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

Trabalho 2 - AEDS 1

Problema de roteamento de veículos

João Victor Graciano Belfort de Andrade
Mateus Henrique Vieira Figueiredo
Vitor Ribeiro Lacerda

Florestal - MG
2022

Sumário

1. Introdução

Esta documentação se refere ao projeto feito no Trabalho Prático II, da disciplina ALGORITMOS E ESTRUTURA DE DADOS I, de código CCF 211. No trabalho, foi sugerido a implementação de um código para resolução de um PRV(Problema de roteamento de Veículos, e a montagem de um um gráfico para analisar seu tempo de execução, a fim de aprender sobre complexidade de algoritmos. O projeto realizado descrito neste documento está disponível na página do GitHub:Vehicle-Routing-Problem.

Algumas bibliotecas externas foram utilizadas para o programa, elas são “time.h”, “stdbool.h”, ”stddef.h”, “limits.h”, “math.h” e “string.h”. “time.h” foi utilizada para pegar o horário da máquina e calcular o tempo de execução do código, a biblioteca “stdbool.h” foi utilizada para inserção do tipo bool e para lógica booleana, ”stddef.h” foi utilizada para a inserção do macro “NULL”, a biblioteca “math.h” foi utilizada para ter acesso a função “ceil”, que foi utilizada no cálculo do número de caminhões, “string.h” foi utilizada para o tratamento de strings e “limits.h” foi utilizada para a definição de “CHAR MAX". O programa foi dividido em 2 pastas, sendo uma para armazenamento de testes utilizados para a montagem dos gráficos e outra para o algoritmo solicitado no trabalho.

2. Organização

Na pasta principal do arquivo podem ser encontrados os arquivos, cada um destes com os seguintes nomes: arquivos (Pasta de arquivos), src (Pasta de arquivos), README.md, .gitignore.txt. Os arquivos mencionados do tipo “pasta de arquivos” ainda contém outros, em “arquivos”: “teste1.txt”, “teste2.txt”, “teste3.txt”, teste4.txt, teste5.txt, teste6.txt, teste7.txt, “teste8.txt”, “teste9.txt”, “teste10.txt”, “teste11.txt”, “teste12.txt”,“teste13.txt”, “teste14.txt” e “teste15.txt”. Já em src temos: “cidade.c”, “cidade.h”, “permutacao.c”, “permutacao.h”, “main.c”.

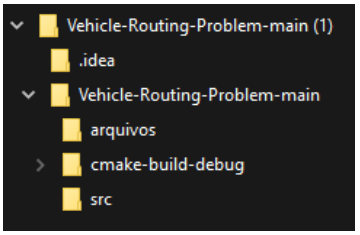

Figura 1- Repositório do projeto

Sendo assim, a pasta principal do projeto contém arquivos referentes ao repositório do github, e as duas pastas contendo os códigos, além de arquivos referentes ao projeto e arquivos criados pela ide CLion na compilação do código. Em “arquivos” tem-se os arquivos criados pelo grupo, variando de "teste1.txt" até “teste15.txt”, cada número do arquivo “teste”, refere-se ao número de cidades utilizadas para o teste. Já a pasta “src” contém os códigos tanto das tads (como o de permutação e das cidades) como do código principal do trabalho,

3. Desenvolvimento

3.1 “cidade.c” e “cidade.h”

Nestes arquivos foi descrita a “TAD cidade”, essa estrutura contém a demanda (do tipo “unsigned int”, para armazenar a demanda) e um booleano (variável do tipo “bool” da biblioteca “stdbool.h”, para registrar a visita à cidade). Foram criadas as seguintes funções:

“getDemanda” e “setDemanda”: Retorna e subscreve respectivamente a demanda da cidade.

“getFoiVisitada” e “setFoiVisitada”: Retorna e subscreve respectivamente se a cidade foi visitada.

3.2 “permutacao.c” e “permutacao.h”

Nestes arquivos foi descrita a “TAD permutacao” , uma estrutura abstrata para o cálculo da permutação. Em “permutacao.h” foram criadas as funções para serem utilizadas na “main.c” e na “permutacao.c”, estes são:

“inicializa”: Essa função inicializa as variáveis globais para serem utilizadas em outras funções. São elas: um ponteiro “array” que contém a quantidade de cidades, um ponteiro para matriz que armazena distância entre as cidades, e a capacidade do caminhão.

