Universidade do Minho

2ºSemestre 2019/20

(MIEI, 3ºAno)

Modelos Estocásticos de Investigação Operacional

Trabalho Prático

Identificação do Grupo

Número:	*Nome completo:*	*Rubrica:*
A81283	Hugo Filipe Oliveira de Sousa Faria	Hugo Faria
A80791	João Diogo Mendes Teixeira da Mota	João Diogo Mota
A82313	Pedro Teixeira Gonçalves	Pedro Gonçalves
A81716	Rodolfo António Vieira da Silva	Rodolfo Silva

Data de entrega: 2020-_05_-_11_

Conteúdo

Lista de Figuras

1 Introdução

O estudo da situação e gestão do sistema permite ao empresário ter uma perceção para instantes de tempo particulares das necessidades diárias de transferência de automóveis. Desta forma, garante-se a disponibilidade de automóveis para alugar, não culminando numa transferência descontrolada, que tornariam o negócio menos lucrativo.

O objetivo deste projeto é adquirir competências relativas à Programação Dinâmica Estocástica. Assim, serão analisados todos os cenários possíveis de acordo com as informações fornecidas no enunciado, permitindo a formulação e resolução do problema.

Com este estudo, pretende-se concluir qual a política ótima de transferência diária de automóveis entre as duas filiais de um empresário, de forma a garantir um resultado proveitoso para o mesmo.

2 Descrição do Problema

O problema em questão relaciona-se com problemas de decisão *Markovianos*, tendo por base problemas com um número infinito de estágios com alternativas. Estes são caracterizados por, para cada horizonte n, poder obter-se uma política ótima, que irá eventualmente variar com n.

Partindo do pressuposto que este método consiste em tomar, como aproximação para o valor de g* (ganho da política ótima para um número infinito de estágios), o valor de g_n obtido para um valor de n elevado, e adotar a política ótima correspondente ao número finito de estágios, n, a fórmula de recorrência é dada por [1]

$$F_n = opt\left\lfloor \left\langle Q_n^k + P_n^k F_n - 1 \right\rangle \right\rfloor$$

Sendo esta utilizada para calcular o resultado final dado por:

$$Dn(Nx1) = Fn - F(n-1)$$

Inicialmente, é pedido que seja gerado um programa que permita a implementação do método de iteração de valor, de modo a obter a solução pretendida. Em seguida, deve ser formulado o problema que permita o empresário obter a política ótima de transferência diária de automóveis entre as suas filiais.

3 Formulação do Problema

Propõe-se o estudo da política óptima de transferência diária de automóveis entre duas filiais. Para este efeito, recorreu-se a um modelo de Programação Dinâmica Estocástica. Para a resolução do problema, e tendo em consideração que o empresário em questão gere duas filiais, são cruciais os dados descritos de seguida.

- São diariamente entregues um número variável de automóveis;
- Cada automóvel entregue apenas fica disponível no dia seguinte;
- Cada filial não pode ter mais do que doze automóveis no final de cada dia;
- O empresário pode transferir um número máximo de três automóveis por dia para a outra filial, para reajustar os stocks;
- O custo de transferir um automóvel entre filiais é de 7€;
- Cada uma das filiais tem um estacionamento limitado a oito automóveis;
- Está associado um custo de 10€/noite, caso a filial tenha mais de oito automóveis.

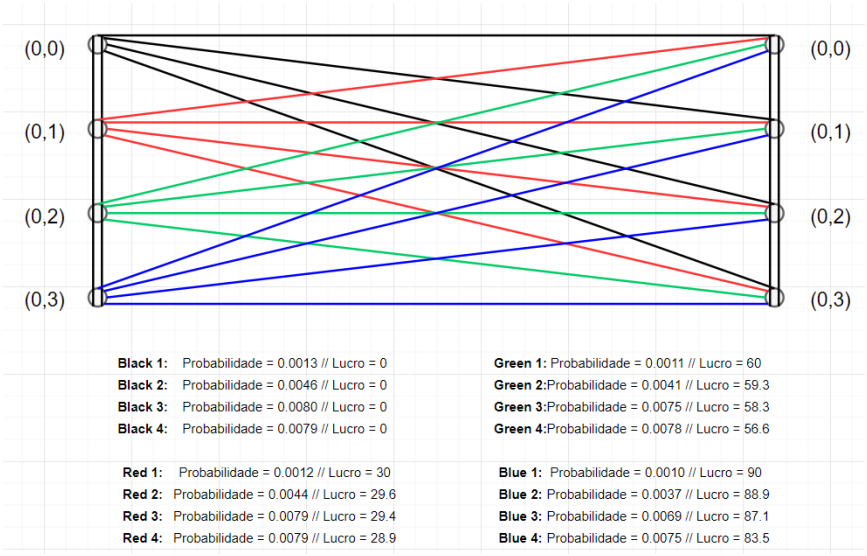
4 Considerações e Simplificações

De forma a tornar possível a resolução do problema apresentado, é necessária a recolha de alguns dados indicados de forma implícita no enunciado apresentado. De destacar:

- **Estados:** Os estados a serem considerados neste problema são o número total de automóveis presentes por filial ao início do dia.
- **Estágios:** Uma vez que os estados são mutuamente exclusivos, ou seja, o sistema só poderá ser descrito por um único estado, num determinado instante ou estágio, e considerando que se pretende avaliar a política ótima de transferência diária de automóveis, o estágio corresponde a 1 dia.
- **Decisões:** Dado que se trata de um problema de estágios infinitos com decisões, foram definidas como decisões a possibilidade de transferência de automóveis entre filais.

- **Simplificação de Probabilidades:** Devido à necessidade de realização de *shifts* de modo a gerar as tabelas que contêm as probabilidades de transferência de automóveis, algumas colunas iriam apresentar valores com probabilidade 0, levando a que a soma dos elementos de uma linha fosse diferente de 1. Uma vez que este resultado iria afetar o D_n, o grupo optou por simplificar estas probabilidades, distribuindo pelas colunas da linha com valor diferente de 0, o resto da subtração de 1 pela soma obtida, de forma estabilizar as mesmas, permitindo obter o valor de probabilidade 1 para cada linha.

- **Transferência de Automóveis:** À falta de indicação, assumiu-se que o empresário poderá realizar a transferência de automóveis entre as suas filiais no dia em que estes são entregues.



