Algoritmos e Estrutura de Dados I

Kayky Sena
Miller Anacleto
Jorge Salhani

I. Descrição do Problema

Neste trabalho abordaremos o Problema do Caixeiro Viajante utilizando contextos de tipos abstratas de dados (TADs) estudados em sala de aula. A citar: Pilhas, Filas e Listas.

Em resumo, o problema consiste em encontrar um caminho capaz de visitar uma única vez um conjunto N de cidades de modo que o trajeto percorrido tenha a menor distância. As considerações para a modelagem do problema são as seguintes:

1) Existem N cidades, representadas por $1, 2, ..., N$;
2) Entre duas cidades quaisquer pode não existir conexão ou, caso exista, este caminho é único entre elas;
3) Cada caminho possui uma distância associada;
4) O percurso é fechado, ou seja, inicia-se na mesma cidade em que deve ser finalizado;
5) A solução deve encontrar o melhor trajeto por abordagem de ”força bruta”, com solução exata.

Nas próximas seções trataremos da modelagem da solução, seguida da análise de complexidade do algoritmo utilizado.

II. Modelagem da Solução

Em primeiro lugar, vamos considerar o formato em que as informações dos trajetos são apresentadas. Vamos supor que tenhamos 4 cidades $\{1, 2, 3, 4\}$. Sendo δ a distância entre um par de cidades, o mapa consiste nos elementos:

4 ← número de cidades
1 ← cidade inicial
1 2 5 ← conexão 1-2, $\delta = 5$
1 4 1 ← conexão 1-4, $\delta = 1$
1 3 7 ← conexão 1-3, $\delta = 7$
2 3 2 ← conexão 2-3, $\delta = 2$
3 4 3 ← conexão 3-4, $\delta = 3$

Utilizando este mapa, o modelo proposto segue:

Em primeiro lugar, sendo $i = 1, 2, ..., N$ e $j = 1, 2, ..., N$ cada uma das cidades, tais que $i \neq j$, para cada cidade i devemos organizar uma lista de cidades j que conectam-se com i, acompanhado de sua respectiva distância. A esta estrutura damos o nome LISTA.

Cada elemento de LISTA deve conter as informações da cidade de referência (fixa) i, o número total de conexões a partir dela, todas as conexões $i - j$ e suas respectivas distâncias δ. A esta estrutura damos o nome CIDADE.

Na figura 1 representamos as dependências e os tipos das variáveis contidas em ambos os TADs mencionados. Já na figura 2 temos uma representação visual (em formato de grafo) do mapa cujas conexões estão exemplificadas acima, junto à sua respectiva LISTA.

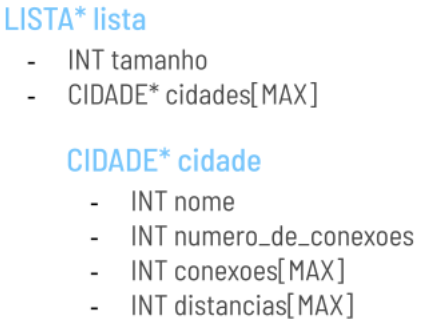

Figura 1: Representação esquemática das dependências do TAD lista LISTA, assim como seus tipos. Este TAD é composto pelo TAD CIDADE.

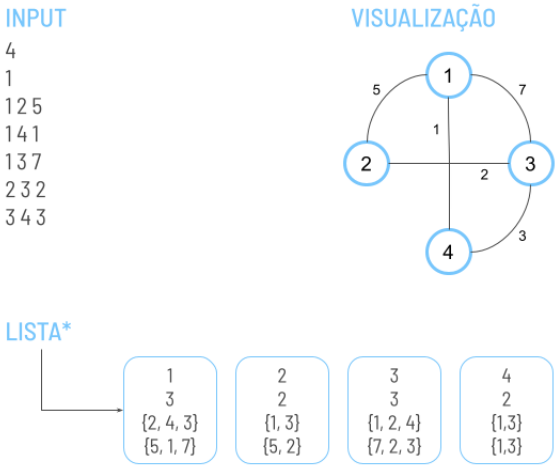

Figura 2: Exemplo de *input* padrão a ser lido pelo código em conjunto com sua representação gráfica. Também representamos o TAD LISTA gerado a partir do *input* exemplo. A ordenação dos valores que aparecem em cada componente de LISTA (i.e. cada elemento CIDADE) segue a ordem, de cima para baixo: nome, numero_de_conexoes, conexoes, distancias; conforme ordenado na representação da figura 1.

