

Instituto Politécnico de Setúbal

Escola Superior de Tecnologia de Setúbal

João Espada (202100660)

Tiago Silva (202000331)

Licenciatura em Engenharia Informática

Sistemas Operativos

Trabalho Prático #1

# **Descrições das soluções**

Foram realizadas 10 execuções para cada versão do programa em diferentes conjuntos de dados.

**Tempo de Execução**:

Até ao ficheiro com a matriz de 8 cidades, a versão avançada apresentou uma menor média de tempo até a identificação do melhor caminho, em comparação com a versão base.

No entanto, a partir do conjunto de dados com 8 cidades, a média de tempo da versão avançada tornou-se superior à da versão base.

**Número de Iterações**:

A versão avançada exibiu uma média de iterações significativamente maior em comparação com a versão base até ao ficheiro com 8 cidades.

Posteriormente, as diferenças entre os valores diminuíram, com a versão avançada mantendo uma ligeira vantagem em número de iterações.

**Encontrar o Caminho Ideal**:

Ambas as versões apresentaram resultados semelhantes em termos de frequência de identificação do caminho ideal.

# **Resultados Testes Versão Base**









# **Resultados Testes Versão Avançada**

# 



# 

# **Descrição do Comportamento dos Programas**

## **Funções gerais**

* initialize\_shared\_memory()

Função que inicia a memória partilhada e o semáforo.

* generate\_random\_path(int \*path, int size)

Inicializa um array que representa um caminho com números de cidade consecutivos e depois baralha o caminho. Este caminho é depois usado como base inicial do algoritmo.

* calculate\_distance(int \*path)

Calcula a distância total de um determinado caminho.

* exchange\_mutation(int \*path)

Implementa uma troca, seleciona aleatoriamente duas posições no caminho e troca as cidades nessas posições.

* get\_elapsed\_time()

Calcula e retorna o tempo em milissegundos desde o início do programa até ao fim.

* run\_algorithm(int process\_id, int num\_processes, int max\_time)

É a lógica principal do algoritmo. Começa por iniciar uma solução aleatória, faz a mutação e avalia a nova solução. Se forem encontradas melhorias atualiza a memória compartilhada.

* main(int argc, char \*argv[])

Inicializa o programa, lê o ficheiro onde está a matriz, inicia a memória compartilhada e cria o semáforo. Também imprime a melhor solução e a informação sobre esta e sobre o programa.

## **Versão Base**

* update\_shared\_memory(Solution \*solution)

Atualiza a memória partilhada com uma solução melhor. Usa um semáforo para garantir que apenas um processo pode atualizar esta memória de cada vez.

## **Versão Avançada**

* update\_shared\_memory(Solution \*solution)

Atualiza a memória partilhada com uma solução melhor. Usa um semáforo para a sincronização e envia um sinal ao processo pai para avisar da atualização.

* synchronize\_processes()

Responsável por sincronizar os processos filho, enviando um sinal para cada um deles.

* handle\_update\_signal(int signo)

É a função que trata e manipula o sinal. Chama a função s*ynchronize\_processes()* para sincronizar os processos filho.

## **Versão base VS avançada**

**Sinais**: A versão base não usa sinais para sincronizar os processos. A versão avançada usa um sinal que vai notificar o processo pai.

**Sincronização de Processos**: A versão base usa semáforos para sincronizar os processos, enquanto a versão avançada usa tanto semáforos como sinais.

**Processos filho**: Na versão base, cada processo filho executa o algoritmo de forma independente. Na versão avançada são criados da mesma forma, mas dentro do loop do algoritmo têm sempre passos de sincronização uns com os outros.