Pequeno exemplo da rede de nodos com 4 dos estados do problema

5 Resolução do Problema

Para a resolução deste exercício, recorreu-se à linguagem de programação *Python*.

Após identificados os dados necessários, e tendo em consideração o ficheiro que contém as probabilidades de pedidos e entregas de automóveis por filial fornecido pela equipa docente, o prosseguiu-se para a construção da matriz das transição.

5.1 Criação das Matrizes de Transição

5.1.1 Matrizes de Transição Por Filial

Uma matriz transição é uma matriz de probabilidades de mudança a partir de um estado para qualquer outro estado. Como tal, o primeiro passo para o desenvolvimento da mesma foi a criação de duas matrizes 13x13, dada a existência de treze estados e duas filiais.

Para dar início ao preenchimento das matrizes, foram criados dois ciclos aninhados com o intuito de preencher as linhas e colunas da matriz. O ciclo mais aninhado contém um caso especial, sendo este o de o estado passar para 12 automóveis presentes na filial. O caso referido deve-se à possibilidade de serem entregues mais automóveis do que o permitido por filial, pelo que estes serão transferidos para outras filiais geridas por diferentes empresários.

Para cada iteração do ciclo, é calculada a probabilidade acumulada que representa a situação em que são efetuados um número de pedidos correspondentes à quantidade de automóveis disponíveis, juntamente com a possibilidade de existirem pedidos que não serão satisfeitos, devido à inexistência de automóveis disponíveis para aluguer. Neste caso, para o sistema mudar de um estado $E1$, com um determinado número de automóveis, para um estado $E2$, recorrer-se-á à probabilidade acumulada referida, juntamente com a probabilidade de serem entregues o número de automóveis de $E2$, sendo essa probabilidade dada por:

$$Probabilidade = (ProbabilidadeAcumulada) * (NumeroAutomoveisE2)$$

Após este cálculo, serão naturalmente adicionados os casos em que o número de pedidos é inferior ao número de automóveis disponíveis. Estas somas serão realizadas num ciclo descendente, em que tanto o número de pedidos como o número de entregas serão decrementados, a fim de serem satisfeitos todos os casos possíveis para a mudança de estado em questão.

Por fim, para o caso especial em que o estado passa para 12 automóveis na filial, o processo é idêntico ao anteriormente mencionado. No entanto, sabendo que a uma filial podem ser entregues automóveis, mesmo que esta se encontre lotada, levando à transferência dos mesmos para filiais de diferentes empresários, é indispensável a consideração destes casos. Para este efeito, foi definida uma variável que irá acumular todas as probabilidades de entrega necessárias para que a filial fique com o número de carros pretendidos no estado destino, desde que hajam 0 pedidos até que hajam 12 pedidos. A consideração dos 12 pedidos deve-se ao facto de este ser o limite de pedidos fornecido no conjunto de dados atribuído para a resolução do problema. Está variável será, de seguida, multiplicada pela probabilidade de pedidos, obtendo a probabilidade final da transição em questão.

5.1.2 Unificação Das Matrizes

Após a obtenção das matrizes de transição 13x13 para cada uma das filias, procedeu-se à união das duas matrizes. Como tal, o número de estados aumentou para 169, devido a todas as combinações possíveis através de um par de números, cada um deles entre 0 e 12. Como o número de estados é de 169 e uma matriz de transição é necessariamente quadrada, de modo a abranger todas as transições de estado possíveis, foi construída uma nova matriz de transição, esta com 169 linhas e 169 colunas.

A construção da nova matriz é realizada através da multiplicação dos valores de ambas as matrizes, obtendo a probabilidade de transição de estados em ambas as filiais.

5.1.3 Matrizes de Transição com Decisão de Transferência > 0

Anteriormente, foi mencionado que este problema se trata de uma problema com um número infinito de estágios com decisões, e que o mesmo contém 7 decisões possíveis quanto à transferência de automóveis entre as duas filiais do empresário. De forma a englobar as decisões na resolução do exercício, foi desenvolvido um algoritmo que calcula as matrizes 13x13 para cada decisão possível, tanto para a filial que transfere os automóveis, como para a filial que os recebe.

Em relação à filial que irá transferir os automóveis, temos dois ciclos aninhados para percorrer a matriz para cada decisão possível (transferir um, dois ou três automóveis). Utilizando como exemplo a transferência de um automóvel, é realizado um *shift* para a esquerda na matriz de transição, uma vez que as probabilidades de mudar de um estado $E1$ para um estado $E2$

correspondem às probabilidades de ir do estado $E1$ para o estado $E2+1$ na matriz em que não há qualquer transferência de automóveis. Dado que as matrizes foram iniciadas com o valor 0 para cada posição, a última coluna, em que o estado final é a filial conter 12 automóveis, ficará com uma probabilidade de 0, o que corresponde à realidade, visto que uma filial que transfira um automóvel não poderá ficar com o número máximo de automóveis permitido, sendo este limite, no contexto do problema apresentado, 12 automóveis por filial. Dado este facto, a soma do valor das probabilidades da linha será um valor inferior a 1, pelo qual o grupo efetuou uma normalização dos valores da linha, tal como indicado na secção "Considerações e Simplificações".

Para as outras duas decisões de transferência de automóveis (transferir dois ou três automóveis), rege-se pelo mesmo procedimento, realizando, no entanto, um *shift* de duas e três colunas, respetivamente, para a esquerda.

No que diz respeito à filial que irá receber os automóveis da transferência, o *shift* terá que ser realizado para a direita, visto que as probabilidades de ir de um estado $E1$ para um estado $E2$ correspondem às probabilidades de ir do estado $E1$ para o estado $E2-1$ na matriz em que não há qualquer transferência de automóveis. Por um motivo idêntico à última coluna se encontrar com todos os valores a 0 nas matrizes da filial que transfere, no caso da matriz que recebe as transferências, a primeira coluna irá ter as probabilidades todas a 0, uma vez que uma filial que recebe automóveis nunca poderá estar num estado em que contenha 0 automóveis.

Após a obtenção das matrizes 13x13 para as 6 decisões restantes, foi aplicado o algoritmo para calcular a matriz 169x169 por cada uma das matrizes.