Temos aqui todas as informações necessárias para utilizar diversos modelos capazes de realizar a busca exaustiva sobre todas as configurações de caminhos e escolher o mais adequado. Em nossa solução, utilizaremos a estrutura de Pilha. Descreveremos então o funcionamento da busca utilizando um TAD Pilha para que, em seguida, possamos pontuar vantagens e desvantagens desta escolha.

Com a estrutura do tipo pilha podemos realizar a busca do melhor caminho de modo semelhante à uma busca de profundidade. Utilizando o exemplo da figura 2, o algoritmo é executado como segue:

Dada uma cidade inicial $i = 1$, suas conexões $c_k \in C$, tal que $k = 1, 2, 3$ e $C = \{2, 4, 3\}$ e respectivas distâncias $\delta_k \in D$, tal que $D = \{5, 1, 7\}$ e $k = 1, 2, 3$, criamos m elementos para cada uma das conexões. O primeiro elemento contém as informações $i = 1, c_1 = 2, \delta_1 = 5$. Chamamos esta estrutura de VIAGEM. Nesta estrutura também adicionamos um TAD LISTA visitadas que armazena as cidades já visitadas, que explicaremos o motivo em seguida.

Para cada conexão $c_k \in C; k = 1, 2, 3$ adicionamos seu respectivo elemento do tipo VIAGEM em um TAD do tipo PILHA. Neste sentido, para a cidade $i = 1$, temos a pilha:

$$\begin{aligned} \{1, 3, 7\} \quad & \leftarrow \text{topo} \\ \{1, 4, 1\} \quad & \\ \{1, 2, 5\} \quad & \leftarrow \text{base} \end{aligned}$$

Com a pilha inicial, caminhamos para a cidade destino do topo da pilha, isto é, para a cidade $i = 3$, percorrendo uma distância $\delta = 7$ (e removemos esta conexão da pilha). Estando nessa nova cidade, executamos um empilhamento análogo ao anterior incrementando, à distância atual, qual a distância acumulada ao longo do percurso. Agora, porém, é importante verificar se a próxima cidade já não foi visitada para evitar ciclos ou contagem dupla. Fica evidente agora o motivo da existência do TAD LISTA dentro de VIAGEM. A estrutura do TAD pilha PILHA pode ser melhor analisado na figura 3. Assim, sendo as conexões da cidade $i = 3$ $C = \{1, 2, 4\}$ e suas respectivas

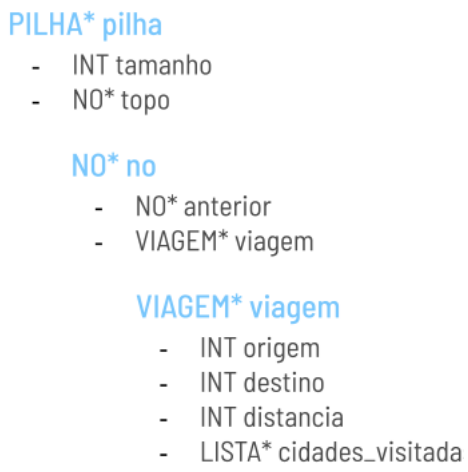

Figura 3: Representação esquemática das dependências do TAD pilha PILHA, assim como seus tipos. PILHA é composto pelo TAD NO, que por sua vez contém o TAD VIAGEM que contém o TAD LISTA. Este último, já definido, pode ser visualizado na figura 1.

distâncias $D = \{7, 2, 3\}$, temos a pilha (agora com as cidades visitadas)

$$\{3, 4, 10\} \quad [1, 3] \qquad \leftarrow \text{topo}$$
$$\{3, 2, 9\} \quad [1, 3]$$
$$\{1, 4, 1\} \quad [1]$$
$$\{1, 2, 5\} \quad [1] \qquad \leftarrow \text{base}$$

Em sequência, o processo se repete. Estamos agora na cidade $i = 4$ com $C = \{1, 3\}$ e, respectivamente, $D = \{1, 3\}$. Removemos o topo da pilha, dado pelo trajeto $3 \rightarrow 4$, e temos agora

$$\{4, 1, 11\} \quad [1, 3, 4] \qquad \leftarrow \text{topo}$$
$$\{3, 2, 9\} \quad [1, 3]$$
$$\{1, 4, 1\} \quad [1]$$
$$\{1, 2, 5\} \quad [1] \qquad \leftarrow \text{base}$$

Na próxima iteração, estaremos na cidade $i = 1$ inicial. Caso todas as cidades tenham sido visitadas, armazenamos a distância percorrida, caso seja menor que as anteriores ou caso seja a primeira encontrada, e desempilhamos esta conexão.