“encontraMelhorRota”: Essa função é usada para calcular todas possibilidades de permutações de combinações de um dado array de inteiros para todas as variáveis de quantidades de itens. Possui como objetivo encontrar a melhor rota.

Já em “permutacao.c” foram criadas funções para serem utilizadas no próprio arquivo. Elas são:

“fazCombinacoes”: Função usada para calcular todas as possíveis combinações de um dado item de inteiros para todas as variáveis de quantidades de itens.

“fazPermutacoes”: Essa função usa a permutação do heap (O algoritmo Heap faz todas as permutações de X objetos, minimizando os movimentos para gerar as próximas permutações. O algoritmo gera cada permutação com base na permutação anterior, trocando um único par de elementos.)para gerar todas as permutações de forma recursiva.

“troca”: Essa função é usada para trocar os valores dos elementos dados na memória.

“checaPermutacao”: Essa função é usada para checar se a rota permutada atual é melhor que as rotas permutadas anteriormente.

“liberaMemoria”: Essa função é usada para liberar a memória das estruturas globais alocadas dinamicamente.

Obs: O grupo preferiu por utilizar além do código de permutação, um código de combinação, para a criação das rotas, pois desta forma as variáveis se mantém mais organizadas, o código fica mais eficiente, mas ainda segue as instruções dadas pelo Trabalho Prático, sem alterar o objetivo principal do trabalho (avaliação da complexidade de um algoritmo exponencial). Os códigos de combinação e permutação foram retirados da internet e adaptados para o trabalho. Os códigos originais podem ser encontrados em: código-de-combinação e código-de-permutação.

3.3 “main.c”

Neste arquivo foi implementado o código principal do programa que utiliza dos demais executáveis e headers, para criar uma interface onde o usuário poderá escolher o arquivo de teste a ser executado pelo programa. O programa retorna tanto o tempo gasto para a execução, quanto a melhor rota encontrada.

Para a checagem, tendo a permutação em mão, primeiramente nós calculamos a demanda total da rota, sendo a rota menor ou igual a capacidade do caminhão seguimos para a próxima checagem. A próxima checagem consiste em comparar a distância total da rota com as anteriores de forma a salvar qual a melhor rota é para uma quantidade especifica de dígitos.Por exemplo:

Sendo 3 cidades para serem calculadas as rotas, primeiramente cada combinação gerada para uma quantidade específica de dígitos, a partir da combinação, permutações são geradas. A partir da permutação fazemos a checagem para decidir qual é a melhor rota.

Voltando para o nosso exemplo:
Combinação: 1 2 3
Permutação:
1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1

Combinação: 1 2
Permutação:
1 2
2 1

Combinação: 1 3
Permutação:
1 3
3 1

Combinação: 2 3
Permutação:
2 3
3 2

Combinação: 1
Permutação:
1

Combinação: 2
Permutação:
2

Combinação: 3
Permutação:
3

No código, começamos pela maior quantidade de dígitos, no caso do exemplo, 3, então checamos se o caminhão pode armazenar, caso não consiga, iremos para a segunda quantidade de dígitos, 2. Para dois dígitos temos 3 combinações, as quais cada permutação é checada para checar se a demanda da rota pode ser atendida. Vamos supor, quando temos 2 dígitos, a demanda para a rotas 23, 32, 13, 31 possam ser atendidas, dessa forma o código decidirá quais dessas rotas tem a menor distância, note que para o cálculo de distâncias, consideramos 0230, 0320, 0130, 0310. Após encontrar a melhor distância, definimos as cidades daquela distância como visitadas. De forma a solucionar a condição que podemos ter múltiplas rotas com o mesmo dígito, iremos rodar o código novamente, na mesma quantidade de dígitos para fazer essa checagem, de forma a ignorar permutações nos quais as cidades já foram visitadas.

O código continua indo para a menor quantidade de dígitos até atingir 1, em outras palavras até ter visitado todas as cidades. Portanto como rota final para o nosso exemplo, podemos ter: 0320010.

