PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

ENGENHARIA DE SOFTWARE

ICEI

PROBLEMAS INTRATÁVEIS

CAMINHÕES E ROTAS

Arthur Jansen Oliveira

Barbara Mattioly Andrade

Laura Enísia Rodrigues Melo

Belo Horizonte

2023

Sumário

1) INTRODUÇÃO

O trabalho proposto visa colocar em prática o estudo de problemas intratáveis, utilizando-se técnicas de programação e algoritmos específicos a fim de encontrar soluções possíveis e mais adequadas para o problema das Rotas de Caminhões de uma empresa de distribuição e logística.

O problema se baseia em uma empresa que possui N caminhões e organiza as entregas em M rotas distribuídas entre as carretas. A empresa tem como objetivo realizar a distribuição de mercadoria de forma que cada caminhão faça a mesma quilometragem. Porém, caso não seja possível que todos possuam a mesma distância, que seja prevalecido o menor caminho.

Desse modo, o grupo dividiu o trabalho de forma que cada integrante elaborou uma solução para determinado desafio proposto: o método de Backtracking, o método de Divisão e Conquista e o método da Programação Dinâmica. Para a implementação das soluções propostas foi utilizada a linguagem de programação Java.

2) BACKTRACKING

O algoritmo de backtracking foi implementado considerando várias sequências de rotas e descartando algumas, a fim de encontrar a melhor solução para distribuir todas as rotas existentes em um conjunto, para 3 caminhões. O algoritmo basicamente constrói e percorre uma árvore de sub-tarefas, de modo recursivo e como a busca em profundidade.

Inicialmente, o algoritmo distribui as rotas de modo ordenado. A ordenação foi um critério escolhido pois ajuda a melhorar o desempenho do algoritmo, isso porque é possível agrupar rotas próximas uma das outras, levando a soluções melhores mais rapidamente. Desse modo, uma vez que as rotas estavam de modo crescente, contribuiu potencialmente para melhorar o desempenho do sistema e a reduzir o espaço de busca para encontrar as menores quilometragens.

Em seguida, foi criado o método de distribuição de rotas, no qual possui uma variável média para calcular a média de rotas por caminhão, e um vetor de rotas

para ser atribuído para os caminhões de modo que a quilometragem seja igual entre os automóveis, ou com a menor diferença. A medida que o código é implementado, o sistema tenta atribuir as rotas para um caminhão. Com isso, é verificado a maior diferença de soma por caminhão e guardado em uma variável de melhor distribuição.

O critério para a poda no sistema é de acordo com a soma de quilometragem por caminhão com a verificação da média (com tolerância de 10%) citada anteriormente. Caso a soma seja maior que a média, significa que essa distribuição não é a melhor, pois excedeu a média das quilometragens já percorrida. Portanto, não é a melhor atribuição para o caminhão, logo o fluxo é podado e inicia novamente a conferência da próxima rota para distribuição a fim de explorar as demais combinações possíveis, otimizando o algoritmo.

Ademais, é importante salientar que no código principal, é implementado a solução do backtracking de forma que contenha 3 caminhões e a distribuição comece com 6 rotas. Em seguida era contabilizado o tempo de execução do backtracking para determinado conjunto de rotas e testado com diferentes valores em um conjunto de tamanho 10. Posteriormente era calculado o tempo médio de execução para o conjunto atual do tamanho de rotas. E por fim, caso o sistema atingisse o limite de 30 segundos, encerrava-se o programa e mostrava-se o tamanho de rotas limite para a condição utilizada.

Outro fator que também é importante salientar é que as medidas de tempo usadas para o cálculo de tempo nas execuções no sistema foram milissegundos e nanosegundos. Isso porque esses parâmetros possuem mais precisão no tempo estimado que o programa utilizou para realizar os métodos com as quantidades específicas de rotas no conjunto.

Por fim, no final da execução do algoritmo implementado do Backtracking, o tempo médio obtido que excedeu o limite de 30 segundos foi o conjunto com 17 rotas, com tempo médio de duração 34432.1ms (34s).

