Knowledge Graph:

- Structure of the Graph:
  1. Nodes represent entities (people, place, concepts)
  2. Edges: relationships between nodes
  3. Properties: Attributes of nodes and edges (date, entity names)

- So usually the graph is pre-made then the system finds those relevant nodes, etc. to input into retrieval part of the generation. So the most important part for RAG is the retrieval part.
  - We get the relevant info from
    1. entity recognition: NER (Name Entity Recog.) (Part of entity linking)
       An NLP technique to identify entity mentions in the input query. Use pre-trained for this (e.g., spaCy, NLTK, or transformer [Bert]).
    2. Candidate Generation: After (1) which helps generate the graph we generate similar entities from the graph to the query.
       - Contextual similarity: finding the similarity in context of entities and context of the query
       - Emedding: cosine similarity
    3. Disambiguation: Selects the most relevant entity from the ranked candidates so use contextual similarity here.

- Graph Traversing:
  - Just use dijkstra

Vector Similarity:

— Given a query vector, vector similarity search compares high-dimensional vectors to find those that are most similar to the query vector.

— The documents are the high-dimensional vectors

· Vector Representation:

- Embeddings: Using models like Bert, Word2Vec, and FastText the user-input is converted from words to the query vector.
- Dimensionality: Vectors have many (a lot) of dimensions, capturing the meaning of the items.
- How to similarity check:
  First, use an ANN, FAISS (the goat), to find the most similar vectors in the dataset. Then, use cosine similarity to find the most similar vector.

· RAG integration:

— Retrieval: retrieves relevant documents using similarity search

— Generation: Using a generation model using the retrieved documents to generate a response