4. Resultados

Devido a interface interativa, foi criado um menu em que o usuário poderia escolher qual arquivo de teste da pasta arquivos o usuário quer abrir. O menu está sendo demonstrado abaixo:

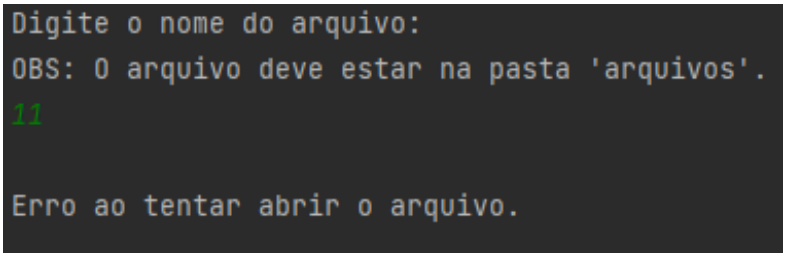

Figura 2- arquivo não encontrado

```
Digite o nome do arquivo:
OBS: O arquivo deve estar na pasta 'arquivos'.
text.txt

Quantidade de caminhoes: 1.

MELHOR ROTA ENCONTRADA:
0 2 1 3 0

Tempo total = 0.0210 segundos.
```

Figura 3- solução

Para o trabalho prático foi pedida a montagem de um gráfico, com os devidos tempos, os gráficos com os tempos estão apresentados abaixo:

