**Trabalho Final**

**Algoritmo Húngaro**

|  |  |
| --- | --- |
| **Alunos:** | |
| Abne Martins Machado | E01797 |
| Ana Claudia Silveira Rodrigues | E01598 |
| Luiz Henrique Silveira Rodrigues | E01599 |
| Pedro Henrique Almeida de Freitas | E01723 |

**2023**

**Problema de alocação de funcionários na Fábrica de Automóveis**

Considere que a fábrica de automóveis precisa transferir trabalhadores de diferentes habilidades para diferentes estações de montagem, de modo a maximizar a eficiência na produção automotiva. Cada trabalhador pode ser visto como uma "unidade" que pode ser alocada em estações de montagem específicas, e a eficiência do trabalhador em cada estação pode ser interpretada como o custo associado a essa alocação.

**Objetivo:**

Maximizar a eficiência global da fábrica, que pode ser representada pela minimização do custo total associado às alocações de trabalhadores para estações de montagem utilizando método do Método Húngaro para alocação.

Para modelar matematicamente o problema de alocação de trabalhadores para estações de montagem, vamos utilizar a Teoria dos Grafos e o Método Húngaro. Vamos definir as variáveis e parâmetros necessários.

**Modelagem matemática**

**Notações:**

* n é o número de trabalhadores e também o número de estações de montagem.
* xij: Variável binária que indica se o trabalhador i alocado na estação de montagem j.
* xij = 0, se o trabalhador i estiver alocado à estação.
* xij = 0, caso contrário.

**Parâmetros:**

Cij é o custo associado à alocação do trabalhador i à estação j. Este custo pode representar a eficiência do trabalhador naquela estação

**Função Objetivo:** Minimizar

**Restrições:**

1. Cada trabalhador só pode ser alocado a uma estação:
2. Cada estação deve receber a quantidade necessária de trabalho:
3. As quantidades de trabalho devem ser não negativas e binárias:

**Método Húngaro:**

Para resolver este problema de alocação, você pode usar o Método Húngaro. Este método é eficiente para resolver problemas de alocação quadrados.

Para resolver este problema de alocação, você pode usar o Método Húngaro. Este método é eficiente para resolver problemas de alocação quadrados.

A matriz de custos Cij deve ser transformada em uma matriz de custos reduzida aplicando os passos do Método Húngaro. A alocação ótima pode ser encontrada a partir da matriz reduzida

**Testes utilizados para o algoritmo implementado**

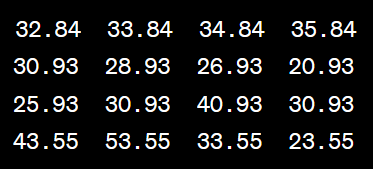
**Problema exemplo 1:**

Dado o contexto da fábrica de automóveis com 4 trabalhadores, 4 estações de trabalho, e cada trabalhador possuindo um custo por hora trabalhada diferente, podemos criar uma função objetivo para minimizar o custo total de alocação de trabalhadores para estações de montagem utilizando método do Método Húngaro para alocação.

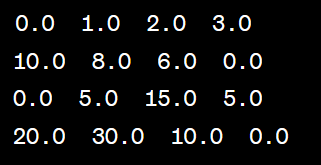
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Funcionários** | **Estação de trabalho – Valor por hora em reais** | | | |
| **1** | **2** | **3** | **4** |
| 1 | R$ 32,84 | R$ 33,84 | R$ 34,84 | R$ 35,84 |
| 2 | R$ 30,93 | R$ 28,93 | R$ 26,93 | R$ 20,93 |
| 3 | R$ 25,93 | R$ 30,93 | R$ 40,93 | R$ 30,93 |
| 4 | R$ 43,55 | R$ 53,55 | R$ 33,55 | R$ 23,55 |

