EP: Marcação de tempo de execução – 2022

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

Bernardo Aquino Capello Coelho
Gustavo de Castro Nogueira
Vítor Lion Guimarães Rodrigues
Guilherme Roberto Ferreira Santos

1) Algoritmo Guloso:

Nesse algoritmo foi modelado 3 classes auxiliares para o mesmo. A classe rota que recebe o Id e a Distância das rotas dos caminhões, CaminhaoComparator que auxilia a decidir qual caminhão tem a maior soma de rotas e por final a classe Caminhão que é semelhante a uma partição que armazena o Id, as rotas e a soma da distância delas para que possam ser distribuídas adequadamente.

O algoritmo recebe a lista de rotas com suas distâncias e o número de caminhões (partições). Organiza-se esses valores em uma PriorityQueue que garante que o caminhão com menor soma de rotas esteja no topo e receba o próximo valor de rota que o algoritmo guloso vai armazenar. O código guloso está a partir da linha 106 do arquivo AppGuloso.

Critério guloso: É comparado o total de km entre dois caminhões e o critério é selecionar aquele em que a soma das rotas totalize um menor valor.

Função: min(sum(total km caminhão))

Testando com o arquivo caminhoes_compacto:

Tempo de Execução(milisegundos): 13.0ms

Caminhão: 0
Quantidade Rotas: 8
Total Km: 566
ID: 4 | Distância: 89
ID: 15 | Distância: 80
ID: 20 | Distância: 80
ID: 10 | Distância: 71
ID: 14 | Distância: 69
ID: 25 | Distância: 66
ID: 23 | Distância: 59
ID: 2 | Distância: 52

Caminhão: 1
Quantidade Rotas: 9
Total Km: 616
ID: 16 | Distância: 87
ID: 19 | Distância: 84

ID: 9 | Distância: 75
ID: 24 | Distância: 73
ID: 17 | Distância: 70
ID: 8 | Distância: 66
ID: 12 | Distância: 56
ID: 18 | Distância: 54
ID: 3 | Distância: 51

Caminhão: 2
Quantidade Rotas: 8
Total Km: 566
ID: 21 | Distância: 89
ID: 13 | Distância: 83
ID: 7 | Distância: 74
ID: 5 | Distância: 72
ID: 11 | Distância: 71
ID: 22 | Distância: 64
ID: 6 | Distância: 62
ID: 1 | Distância: 51

Programação Dinâmica:

Esse algoritmo foi baseado no problema da partição. O objetivo utilizando programação dinâmica é monitorar as formas ótimas de particionamento que serão salvas em uma tabela auxiliar. A tabela auxiliar **tabela[n][k]** na linha 104 do arquivo AppProgDinam possui como número de linhas o tamanho do array de rotas(n) e como número de colunas(k) o número de caminhões(partições)

Sendo assim, cada elemento de tabela[i][j] vai ser calculado minimizando a soma máxima da partição quando o array fornecido for dividido em j começando pelo índice i.

rotas: {s[0], s[1], ... , s[n-1]}

tabela[i][j] = min(max(tabela[x][j - 1], s[i] + s[i+1] + ... + s[x]));

x: de 0 até i-1

Evitando que ocorra somas parciais diversas vezes, o algoritmo calcula a soma cumulativa dos elementos. Exemplificando, a soma do índice i é calculada a partir da soma até o índice i -1.

soma[i] = soma[i - 1] + s[i]

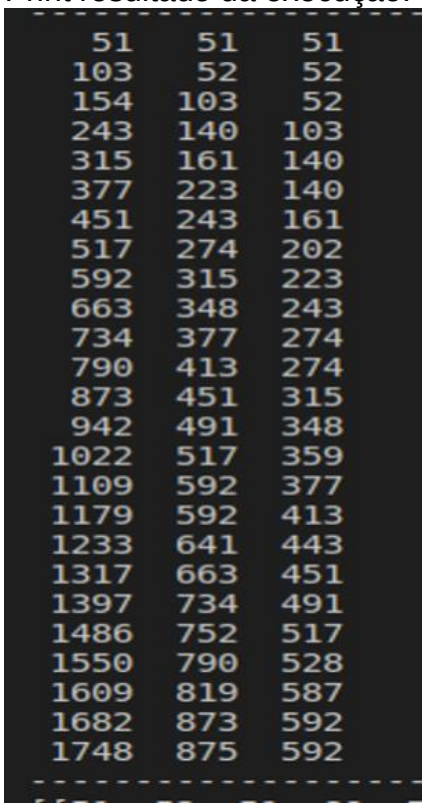
Calculando a soma de i até m é a mesma coisa que:
soma[m] - soma[i - 1]

Dessa forma é eliminada a repetição no cálculo de somas com mesmo valor.

