In [2]:

```
# 1. Importing the Necessar Modules

import nltk
import spacy
import numpy as np
import networkx as nx
from sklearn.metrics import jaccard_score
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import MultiLabelBinarizer
```

In [16]:

```
# 2. Downloading the Necessary Libraries and Modules

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('punkt_tab')

from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

In [5]:

```
# 3. Load spaCy model for vectorization

nlp = spacy.load("en_core_web_sm")
```

In [6]:

```
# 4. Tokenize documents into sentences

def tokenize_sentences(text):
    return sent_tokenize(text)
```

In [7]:

```
# 5. Preprocess each sentence

def preprocess_sentence(sentence):
    stop_words = set(stopwords.words('english'))
    words = word_tokenize(sentence.lower())
    return [word for word in words if word.isalnum() and word not in stop_words]
```

In [8]:

```
# 6. Extract key phrases using CountVectorizer

def extract_key_phrases(sentences):
    preprocessed_sentences = [' '.join(preprocess_sentence(s)) for s in sentence
    vectorizer = CountVectorizer().fit(preprocessed_sentences)
    key_phrases = vectorizer.get_feature_names_out()
    return key_phrases
```

In [11]:

```
# 7. Jaccard Similarity Matrix between sentences and key phrases

def build_similarity_matrix(sentences, key_phrases):
    binarizer = MultiLabelBinarizer(classes=key_phrases)
    sentence_sets = [set(preprocess_sentence(s)) for s in sentences]
    binary_matrix = binarizer.fit_transform(sentence_sets)
    n = len(sentences)
    similarity_matrix = np.zeros((n, n))

    for i in range(n):
        for j in range(n):
            if i != j:
                similarity_matrix[i][j] = jaccard_score(binary_matrix[i], binary

    return similarity_matrix
```

In [12]:

```
# 8. Rank Sentences

def rank_sentences(similarity_matrix):
    graph = nx.from_numpy_array(similarity_matrix)
    scores = nx.pagerank(graph)
    return scores
```

In [13]:

```
# 9. Get summary

def textrank_summarize(text, summary_ratio=0.3):
    sentences = tokenize_sentences(text)
    key_phrases = extract_key_phrases(sentences)
    similarity_matrix = build_similarity_matrix(sentences, key_phrases)
    scores = rank_sentences(similarity_matrix)

    ranked_sentences = sorted(((scores[i], s) for i, s in enumerate(sentences)),

    top_n = int(len(sentences) * summary_ratio)
    summary = ' '.join([sent for _, sent in ranked_sentences[:top_n]])

    return summary
```

In [17]:

```
# 10. Implementing  the Model

text = """
Natural Language Processing (NLP) is a sub-field of artificial intelligence.
It involves understanding and generating human language.
One of the most interesting tasks in NLP is text summarization.
There are two main approaches to summarization: extractive and abstractive.
TextRank is an extractive summarization algorithm.
It is inspired by the PageRank algorithm used by Google.
TextRank builds a graph of sentences based on similarity.
Then, it ranks the sentences to pick the most important ones for the summary.
"""
print("Summary of the document :- ")
print(textrank_summarize(text))
```

```
Summary of the document :- 
TextRank is an extractive summarization algorithm. There are two main approaches
to summarization: extractive and abstractive.
```