codes should build a directed weighted graph from this dataset in the **ExtractGraph initialization step**.
Each node represents a word.
Each edge connecting a head word and a tail word.
The tail word is the next word of the head word.
The edge weight is the probability of the next word appearing after head word.

Also, you need to implement **getProb()**,
which reads the graph and returns the probability of the next word appearing after head word.

**Task 2: Implement Beam Search on the graph, to generate the sentences with max score.**

In this task, you will work on BeamSearch.py.

Basic beam search should be implemented as beamSearchV1(pre_words, beamK, maxToken). The function should accept these parameters:

- pre_words is the existing words in the sentence
  and you will need to predict the next and following words to finish the sentence.
- beamK is width of beam.
- maxToken is the maximum words of a valid sentence, including the pre_words.

Searched/generated sentence along with its score

should be returned in form of *StringDouble*.

The score is defined as:

$$score(y) = log\ P(y|x)\ ,$$

Where $y$ is the sentence, and $x$ is the pre_words.

$P(y|x)$ is the probability of the sentence given input pre_words.

$log$ function can help keep the accuracy of computation by replacing probability multiplication with log probability sum.

Resources:
https://www.youtube.com/watch?v=RLWuzLLSIgw
https://geekyisawesome.blogspot.com/2016/10/using-beam-search-to-generate-most.html

Then sentence length-normalization enhanced beam search should be implemented in beamSearchV2(pre_words, beamK, param_lambda, maxToken).

- pre_words is the existing words in the sentence
  and you will need to predict the next and following words to finish the sentence.
- beamK is width of beam.
- maxToken is the maximum words of a valid sentence, including the pre_words.
- param_lambda is the hyper parameter to control the sentence length normalization.

The length-normalization enhanced beam search is defined as

$$score(y) = \frac{1}{|y|^{\lambda}}\ logP(y|x),$$

Where |y| is the length of sentence, and $\lambda$ (param_lambda) is defined as 0.7 in main function.

Searched/generated sentence with its score
should be returned in form of *StringDouble*.

Search termination:

1. "<\s>" appears in the generated sentence.
   <\s> is the end of a sentence.
2. The count of words in the sentence is bigger than maxToken.

Paper related to length-normalization in beam search:
https://arxiv.org/pdf/1707.01830.pdf
https://arxiv.org/pdf/1609.08144.pdf%20(7.pdf

**Tips**: You may want to implement beamSearchV2() first,
then beamSearchV1() can directly call beamSearchV2() with param_lambda set as 0.

**Task 3: Discuss the effect of length-normalization module.**

Tune param_lambda, play with more pre_words, and compare the sentences from two versions of beam search to explore the effect of length-normalization.
Please write your findings and evidence in a **txt** file.
You can also read papers for reference, and paper citation is welcome.

**Requirements and Reminders**

1. You **CANNOT** change the classes' names or the required methods' names. However, you can add new variables, constants, and methods in these classes and create new classes if necessary.
2. **Assignment1Main** is the main class for running your assignment 1. You are **NOT** allowed to change anything in this file.
3. You **CAN ONLY** use Python in this assignment.
4. You **CANNOT** use external Python packages.

## Grading

Your submission will be graded based on the following:

1. Correctness of the implementation of the required functions (50%).
2. The efficiency of your implementation, try your best to improve your algorithm running time and memory space (15%).
3. Necessary program annotations and comments (10%).
4. A clear and comprehensive discussion from Task 3 (25%).

## Submission Requirements

A zipped file package with the naming convention as "PITTIDS_a1".
For example, suppose the PITT ID is jud1; then the submission package should be jud1_a1.zip.
The file package should contain the following:

1. All the scripts/programs you used for this assignment (**src folder**).
2. Your output on the screen. (This should be in **txt file**.)
3. Your discussion reports. (This should be in **txt file**.)

**Do not upload the assign1_sentences.txt.**