5.2 Criação das Matrizes de Contribuição

O processo da criação das matrizes de contribuição foi semelhante ao da construção das matrizes de transição. Dado que existem várias possibilidades para chegar de um estado a outro, o cálculo do ganho que o empresário terá com essa transição será obtida através de uma média dos custos de cada transição possível para chegar de um determinado estado a outro. Assim sendo, para cada par pedido/entrega que satisfaça a transição de estado, o custo será obtido através da seguinte expressão

$$custo+= \frac{probabilidadeAcumulada * pEntrega[entrega])}{matrizP[i][x]} * (pedido * 30)$$

em que *pEntrega[entrega]* é a probabilidade de entrega para o índice "entrega", e *matrizP[i][x]* é a probabilidade de transição de um estado i para

um estado x. Além disso, *pedido*30* representa o que o valor em que o empresário é creditado consoante o número de veículos que são alugados.

É importante notar que, tal como indicado no problema, caso uma filial contenha mais do que 8 automóveis, o empresário terá que pagar uma taxa de estacionamento de 10€. Como tal, será necessária uma condição que verifica o número de automóveis presentes na filial e, caso este valor seja superior a 8, terá que ser debitado um valor de 10€ à variável custo.

5.2.1 Unificação Das Matrizes

Do mesmo modo que se procedeu com as matrizes de transição, foi também construída uma matriz 169x169 resultante da unificação das duas matrizes de contribuição obtidas. Para tal, foram construídos dois ciclos aninhados para percorrerem todas as posições das matrizes 13x13 construídas previamente e, para cada posição, vai ser somado o custo presente em cada uma das duas matrizes, visto que ambas as filiais pertencem ao mesmo empresário.

5.2.2 Matrizes de Contribuição com Decisão de Transferência>0

Em relação à filial que irá transferir os automóveis, são construídos dois ciclos aninhados para preencher a matriz com o custo da transição de estado consoante a política de transferência que decidir tomar. Assim como com as matrizes de transição, será realizado um ou mais*shifts* para a esquerda, consoante o número de automóveis disponíveis. Neste passo, foram tidos em consideração os casos em que o número de veículos a transferir pela filial faça com que o número de automóveis nesta, estando previamente acima de 8, desça para um valor inferior, pois não será mais necessário que o empresário pague a taxa de estacionamento necessária, desde logo somando o valor dessa taxa (10€) ao custo. Além disso, terá que se ter em consideração a taxa de transferência de automóveis de uma filial para a outra (7€), sendo este valor (multiplicado pelo número de carros transferidos), subtraído ao custo.

Para a filial que recebe, o processo dos *shifts* será inverso. Neste caso, tem também que se ter em consideração os casos em que o estado transita de um número de automóveis inferior a 8 para um superior. Caso isto se verifique, como o empresário terá que pagar a taxa de utilização do espaço extra, serão subtraídos 10€ ao custo.

Por fim, é necessário aplicar o algoritmo de junção das matrizes para que cada par de matrizes de decisão dê origem a uma matriz 169x169.

5.3 Cálculo da Solução

Dada como concluída a fase de construção das matrizes de transição e contribuição, foi desenvolvido o algoritmo para chegar à solução pretendida. Como tal, o primeiro passo foi o cálculo das matrizes de esperança de contribuição de estágio (Q). Esta matriz, sendo uma matriz constituída apenas por uma coluna, é obtida através do somatório das multiplicações da matriz de transição pela respectiva matriz de contribuição.

De seguida, é calculada a matriz $Pn * Fn - 1$. Esta matriz, juntamente com a matriz Q, permitem calcular a matriz de esperança total da contribuição (Vn).

Para calcular a nova matriz, Fn, e visto que se trata de um problema em que queremos maximizar o lucro do empresário, serão selecionados os maiores valores de Vn por linha.

É também necessário o cálculo de Dn através da expressão

$$Dn(Nx1) = Fn - F(n-1)$$

Uma vez que este processo é iterativo, após um número suficientemente grande de iterações, verifica-se que os valores de Dn começam a estagnar, momento o qual se poderá parar a execução.

6 Análise de Dados

Depois de se ter executado o programa múltiplas vezes, atingiu-se a um valor D_n praticamente constante, sendo esse valor ≈ 217. Este resultado pode ser corroborado com o ficheiro *excel* remetido, representante de 200 iterações realizadas). Verificou-se ainda a estabilidade da política ótima a escolher para um aumento do número de iterações realizadas.

Legenda da política

- **0:** transferir 0 automóveis entre as filiais
- **1:** transferir 1 automóveis da filial 1 para a 2
- **2:** transferir 2 automóveis da filial 1 para a 2
- **3:** transferir 3 automóveis da filial 1 para a 2
- **4:** transferir 1 automóveis da filial 2 para a 1
- **5:** transferir 2 automóveis da filial 2 para a 1
- **6:** transferir 3 automóveis da filial 2 para a 1

Apesar da incerteza, os resultados aparentam encontrar-se dentro dos valores esperados, uma vez que, analisando as probabilidades inicialmente fornecidas, verifica-se que a filial 2 tem menores hipóteses de receber um elevado número de automóveis num dia quando em comparação com a filial 1. Por outro lado, a filial 2 apresenta hipóteses mais elevadas de alugar um grande número de automóveis por dia. Deste modo, é evidente que o método 3 seja o mais utilizado, uma vez que permite mais facilmente assegurar um bom número de vendas na filial 2.

7 Conclusões e Recomendações

O desenvolvimento deste estudo permitiu a todos os elementos do grupo construir uma ideia mais consistente da necessidade de executar um estudo prévio no que toca à gestão de inventários, gestão dos carros em cada filial, antes de definir medidas fixas a implementar, permitindo um aumento na qualidade de serviço.

Através de todo o processo efetuado, é agora sugerido pelo grupo, a transferência de 3 veículos da filial 1 para a filial 2, pois para a maioria dos casos, é assim se consegue obter o maior rendimento possível para o empresário. Caso essa transferência não seja possível, nos anexos deste relatório estão descritas quais as outras ações que permitem também obter bons rendimentos.

Em suma, este projeto permitiu ao grupo simular o planeamento do funcionamento de um consórcio de aluguer de automóveis, mostrando quais as melhores e/ou piores ações ano no que toca à gestão dos automóveis nas filiais. Além disso, permitiu ao grupo perceber a importância da programação dinâmica para a resolução deste género de problemas.

