Solução 1. Backtracking

****Vide o arquivo anexado "backtracking.c" contendo o algoritmo comentado.**

Resumo:

Assumimos que todas as cidades são numeradas de [*1, 2, … , n],* e que nós temos as distâncias entre elas dadas por uma matriz de custos. Logo, a distância entre as cidades *i* e *j* poderia ser facilmente encontrada - *CostMatrix[i][j]*.

Implementamos um simples algoritmo do tipo *força-bruta* para solucionar o *TSP* (*Traveling Salesman Problem*). Tal algoritmo gera todas as soluções válidas - no caso todos os caminhos válidos - uma por uma, e retorna aquela de menor custo. A estratégia imediata para gerar todas as rotas é por meio de um algoritmo recursivo.

Portanto, para encontrar o caminho mais curto, retornando ao ponto de partida, iniciamos a jornada do caixeiro viajante na cidade 1 (ou no caso do nosso algoritmo, cidade zero). Fixamos a cidade 1 e calculamos todas as permutações possíveis entre as cidades restantes. Dentre todas elas, devolvemos a de custo mínimo.

Apenas para efeito de exemplo, sejam as cidades a serem visitadas: [1, 2, 3, 4 e 5]. Fixamos a cidade 1 (cidade de partida) e permutamos as demais. Como resultado teríamos: "12345", "13245", "14352" etc. Dentre todas as combinações possíveis das cidades [2, 3, 4, …, n], retornamos aquela de peso mínimo. Evidente que no custo total é contabilizado o custo de retorno à cidade inicial.

Complexidade:

Nosso algoritmo sempre fixa a cidade 1 como a de partida e, posteriormente, gera todas as permutações possíveis das cidades remanescentes [2,…,n] - isto é, para o restante da rota. Com isso há (n - 1)! permutações e, para cada uma delas, nós gastamos um tempo O(n) para computar o comprimento (custo) total do tour correspondente. Por fim, nós teriamos um tempo total de n.(n - 1)! e, portanto, o algoritmo possui complexidade O(n!). Para concluir, somos conscientes que essa é

uma forma completamente fácil de encontrar uma solução, entretanto, não encontramos uma solução ótima.