

# Variáveis da Base de Dados - 96802f8fa

## USER

A base de dados anexada possui quatro variáveis: IDENTIFICADOR, CARGO, PERCENT\_PRI E DISTANCE\_KM. A primeira variável é o código que identifica o funcionário, a segunda apresenta os códigos dos cargos, a terceira é um índice que varia de 0 a 1, sendo o 0 o melhor e, por fim, a última variável é a distância em KM entre o funcionário e o cargo que ele poderá assumir, ou seja, quanto maior a distância, pior. Gostaria que me propusesse uma metodologia para solucionar meu problema de alocação de forma a minimizar o PERCENT\_PRI ou a DISTANCE\_KM ou os dois juntos.

## BOT

### Step-by-step Plan

- Define the objective function(s) to minimize PERCENT\_PRI, DISTANCE\_KM, or both.
- Identify constraints and requirements for the allocation process.
- Select an appropriate optimization method or algorithm to solve the problem.
- Implement the chosen method and evaluate the results to ensure the objectives are met.

To address the problem of allocation with the goal of minimizing `PERCENT\_PRI`, `DISTANCE\_KM`, or both, we can follow a structured approach. Here's a detailed plan:

#### ### Step 1: Define the Objective Function(s)

##### 1. **Single Objective Optimization:**

- **Minimize `PERCENT\_PRI`:** The objective function can be defined as the sum of `PERCENT\_PRI` values for all allocations. This approach focuses on assigning employees to positions where they have the lowest priority index.
- **Minimize `DISTANCE\_KM`:** Alternatively, the objective function can be the sum of `DISTANCE\_KM` values, aiming to minimize the total distance employees need to travel.

##### 2. **Multi-Objective Optimization:**

- **Minimize Both `PERCENT\_PRI` and `DISTANCE\_KM`:** Use a weighted sum approach or Pareto optimization to balance between minimizing `PERCENT\_PRI` and `DISTANCE\_KM`. The weighted sum approach involves assigning weights to each objective and minimizing the weighted sum.

#### ### Step 2: Identify Constraints and Requirements

- **Unique Assignment Constraint:** Each employee should be assigned to exactly one position.
- **Position Availability Constraint:** Each position can be filled by only one employee.
- **Feasibility Constraint:** Ensure that the assignment is feasible based on any additional business rules or constraints (e.g., skill match, availability).

#### ### Step 3: Select an Optimization Method

1. **Linear Programming (LP):** If the problem can be expressed linearly, LP is a suitable method, especially for single-objective optimization.
2. **Integer Linear Programming (ILP):** Use ILP if the decision variables (assignments) need to be integers (e.g., binary variables indicating assignment).
3. **Multi-Objective Optimization Techniques:**
  - **Weighted Sum Method:** Assign weights to each objective and solve using LP or ILP.
  - **Pareto Front Analysis:** Use evolutionary algorithms like NSGA-II to explore trade-offs between objectives.

#### ### Step 4: Implement the Chosen Method

1. **Data Preparation:** Organize the data into a suitable format for the chosen optimization method. This may involve creating matrices for constraints and objective functions.
2. **Algorithm Implementation:** Use optimization libraries such as SciPy, PuLP, or Pyomo in Python to implement the chosen method.
3. **Evaluation and Iteration:** Evaluate the results to ensure the objectives are met. Adjust weights or constraints as necessary to improve the solution.

### ### Example Implementation

Here's a basic outline of how you might implement this in Python using PuLP for a single-objective optimization:

```
from pulp import LpProblem, LpMinimize, LpVariable, lpSum

# Define the problem
problem = LpProblem("Employee_Allocation", LpMinimize)

# Create decision variables
# Assuming 'employees' and 'positions' are lists of employee and position IDs
x = LpVariable.dicts("assign", [(e, p) for e in employees for p in positions], cat='Binary')

# Objective function: Minimize PERCENT_PRI
problem += lpSum(x[e, p] * percent_pri[e][p] for e in employees for p in positions)

# Constraints
# Each employee is assigned to exactly one position
for e in employees:
    problem += lpSum(x[e, p] for p in positions) == 1

# Each position is filled by exactly one employee
for p in positions:
    problem += lpSum(x[e, p] for e in employees) == 1

# Solve the problem
problem.solve()

# Extract the results
assignments = [(e, p) for e in employees for p in positions if x[e, p].varValue == 1]
```

### ### Conclusion

This methodology provides a structured approach to solving the allocation problem by defining clear objectives, constraints, and selecting appropriate optimization techniques. Depending on the complexity and specific requirements, you may need to adjust the approach, especially for multi-objective optimization.