**Interpretação dos resultados apresentados pelo algoritmo implementado neste teste**

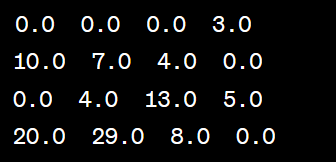
**Matriz Inicial:**

****

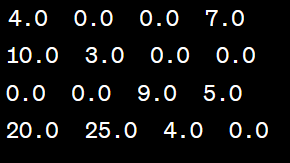
**Redução de Linhas**

****

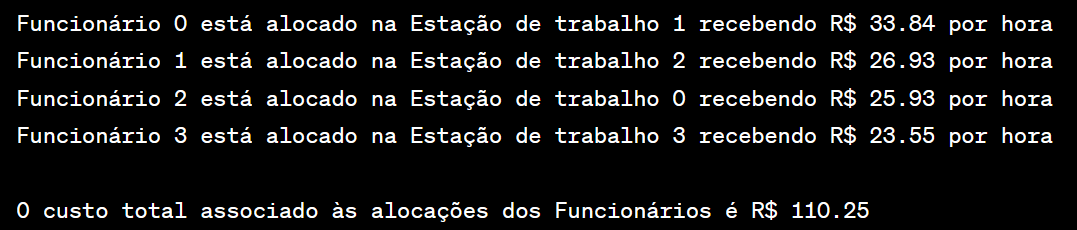
**Redução de Colunas**

****

**Matriz após movimentações**

****

**Resultado Final:**



A interpretação final indica que o Funcionário 0 está alocado na Estação de trabalho 1 recebendo R$ 33.84 por hora, o Funcionário 1 está alocado na Estação de trabalho 2 recebendo R$ 26.93 por hora, e assim por diante.

O custo total associado a essas alocações é R$ 110.25.

Essencialmente, o algoritmo tenta minimizar o custo total das alocações, e os resultados indicam como os funcionários foram alocados em diferentes estações de trabalho para atingir esse objetivo. Isso significa que o menor custo que a empresa terá com a alocação destes quatro funcionários nas quatro estações de trabalho será de R$ 110,25.

**Problema exemplo 2:**

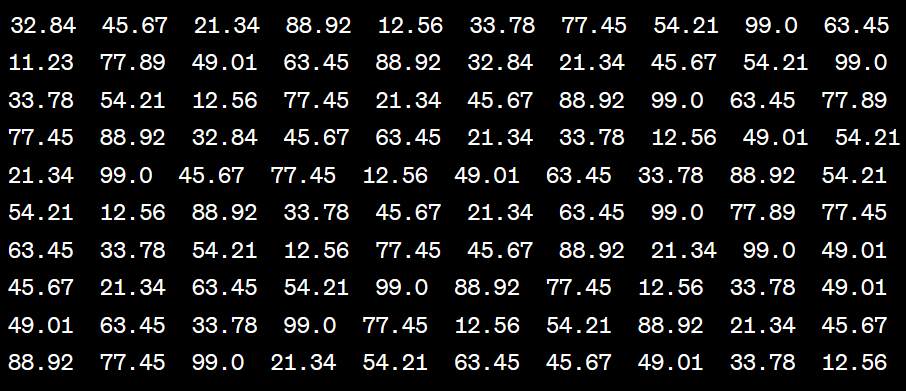
Dado o contexto da fábrica de automóveis com 10 trabalhadores, 10 estações de trabalho, e cada trabalhador possuindo um custo por hora trabalhada diferente, podemos criar uma função objetivo para minimizar o custo total de alocação de trabalhadores para estações de montagem utilizando método do Método Húngaro para alocação.

