SERVIÇO PÚBLICO FEDERAL · MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE VIÇOSA · UFV
CAMPUS FLORESTAL

Trabalho Prático 2 - AEDS 1

PROBLEMA DE ROTEAMENTO DE VEÍCULOS

IURY MARTINS PEREIRA [EF04671]
JOÃO VITOR GONÇALVES VIEIRA [EF04212]

Florestal - MG
2022

Sumário

1. Introdução

Este trabalho tem como foco criar um programa que diminua o gasto operacional, a implementação será baseada no Problema de roteamento de veículos(PRV) que consiste no atendimento a um conjunto de consumidores onde que para atendê los serão utilizados caminhões com capacidades limitadas e a nossa função será encontrar uma rota com o menor custo e atendendo o máximo de clientes possíveis, assim a principal tarefa é minimizar o custo total da operação e observar se a implementação usando PRV será realmente boa para o nosso objetivo.

2. Organização

Primeiramente organizamos cada função do trabalho onde que Iury ficou com a implantação do código e João ficou responsável para auxiliá-lo e fazer a parte escrita, e durante a primeira reunião foi criado nosso repositório online no Github onde fizemos as pastas principais do projeto.

Na Figura 1 é possível visualizar que foram criadas duas pastas a primeira /Libs onde ficaram os arquivos cabeçalho, esses arquivos são aqueles que contêm as declarações das funções utilizadas no projeto e por último a pasta /Sources onde ficam as implementações das funções que foram declaradas nos arquivos .h contidos dentro da pasta /Libs.

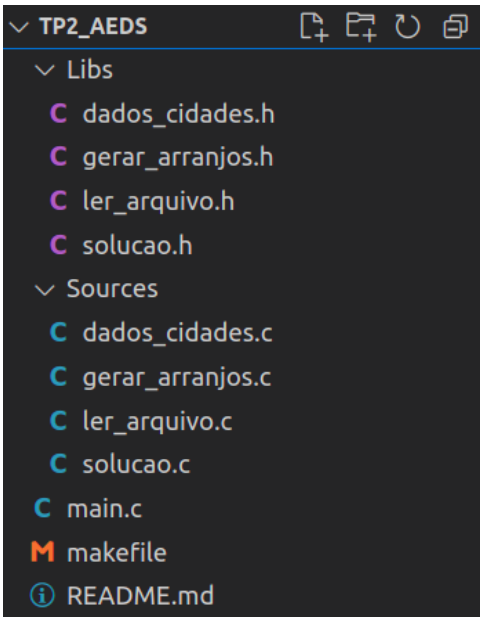

Figura 1 - Repositório do projeto.

Para executar o projeto, foi utilizado um arquivo Makefile com os comandos utilizados para compilar e executar os códigos. Por fim, foi criado um README para que auxiliasse os usuários de Linux e Mac a como executar o código usando os comandos declarados dentro do makefile.

3. Desenvolvimento

3.1 Cronologia

25 de Janeiro - Terça

Local de armazenamento do trabalho: Github.

26 de Janeiro - Quarta

Monitoria de explicação sobre o trabalho

28 de Janeiro - Sexta

Procura do algoritmo de arranjo que foi encontrado no site que está nas referências do projeto.

30 de Janeiro - Domingo

Início de desenvolvimento do TAD gerar_arranjo e ler_arquivo

31 de Janeiro - Segunda

Correções nas funções de arranjo e ler arquivo.

Início do TAD dados_cidades

03 de Fevereiro - Quinta

Correções nas funções de de dados cidades

05 de Fevereiro - Sábado

Criação do TAD solução

Início das implementações das condições impostas

07 de Fevereiro - Segunda

Todos os TAD finalizados e corrigidos.

08 de Fevereiro - Terça

Testes de desempenho de tempo.

09 de Fevereiro - Quarta

Confecção do gráfico e Inicio da documentação

11 de Fevereiro - Sexta

Finalização da documentação.

3.2 Explicação do Código

O código de arranjo foi encontrado em um site que o endereço se encontrar na bibliografia, o algoritmo fazia arranjo com char mas foi adaptado para que pudesse ser útil para implementação do trabalho, a função gerar_arranjo começa com um if que se P==0 retorna para main, esse P vai ser utilizado para deixar nossa função recursiva,pois sempre que a função é executada ela diminui 1 do valor do P, P também significar o número de cidades possíveis para a permutação dos arranjos, após isso criamos um array com nome aux de inteiros usando calloc de tamanho P+1, depois teremos um while com um for dentro que irá conferir se tem alguma repetição de arranjo e caso não tenha irá armazenar os números da rota no vetor aux e quando não tem repetição irá armazenar na variável soma_aux a soma de cada rota, e se a soma for menos que a capacidade do caminhão a rota será alocada nos arranjos_uteis, após tem um for que vai limitar o loop do while e assim encerrando a função gerar_arranjo.

Após gerar cara arranjo o algoritmo verifica se a soma da demanda destes arranjo é menor ou igual a capacidade do caminhão, caso esta condição seja atendida o arranjo gerado e copiado para o vetor arranjos que guarda todos os arranjos que satisfaçam esta condição, também é incrementado +1 no campo arranjos_uteis que guarda a quantidade de arranjos úteis para fazer a solução. A posição deste arranjo no vetor arranjos é definida pela ordem que são gerados.