O percurso seguinte (i.e. conexão: $\{3, 2, 9\}$) nos leva para a cidade $i = 2$, cujas conexões e distâncias são $C = \{1, 3\}$, $D = \{5, 2\}$. Temos então a pilha

$$\{2, 1, 14\} \quad [1, 3, 2] \qquad \leftarrow \text{topo}$$
$$\{1, 4, 1\} \quad [1]$$
$$\{1, 2, 5\} \quad [1] \qquad \leftarrow \text{base}$$

E assim por diante, até que todos os trajetos sejam verificados e o menor caminho contabilizado, caso todas as cidades tenham sido necessariamente visitadas e apenas uma única vez.

A evolução do percurso no empilhar/desempilhar do TAD pilha pode ser observado na figura 4, onde podemos concluir que apenas um caminho perpassa todas as cidades, e sua distância é igual a 11.

III. Discussão

Em analogia às estruturas estudadas em aula, temos o paralelo entre o TAD ITEM $\leftrightarrow$ CIDADE. Os demais TADs utilizam mesma nomenclatura usual, ou seja, LISTA e PILHA.

Em primeiro lugar, um TAD CIDADE (ou ITEM) permite a aglutinação de diferentes tipos base de dados em um único elemento, tais como *int* e *int**. Neste mesmo sentido, quando definimos um TAD LISTA que comporta elementos do tipo CIDADE, podemos armazenar uma sequência de elementos na memória. O mesmo vale para o caso do TAD VIAGEM definido para este projeto.

Destacamos aqui o uso de uma lista sequencial no TAD LISTA. Como sabemos que o número de cidades é pequeno (menor que 12, por conta do mecanismo de força bruta ser computacionalmente demorado em casos de

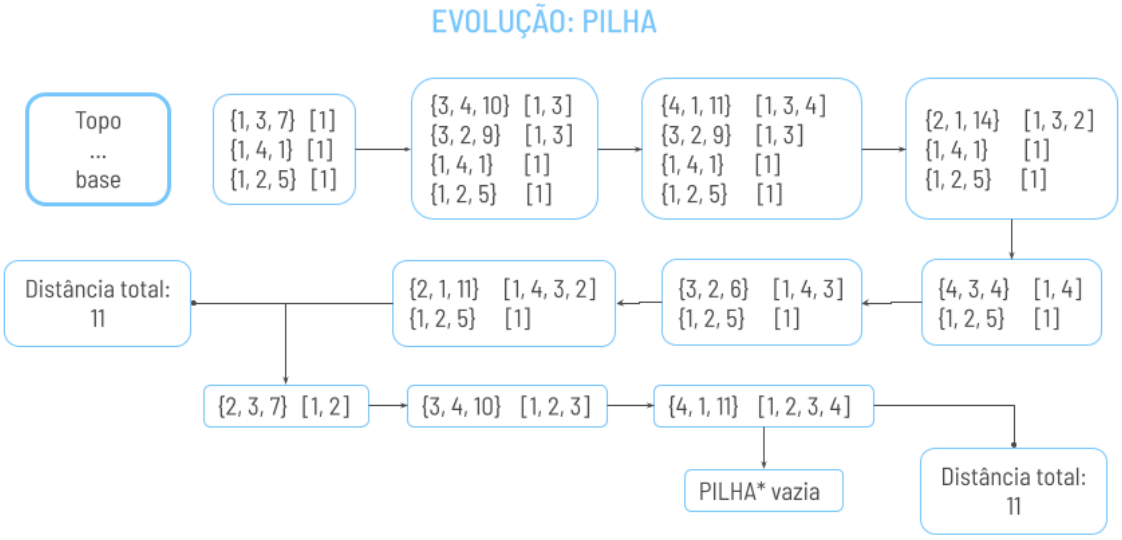

Figura 4: Evolução do TAD pilha PILHA conforme o percurso é mapeado pelo caminhante. O empilhar/desempilhar utiliza o mapa exemplo dado no início deste trabalho. O primeiro quadro retangular em destaque indica que, em cada quadro, o topo da pilha está na posição mais acima e a base, na posição mais abaixo.

maior número) e invariável, não há expressiva perda de memória caso não seja utilizadas todas as 12 posições possíveis, e sua facilidade de implementação fez dessa implementação a mais adequada.