**As demandas de cada estação, os custos por hora dos funcionários e a capacidade de horas trabalhadas por eles em 30 dias estão listadas na tabela a seguir.**

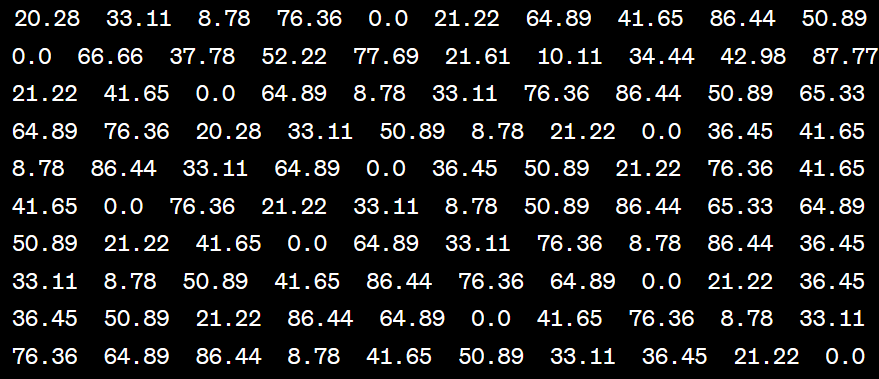
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Funcionário** | **Estação de trabalho – Valor por hora em reais** | | | | | | | | | |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| 1 | 32,84 | 45,67 | 21,34 | 88,92 | 12,56 | 33,78 | 77,45 | 54,21 | 99,00 | 63,45 |
| 2 | 11,23 | 77,89 | 49,01 | 63,45 | 88,92 | 32,84 | 21,34 | 45,67 | 54,21 | 99,00 |
| 3 | 33,78 | 54,21 | 12,56 | 77,45 | 21,34 | 45,67 | 88,92 | 99,00 | 63,45 | 77,89 |
| 4 | 77,45 | 88,92 | 32,84 | 45,67 | 63,45 | 21,34 | 33,78 | 12,56 | 49,01 | 54,21 |
| 5 | 21,34 | 99,00 | 45,67 | 77,45 | 12,56 | 49,01 | 63,45 | 33,78 | 88,92 | 54,21 |
| 6 | 54,21 | 12,56 | 88,92 | 33,78 | 45,67 | 21,34 | 63,45 | 99,00 | 77,89 | 77,45 |
| 7 | 63,45 | 33,78 | 54,21 | 12,56 | 77,45 | 45,67 | 88,92 | 21,34 | 99,00 | 49,01 |
| 8 | 45,67 | 21,34 | 63,45 | 54,21 | 99,00 | 88,92 | 77,45 | 12,56 | 33,78 | 49,01 |
| 9 | 49,01 | 63,45 | 33,78 | 99,00 | 77,45 | 12,56 | 54,21 | 88,92 | 21,34 | 45,67 |
| 10 | 8,92 | 77,45 | 99,00 | 21,34 | 54,21 | 63,45 | 45,67 | 49,01 | 33,78 | 12,56 |

