**RELATÓRIO FINAL TP FPAA – 11/12/2023**

Eric Rodrigues Diniz

Diego Machado Cordeiro

Luiz Felipe Vieira

O propósito deste relatório é destacar os resultados alcançados nas implementações dos algoritmos estudados ao longo do segundo semestre de 2023 no âmbito do curso de Engenharia de Software da PUC MINAS, especificamente na disciplina de Fundamentos de Projeto e Análise de Algoritmos.

Link repositório: <https://github.com/FelipeVieir4/TP-FPAA>

Abaixo contém uma tabela com o algoritmo implementado para resolver o problema de otimização de rotas e o responsável pela implementação.

Tabela de divisão dos trabalhos e implementação.

|  |  |
| --- | --- |
| Algoritmo | Responsável |
| Luiz Felipe Vieira | Back Tracking |
| Diego Machado Cordeiro | Divisão e Conquista |
| Eric Rodrigues Diniz | Programação Dinâmica |

Critério de avaliação: *Todo mundo fez tudo*

Cenários de testes:

\* Conjunto de rotas 1: 40;36;38;29;32;28;31;35;31;30;32;30;29;39;35;38;39;35;32;38;32;33;29; 33;29;39;28

\* Conjunto de rotas 2:32;51;32;43;42;30;42;51;43;51;29;25;27;32;29;55;43;29;32;44;55;29;53; 30;24;27

As próximas páginas contêm as análises de cada algoritmo e seus resultados.

**BACK TRACKING**

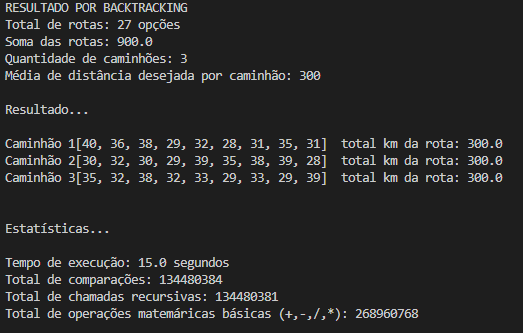
**Decisões de projeto.**

Atendendo aos requisitos estabelecidos para a **otimização de rotas**, visando a adequação da quantidade necessária de caminhões e um limite de **tempo de execução do código de no máximo 30 segundos**, o algoritmo foi desenvolvido conforme descrito abaixo.

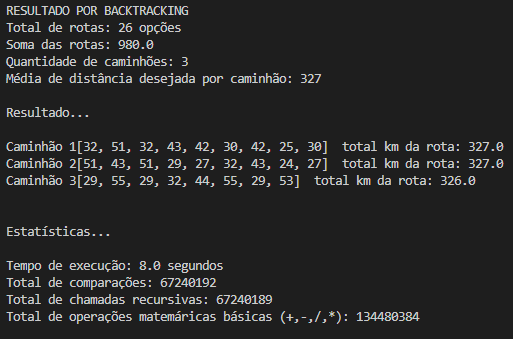
1. Definiu-se um alvo para cada caminhão somando a distância de todas as rotas e dividindo pela quantidade de caminhão.
2. Armazenou-se essas rotas de forma global na classe.
3. Então, executa-se o backTracking com o objetivo de identificar entre o conjunto de rotas um subconjunto cuja soma seja o mais próximo possível do alvo.
4. A poda ocorre caso o conjunto formado com a adição da próxima rota seja pior naquele momento do que o conjunto atual considerado como melhor. O Algoritmo analisa o módulo da diferença entre a solução atual e o alvo, e o módulo da nova solução e o alvo, escolhendo aquela que fornece um número menor, ou seja, mais próximo.
5. Após o subconjunto ser encontrado, ele é removido do conjunto de rotas disponíveis e então o código é executado para o próximo caminhão.