3) DIVISÃO E CONQUISTA

O algoritmo de divisão e conquista aplicado na resolução do problema em questão apresenta implementação relativamente semelhante ao algoritmo para resolução da subsequência de soma máxima apresentado em aula. Ele divide o conjunto inicial ao meio recursivamente até atingir o caso base, “quando x for pequeno”, ou seja, no caso do algoritmo acontece quando existem apenas dois elementos a serem percorridos (quando a posição final menos a posição inicial é menor ou igual a dois).

Ao atingir o caso base, cada subconjunto é resolvido separadamente para que depois aconteça a conquista, para a base existem três possibilidades de solução, apenas o elemento mais à direita, apenas o elemento mais à esquerda e a soma de ambos os elementos.

Para cada subconjunto sendo testado, é encontrado o valor que mais se aproxima da média (soma das quilometragens mais a tolerância (utilizamos 8%) dividido pelo número de caminhões) e o valor que mais se distancia da média, esses valores são armazenados em 2 listas temporárias. Ao resolver o primeiro subconjunto da direita e o primeiro subconjunto da esquerda, acontece a conquista onde os valores mais próximos da média são comparados e o mais próximo é armazenado na lista temporária (caso ele encontre um valor mais próximo da média do que o armazenado na lista, e o somatório desse valor com o que já está na lista ultrapasse a média, a lista temporária é limpada e os novos valores são inseridos nela). Esse processo se repete após a conquista ter acontecido com todos os subconjuntos.

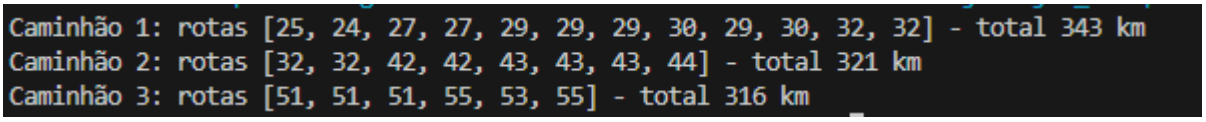
No final da conquista, as listas de elementos que mais se aproximam da média e mais se distanciam são comparadas, caso o somatório dos elementos fique dentro do valor da média todos são armazenados no conjunto de solução do caminhão, caso contrário apenas os valores presentes na lista que mais se aproxima da média são armazenados. Ao encontrar a solução para o primeiro caminhão as listas temporárias são limpas, os elementos que entraram no conjunto de solução são removidos da rota a ser percorrida e o processo inicia novamente para os demais caminhões.

Para resolver o algoritmo,foi aplicada uma tolerância de 8% ao valor da média, exemplo, caso o somatório dos elementos seja igual a 100, utilizamos uma média de 108. Essa tolerância foi aplicada pelo fato de que não é possível atingir o caso ótimo para todas as combinações aleatórias de rotas que são geradas, então a tolerância é utilizada para tentar obter a menor distância possível entre as rotas.A solução ótima não é atingida caso a média (sem a tolerância) seja igual a um número fracionário e por conta desses casos a tolerância é aplicada.

O valor de 8% foi escolhido já que à medida que a tolerância aumenta, a diferença de distância entre os caminhões aumentava e para alguns conjuntos, uma tolerância menor também aumentava a discrepância dos dados.

A imagem abaixo mostra a execução do algoritmo com tolerância de 8%, para esse caso a diferença máxima de distância foi de 27.

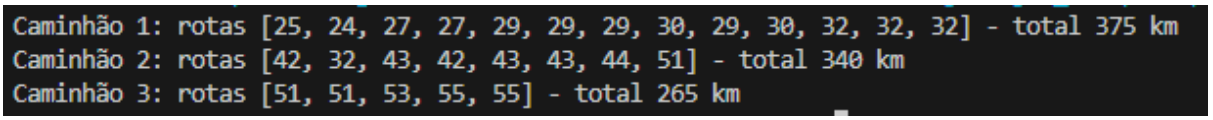
Figura 1: Execução do algoritmo com tolerância de 8%



Fonte: Autores

Já a imagem abaixo mostra a execução do algoritmo com tolerância de 15%, para esse caso a diferença máxima de distância foi de 110.