**Interpretação dos resultados apresentados pelo algoritmo implementado neste teste**

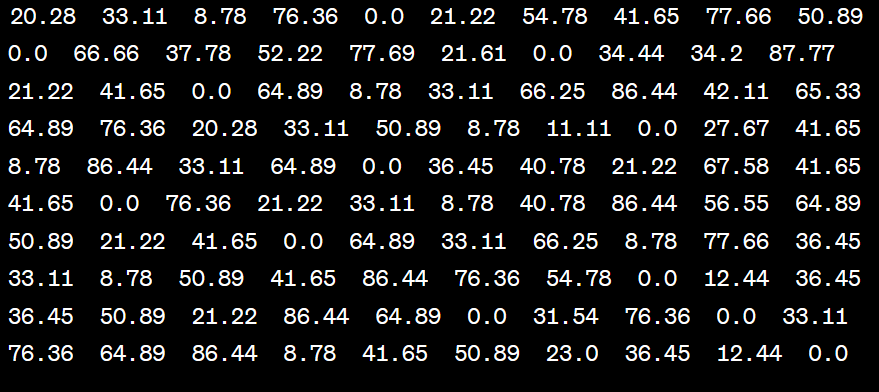
**Matriz Inicial:**

****

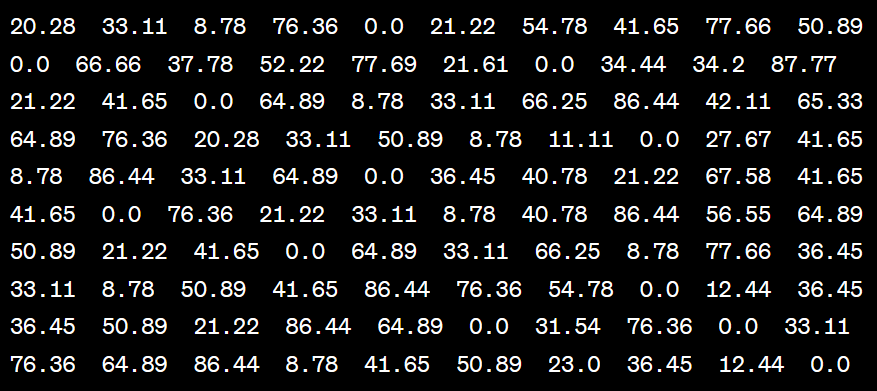
**Redução de Linhas**

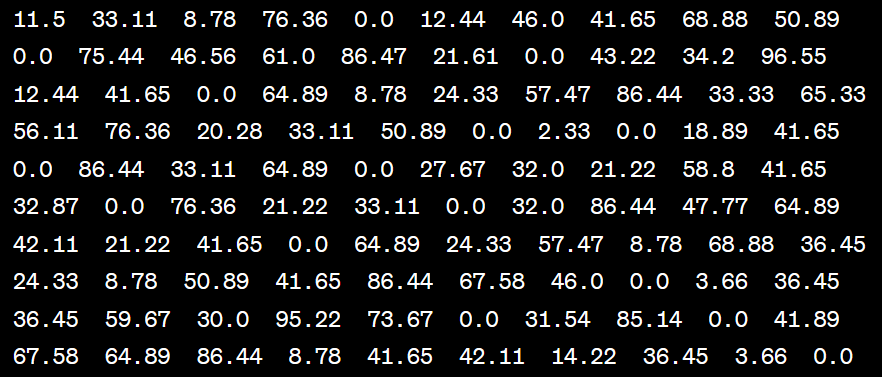
****

**Redução de Colunas**

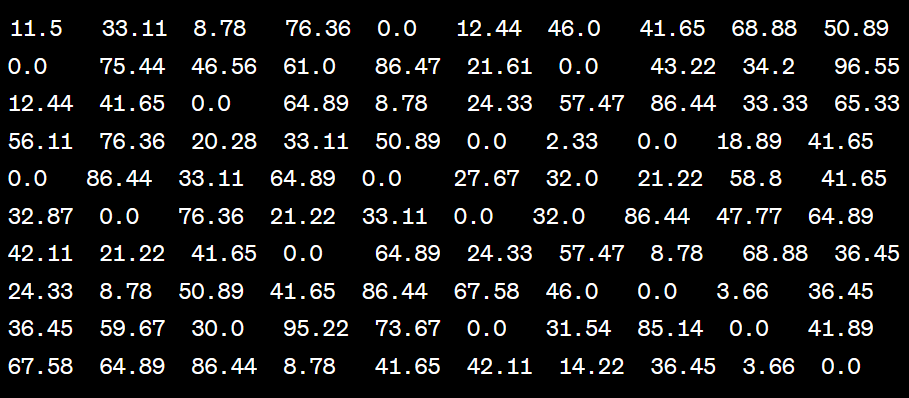
****

**Movimentações**

****

****

**Matriz após movimentações**

****

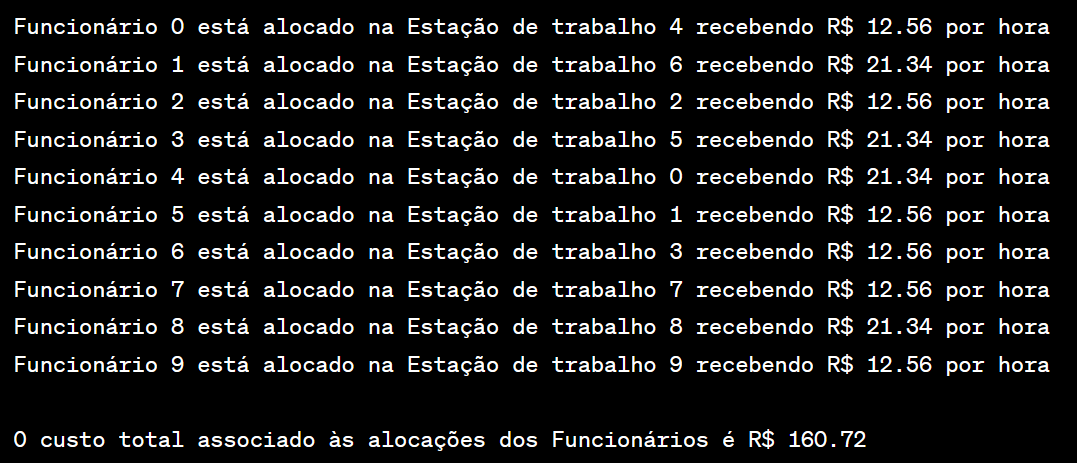
**Interpretação:**

Cada linha da matriz após movimentações representa um funcionário, e cada coluna representa uma estação de trabalho.

Os valores não nulos indicam a alocação dos funcionários nas estações de trabalho correspondentes.

O custo total associado a essas alocações é R$ 160.72.

**Resultado Final:**

****

O algoritmo otimizou a alocação para minimizar o custo total, levando em consideração as preferências dos funcionários e as capacidades das estações de trabalho.

Os custos associados a cada alocação refletem a eficiência global da fábrica, sendo minimizados na alocação final apresentada.

**Apresentação do Algoritmo**

O código implementa o Algoritmo Húngaro, uma técnica de otimização usado para resolver problemas de atribuição. Esses problemas geralmente envolvem a alocação de recursos de maneira eficiente, minimizando ou maximizando algum critério, como custo. Vou fornecer uma explicação passo a passo do código:

**Definição de Nomes e Unidade de Medida:**

As variáveis eixoHorizontal e eixoVertical representam os rótulos para as duas dimensões do problema (por exemplo, funcionários e estações de trabalho). unidadeMedida é a unidade associada ao custo.