Quando, por outro lado, definimos o TAD PILHA, optamos por uma implementação encadeada ou invés de sequencial. Esta escolha deve-se aos fatos que seguem abaixo.

Em primeiro lugar, o número de conexões pode ser muito maior que o número de cidades (por exemplo em grafos completamente conexos, onde todas as cidades se conectam, para n cidades temos $n!/2$ conexões, considerando conexões de via única) e como cada elemento empilhado representa uma conexão percorrida pelo viajante, o uso de memória não sequencial é vantajoso neste cenário.

Outro ponto importante é a alta frequência de remoções e inserções na pilha. Como a cada nova cidade visitada existirá um conjunto de novas conexões que podem ser empilhadas, o tamanho desta estrutura varia consideravelmente. Como o acesso se dá apenas ao primeiro elemento, com operações empilhar/desempilhar com complexidade $\mathcal{O}(1)$, o fator mais relevante está na memória alocada e na variação do tamanho da pilha.

Por fim, o uso de uma estrutura PILHA foi escolhido por conta da possibilidade de criar (empilhar) e desconsiderar (desempilhar) caminhos que são ramificações de um caminho pré-definido. Por exemplo, acompanhando a visualização da figura 2, dada a cidade inicial $i = 1$ sabemos que existem 3 caminhos possíveis. Destes, vamos supor que seja escolhido o caminho $1 \rightarrow 3$. Agora, nos restam apenas dois. Caso a próxima cidade seja 2, nos resta voltarmos para $i = 1$, que conclui esta bifurcação. Precisamos agora fazer um caminho reverso até que pos-

samos escolher pela bifurcação não escolhida, neste caso o caminho $3 \rightarrow 4$ ao invés de $3 \rightarrow 2$.

Estas decisões sucessivas entre bifurcações que foram acumuladas ao longo do caminho podem ser modeladas com uso de pilhas justamente por facilitar a retomada de ramificações cujos caminhos ainda não foram escolhidos. Por isso comparamos este tipo de busca à busca em profundidade realizada comumente em estruturas de árvore.

Nesta parte final, vamos apresentar os resultados de complexidade do algoritmo em relação ao tempo de execução para diferentes grafos com crescente número de cidades e conexões.

O gráfico apresentado na figura 5 representa o tempo de execução do algoritmo para um número crescente de cidades completamente conexas, ou seja, cada cidade i possui conexões com todas as demais j ($i \neq j$). É interessante notarmos que, como esperado, o tempo de execução segue, em ordem de grandeza, a complexidade do algoritmo $\sim \mathcal{O}(n!)$.

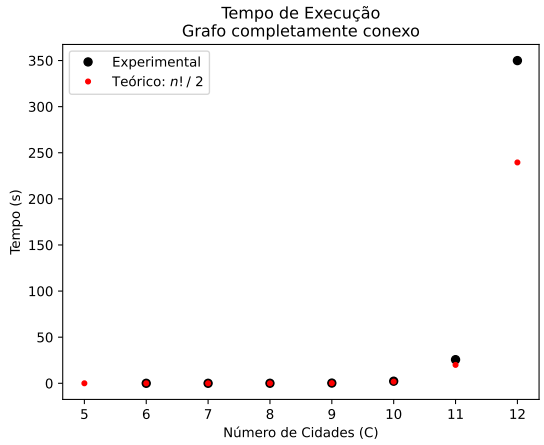

Figura 5: Relação entre o tempo de execução (segundos) e o número de cidades (C) considerando grafos (ou mapas) completamente conexos. Os círculos maiores em preto representam os dados obtidos experimentalmente e os pontos menores em vermelho, o respectivo resultado teórico para $n!/2$, considerando normalização por 10^6.

Também vale a pena destacar o fator multiplicativo 2×10^6. O fator 10^6 é considerado para que aproximar a curva teórica com o tempo de execução, dado em segundos, visto que ocorrem verificações de mais de um caminho por segundo. Já o fator 2 decorre da consideração de caminhos unidirecionais. Com isso, o caminho $2 \rightarrow 1$ é igual ao caminho $1 \rightarrow 2$, e para evitar contagem dupla, reduzimos na metade o número de combinações de trajetos entre as cidades consideradas.

Concluímos, portanto, os motivos de escolha das estruturas definidas ao longo deste trabalho, em destaque à utilização de estruturas em pilha e, também, a coerência entre os resultados experimentais do tempo computacional para a execução do algoritmo para busca ao longo de todo o espaço de possibilidades (força bruta) em relação à complexidade do algoritmo utilizado.