Figura 2: Execução do algoritmo com tolerância de 15%



Fonte: Autores

O cálculo do tempo médio foi feito da mesma forma que o utilizado para o backtracking, as medidas de tempo utilizadas foram milissegundos e nanosegundos, já que eles possuem maior precisão. O algoritmo se mostrou eficiente para a resolução de grandes conjuntos de dados, executando em um tempo médio de 1.13 milissegundos para 100 rotas e realizando o teste 20 vezes.

4) PROGRAMAÇÃO DINÂMICA

O algoritmo de programação dinâmica foi construído visando resolver o problema das rotas entre N caminhões. Para isso, o sistema inicia pegando as rotas disponíveis e ordenando-as de forma crescente. Foi realizada a ordenação para ajudar na otimização e para mostrar o resultado final.

Após a realização da ordenação, as rotas são enviadas por parâmetro para um método que gera limites. Os valores aceitos serão utilizados como limitadores na tabela dinâmica. Com isso, é retornado o limite e é iniciada a construção da tabela.

Utilizando dois for, estruturas de repetições, é possível percorrer a matriz de elementos das rotas já geradas. É usado o tamanho do vetor de limite para a colunas(j), e o tamanho do vetor de rotas geradas, para a quantidade de linhas(i). Desse modo, é obtido uma tabela de tamanho rotas geradas X vetor de limites. Foi usado esse padrão, pois percebi através de testes, que dessa forma chegaria ao resultado mais rápido sem redundâncias ou outras tabelas.

Em seguida, é realizado o preenchimento das linhas e colunas da tabela. Um exemplo das regras utilizadas é caso o valor seja maior que o limite, é utilizado o valor da coluna anterior, e repete esse valor na célula da tabela atual.

Ao final desse processo, é chamado outro método que utiliza a tabela e faz o caminho inverso a fim de descobrir quais rotas foram utilizadas. Com essas rotas agrupadas em um vetor de inteiro, ele retorna o conjunto de forma ordenada em decrescente.

O algoritmo é feito de modo para descobrir um conjunto de rotas válidas para um caminhão. Esse processo se repete N vezes de acordo com a quantidade de caminhões, e em todas essas vezes é retirado as rotas já utilizadas pelos caminhões passados. Durante a implementação foram encontrados problemas devido números repetidos no conjunto de rotas eram tratados como iguais, então foram realizadas correções para tratar esse ponto, aplicando a retirada por índice e não por valor, preservando rotas repetidas. A ideia é seguir e recriar a regra padrão da tabela dinâmica, porém com repetição e retirada de rotas disponíveis em um vetor.

A tabela para resolução do problema foi organizada da seguinte forma:

- **Linhas:** as rotas que podem ser usadas
- **Colunas:** o limite (capacidade) da soma das rotas
- **Células:** a soma das rotas que podem ser adicionadas (que serão comparadas com o valor de média para encontrar a solução)
- **Função:** É diferente da função original, pois temos uma subtração da coluna, pelo valor da linha, para ver se o valor dessa célula nova encontrada + o valor que queremos adicionar é melhor do que a célula anterior.
  max(T[i - 1,j]; T[i - 1, j - v] + v), sendo V o valor da rota.

Para encontrar o melhor resultado, foi aplicado tolerância de 10% no valor da média, para aqueles casos em que não é possível atingir o resultado ótimo, ter um valor a partir da média.

5) CONCLUSÃO

Em síntese, o trabalho proposto foi muito importante para esclarecer e concretizar os conhecimentos adquiridos durante o semestre letivo.

Para realização da comparação entre os algoritmos, foram executados primeiramente, consecutivos testes para o algoritmo de backtracking conforme apresentado anteriormente para conjuntos de dados de tamanho crescente, até atingir um tamanho T que não consiga ser resolvido em 30 segundos pelo algoritmo, realizando testes para 3 caminhões e começando com 6 rotas até atingir o tempo limite. O tempo limite foi atingido para o conjunto de 17 rotas. Para realizar comparações do tempo de execução dos algoritmos, foram armazenadas todas as 10 rotas de 17 elementos utilizados para teste no backtracking e realizado a comparação com o algoritmo de divisão e conquista.