**Matriz de Custo:**

A matriz matriz1 representa os custos associados à alocação de cada "funcionário" a uma "estação de trabalho". Existem diferentes matrizes de exemplo no código que podem ser alternadas para testar o algoritmo.

**Redução de Linhas e Colunas:**

reduzLinhas e reduzColunas são funções que realizam a redução dos valores nas linhas e colunas da matriz, respectivamente.

**Encontrar Traços:**

encontraTracos identifica traços na matriz. Traços são zeros isolados em linhas ou colunas.

Encontrar Zeros e Marcar Linhas/Colunas:

encontraZeros identifica zeros isolados e marca linhas e colunas correspondentes.

Terminar Linhas/Colunas:

terminaLinhasHorizontal e terminaLinhasVertical são funções recursivas que realizam a marcação de linhas e colunas para encontrar uma solução viável.

**Resolução de Entradas não Riscadas:**

resolvendoEntradasNaoRiscadas ajusta os valores da matriz para continuar a busca pela solução ótima.

**Retornar Resultado:**

retornaResultado exibe os resultados finais, mostrando como os funcionários são alocados nas estações de trabalho e o custo total associado.

**Cópia de Matriz:**

copiaMatriz cria uma cópia de uma matriz.

**Impressão de Matriz:**

printaMatrizNumerica imprime uma matriz numérica.

**Arredondamento de Números:**

arredondarDouble é uma função auxiliar para arredondar números para duas casas decimais.

O código utiliza diferentes métodos e recursividade para percorrer as etapas do algoritmo. O Algoritmo Húngaro é um método eficiente para resolver problemas de atribuição, e o código implementa a lógica necessária para encontrar uma solução ótima.

