UNIVERSIDADE DE SÃP PAULO - USP
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO

O Problema do Caixeiro Viajante.

Bruno da Freiria Mischiati Borges
Daniel Coutinho Ribeiro
Yan Köhler de Araujo

São Paulo, 28 de outubro de 2021

Resumo

Esse trabalho apresenta uma solução pelo método de força bruta para o Problema do Caixeiro Viajante. Além disso, é apresentada a Estrutura de Dados utilizada na resolução do Problema e também é feita a análise de desempenho de todas as funções criadas para implementação do algoritmo de solução.

Sumário

1 INTRODUÇÃO 4

2 MODELAGEM DA SOLUÇÃO 5
2.1 Estrutura de Dados Escolhida 5
2.2 Vantagens e Limitações da Listas Dinâmicas Duplamente Encadeadas 5
2.3 Lógica utilizada para solução do problema 6

3 ANÁLISE DE COMPLEXIDADE 7
3.1 Análise de assintótica de complexidade das funções implementadas 7
3.2 Gráfico: Número de Cidades X Tempo 9

4 REFERÊNCIAS 10

1 Introdução

Este relatório contém a modelagem para solução do Problema do Caixeiro Viajante e a análise assintótica de complexidade computacional da solução proposta. Na modelagem, são explicitadas a lógica utilizada para resolução do Problema e a justificativa para o tipo de Estrutura de Dados utilizada na implementação da solução. Já na análise assintótica, é apresentada a complexidade de cada função empregada na implementação do algoritmo de solução (utilizando a notação Big-Oh).

2 Modelagem da Solução

Nesse capítulo será apresentado o tipo de Estrutura de Dados escolhido para resolução do Problema do Caixeiro Viajante, bem como as vantagens e limitações que essa estrutura apresenta. Além disso, também será apresentada a lógica empregada na resolução do problema proposto.

2.1 Estrutura de Dados Escolhida

A Estrutura de Dados escolhida para modelar o problema do Caixeiro Viajante foi a de Listas Lineares Dinâmicas Duplamente Encadeadas. A escolha dessa estrutura se deu pela necessidade de efetuar, constantemente, operações de inserção, remoção, busca e acesso nas listas elaboradas para solução do problema. Além disso, o método escolhido para solução do problema envolve a criação de inúmeros nós e de outras listas. Essas tarefas são predominantemente presentes – e facilitadas – pela Estrutura de Dados escolhida.

2.2 Vantagens e Limitações da Listas Dinâmicas Duplamente Encadeadas

A implementação das Listas Duplamente Encadeadas beneficia operações de remoção e inserção, além de torna-lás mais eficientes em relação ao tempo. Por outro lado, a implementação desse tipo de Estrutura de Dados é de alta complexidade.
As operações citadas são beneficiadas porque o tipo de implementação escolhida diminui a quantidade de variáveis auxiliares necessárias para essas operações. Além disso, acessar e percorrer as listas duplamente encadeadas é mais fácil, umas vez que é possível percorrê-la em dois sentidos: do começo para o fim e do fim para o começo.
Já o desempenho em relação ao tempo decorre do fato de que a lista utilizada permite que remoções e inserções ocorram mais rapidamente, pois os elementos da lista não estão em regiões contíguas de memória e, dessa forma, para acessá-los, basta utilizar referências.
Entretanto, como já citado, a limitação do tipo de Estrutura de Dados escolhido é o fato de que a implementação de uma lista duplamente encadeada é de alta complexidade, pois envolve a implementação de inúmeras referências entre os nós da lista. Ademais, o uso incorreto de referências pode ocasionar perda total ou parcial da lista.

2.3 Lógica utilizada para solução do problema

A lógica utilizada para solução do problema é descrita por dois processos: gerar todas as combinações de caminhos possíveis e, posteriormente, analisar qual caminho é o que possuí menor distância a ser percorrida.

Para gerar todas as combinações de caminhos possíveis foi utilizada a ordem lexicográfica. A ordem lexicográfica implica ordenação dos caminhos gerados e, dessa forma, também implica obter todas as combinações possíveis com garantia de sucesso.

Basicamente, a ordem lexicográfica é definida pela ordenação de um conjunto de elementos de acordo com uma ordenação lógica preestabelecida deles. Ou seja, supondo que cada cidade do problema do caixeiro viajante corresponda à um número, duas ordenações possíveis de uma rota seriam (considerando um total de 4 cidades, por exemplo): $1 \rightarrow 2 \rightarrow 4 \rightarrow 3$ e $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$. Mas, como $1 < 2 < 3 < 4$, seguindo a ordem lexicogŕafica, na ordem das permutações geradas, $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ seria gerado primeiro e $1 \rightarrow 2 \rightarrow 4 \rightarrow 3$ seria gerado depois. Com isso, as permutações das cidades são feitas de maneira lógica.

Posteriormente, uma vez geradas todas as permutações possíveis de todas as rotas possíveis é preciso escolher aquela com menor distância a ser percorrida. Para isso, dada uma rota qualquer, as distâncias entre cada cidade dessa rota são somadas e armazenadas. Assim, uma vez que todas as rotas foram obtidas e suas respectivas distâncias também, resta selecionar a rota com a menor distância (o que foi realizado através de um algoritmo de busca sequencial).

3 Análise de Complexidade

Nesse capítulo será feita a análise assintótica da complexidade temporal da solução proposta para o Problema do Caixeiro Viajante.Para isso, será apresentada uma tabela em que cada linha contem uma função e sua respectiva complexidade. Além disso, também será apresentado um gráfico com as medidas de tempo de execução em função do tamanho do conjunto de entrada.