Conforme apresentado na tabela abaixo, o tempo médio de execução para encontrar a solução em um conjunto de 17 rotas foi muito maior no backtracking do que no divisão e conquista, isso se deve a complexidade para a execução de cada um dos algoritmos, enquanto o algoritmo de backtracking possui complexidade exponencial, o algoritmo de divisão e conquista possui forte tendência a

complexidade logarítmica. Dessa forma, executando em um tempo muito menor que o do backtracking (no exemplo abaixo, a diferença de tempo de execução entre os dois algoritmos foi de aproximadamente 34.4314 segundos) uma vez que divide o problema em subproblemas menores.

Tabela 1:

Tempo de Execução (ms)		
Rotas	**Backtracking**	**Divisão e Conquista**
17	33893.6	0.26

Fonte: Autores

Com relação a qualidade dos dados, os resultados encontrados com o algoritmo de backtracking foram melhores que os encontrados com algoritmo de divisão e conquista como pode ser observado na tabela 2 abaixo comparando a diferença entre as distâncias, já que ele sempre encontra o resultado ótimo. Caso encontre um resultado que não atende a solução ele faz o retrocesso.

Tabela 2 - Resultados do somatório de km para as dez rotas

idRota	Total Diferença de Distâncias (Km) - 17 elementos				
	Caminhão	Backtracking	Diferença - BT	Divisão e Conquista	Diferença - DC
1	1	88	2	85	25
	2	87		85	
	3	86		110	
2	1	81	1	82	9
	2	81		91	
	3	82		90	
3	1	78	4	82	9
	2	80		91	
	3	82		86	
4	1	94	2	91	23
	2	91		91	
	3	92		114	
5	1	84	0	82	26
	2	83		80	
	3	82		106	
6	1	93	0	100	15
	2	93		85	
	3	93		94	
7	1	89	0	85	29
	2	89		83	
	3	89		112	
8	1	93	3	95	14
	2	96		81	
	3	91		91	
9	1	91	0	84	12
	2	91		96	
	3	91		93	
10	1	91	2	83	13
	2	91		96	
	3	89		92	

Fonte: Autores

Para analisar a qualidade dos dados entre os algoritmos de divisão e conquista, backtracking e programação dinâmica, e o motivo pelos quais alguns resultados atingiram resultados melhores que os demais, como exemplo os resultados encontrados para a rota de número 6 apresentada acima (com resultado ótimo, diferença igual a 0) e a rota de número 3 (diferença igual a 4). Analisamos os casos onde não foi possível encontrar o resultado ótimo e observamos que quando a média de quilometragem entre os caminhões era um número fracionário, não era possível encontrar a solução ótima. Por conta desses casos, aplicamos uma tolerância ao valor da média dos algoritmos, para que a diferença entre as distâncias fosse a menor possível. Quanto maior o valor da tolerância aplicado, mais discrepante os dados se tornavam (como foi possível observar nas figuras 1 e 2

apresentadas anteriormente) e para alguns conjuntos de dados valores de tolerância muito pequenos também aumentava a discrepância dos dados, então foi aplicado um valor intermediário para os algoritmos.

A análise feita acima pode ser observada com mais detalhes na imagem abaixo de execução do algoritmo para dois conjuntos e três caminhões utilizando backtracking.

Figura 2: Resultados da execução do backtracking para três caminhões e conjuntos de 16 rotas

```
Conjunto 3: [13, 14, 14, 15, 15, 16, 16, 16, 17, 17, 17, 18, 19, 19, 19, 19]
-Caminhão 1: rotas 16, 17, 17, 19, 19 - total 88km
-Caminhão 2: rotas 16, 16, 18, 19, 19 - total 88km
-Caminhão 3: rotas 13, 14, 14, 15, 15, 17 - total 88km
Tempo de execução: 6202 ms
-----------------------------------------
Conjunto 4: [13, 13, 13, 14, 14, 15, 15, 15, 16, 17, 18, 18, 18, 18, 18, 19]
-Caminhão 1: rotas 15, 17, 18, 18, 19 - total 87km
-Caminhão 2: rotas 15, 15, 18, 18, 18 - total 84km
-Caminhão 3: rotas 13, 13, 13, 14, 14, 16 - total 83km
Tempo de execução: 6652 ms
```

Fonte: Autores

Analisando a imagem acima, temos:

- **Conjunto 3: [13, 14, 14, 15, 15, 16, 16, 16, 17, 17, 17, 18, 19, 19, 19, 19]**
    - Média dos resultados: Somatório/Número de Caminhões
        - Média = 255/3 = 85 (Número inteiro)
        - Dessa forma é possível dividir igualitariamente as rotas entre os caminhões, encontrando assim resultado ótimo;

- **Conjunto 4: [13, 13, 13, 14, 14, 15, 15, 15, 16, 17, 18, 18, 18, 18, 18, 19]**
    - Média dos resultados: Somatório/Número de Caminhões
        - Média = 254/3 = 84,66666 (Número fracionário)
        - Dessa forma não é possível dividir de maneira igual as rotas entre os caminhões e encontrar o resultado ótimo, então é aplicada um valor de tolerância a essa média

para que seja possível encontrar a menor distância possível;

Após análise de todos os algoritmos e as comparações realizadas, foi possível deduzir que:

1. O método do Backtracking é um método de tentativa e erro, em que a busca completa nas rotas para encontrar a melhor distribuição para os caminhões possui comportamento exponencial. Esse resultado foi baseado nos testes realizados na tabela abaixo:

Tabela 3: tempo de execução para o backtracking

Tempo de Execução (ms)	
Rotas	**Backtracking**
6	0.4
7	1.6
8	4.2
9	10.2
10	24.1
11	47.6
12	122.5
13	334.9
14	1037.4
15	3265.2
16	9490.4
17	33893.6

Fonte: autores

Desse modo, com a análise da tabela acima é possível constatar que o algoritmo possui crescimento exponencial no tempo de execução do algoritmo. Isso é esperado pelo algoritmo de backtracking dado que ele é um refinamento do algoritmo de força bruta. Ele refina a busca exaustiva de modo que elimine soluções desnecessárias sem examinar.

Além disso, um outro ponto é que para um grande conjunto de dados ele é ineficiente, o que leva ele a explorar muitas soluções candidatas, das quais podem não funcionar. Já o método da Divisão e Conquista, subdivide o problema em subproblemas menores que são similares ao problema original. E a solução torna-se eficiente justamente pelo fato dos subproblemas poderem ser resolvidos independentemente e de modo paralelo. É possível provar os conceitos com as seguintes análises realizadas com o algoritmo de Divisão e conquista a seguir.

2. O algoritmo de divisão e conquista, recursivo, implementado seguindo os critérios e decisões descritas anteriormente se mostrou eficiente, já que executou em uma complexidade logarítmica e em tempo menor que os demais algoritmos em testes para os mesmos conjuntos, porém para a natureza específica do problema de distribuição de rotas e a forma com que ele foi implementado (descrita anteriormente) em muitos casos não retornou o resultado ótimo como pode ser observado na tabela de comparação com o backtracking.

Para analisar o tempo de execução utilizando o algoritmo de divisão e conquista, foram realizados testes para diferentes rotas e calculado o tempo médio para 10 conjuntos, e mesmo com um grande conjunto de dados o algoritmo executa de forma rápida, com crescimento logarítmico. Para o maior conjunto de dados testado (42 rotas) o algoritmo executa em um tempo médio de 0.30 milissegundos.

Tabela 4: tempo médio de execução para o divisão e conquista

Tempo Médio de Execução (ms) - 10 conjuntos	
Rotas	Divisão e Conquista
6	0.17
12	0.22
24	0.23
36	0.30
42	0.36