Depois que todos os arranjos úteis são salvos no vetor arranjos, o algoritmo entra na função melhor_solucao que irá escolher a melhor solução possível para o problema atendendo as condições impostas. Para isso, o algoritmo irá gerar todos os arranjos possíveis entre as posições dos arranjos no vetor arranjos do tipo (A n,p) onde n representa o total de arranjos úteis guardado no campo arranjos_uteis é p representa a quantidade de caminhões que é dado pela divisão entre a soma total da demanda pela capacidade do caminhão. Cada arranjo deste gerado fica armazenado no vetor índice que contém as posições dos arranjos(das cidades) no vetor arranjos que formaram uma possível solução do problema. Depois o algoritmo faz uma possível solução pegando os arranjos(das cidades) através das suas

posições que estão no vetor indice, e armazena essa possível solução no vetor possivel_solucao, após isso verifica se essa possível solução tem arranjos distintos que contém uma mesma cidade é verifica se a possível solução atende todas as cidades. Caso a possível solução atenda essas condições ela se torna a melhor solução e é armazenada no vetor melhor_solucao caso seja a primeira solução encontrada pelo algoritmo, caso não seja a primeira solução encontrada é verificado se essa nova solução tem uma distância percorrida pelo caminhão menor que a distância da melhor solução que existe até então, caso esse condição seja atendida ela se torna a melhor solução, do contrário ela é descartada. O algoritmo faz todas essas verificações até acabar todos os arranjos do tipo (A n,p) onde n representa o total de arranjos úteis guardado no campo arranjos_uteis é p representa a quantidade de caminhões.

4. Resultados

Assim que terminado o algoritmo foi iniciado a fase de testes com cinco arquivos com tamanho N diferente, e como resultado obtivemos que os arquivos com os valores inferiores a oito rodaram abaixo de 1 segundo e quando foi utilizado N acima de oito tivemos a percepção que o tempo aumentava exponencialmente, a seguir a imagem 2 apresenta o gráfico que foi utilizado para analisar o tempo gasto.

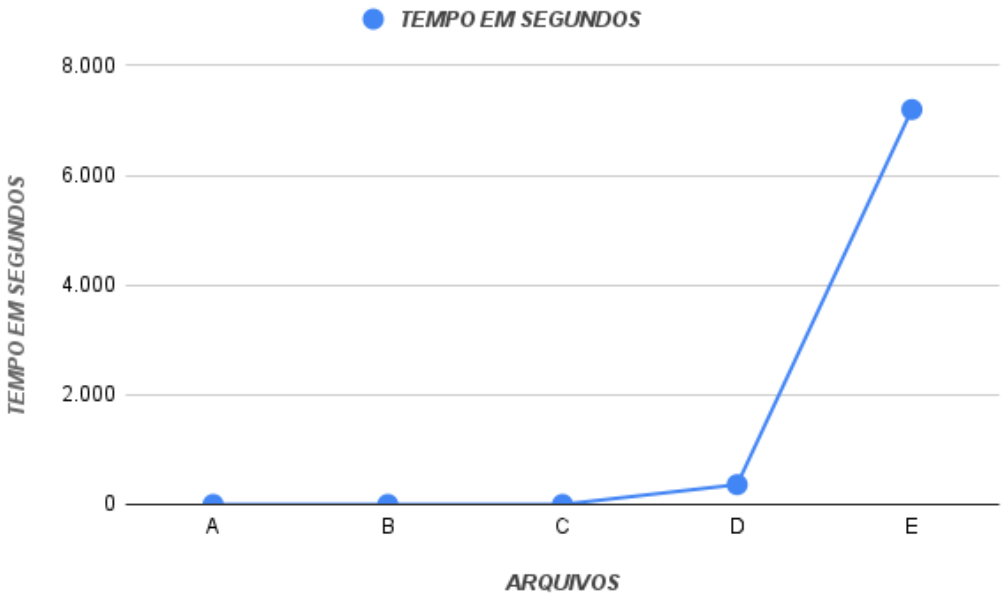

Figura 2 - Gráfico de linha representando o tempo de cada arquivo

4.1 Tempo de cada arquivo

Arquivo A com N = 2 teve o tempo de execução 0,000212 de segundos.

Arquivo B com N = 4 teve o tempo de execução de 0,00043 segundos.

Arquivo C com N = 6 teve o tempo de execução de 0,054088 segundos.

Arquivo D com N = 8 teve o tempo de execução de 361 segundos, aproximadamente 6 minutos.

Arquivo E com N = 10 teve o tempo de execução maior de 2 Horas, que em segundos é 7200.

5. Conclusão

Como observado na parte de resultados é inviável em alguns casos usar o algoritmo baseado no método de PRV para certos valores de N, como o exemplo de N=10 onde que o tempo de execução foi maior que 2 horas, iríamos fazer um teste com um arquivo de N=12 mas optamos em não fazer já que no caso N=10 foi um tempo grande em comparação com N=8 de apenas 361 segundos que também é um tempo considerável se fosse colocado numa situação na vida real, deduzimos então que o algoritmo é viável apenas em casos menores, pois quando o valor de N for grande, maior será o tempo de execução utilizado para encontrar a melhor rota.

6. Referências

[1] Github. Disponível em <https://github.com/joaoVGvieira/TP2_AEDS> Último acesso em: 08 de Fevereiro de 2022.

[2] Daemons Labs. Disponível em <Codigo Em C Para Permutar Vetores | Dæmonio Labs> Último acesso em: 08 de Dezembro de 2022.