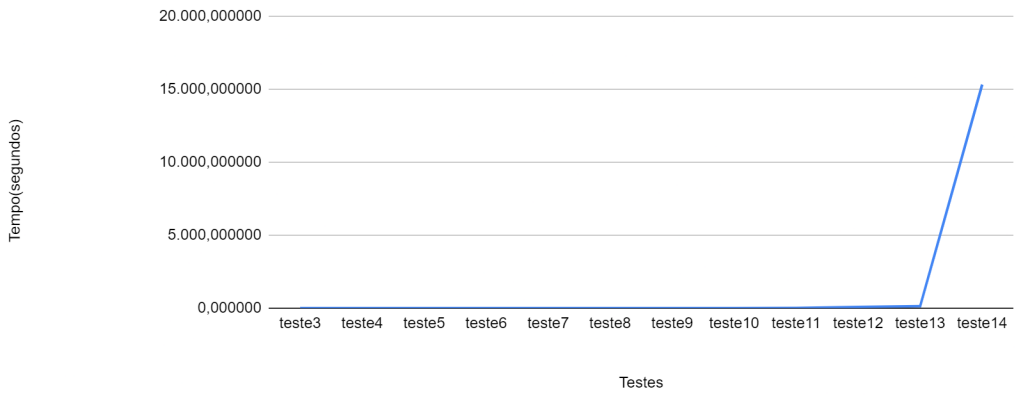

Figura 4- Gráfico com 14 testes

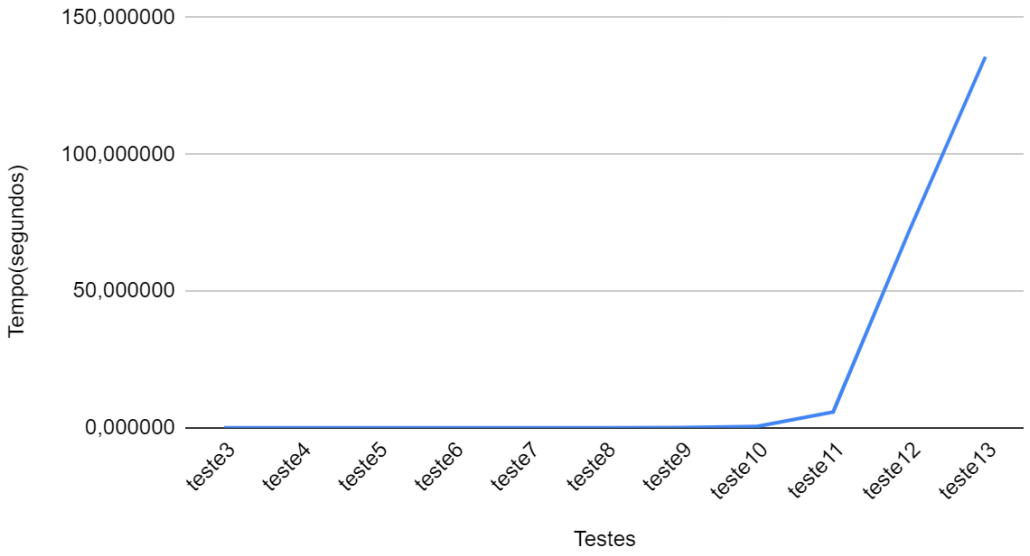

Figura 5- Gráfico com 13 testes

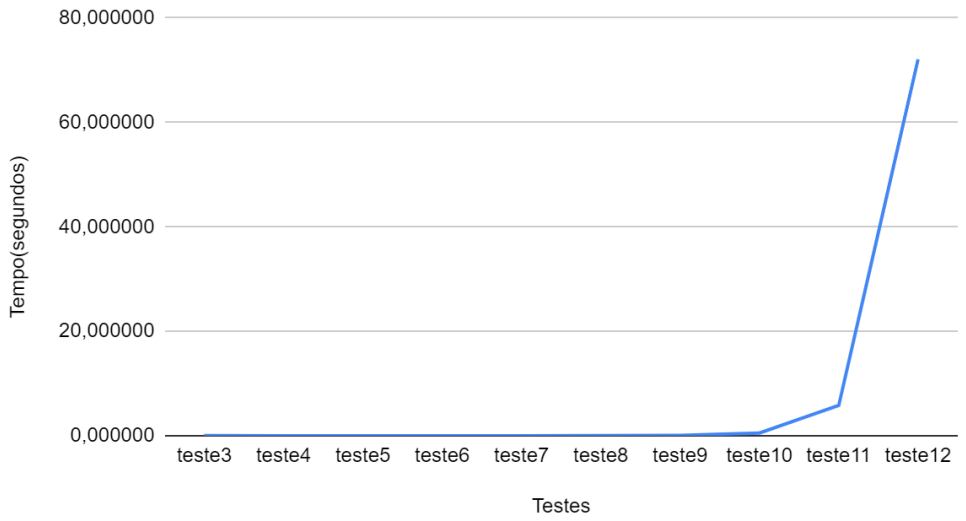

Figura 6- Gráfico com 12 testes

Ao analisar o gráfico gerado pela diferença de entre os tempos podemos observar que, quanto maior o tempo, maior é a diferença entre este e o tempo anterior sendo que no gráfico com 14 testes os demais tem um tempo, praticamente, igualmente menor do que o teste 14. Ao observar que o tempo crescia exponencialmente decidimos encerrar os testes no 14º pois o 15º seria muito demorado, como o tempo não pode ser analisado corretamente nos gráficos, abaixo segue uma planilha com os tempos de teste:

Testes	Tempo(segundos)
teste3	0,002000
teste4	0,003000
teste5	0,003000
teste6	0,004000
teste7	0,004000
teste8	0,009000
teste9	0,053000
teste10	0,520000
teste11	5,800000
teste12	72,090000
teste13	135,680000
teste14	15.319,233000

5. Conclusão

De forma geral, foram implementados TADs e códigos, de forma organizada, comentada e de fácil entendimento para o problema solicitado pelo trabalho, foi decidida uma abordagem mais interativa com o usuário, que ao fim foi uma ótima opção abstraindo o código, e o deixando mais fácil de ser visualizado e compreendido pelo usuário. Os resultados saíram assim como o esperado, sendo que algumas implementações feitas, apesar de deixarem o código mais extenso, facilitam seu uso e entendimento, sendo assim o trabalho, de forma geral atende as exigências passadas pelo Professor do curso de AEDES-2021/2.

6. Referências

Para versionar o projeto foi utilizado o Github [1].
Código utilizado para realizar as combinações [2].
Código utilizado para realizar as permutações [3].

[1] Github. Disponível em: <https://github.com/Mateus-Henr/Vehicle-Routing-Problem> Último acesso em: 08 de fevereiro de 2022.

[2] Geeks for Geeks, **Print all possible combinations of r elements in a given array of size n,** 19 de Janeiro de 2022. Disponivel em: <https://www.google.com/amp/s/www.geeksforgeeks.org/print-all-possible-combinations-of-r-elements-in-a-given-array-of-size-n/amp/> Último acesso em: 07 de fevereiro de 2022.

[3] Geeks for Geeks, **Write a program to print all permutations of a given string,** 18 de Janeiro de 2022. Disponivel em : <https://www.google.com/amp/s/www.geeksforgeeks.org/write-a-c-program-to-print-all-permutations-of-a-given-string/amp/> Último acesso em: 07 de fevereiro de 2022.