Graph Problem: **Shortest-path**

The shortest path problem is a fundamental problem in graph theory that involves finding the shortest path between two nodes (vertices) in a graph. The problem is defined as follows: given a weighted graph and two nodes in the graph, find the shortest path between the two nodes that minimizes the sum of the weights of the edges in the path.

There are various algorithms for solving the shortest path problem, and the choice of algorithm depends on the specific problem and the properties of the graph. Some of the most commonly used algorithms include Dijkstra's algorithm, the Bellman-Ford algorithm, and the Floyd-Warshall algorithm.

*def* getRecommendMusicByName(*self*, *musicTitle*):

        # Define the features to use for similarity calculation

        features = ['beats.per.minute', 'energy', 'danceability',

'loudness.dB', 'liveness', 'valance', 'length', 'acousticness', 'speechiness']

        # Normalize the features

        self.df[features] = self.df[features].apply(*lambda* *x*: (x - x.mean()) / x.std(), *axis*=0)

        # Compute the pairwise similarity matrix using cosine similarity

        sim\_matrix = np.zeros((len(self.df), len(self.df)))

        for i in range(len(self.df)):

            for j in range(len(self.df)):

                if i != j:

                    vec1 = self.df.loc[i, features].values

                    vec2 = self.df.loc[j, features].values

                    sim\_matrix[i, j] = np.dot(vec1, vec2) / (np.linalg.norm(vec1) \* np.linalg.norm(vec2))

        # Define a function to compute the shortest paths between two nodes in a graph

*def* shortest\_paths(*matrix*):

            n = matrix.shape[0]

            for k in range(n):

                for i in range(n):

                    for j in range(n):

                        if matrix[i, k] + matrix[k, j] < matrix[i, j]:

                            matrix[i, j] = matrix[i, k] + matrix[k, j]

            return matrix

        # Compute the shortest paths between all pairs of songs

        dist\_matrix = shortest\_paths(1 - sim\_matrix)

        # Define a function to get recommendations based on a song

*def* get\_recommendations(*title*, *data*=self.df, *dist\_matrix*=dist\_matrix, *top\_n*=5):

            # Get the index of the song that matches the title

            idx = data[data['title'] == title].index[0]

            # Get the shortest distances to all other songs

            distances = dist\_matrix[idx, :]

            # Sort the songs based on the shortest distances

            sorted\_distances = sorted(enumerate(distances), *key*=*lambda* *x*: x[1])

            # Get the top n most similar songs

            song\_indices = [i[0] for i in sorted\_distances[1:top\_n+1]]

            # Return the titles of the top n songs

            return data['title'].iloc[song\_indices].tolist()

        # Example usage

        return get\_recommendations(musicTitle)

**getRecommendMusicByName** which takes a music title as input and returns a list of recommended music titles based on their similarity to the input music.

To compute the similarity between music tracks, the code uses a cosine similarity measure on a set of audio features such as beats per minute, energy, danceability, loudness, liveness, valence, length, acousticness, and speechiness. The features are first normalized by subtracting the mean and dividing by the standard deviation. Then, a pairwise similarity matrix is calculated using the cosine similarity measure.

To find the shortest path between all pairs of music tracks, the code defines a function shortest\_paths that computes the Floyd-Warshall algorithm for finding the shortest path between all pairs of nodes in a graph. This algorithm iteratively updates the distance matrix by checking if there is a shorter path through each intermediate node. The distance matrix is then subtracted from 1 to convert distances to similarities.

Finally, to get the recommendations for a given music title, the code defines a function get\_recommendations that takes the music title, data, and distance matrix as inputs. It then finds the index of the input music track and calculates the shortest distance to all other music tracks using the distance matrix. The distances are sorted, and the top n most similar tracks are returned as a list of music titles.

Regarding the graph problem, the code defines a graph problem of finding the shortest path between all pairs of nodes in a complete graph where each node represents a music track, and the edges represent the similarity between the tracks. The problem is solved using the Floyd-Warshall algorithm, which has a time complexity of O(n^3), where n is the number of nodes in the graph. In this case, the time complexity is O(n^3), which can be computationally expensive for large datasets.