Travelling Salesman Problem with Minimum Spanning Tree

by **Álvaro S. Alves** - *2023201643*

First Assignment for the discipline of `Técnicas de Busca e Ordenação (Search and Sorting Techniques)`.

Computer Science Class :computer:

Federal University of Espírito Santo (UFES)

Relatório

Introdução

Esse relatório está divido do em metodologia (que contém detalhes de implementação, análises e testes), conclusão e bibliografia.

Metodologia

Adiante, os detalhes acerca da metodologia de desenvolvimento usada.

Implementação O programa é dividido nos arquivos `tsp` (para o Travelling Salesman Problem), `mst` (para a Minimum Spanning Tree), `tour` (para funções do Tour), `utils` (funções gerais, usadas em todos os outros 4 módulos) e `main` (módulo principal), onde a implementação das funções e interface dos tipos abstratos de dados (TADs) se encontram em arquivos de extensão *.c* no diretório **source** e suas definições e protótipos se encontram em arquivos de extensão *.h* no diretório **include**.

TADs A escolha das estruturas de cada TAD foi feita após a análise das funcionalidades pedidas, visando facilidade, clareza e versatilidade para o programa, porém ainda mantendo a preocupação com a performance.

- **TSP**: TAD contendo o nome do problema, dimensão (quantidade de cidades) e lista de cidades. > *City*: TAD contendo ID e coordenadas euclidianas (x, y) de cada cidade.
- **Graph**: TAD utilizado para o grafo geral de cidades (arestas de uma cidade para todas as outras) e para a `MST` contendo quantidade de vértices, quantidade de arestas, lista de arestas e custo mínimo para percorrer todos os vértices (no caso da `MST`). > *Edge*: TAD contendo vértice de origem, vértice de destino e peso (tamanho) de cada aresta.
- **Subset**: TAD contendo ID da aresta pai do conjunto e seu ranking (para definir qual será o conjunto abrangente/pai).

- **Tour**: TAD contendo a quantidade de vértices e a lista de vértices visitados em ordem.

Funções Para a criação da Árvore Geradora Mínina, foi utilizado o Algoritmo de Kruskal, que ordena todas as arestas de um grafo por peso e escolhe as arestas com o menor peso que permitem traçar um caminho que ligue todos os vértices. Para a ordenação das arestas, optou-se pelo algoritmo QuickSort através da função `qsort` da biblioteca `stdlib`. Para a separação e escolha de arestas com base nos seus vértices, foi utilizado o conceito de conjuntos e métodos simples como `unionSet` e `findSet`.

Para o Tour, optou-se pela utilização de funções recursivas para empilhar os acessos às arestas da MST de forma a manter a clareza do código enquanto ocorre o caminhamento pela árvore.

Análise de Complexidade das Principais Funcionalidades A seguir, algumas das principais funções e sua complexidade com base na análise do código:

```
// O(N)
void readTSPFile(char *fileName, TravellingSalesmanProblem tsp);

// O(N * lg(N)) + O(N^2)
Graph buildMST(Graph graph);
// O(N)
bool isEdgeInGraph(Edge *edgesArray, int edgesAmount, int v1, int v2);
// O(N^2)
void calculateDistanceBetweenCities(TravellingSalesmanProblem tsp, Graph graph);
// O(N)
void writeMSTFile(char *fileSteam, Graph mst);

// O(M * N * P)
Tour buildTour(Graph mst);
// O(N)
void writeTourFile(char *fileSteam, Tour tour);
```

A maioria das funções acima, como as funções de escrita e leitura em arquivos e a função de consulta ao grafo, possuem uma complexidade linear (`O(N)`), que podem ser consideradas aceitáveis.
Porém, a função de construção da árvore conta com ordenação (`O(N * lg(N))`) e pesquisa no grafo geral (`O(N^2)`), assumindo uma complexidade muito alta que pode acarretar em lentidão na execução.
Por fim, a função de construção do tour assume grande complexidade por se utilizar de caminhamento pelos vértices (`O(M)`) enquanto pesquisa nas arestas do grafo geral (`O(N)`) e empilha sua execução via recursão (`O(P)`), assumindo uma complexidade ainda mais alta que a de construção da árvore.

Análise Empírica com Base em Testes A seguir, alguns resultados de execução (tempo em segundos):

```
berlin52
Read TSP Interval:         0.00010
Build Graph Interval:      0.00016
Build MST Interval:        0.00026
Build Tour Interval:       0.00028
Full Execution Interval:   0.00090
OPT TOUR LEN:    7544.365901904087
DEF TOUR LEN:   10203.849730879328
MY TOUR LEN:    10403.860360720962


eil101
Read TSP Interval:         0.00025
Build Graph Interval:      0.00116
Build MST Interval:        0.00168
Build Tour Interval:       0.00334
Full Execution Interval:  0.00683
OPT TOUR LEN:   642.3095357906022
MY TOUR LEN:    886.1115549507533
DEF TOUR LEN:   913.8415442402764


tsp225
Read TSP Interval:         0.00063
Build Graph Interval:      0.00678
Build MST Interval:        0.00813
Build Tour Interval:       0.01476
Full Execution Interval:  0.03092
OPT TOUR LEN:   3859.0
DEF TOUR LEN:   5276.38793936093
MY TOUR LEN:    5324.876857440186


a280
Read TSP Interval:         0.00055
Build Graph Interval:      0.00912
Build MST Interval:        0.00893
Build Tour Interval:       0.03144
Full Execution Interval:  0.05088
OPT TOUR LEN:   2586.7696475631606
MY TOUR LEN:    3480.2108772142674
DEF TOUR LEN:   3878.37390654869
```

```
pr1002
Read TSP Interval:       0.00144
Build Graph Interval:    0.06983
Build MST Interval:      0.20555
Build Tour Interval:     1.54358
Full Execution Interval: 1.84821
OPT TOUR LEN:  259066.6630526768
MY TOUR LEN:   351350.5630052579
DEF TOUR LEN:  359325.00846265815
```

```
d18512
Read TSP Interval:          0.01357
Build Graph Interval:      18.03470
Build MST Interval:       148.88872
Build Tour Interval:     9260.80673
Full Execution Interval: 9459.46471
MY TOUR LEN:   889565.7047727136
DEF TOUR LEN:  928233.6749880624
```

É notório que, para todos os casos, a leitura dos arquivos do TSP e a construção do grafo geral ocorreram de forma rápida.
A construção e escrita da MST também obteve bons resultados na maioria dos casos de teste (até o `pr1002`).
Porém, durante a execução do teste `d18512`, ficou perceptível a má performance do algoritmo para a construção do tour.

Conclusão

Após análises, é possível perceber que o gargalo da aplicação se encontra principalmente na construção do Tour, que executa em menos de 2 segundos para pouco mais de 1000 cidades, porém tem dificuldades ao processar 18000 cidades (mais de 2h executando).

Bibliografia

Abaixo, a bibliografia consultada para a compreensão do problema e sua possibilidade de solução.

Travelling Salesman Problem | Wikipedia
Kruskal Algorithm: Overview & Create Minimum Spanning Tree | Simplilearn
Kruskal's Minimum Spanning Tree (MST) Algorithm | Geeks for Geeks