3.1 Análise de assintótica de complexidade das funções implementadas

A partir da análise assintótica de cada função utilizada na solução do Problema do Caixeiro Viajante, foi possível definir a complexidade de cada uma dessas funções. A seguir, será apresentada uma tabela contendo cada função e sua respectiva complexidade computacional.

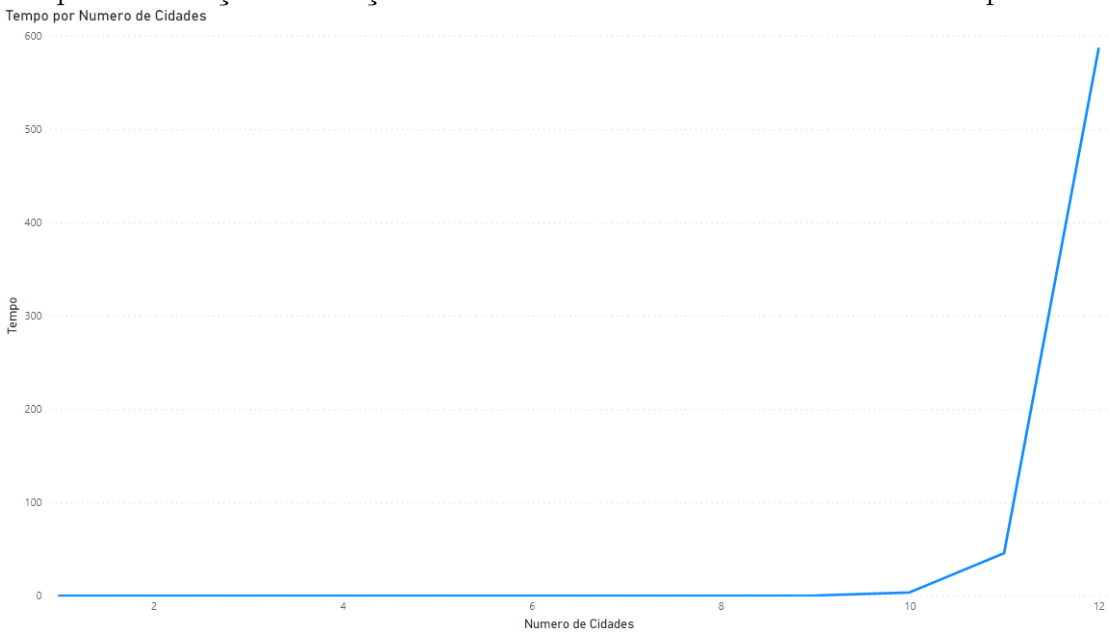
Analise da Complexidade Temporal	
Função	Complexidade
tsp_read_distance_file	n
tsp_solve_bruteforce	n!
distance_list_filter_out_start	n
distance_list_filter_start	n
distance_list_node_get_distance_from_to	n
distance_list_print	n
distance_list_is_full	1
distance_list_unshift	1
distance_list_insert_after	1
distance_list_insert_before	1
distance_list_push	1
distance_list_free	1
distance_list_delete_head	1
distance_list_delete_tail	1
distance_list_search_with_from_to	n
distance_list_search	n
distance_list_delete	1
distance_list_get_size	n
distance_list_set_head	1

Analise da Complexidade Temporal	
Função	Complexidade
distance_list_set_tail	1
distance_list_get_head	1
distance_list_get_tail	1
distance_list_is_empty	1
distance_list_new	1
distance_list_node_get_prev	1
distance_list_node_get_next	1
distance_list_node_set_prev	1
distance_list_node_set_next	1
distance_list_node_create	1
distance_list_node_free	1
distance_list_node_get_key	1
distance_list_node_set_key	1
distance_list_node_get_to	1
distance_list_node_set_to	1
distance_list_node_get_from	1
distance_list_node_set_from	1
distance_list_node_get_distance	1
distance_list_node_set_distance	1
distance_list_node_print	1
brute_force	n!
path_unshift	1
path_copy_to	n
path_populate	n^2
path_get_next_permutation	n^2
path_calculate_distance	n^2
path_insert_after	1
path_insert_before	1
path_print	n
path_print_with_start	n
path_is_full	n
path_push	1
path_free	1
path_swap	n
path_delete_head	1
path_delete_tail	1
path_search	n

Analise da Complexidade Temporal	
Função	Complexidade
path_delete	1
path_get_size	n
path_set_head	1
path_set_tail	1
path_get_head	1
path_get_tail	1
path_is_empty	1
path_new	1
path_node_get_prev	1
path_node_get_next	1
path_node_set_prev	1
path_node_set_next	1
path_node_create	1

3.2 Gráfico: Número de Cidades X Tempo

A partir da análise do tempo de execução do código desenvolvido para solução do problema proposto, foi possível criar um gráfico que apresenta as medidas de tempo de execução em função do número de cidades fornecidas como arquivo de entrada.



4 Referências

[1]. Wikipedia. Travelling salesman problem. Wikipedia, 2021. Disponível em: https://en.wikipedia.org/wiki/Travelling_salesman_problem. Acesso em: 01/10/2021.

[2]. FORISEK, Michal. How would you explain an algorithm that generates permutations using lexicographic ordering?. Quora, 2021. Disponível em: https://www.quora.com/How-would-you-explain-an-algorithm-that-generates-permutations-using-lexicographic-ordering. Acesso em: 01/10/2021.