Referências

[1] DPS, EE, Universidade do Minho (2020). Modelos Estocásticos de Investigação Operacional

8 Anexos

Anexos das matrizes de probabilidades e de custos presentes no ficheiro *excel*, remetido juntamente com o relatório.

```
                0       1       2       3       4       5       6       7       8       9       10      11      12
f1Pedido =  [0.0528, 0.0904, 0.1216, 0.1396, 0.1184, 0.1112, 0.0988, 0.0852, 0.0596, 0.0568, 0.0368, 0.0220, 0.0068]
f1Entrega = [0.0348, 0.0820, 0.1180, 0.1396, 0.1264, 0.1188, 0.0932, 0.0888, 0.0640, 0.0556, 0.0436, 0.0256, 0.0096]

f2Pedido =  [0.0352, 0.0724, 0.1024, 0.1336, 0.1388, 0.1236, 0.1048, 0.0900, 0.0704, 0.0608, 0.0380, 0.0220, 0.0080]
f2Entrega = [0.0384, 0.1312, 0.2312, 0.2296, 0.1516, 0.1156, 0.0584, 0.0288, 0.0080, 0.0056, 0.0012, 0.0004, 0.0000]
```

Probabilidades que nos foram fornecidas

DN = [	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	
	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	
	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	
	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	
	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	
	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	
	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	].
	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679		
	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679		
	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679		
	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679		
	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679		
	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679		
	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679		
	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679		
	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679		
	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679		
	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679		
	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679		
	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679		
	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679		
	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679		
	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679		
	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679		
	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679		
	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679		
	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679	217.793067679		

Resultado do DN ao fim de 200 iterações

PO = [	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[2, 3, 1, 0, 4, 5, 6]	[2, 3, 1, 0, 4, 5, 6]	[4, 3, 2, 5, 1, 0, 6]	
	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[2, 3, 1, 0, 4, 5, 6]	[2, 1, 3, 0, 4, 5, 6]	[2, 3, 1, 0, 4, 5, 6]	[4, 3, 2, 5, 1, 0, 6]	
	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[2, 1, 3, 0, 4, 5, 6]	[2, 1, 0, 3, 4, 5, 6]	[2, 1, 3, 0, 4, 5, 6]	[4, 5, 3, 2, 1, 0, 6]	
	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[1, 0, 2, 3, 4, 5, 6]	[1, 0, 2, 3, 4, 5, 6]	[2, 0, 1, 3, 4, 5, 6]	[4, 5, 3, 2, 1, 0, 6]	
	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 0, 2, 1, 4, 5, 6]	[0, 1, 2, 3, 4, 5, 6]	[0, 1, 2, 3, 4, 5, 6]	[0, 2, 1, 3, 4, 5, 6]	[4, 5, 3, 2, 1, 0, 6]	
	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 0, 1, 4, 5, 6]	[0, 3, 1, 2, 4, 5, 6]	[0, 1, 2, 3, 4, 5, 6]	[0, 1, 2, 3, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[4, 3, 2, 5, 1, 0, 6]	
	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 0, 1, 4, 5, 6]	[0, 1, 3, 2, 4, 5, 6]	[0, 1, 2, 3, 4, 5, 6]	[1, 0, 2, 3, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 4, 2, 1, 5, 0, 6]	].
	[3, 2, 0, 1, 4, 5, 6]	[3, 0, 2, 1, 4, 5, 6]	[0, 3, 1, 2, 4, 5, 6]	[0, 1, 2, 3, 4, 5, 6]	[2, 3, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]		
	[3, 2, 0, 1, 4, 5, 6]	[3, 0, 2, 1, 4, 5, 6]	[3, 0, 1, 2, 4, 5, 6]	[3, 1, 2, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 4, 0, 6, 5]		
	[3, 0, 2, 1, 4, 5, 6]	[3, 2, 0, 1, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 4, 0, 6, 5]		
	[3, 2, 0, 1, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 4, 0, 5, 6]		
	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 4, 0, 5, 6]		
	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 4, 0, 5, 6]		
	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[2, 3, 1, 0, 4, 5, 6]	[2, 3, 1, 0, 4, 5, 6]	[3, 2, 1, 4, 0, 5, 6]		
	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[2, 3, 1, 0, 4, 5, 6]	[2, 3, 1, 0, 4, 5, 6]	[2, 3, 4, 1, 0, 5, 6]		
	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[2, 1, 3, 0, 4, 5, 6]	[2, 1, 3, 0, 4, 5, 6]	[4, 2, 3, 1, 0, 5, 6]		
	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[1, 2, 0, 3, 4, 5, 6]	[1, 2, 0, 3, 4, 5, 6]	[4, 2, 3, 1, 0, 5, 6]		
	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[0, 1, 2, 3, 4, 5, 6]	[0, 1, 2, 3, 4, 5, 6]	[0, 1, 2, 3, 4, 5, 6]	[4, 3, 2, 0, 1, 5, 6]		
	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 0, 1, 4, 5, 6]	[0, 1, 2, 3, 4, 5, 6]	[0, 1, 2, 3, 4, 5, 6]	[0, 1, 2, 3, 4, 5, 6]	[4, 3, 2, 1, 0, 5, 6]		
	[3, 2, 1, 0, 4, 5, 6]	[3, 0, 2, 1, 4, 5, 6]	[0, 1, 2, 3, 4, 5, 6]	[0, 1, 2, 3, 4, 5, 6]	[2, 1, 3, 0, 4, 5, 6]	[3, 2, 4, 1, 0, 5, 6]		
	[3, 2, 0, 1, 4, 5, 6]	[0, 3, 2, 1, 4, 5, 6]	[0, 1, 2, 3, 4, 5, 6]	[0, 1, 2, 3, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 4, 0, 5, 6]		
	[3, 0, 2, 1, 4, 5, 6]	[3, 0, 2, 1, 4, 5, 6]	[0, 1, 3, 2, 4, 5, 6]	[1, 2, 3, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[4, 3, 2, 6, 5, 1, 0]		
	[3, 0, 2, 1, 4, 5, 6]	[3, 0, 2, 1, 4, 5, 6]	[3, 1, 2, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 6, 5]	[4, 3, 2, 6, 5, 1, 0]		
	[3, 2, 0, 1, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[4, 3, 2, 5, 6, 1, 0]		
	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[4, 3, 2, 5, 1, 6, 0]		
	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[4, 3, 2, 5, 1, 6, 0]		
	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[2, 3, 1, 0, 4, 5, 6]	[3, 2, 1, 0, 4, 5, 6]	[4, 3, 2, 1, 5, 0, 6]		