**Funcionamento do Programa**

O algoritmo é iterativo e utiliza diversas estratégias para marcar e ajustar valores na matriz, buscando encontrar uma alocação que minimize ou maximize o custo total, dependendo da natureza do problema.

O código é estruturado de forma modular, com funções específicas para cada etapa do algoritmo. A lógica é baseada nas propriedades dos problemas de atribuição e na busca por soluções viáveis de maneira eficiente.

**Execução do código do programa implementado explicando os parâmetros de entrada e os parâmetros de saída (solução)**

**Parâmetros de Entrada:**

Matriz de Custos (matriz1):

A matriz que representa os custos associados à atribuição. Cada elemento matriz1[i][j] representa o custo de atribuir a i-ésima linha à j-ésima coluna.

Nomes das Variáveis (eixoHorizontal, eixoVertical, unidadeMedida):

Strings que fornecem rótulos para as variáveis, como "Funcionário", "Estação de trabalho", e "por hora". Esses rótulos são usados para imprimir resultados.

**Parâmetros de Saída (Solução):**

Alocações e Custos:

O programa imprime na tela a alocação resultante, indicando qual "Funcionario" está alocado a qual "Estacao de trabalho". O custo total associado a essas alocações também é exibido.

Matriz Após Movimentações (matrizReduzida):

A matriz original é processada por diversas etapas (redução de linhas, redução de colunas, marcação de traços, etc.), e a matriz resultante é exibida na tela.

Iterações e Passos Intermediários:

O código imprime várias matrizes intermediárias durante as iterações, mostrando a redução de linhas, redução de colunas, marcação de zeros, entre outros passos.

**Funcionamento Geral do Programa:**

Inicialização:

O programa inicia com a definição da matriz de custos e outros parâmetros.

Redução de Linhas e Colunas:

São realizadas operações para reduzir as linhas e colunas da matriz original, visando simplificar o problema.

Marcação de Traços e Zeros:

O código encontra zeros na matriz reduzida e os marca. Em seguida, traços são colocados nas linhas e colunas contendo zeros, com o objetivo de encontrar uma solução inicial.

Designação de Linhas e Colunas:

A alocação é aprimorada iterativamente, alternando entre designação de linhas e colunas, para encontrar uma solução aprimorada.

Saída Final:

O programa exibe na tela a matriz após todas as movimentações e a alocação final, indicando qual funcionário está alocado em qual estação de trabalho.

**Observações:**

O código é projetado para ser adaptável a diferentes matrizes de custo.

Durante as iterações, o programa utiliza estratégias heurísticas para melhorar as alocações.

O resultado final é uma alocação que minimiza o custo total.

**Possíveis Melhorias:**

A implementação usa heurísticas para encontrar uma solução eficiente, mas não garante a solução globalmente ótima.

Pode haver variação nos resultados dependendo da implementação específica das heurísticas.

Este é um algoritmo bem conhecido para resolver problemas de atribuição e é eficaz na prática. O código é estruturado de maneira clara, facilitando a compreensão das etapas do algoritmo.

**Comparação com a Modelagem Matemática:**

Quanto a eficiência

O algoritmo húngaro é conhecido por ser eficiente, mas dependendo da implementação e do problema específico, pode haver casos em que uma modelagem matemática e um solver linear, como o Simplex, também são eficientes.

Precisão e confiabilidade

Ambos são precisos em suas soluções e também confiáveis.

Facilidade de Implementação e Manutenção:

O algoritmo húngaro é relativamente direto de implementar, enquanto a modelagem matemática pode exigir o uso de bibliotecas ou solvers externos.

Adaptabilidade a Diferentes Problemas:

Alguns métodos podem ser mais flexíveis em lidar com variações nos requisitos do problema.

Escalabilidade:

O desempenho pode variar à medida que o tamanho do problema aumenta.