Exemplo de execução(caminhoes_compacto):

Número de caminhões: 3
Rotas Utilizadas: [51, 52, 51, 89, 72, 62, 74, 66, 75, 71, 71, 56, 83, 69, 80, 87, 70, 54, 84, 80, 89, 64, 59, 73, 66]

Print resultado da execução:



51	51	51
103	52	52
154	103	52
243	140	103
315	161	140
377	223	140
451	243	161
517	274	202
592	315	223
663	348	243
734	377	274
790	413	274
873	451	315
942	491	348
1022	517	359
1109	592	377
1179	592	413
1233	641	443
1317	663	451
1397	734	491
1486	752	517
1550	790	528
1609	819	587
1682	873	592
1748	875	592

Tabela:

	1	2	3
51	51	51	51
52	103	52	52
51	154	103	52
89	243	140	103
72	315	161	140
62	377	223	140
74	451	243	161
66	517	274	202
75	592	315	223
71	663	348	243
71	734	377	274
56	790	413	274
83	873	451	315
69	942	491	348
80	1022	517	359
87	1109	592	377
70	1179	592	413
54	1233	641	443
84	1317	663	451
80	1397	734	491
89	1486	752	517
64	1550	790	528
59	1609	819	587
73	1682	873	592
66	1748	875	592

Backtracking:

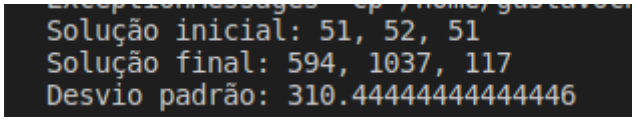
Para dividir as rotas entre os caminhões (partições), tal que a diferença entre o total de km a ser percorrido por cada caminhão seja mínima utilizando um algoritmo baseado em backtracking. O critério utilizado foi: a melhor solução é aquela que apresentar o menor desvio padrão (diferença entre média e o valor) considerando o total de km do caminhão e a média gerada a partir da soma de km de todas as rotas dividida pela quantidade de caminhões.

Algoritmo:

- Inicializamos uma pilha com soluções a serem testadas
- iniciamos a primeira solução atribuindo aos caminhões vazios os elementos de índice 0 a n (quantidade de caminhões) da lista de rotas a serem atribuídas e empilhamos;
- Em seguida enquanto houver soluções na pilha, desempilhamos e chamamos recursivamente o algoritmo de backtracking passando como solução atual a solução a ser testada desempilhada.
- O algoritmo de backtracking gera as possíveis soluções e poda a pior entre elas para reduzir a quantidade de testes realizados,
    - isso acontece da seguinte maneira da seguinte maneira: para cada caminhão adicionar a rota selecionada, empilhar nova solução, remover rota selecionada e ir para próximo caminhão. Quando não existir mais rotas a serem selecionadas, o valor da solução é comparado com a melhor solução e caso esse valor seja melhor (o

desvio padrão da solução for menor que o desvio padrão da melhor solução), a melhor solução passa a ser a solução atual.

Resultado utilizando arquivo “caminhoes_compacto.txt”:



Podemos verificar que infelizmente o algoritmo encontra uma solução diferente da solução ótima, a provável causa disso são as podas da pior solução que ocasionam no não teste de todas as soluções, além de possíveis ineficiências no código desenvolvido.

2) Divisão e Conquista:

Para descobrir o período que houve o maior acumulo de temperatura no ano utilizando divisão e conquista, primeiro divide-se o array com as temperaturas em 2 metades.
Após isso, retorna-se o subArray com valor máximo da seguinte árvore:

- subArray com soma máxima na metade da esquerda(Chamada recursiva)
- subArray com soma máxima na metade da direita(Chamada recursiva)
- subArray com soma máxima de modo que o subArray ultrapasse/cruze o ponto médio

Para encontrar a soma do subArray que cruza o ponto médio, encontra-se a soma máxima começando no indice médio e terminando em algum índice à esquerda do meio. Depois encontra-se a soma máxima começando do meio + 1 e terminando em algum ponto à direita do meio + 1. Por fim, combina-se os dois valores e retorna o máximo entre esquerda, direita e a combinação de ambos.

Após retornar os períodos com maior temperatura de cada ano, verifica-se se ouve coincidências entre os dias de um ano com os outros. Por final, roda-se o algoritmo novamente dessa vez para verificar o período com maior temperatura agrupando os anos (Juntando todos eles).

Exemplo (dois primeiros anos do arquivo):

```
Separado por ano:


[212, 338, 22]
Início[Posição/Dia]: 213
Fim[Posição/Dia]: 339
Soma: 22


[207, 340, 22]
Início[Posição/Dia]: 208
Fim[Posição/Dia]: 341
Soma: 22
```

```
Coincidencias entre os anos 1 e 2: [213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239
, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 27
3, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 3
07, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339]
```