Politica de escolha óptima ao fim de 200 iterações

```python
def matrixCalculator(filial):
    if(filial == 1):
        pEntrega = f1Entrega
        pPedido = f1Pedido
    else:
        pEntrega = f2Entrega
        pPedido = f2Pedido

    for i in range(13):
        lineTotal = 0

        for x in range(12):
            pedido = i
            entrega = x
            probability = 0

            probabilidadeAcumulada = 0
            for l in range(pedido,13):
                probabilidadeAcumulada += pPedido[l]

            probability += probabilidadeAcumulada * pEntrega[entrega]
            pedido -= 1
            entrega -= 1

            while entrega >= 0 and pedido >= 0:
                probability += pPedido[pedido] * pEntrega[entrega]
                pedido -= 1
                entrega -= 1

            if(filial == 1):
                matrixF1[i][x] += probability
            else:
                matrixF2[i][x] += probability

            lineTotal += probability
```

```
        pedido = i
        entrega = 12
        probability = 0
        probabilidadeEntregasAc = 0

        probabilidadeEntregasAc += pEntrega[entrega]
        probabilidadeAcumulada = 0

        for k in range(pedido,13):
            probabilidadeAcumulada += pPedido[k]

        probability += probabilidadeAcumulada * probabilidadeEntregasAc
        pedido-=1
        entrega-=1

        while pedido >= 0 :
            probabilidadeEntregasAc += pEntrega[entrega]
            probability += pPedido[pedido] * probabilidadeEntregasAc
            pedido-=1
            entrega-=1

        lineTotal += probability

        if(filial == 1):
            matrixF1[i][12] += probability
        else:
            matrixF2[i][12] += probability
```

```
def bigMatrixCalculator():
    coluna = 0
    linha = 0
    for f1Linha in range(13):
        for f1Coluna in range(13):
            coluna = 0
            for f2Linha in range(13):
                for f2Coluna in range(13):
                    matrixF1xF2[linha][coluna] = matrixF1[f1Linha][f2Linha] * matrixF2[f1Coluna][f2Coluna]
                    matrixF1_C1_F2_F1xF2[linha][coluna] = matrixF1_C1_F2_F1[f1Linha][f2Linha] * matrixF1_C1_F2_F2[f1Coluna][f2Coluna]
                    matrixF1_C2_F2_F1xF2[linha][coluna] = matrixF1_C2_F2_F1[f1Linha][f2Linha] * matrixF1_C2_F2_F2[f1Coluna][f2Coluna]
                    matrixF1_C3_F2_F1xF2[linha][coluna] = matrixF1_C3_F2_F1[f1Linha][f2Linha] * matrixF1_C3_F2_F2[f1Coluna][f2Coluna]
                    matrixF2_C1_F1_F1xF2[linha][coluna] = matrixF2_C1_F1_F1[f1Linha][f2Linha] * matrixF2_C1_F1_F2[f1Coluna][f2Coluna]
                    matrixF2_C2_F1_F1xF2[linha][coluna] = matrixF2_C2_F1_F1[f1Linha][f2Linha] * matrixF2_C2_F1_F2[f1Coluna][f2Coluna]
                    matrixF2_C3_F1_F1xF2[linha][coluna] = matrixF2_C3_F1_F1[f1Linha][f2Linha] * matrixF2_C3_F1_F2[f1Coluna][f2Coluna]

                    matrixC_F1xF2[linha][coluna] = matrixC_F1[f1Linha][f2Linha] + matrixC_F2[f1Coluna][f2Coluna]
                    matrixC_F1_C1_F2_F1xF2[linha][coluna] = matrixC_F1_C1_F2_F1[f1Linha][f2Linha] + matrixC_F1_C1_F2_F2[f1Coluna][f2Coluna]
                    matrixC_F1_C2_F2_F1xF2[linha][coluna] = matrixC_F1_C2_F2_F1[f1Linha][f2Linha] + matrixC_F1_C2_F2_F2[f1Coluna][f2Coluna]
                    matrixC_F1_C3_F2_F1xF2[linha][coluna] = matrixC_F1_C3_F2_F1[f1Linha][f2Linha] + matrixC_F1_C3_F2_F2[f1Coluna][f2Coluna]
                    matrixC_F2_C1_F1_F1xF2[linha][coluna] = matrixC_F2_C1_F1_F1[f1Linha][f2Linha] + matrixC_F2_C1_F1_F2[f1Coluna][f2Coluna]
                    matrixC_F2_C2_F1_F1xF2[linha][coluna] = matrixC_F2_C2_F1_F1[f1Linha][f2Linha] + matrixC_F2_C2_F1_F2[f1Coluna][f2Coluna]
                    matrixC_F2_C3_F1_F1xF2[linha][coluna] = matrixC_F2_C3_F1_F1[f1Linha][f2Linha] + matrixC_F2_C3_F1_F2[f1Coluna][f2Coluna]

                    coluna += 1
            linha += 1
```

```
def matrixBuilder_TrfXCarros():
    for linha in range(0,13):
        for coluna in range(0,12):
            matrixF1_C1_F2_F1[linha][coluna] = matrixF1[linha][(coluna+1)]
            matrixF2_C1_F1_F2[linha][coluna] = matrixF2[linha][(coluna+1)]

    for linha in range(0,13):
        for coluna in range(0,11):
            matrixF1_C2_F2_F1[linha][coluna] = matrixF1[linha][(coluna+2)]
            matrixF2_C2_F1_F2[linha][coluna] = matrixF2[linha][(coluna+2)]

    for linha in range(0,13):
        for coluna in range(0,10):
            matrixF1_C3_F2_F1[linha][coluna] = matrixF1[linha][(coluna+3)]
            matrixF2_C3_F1_F2[linha][coluna] = matrixF2[linha][(coluna+3)]

    for linha in range(0,13):
        sum1,sum2 = 0,0
        for coluna in range(0,13):
            sum1 += matrixF1_C1_F2_F1[linha][coluna]
            sum2 += matrixF2_C1_F1_F2[linha][coluna]
        for coluna in range(0,12):