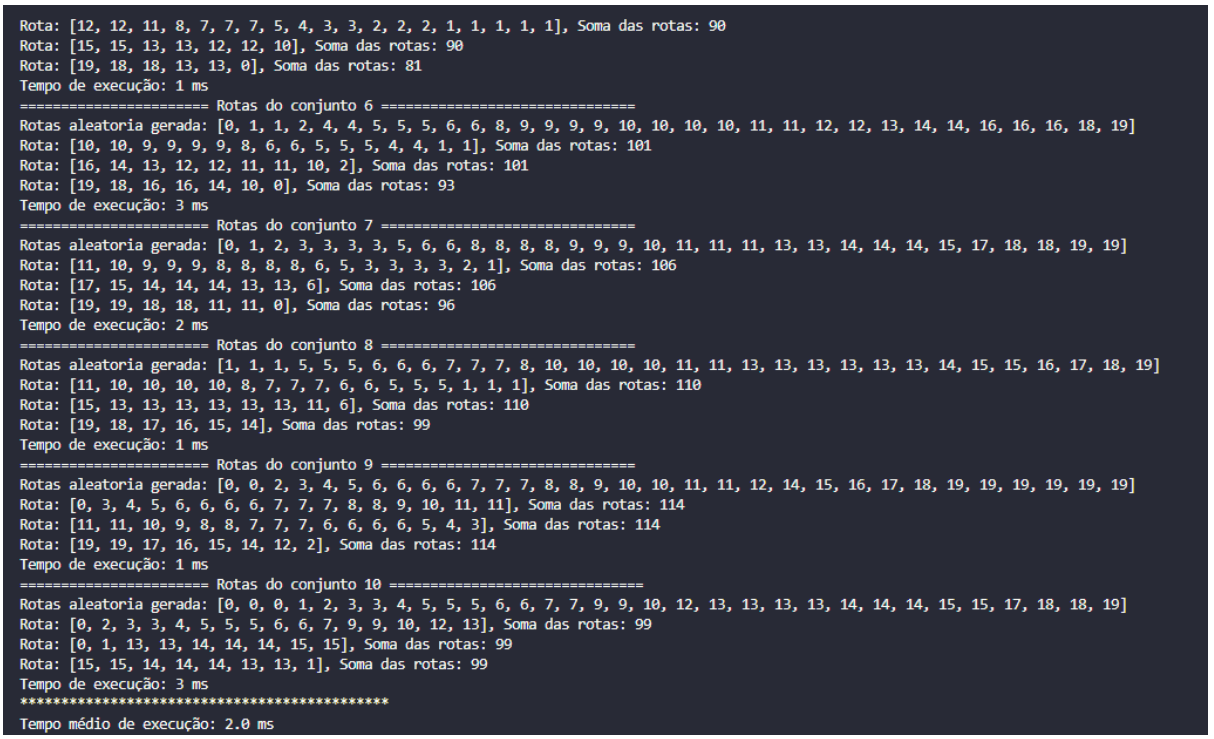
Fonte: Autores

3. O algoritmo de Programação Dinâmica é implementado usando a regra de criação de uma tabela, e percorrendo ela para achar o melhor valor (aquele que mais se aproxima da média de quilometragem) na última célula da tabela. Dessa forma foi fácil a implementação da tabela, porém não consegui achar um valor excelente em todos os casos, pois há um erro na retirada de rotas.

Após a execução do código, foi observado que mesmo percorrendo a tabela duas vezes, uma para realizar o preenchimento e outra para encontrar a solução das melhores rotas, o algoritmo se mostrou eficiente, executando em um tempo médio de 2ms com 32 rotas e 10 conjuntos. Ao realizar o teste com diversas rotas e conjuntos (mesmo conjunto de dados executados no backtracking até atingir 30 segundos), foi percebido que o tempo de execução do algoritmo ia crescendo à medida que o número de rotas aumentava, devido ao consumo de memória na criação das lista e dos arrays e a complexidade de execução do algoritmo. Mesmo com esses fatos, o algoritmo se mostrou eficiente executando em tempo melhor que o backtracking e cerca de 0.7ms mais lento que o divisão e conquista implementado para os testes realizados.

Assim como os demais algoritmos, foi aplicada uma taxa de tolerância para que as rotas sejam divididas com a menor distância possível nos casos em que não é possível atingir a solução ótima. O algoritmo implementado atingiu bons resultados, com uma diferença de quilometragem pequena, mas não atingiu o resultado ótimo devido a forma como foi implementado devido a retirada de valores já usados no vetor

Figura 4: Tempo médio de execução e resultados para o divisão e conquista



Fonte: Autores

6) REPOSITÓRIO

O repositório pode ser acessado através do link: https://github.com/lauramelo28/fpaa-tpf-g1

7) REFERÊNCIAS

https://www.cos.ufrj.br/uploadfile/1368210356.pdf

http://www.decom.ufop.br/marco/site_media/uploads/pcc104/15_aula_15.pdf

https://pucminas.instructure.com/courses/157701/files/9763471?module_item_id=3728167 (slides do professor João Caram)