Limitações e Restrições:

O algoritmo húngaro é especialmente eficaz para o Problema de Atribuição, mas pode ter limitações em outros contextos. A modelagem matemática pode oferecer uma abordagem mais geral, mas pode ter limitações de desempenho para problemas específicos.

**Algoritmo implementado em java**

**import** java.text.DecimalFormat;

**import** java.util.Arrays;

**public** **class** CantoNoroeste {

**public** **static** **void** main(String[] args) {

// Definir parâmetros

// Teste 1

// int[] supply = { 90, 60, 120, 150, 150, 150 };

// int[] demand = { 50, 60, 30, 20, 40, 300 };

// Teste 2

**int**[] supply = { 90, 60, 120, 150, 150, 150, 240, 270, 180, 120 };

**int**[] demand = { 12, 10, 1, 10, 10, 10, 12, 14, 16, 20, 25, 30, 8, 2, 6 };

// Verificar e ajustar o tamanho de oferta e demanda

**int**[] adjustedSizes = *adjustSupplyDemand*(supply, demand);

supply = Arrays.*copyOf*(adjustedSizes, adjustedSizes.length / 2);

demand = Arrays.*copyOfRange*(adjustedSizes, adjustedSizes.length / 2, adjustedSizes.length);

// Custo teste 1

// double[][] costs = {

// { 32.84, 32.84, 32.84, 32.84, 32.84, 32.84 },

// { 30.93, 30.93, 30.93, 30.93, 30.93, 30.93 },

// { 30.93, 30.93, 30.93, 30.93, 30.93, 30.93 },

// { 33.55, 33.55, 33.55, 33.55, 33.55, 33.55 }

// };

// Custo teste 2

**double**[][] costs = {

{32.84, 32.84, 32.84, 32.84, 32.84, 32.84, 32.84, 32.84, 32.84, 32.84, 32.84, 32.84, 32.84, 32.84, 32.84},

{32.84, 32.84, 32.84, 32.84, 32.84, 32.84, 32.84, 32.84, 32.84, 32.84, 32.84, 32.84, 32.84, 32.84, 32.84},

{32.84, 32.84, 32.84, 32.84, 32.84, 32.84, 32.84, 32.84, 32.84, 32.84, 32.84, 32.84, 32.84, 32.84, 32.84},

{30.93, 30.93, 30.93, 30.93, 30.93, 30.93, 30.93, 30.93, 30.93, 30.93, 30.93, 30.93, 30.93, 30.93, 30.93},

{30.93, 30.93, 30.93, 30.93, 30.93, 30.93, 30.93, 30.93, 30.93, 30.93, 30.93, 30.93, 30.93, 30.93, 30.93},

{30.93, 30.93, 30.93, 30.93, 30.93, 30.93, 30.93, 30.93, 30.93, 30.93, 30.93, 30.93, 30.93, 30.93, 30.93},

{35.72, 35.72, 35.72, 35.72, 35.72, 35.72, 35.72, 35.72, 35.72, 35.72, 35.72, 35.72, 35.72, 35.72, 35.72},

{35.72, 35.72, 35.72, 35.72, 35.72, 35.72, 35.72, 35.72, 35.72, 35.72, 35.72, 35.72, 35.72, 35.72, 35.72},

{33.55, 33.55, 33.55, 33.55, 33.55, 33.55, 33.55, 33.55, 33.55, 33.55, 33.55, 33.55, 33.55, 33.55, 33.55},

{33.55, 33.55, 33.55, 33.55, 33.55, 33.55, 33.55, 33.55, 33.55, 33.55, 33.55, 33.55, 33.55, 33.55, 33.55}};

// Resolução usando o método do Método Húngaro

**double**[][] solution = *northwestCornerMethod*(supply, demand, costs);

// Imprimir a solução

System.***out***.println("Solução usando o método do Método Húngaro:");

*printSolution*(solution);

// Calcular e imprimir o custo total

**double** totalCost = *calculateTotalCost*(solution, costs);

System.***out***.println("Z = R$" + *formatDecimal*(totalCost));