**Resultados após implementação.**

Resultado para “rotas 1”



Resultado para “rotas 2”



**Conclusão**.

Com a estratégia de poda atual, o algoritmo se aproximou consideravelmente da abordagem de força bruta, uma vez que não foi implementada uma solução de compromisso. Como resultado, o algoritmo explora praticamente todas as possibilidades do conjunto de rotas, evidenciado pelo número expressivo de comparações e chamadas recursivas no código, resultando em uma complexidade aparente de O()

Este cenário torna o código rapidamente inviável à medida que a entrada aumenta, dada a sua complexidade exponencial. Para ilustrar essa situação, ao comparar os exemplos de rota 2 e 1, observamos que a adição de apenas mais uma rota resultou em um aumento de 87,5% no tempo de execução, enquanto a quantidade de comparações e chamadas recursivas aumentou aproximadamente 100%.

Com a inclusão de uma rota adicional, o algoritmo já ultrapassou o limite de tempo estabelecido, atingindo o timeout de 30 segundos. Este comportamento indica a necessidade de otimizações para tornar o algoritmo mais eficiente e escalável diante de entradas mais complexas.

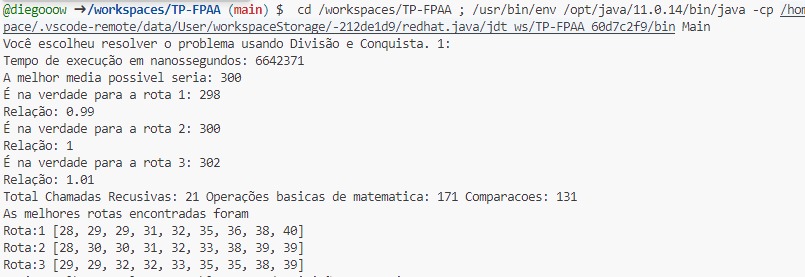
Uma opção considerada, embora ainda não implementada devido à restrição de tempo e dificuldades técnicas, seria incorporar uma solução de compromisso. Nessa abordagem, sempre que o código se aproximar do objetivo ou exceder ligeiramente (dentro de uma margem de erro aceitável), consideraria isso como uma resolução final para a rota. Além disso, propõe-se ordenar as rotas de forma decrescente, garantindo que as rotas mais longas sejam processadas primeiro, mitigando potenciais impactos negativos nas rotas subsequentes.

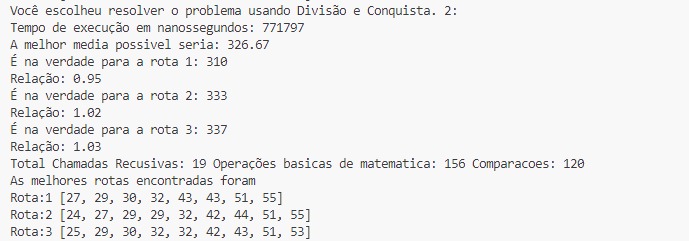
**DIVISÃO E CONQUISTA**

Esta técnica consiste em dividir um problema maior recursivamente em problemas menores até que o problema possa ser resolvido diretamente. Então a solução do problema inicial é dada através da combinação dos resultados de todos os problemas menores computados.

O código implementa a estratégia de divisão e conquista para otimização de rotas de caminhões. A função resolverProblemaDosCaminhoes recebe um array de rotas, o número de caminhões disponíveis e os índices de início e fim do subproblema atual. A abordagem adotada verifica se o número de rotas é menor ou igual ao número de caminhões, indicando que o subproblema é pequeno o suficiente para ser resolvido diretamente. Em caso afirmativo, a função retorna uma lista de listas representando as rotas distribuídas entre os caminhões. Em caso negativo é dividido o problema no meio, e chamando recursivamente a função pelo lado esquerdo(inicio ao meio) e depois pelo direito(meio +1 ao fim) até chegar no caso base. Depois é chamado a função combinaRotas passando o lado esquerdo e lado direito juntando em apenas uma lista de listas, sendo que o lado esquerdo possui as menores rotas e o direito possui as maiores.(Isso só funciona porque o vetor foi previamente ordenado). Então é combinado a soma das maiores rotas com as menores e as do meio continuam com a do meio.

Resultado para “rotas 1”

  
Resultado para “rotas 2”



A abordagem de divisão e conquista não garante necessariamente um resultado ótimo em todos os casos, mas nesse caso chega em um bom resultado em um tempo de execução bom. Em resumo, a abordagem de divisão e conquista implementada no código oferece uma solução sólida para o problema de otimização de rotas de caminhões.