            matrixF1_C1_F2_F1[linha][coluna] += ((1-sum1) * (matrixF1_C1_F2_F1[linha][coluna] / sum1))
            matrixF2_C1_F1_F2[linha][coluna] += ((1-sum2) * (matrixF2_C1_F1_F2[linha][coluna] / sum2))

    for linha in range(0,13):
        sum1,sum2 = 0,0
        for coluna in range(0,13):
            sum1 += matrixF1_C2_F2_F1[linha][coluna]
            sum2 += matrixF2_C2_F1_F2[linha][coluna]
        for coluna in range(0,11):
            matrixF1_C2_F2_F1[linha][coluna] += ((1-sum1) * (matrixF1_C2_F2_F1[linha][coluna] / sum1))
            matrixF2_C2_F1_F2[linha][coluna] += ((1-sum2) * (matrixF2_C2_F1_F2[linha][coluna] / sum2))

    for linha in range(0,13):
        sum1,sum2 = 0,0
        for coluna in range(0,13):
            sum1 += matrixF1_C3_F2_F1[linha][coluna]
            sum2 += matrixF2_C3_F1_F2[linha][coluna]
        for coluna in range(0,11):
            matrixF1_C3_F2_F1[linha][coluna] += ((1-sum1) * (matrixF1_C3_F2_F1[linha][coluna] / sum1))
            matrixF2_C3_F1_F2[linha][coluna] += ((1-sum2) * (matrixF2_C3_F1_F2[linha][coluna] / sum2))
```

```
def matrixBuilder_RcbXCarros():
    for linha in range(0,13):
        for coluna in range(1,13):
                matrixF1_C1_F2_F2[linha][coluna] = matrixF2[linha][(coluna-1)]
                matrixF2_C1_F1_F1[linha][coluna] = matrixF1[linha][(coluna-1)]

    for linha in range(0,13):
        for coluna in range(2,13):
                matrixF1_C2_F2_F2[linha][coluna] = matrixF2[linha][(coluna-2)]
                matrixF2_C2_F1_F1[linha][coluna] = matrixF1[linha][(coluna-2)]

    for linha in range(0,13):
        for coluna in range(3,13):
                matrixF1_C3_F2_F2[linha][coluna] = matrixF2[linha][(coluna-3)]
                matrixF2_C3_F1_F1[linha][coluna] = matrixF1[linha][(coluna-3)]

    for linha in range(0,13):
        sum1,sum2 = 0,0
        for coluna in range(0,13):
            sum1 += matrixF1_C1_F2_F2[linha][coluna]
            sum2 += matrixF2_C1_F1_F1[linha][coluna]
        for coluna in range(1,13):
            matrixF1_C1_F2_F2[linha][coluna] += ((1-sum1) * (matrixF1_C1_F2_F2[linha][coluna] / sum1))
            matrixF2_C1_F1_F1[linha][coluna] += ((1-sum2) * (matrixF2_C1_F1_F1[linha][coluna] / sum2))

    for linha in range(0,13):
        sum1,sum2 = 0,0
        for coluna in range(0,13):
            sum1 += matrixF1_C2_F2_F2[linha][coluna]
            sum2 += matrixF2_C2_F1_F1[linha][coluna]
        for coluna in range(2,13):
            matrixF1_C2_F2_F2[linha][coluna] += ((1-sum1) * (matrixF1_C2_F2_F2[linha][coluna] / sum1))
            matrixF2_C2_F1_F1[linha][coluna] += ((1-sum2) * (matrixF2_C2_F1_F1[linha][coluna] / sum2))

    for linha in range(0,13):
        sum1,sum2 = 0,0
        for coluna in range(0,13):
            sum1 += matrixF1_C3_F2_F2[linha][coluna]
            sum2 += matrixF2_C3_F1_F1[linha][coluna]
        for coluna in range(3,13):
            matrixF1_C3_F2_F2[linha][coluna] += ((1-sum1) * (matrixF1_C3_F2_F2[linha][coluna] / sum1))
            matrixF2_C3_F1_F1[linha][coluna] += ((1-sum2) * (matrixF2_C3_F1_F1[linha][coluna] / sum2))
```

```
def costMatrixCalculator(filial):
    if(filial == 1):
        pEntrega = f1Entrega
        pPedido = f1Pedido
        matrizP = matrixF1
    else:
        pEntrega = f2Entrega
        pPedido = f2Pedido
        matrizP = matrixF2

    for i in range(13):
        lineTotal = 0

        for x in range(12):
            pedido = i
            entrega = x
            custo = 0

            probabilidadeAcumulada = 0
            for l in range(pedido,13):
                probabilidadeAcumulada += pPedido[l]

            custo += ( (probabilidadeAcumulada * pEntrega[entrega]) / matrizP[i][x] ) * (pedido * 30)
            #custo += (probabilidadeAcumulada * (pedido * 30)) * pEntrega[entrega]
            pedido -= 1
            entrega -= 1

            while entrega >= 0 and pedido >= 0:
                custo += ( (pPedido[pedido] * pEntrega[entrega]) / matrizP[i][x] ) * (pedido * 30)
                #custo += (pPedido[pedido] * (pedido * 30)) * pEntrega[entrega]
                pedido -= 1
                entrega -= 1

            if( x >= 9):
                custo -= 10

            #print("Custo de ir do estado ",i," para o estado ",x," :",custo)

            if(filial == 1):
                matrixC_F1[i][x] += custo
            else:
                matrixC_F2[i][x] += custo

            lineTotal += custo
```

```
    pedido = i
    entrega = 12
    custo = 0
    probabilidadeEntregasAc = 0

    probabilidadeEntregasAc += pEntrega[entrega]
    probabilidadeAcumulada = 0

    for k in range(pedido,13):
        probabilidadeAcumulada += pPedido[k]

    if(matrizP[i][12] != 0):
        custo += ( (probabilidadeAcumulada * probabilidadeEntregasAc) / matrizP[i][12] ) * (pedido * 30)

