Passo a passo para entender o funcionamento dos algoritmos

A função generate recebe um dicionário locations como entrada, onde as chaves representam o nome das localidades e os valores representam suas coordenadas geográficas. A função então processa esses dados e retorna uma lista otimizadedList contendo as localidades otimizadas e seus links do Google Maps.

Aqui está uma explicação passo a passo do algoritmo:

1. **Validação das Coordenadas:**
   * A função parse\_coordinates verifica se o valor de cada localidade é uma string.
   * Se for uma string, tenta separá-la por vírgulas e converter cada elemento em um número flutuante (latitude e longitude).
   * Coordenadas inválidas são ignoradas e uma mensagem é impressa.
2. **Criação da Matriz de Coordenadas:**
   * Um array NumPy coords é criado contendo apenas as coordenadas válidas extraídas do dicionário locations.
   * Se o número de localidades válidas for menor que o número total de localidades, uma mensagem é impressa informando que algumas localidades foram ignoradas.
3. **Cálculo da Matriz de Distâncias:**
   * A função distance\_matrix do pacote scipy.spatial é utilizada para calcular a distância entre todas as pares de coordenadas em coords.
   * O resultado é uma matriz quadrada dist\_matrix onde cada elemento dist\_matrix[i, j] representa a distância entre a localidade i e a localidade j.
4. **Função total\_distance:**
   * Esta função calcula a distância total de uma rota específica.
   * Ela percorre a rota (uma lista de índices de localidades) e soma as distâncias entre localidades consecutivas na matriz dist\_matrix.
   * A distância de partida (do primeiro ponto para o segundo) e a distância de chegada (do último ponto para o primeiro) também são consideradas.
5. **Algoritmo de Rota Otimizada (Exaustivo):**
   * A função rotaOtimizada utiliza o módulo itertools para gerar todas as permutações possíveis das localidades intermediárias (excluindo a primeira e a última).
   * Para cada permutação (rota), a função total\_distance é usada para calcular a distância total.
   * A permutação com a menor distância total é retornada como a rota otimizada.
6. **Algoritmo de Aproximação (Nearest Neighbor):**
   * A função nearest\_neighbor implementa um algoritmo heurístico chamado "Vizinho Mais Próximo".
   * Ele recebe um índice inicial e a matriz de distâncias como parâmetros.
   * Começando pela localidade inicial, ele iterativamente adiciona a localidade não visitada mais próxima à rota.
   * O processo continua até que todas as localidades sejam visitadas.
   * Este algoritmo é menos preciso que o algoritmo exaustivo, mas é muito mais rápido para grandes quantidades de localidades.
7. **Função rotaOtimizadaComTimeout:**
   * Esta função tenta encontrar a rota otimizada usando o algoritmo exaustivo (rotaOtimizada).
   * No entanto, ela define um limite de tempo (timeout) de 5 segundos.
   * Se o algoritmo exaustivo não encontrar a rota otimizada dentro do tempo limite, a função muda para o algoritmo de aproximação (nearest\_neighbor) e retorna a rota gerada por ele.
8. **Gerando a Lista Otimizada:**
   * A função principal generate chama rotaOtimizadaComTimeout para encontrar a rota.
   * Dependendo do algoritmo utilizado (exaustivo ou aproximação), a lista optimized\_localities é criada contendo as localidades na ordem otimizada.
   * Finalmente, a função gera a lista otimizadedList contendo o nome de cada localidade otimizada e o link correspondente no Google Maps.

**Observações:**

* O algoritmo exaustivo pode se tornar muito lento para um grande número de localidades devido ao crescimento exponencial do número de permutações.
* O algoritmo de aproximação (Nearest Neighbor) é mais rápido, mas pode não encontrar a rota ideal.
* A função rotaOtimizadaComTimeout permite balancear precisão e tempo de execução.