}

// Método para ajustar o tamanho de oferta e demanda

**private** **static** **int**[] adjustSupplyDemand(**int**[] supply, **int**[] demand) {

**int** max = Math.*max*(supply.length, demand.length);

**int**[] adjustedSizes = **new** **int**[max \* 2];

Arrays.*fill*(adjustedSizes, -1); // Preencher com -1 para indicar valores não ajustados

System.*arraycopy*(supply, 0, adjustedSizes, 0, supply.length);

System.*arraycopy*(demand, 0, adjustedSizes, max, demand.length);

**return** adjustedSizes;

}

// Método do Método Húngaro para alocação

**private** **static** **double**[][] northwestCornerMethod(**int**[] supply, **int**[] demand, **double**[][] costs) {

**int** m = supply.length;

**int** n = demand.length;

**double**[][] solution = **new** **double**[m][n];

**int** i = 0, j = 0;

// Alocar até o esgotamento da oferta ou da demanda

**while** (i < m && j < n) {

**double** quantity = Math.*min*(supply[i], demand[j]);

// Verificar se estamos dentro dos limites da matriz

**if** (i < m && j < n) {

solution[i][j] = quantity;

supply[i] -= quantity;

demand[j] -= quantity;

**if** (supply[i] == 0)

i++;

**if** (demand[j] == 0)

j++;

} **else** {

// Tratar o caso em que estamos fora dos limites da matriz

**break**;

}

}

**return** solution; // Adicione esta linha para corrigir o erro

}

// Método para imprimir a matriz de solução

**private** **static** **void** printSolution(**double**[][] solution) {

**for** (**double**[] row : solution) {

System.***out***.println(Arrays.*toString*(row));

}

}

// Método para calcular o custo total da alocação

**private** **static** **double** calculateTotalCost(**double**[][] solution, **double**[][] costs) {

**double** totalCost = 0;

**int** m = solution.length;

**int** n = solution[0].length;

**for** (**int** i = 0; i < m; i++) {

**for** (**int** j = 0; j < n; j++) {

// Verificar se a célula está dentro dos limites da matriz de custos

**if** (i < costs.length && j < costs[0].length) {

totalCost += solution[i][j] \* costs[i][j];

}

}

}

**return** totalCost;

}

// Método para formatar um número decimal com duas casas decimais

**private** **static** String formatDecimal(**double** value) {

DecimalFormat decimalFormat = **new** DecimalFormat("#,##0.00");

**return** decimalFormat.format(value);

}

}

**Observações a fechar no trabalho**

O que deve ser entregue

• O Link de um vídeo do YouTube de 5 minutos apresentando a solução e contendo obrigatoriamente:

O nome completo de todos os integrantes

A descrição do problema

A modelagem matemática do problema

A apresentação do algoritmo utilizado

Funcionamento do programa implementado

Execução do código do programa implementado explicando os parâmetros de entrada e os parâmetros de saída (solução)

Interpretação da resposta apresentada após execução do programa

• Link do repositório GitHub que deverá conter obrigatoriamente:

Um arquivo README.md, contento o nome de todos os integrantes do grupo, a contextualização do problema, a modelagem matemática, um exemplo de solução utilizando o algoritmo implementado e a interpretação dos resultados

O código completo e todos os arquivos necessários para implementação do algoritmo escolhido para solução do problema.

Atenção:

• Como a solução do problema será resolvida utilizando-se uma linguagem de programação (à escolha do grupo), os problemas apresentados devem possuir um número elevado de variáveis.

• Não serão aceitos problemas resolvidos utilizando o solver ou ferramentas online, ou seja, somente serão aceitos problemas resolvidos por alguma implementação disponível na literatura.

• A qualidade do vídeo e do áudio, a coerência e clareza da apresentação também fazem parte da avaliação do trabalho.

• A apresentação do vídeo pode ser feita por apenas um dos integrantes do grupo.

• Não utilizar fundo musical alto ou vozes artificiais para apresentação