    #custo += (probabilidadeAcumulada * (pedido * 30)) * probabilidadeEntregasAc
    pedido-=1
    entrega-=1

    while pedido >= 0 :
        probabilidadeEntregasAc += pEntrega[entrega]
        if(matrizP[i][12] != 0):
            custo += ( (pPedido[pedido] * probabilidadeEntregasAc) / matrizP[i][12] ) * (pedido * 30)
        #custo += (pPedido[pedido]*(pedido * 30)) * probabilidadeEntregasAc
        pedido-=1
        entrega-=1

    custo -= 10

    lineTotal += custo

    #print("Custo de ir do estado ",i," para o estado 12 :",custo)
    if(filial == 1):
        matrixC_F1[i][12] += custo
    else:
        matrixC_F2[i][12] += custo
```

```
def costMatrixBuilder_TrfXCarros():
    for linha in range(0,13):
        for coluna in range(0,12):
            if( (coluna <= 8) and ((coluna + 1) >= 9) ):
                matrixC_F1_C1_F2_F1[linha][coluna] = matrixC_F1[linha][(coluna+1)] - 7 + 10
                matrixC_F2_C1_F1_F2[linha][coluna] = matrixC_F2[linha][(coluna+1)] - 7 + 10
            else:
                matrixC_F1_C1_F2_F1[linha][coluna] = matrixC_F1[linha][(coluna+1)] - 7
                matrixC_F2_C1_F1_F2[linha][coluna] = matrixC_F2[linha][(coluna+1)] - 7

    for linha in range(0,13):
        for coluna in range(0,11):
            if( (coluna <= 8) and ((coluna + 2) >= 9) ):
                matrixC_F1_C2_F2_F1[linha][coluna] = matrixC_F1[linha][(coluna+2)] - 14 + 10
                matrixC_F2_C2_F1_F2[linha][coluna] = matrixC_F2[linha][(coluna+2)] - 14 + 10
            else:
                matrixC_F1_C2_F2_F1[linha][coluna] = matrixC_F1[linha][(coluna+2)] - 14
                matrixC_F2_C2_F1_F2[linha][coluna] = matrixC_F2[linha][(coluna+2)] - 14

    for linha in range(0,13):
        for coluna in range(0,10):
            if( (coluna <= 8) and ((coluna + 3) >= 9) ):
                matrixC_F1_C3_F2_F1[linha][coluna] = matrixC_F1[linha][(coluna+3)] - 21 + 10
                matrixC_F2_C3_F1_F2[linha][coluna] = matrixC_F2[linha][(coluna+3)] - 21 + 10
            else:
                matrixC_F1_C3_F2_F1[linha][coluna] = matrixC_F1[linha][(coluna+3)] - 21
                matrixC_F2_C3_F1_F2[linha][coluna] = matrixC_F2[linha][(coluna+3)] - 21

    for linha in range(0,13):
        matrixC_F1_C1_F2_F1[linha][12] = -10000
        matrixC_F2_C1_F1_F2[linha][12] = -10000
        matrixC_F1_C2_F2_F1[linha][12] = -10000
        matrixC_F2_C2_F1_F2[linha][12] = -10000
        matrixC_F1_C2_F2_F1[linha][11] = -10000
        matrixC_F2_C2_F1_F2[linha][11] = -10000
        matrixC_F1_C3_F2_F1[linha][12] = -10000
        matrixC_F2_C3_F1_F2[linha][12] = -10000
        matrixC_F1_C3_F2_F1[linha][11] = -10000
        matrixC_F2_C3_F1_F2[linha][11] = -10000
        matrixC_F1_C3_F2_F1[linha][10] = -10000
        matrixC_F2_C3_F1_F2[linha][10] = -10000
```

```
def costMatrixBuilder_RcbXCarros():
    for linha in range(0,13):
        for coluna in range(1,13):
            if( coluna >= 9 and (coluna-1) <= 8):
                matrixC_F1_C1_F2_F2[linha][coluna] = matrixC_F2[linha][(coluna-1)] - 10
                matrixC_F2_C1_F1_F1[linha][coluna] = matrixC_F1[linha][(coluna-1)] - 10
            else:
                matrixC_F1_C1_F2_F2[linha][coluna] = matrixC_F2[linha][(coluna-1)]
                matrixC_F2_C1_F1_F1[linha][coluna] = matrixC_F1[linha][(coluna-1)]

    for linha in range(0,13):
        for coluna in range(2,13):
            if( coluna >= 9 and (coluna-2) <= 8):
                matrixC_F1_C2_F2_F2[linha][coluna] = matrixC_F2[linha][(coluna-2)] - 10
                matrixC_F2_C2_F1_F1[linha][coluna] = matrixC_F1[linha][(coluna-2)] - 10
            else:
                matrixC_F1_C2_F2_F2[linha][coluna] = matrixC_F2[linha][(coluna-2)]
                matrixC_F2_C2_F1_F1[linha][coluna] = matrixC_F1[linha][(coluna-2)]

    for linha in range(0,13):
        for coluna in range(3,13):
            if( coluna >= 9 and (coluna-3) <= 8):
                matrixC_F1_C3_F2_F2[linha][coluna] = matrixC_F2[linha][(coluna-3)] - 10
                matrixC_F2_C3_F1_F1[linha][coluna] = matrixC_F1[linha][(coluna-3)] - 10
            else:
                matrixC_F1_C3_F2_F2[linha][coluna] = matrixC_F2[linha][(coluna-3)]
                matrixC_F2_C3_F1_F1[linha][coluna] = matrixC_F1[linha][(coluna-3)]

    for linha in range(0,13):
        matrixC_F1_C1_F2_F2[linha][0] = -10000
        matrixC_F2_C1_F1_F1[linha][0] = -10000
        matrixC_F1_C2_F2_F2[linha][0] = -10000
        matrixC_F2_C2_F1_F1[linha][0] = -10000
        matrixC_F1_C2_F2_F2[linha][1] = -10000
        matrixC_F2_C2_F1_F1[linha][1] = -10000
        matrixC_F1_C3_F2_F2[linha][0] = -10000
        matrixC_F2_C3_F1_F1[linha][0] = -10000
        matrixC_F1_C3_F2_F2[linha][1] = -10000
        matrixC_F2_C3_F1_F1[linha][1] = -10000
        matrixC_F1_C3_F2_F2[linha][2] = -10000
        matrixC_F2_C3_F1_F1[linha][2] = -10000
```

```
def calculateQ():
    for linha in range(169):
        for coluna in range(169):
            q0[linha][0] += matrixF1xF2[linha][coluna] * matrixC_F1xF2[linha][coluna]
            qF1_1_F2[linha][0] += matrixF1_C1_F2_F1xF2[linha][coluna] * matrixC_F1_C1_F2_F1xF2[linha][coluna]
            qF1_2_F2[linha][0] += matrixF1_C2_F2_F1xF2[linha][coluna] * matrixC_F1_C2_F2_F1xF2[linha][coluna]
            qF1_3_F2[linha][0] += matrixF1_C3_F2_F1xF2[linha][coluna] * matrixC_F1_C3_F2_F1xF2[linha][coluna]
            qF2_1_F1[linha][0] += matrixF2_C1_F1_F1xF2[linha][coluna] * matrixC_F2_C1_F1_F1xF2[linha][coluna]
            qF2_2_F1[linha][0] += matrixF2_C2_F1_F1xF2[linha][coluna] * matrixC_F2_C2_F1_F1xF2[linha][coluna]
            qF2_3_F1[linha][0] += matrixF2_C3_F1_F1xF2[linha][coluna] * matrixC_F2_C3_F1_F1xF2[linha][coluna]
```

```
def beginAlgoritmo(iteracoes,fnAnterior):

    if(iteracoes < itsAFazer):
        pnXfn = [[0.0 for h in range(1)] for o in range(169)]
        f1c1f2 = [[0.0 for h in range(1)] for o in range(169)]
        f1c2f2 = [[0.0 for h in range(1)] for o in range(169)]
        f1c3f2 = [[0.0 for h in range(1)] for o in range(169)]
        f2c1f1 = [[0.0 for h in range(1)] for o in range(169)]
        f2c2f1 = [[0.0 for h in range(1)] for o in range(169)]
        f2c3f1 = [[0.0 for h in range(1)] for o in range(169)]

        # VN = (Pn * FnAnterior) + Q
        vn = [[0.0 for h in range(1)] for o in range(169)]
        vnf1c1f2 = [[0.0 for h in range(1)] for o in range(169)]
        vnf1c2f2 = [[0.0 for h in range(1)] for o in range(169)]
        vnf1c3f2 = [[0.0 for h in range(1)] for o in range(169)]
        vnf2c1f1 = [[0.0 for h in range(1)] for o in range(169)]
        vnf2c2f1 = [[0.0 for h in range(1)] for o in range(169)]
        vnf2c3f1 = [[0.0 for h in range(1)] for o in range(169)]

        fn = [[0.0 for h in range(1)] for o in range(169)]

        for linha in range(169):
            somaLinha1, somaLinha2, somaLinha3, somaLinha4, somaLinha5, somaLinha6, somaLinha7 = 0,0,0,0,0,0,0

            for coluna in range(169):
                somaLinha1 += matrixF1xF2[linha][coluna] * fnAnterior[coluna][0]
                somaLinha2 += matrixF1_C1_F2_F1xF2[linha][coluna] * fnAnterior[coluna][0]
                somaLinha3 += matrixF1_C2_F2_F1xF2[linha][coluna] * fnAnterior[coluna][0]
                somaLinha4 += matrixF1_C3_F2_F1xF2[linha][coluna] * fnAnterior[coluna][0]
                somaLinha5 += matrixF2_C1_F1_F1xF2[linha][coluna] * fnAnterior[coluna][0]
                somaLinha6 += matrixF2_C2_F1_F1xF2[linha][coluna] * fnAnterior[coluna][0]
                somaLinha7 += matrixF2_C3_F1_F1xF2[linha][coluna] * fnAnterior[coluna][0]

            pnXfn[linha][0] = somaLinha1
            f1c1f2[linha][0] = somaLinha2
            f1c2f2[linha][0] = somaLinha3
            f1c3f2[linha][0] = somaLinha4
            f2c1f1[linha][0] = somaLinha5
            f2c2f1[linha][0] = somaLinha6
            f2c3f1[linha][0] = somaLinha7

        for linha in range(169):
            vn[linha][0] = pnXfn[linha][0] + q0[linha][0]
            vnf1c1f2[linha][0] = f1c1f2[linha][0] + qF1_1_F2[linha][0]
            vnf1c2f2[linha][0] = f1c2f2[linha][0] + qF1_2_F2[linha][0]
            vnf1c3f2[linha][0] = f1c3f2[linha][0] + qF1_3_F2[linha][0]
            vnf2c1f1[linha][0] = f2c1f1[linha][0] + qF2_1_F1[linha][0]
            vnf2c2f1[linha][0] = f2c2f1[linha][0] + qF2_2_F1[linha][0]
            vnf2c3f1[linha][0] = f2c3f1[linha][0] + qF2_3_F1[linha][0]
```

```
for linha in range(169):
    fn[linha][0] = max(vn[linha][0],vnf1c1f2[linha][0],vnf1c2f2[linha][0],vnf1c3f2[linha][0],vnf2c1f1[linha][0],vnf2c2f1[linha][0],vnf2c3f1[linha][0])
    dn[linha][0] = fn[linha][0] - fnAnterior[linha][0]

    if(iteracoes == itsAFazer - 1):
        vns = [vn[linha][0],vnf1c1f2[linha][0],vnf1c2f2[linha][0],vnf1c3f2[linha][0],vnf2c1f1[linha][0],vnf2c2f1[linha][0],vnf2c3f1[linha][0]]
        vns.sort(reverse=True)
        for insert in range(7):
            if(vns[insert] == vn[linha][0]):
                politica[linha][insert] = 0
            if(vns[insert] == vnf1c1f2[linha][0]):
                politica[linha][insert] = 1
            if(vns[insert] == vnf1c2f2[linha][0]):
                politica[linha][insert] = 2
            if(vns[insert] == vnf1c3f2[linha][0]):
                politica[linha][insert] = 3
            if(vns[insert] == vnf2c1f1[linha][0]):
                politica[linha][insert] = 4
            if(vns[insert] == vnf2c2f1[linha][0]):
                politica[linha][insert] = 5
            if(vns[insert] == vnf2c3f1[linha][0]):
                politica[linha][insert] = 6

iteracoes += 1
beginAlgoritmo(iteracoes,fn)
```

Código *Python* para descobrir a política ótima