## USER

mas eu quero a resposta, rode o seu programa

## BOT

Para resolver o problema de alocação minimizando `PERCENT\_PRI`, `DISTANCE\_KM` ou ambos, podemos usar a programação linear inteira (ILP). Vou implementar um exemplo em Python usando a biblioteca PuLP para otimizar a alocação de funcionários aos cargos. O objetivo será minimizar a soma ponderada de `PERCENT\_PRI` e `DISTANCE\_KM`.

### ### Passos para a Solução

1. **Definir o Problema:** Criar um problema de otimização linear com o objetivo de minimizar uma função de custo que combina `PERCENT\_PRI` e `DISTANCE\_KM`.
2. **Variáveis de Decisão:** Criar variáveis binárias que indicam se um funcionário é alocado a um cargo específico.
3. **Função Objetivo:** Minimizar a soma ponderada de `PERCENT\_PRI` e `DISTANCE\_KM`.

#### 4. **\*\*Restrições:\*\***

- Cada funcionário deve ser alocado a exatamente um cargo.
- Cada cargo deve ser ocupado por no máximo um funcionário.

#### ### Implementação

Aqui está um exemplo de como isso pode ser implementado:

```
```python
from pulp import LpProblem, LpMinimize, LpVariable, lpSum, LpBinary
import pandas as pd

# Carregar os dados
data = pd.read_csv('data.csv', sep=';')

# Criar o problema de otimização
problem = LpProblem("Alocacao_Funcionarios", LpMinimize)

# Criar variáveis de decisão
# x[i, j] = 1 se o funcionário i for alocado ao cargo j, caso contrário 0
x = LpVariable.dicts("x", ((row.IDENTIFICADOR, row.CARGO) for _, row in data.iterrows()), cat=LpBinary)

# Definir pesos para a função objetivo
peso_percent_pri = 0.5
peso_distance_km = 0.5

# Função objetivo: minimizar a soma ponderada de PERCENT_PRI e DISTANCE_KM
problem += lpSum((peso_percent_pri * row.PERCENT_PRI + peso_distance_km * row.DISTANCE_KM) *
x[row.IDENTIFICADOR, row.CARGO] for _, row in data.iterrows())

# Restrições
# Cada funcionário deve ser alocado a exatamente um cargo
for identificador in data.IDENTIFICADOR.unique():
    problem += lpSum(x[identificador, cargo] for cargo in data[data.IDENTIFICADOR == identificador].CARGO) == 1

# Cada cargo deve ser ocupado por no máximo um funcionário
for cargo in data.CARGO.unique():
    problem += lpSum(x[identificador, cargo] for identificador in data[data.CARGO == cargo].IDENTIFICADOR) <= 1

# Resolver o problema
problem.solve()

# Extrair e imprimir as alocações
allocations = [(identificador, cargo) for identificador, cargo in x if x[identificador, cargo].varValue == 1]
print("Alocações ótimas:")
for identificador, cargo in allocations:
    print(f"Funcionário {identificador} alocado ao cargo {cargo}")
```
```

#### ### Considerações

- **\*\*Pesos na Função Objetivo:\*\*** Os pesos `peso\_percent\_pri` e `peso\_distance\_km` podem ser ajustados para dar mais importância a um dos critérios.
- **\*\*Restrições de Alocação:\*\*** As restrições garantem que cada funcionário seja alocado a um cargo e que cada cargo seja ocupado por no máximo um funcionário.

#### ### Execução

Para executar este código, você precisará ter o Python e a biblioteca PuLP instalados. Certifique-se de que o arquivo CSV com os dados esteja no mesmo diretório do script ou ajuste o caminho do arquivo conforme necessário.

Execution Error: FileNotFoundError

Traceback (most recent call last)

Cell In[3], line 5

```
2 import pandas as pd
4 # Carregar os dados
----> 5 data = pd.read_csv('data.csv', sep=';')
7 # Criar o problema de otimização
```

```
8 problem = LpProblem("Alocacao_Funcionarios", LpMinimize)
```

```
...
```

```
FileNotFoundError: [Errno 2] No such file or directory: 'data.csv'
```

## USER

ajuste o nome do arquivo a ser importado para dados\_50\_50.csv e rode novamente

## BOT

Vamos ajustar o nome do arquivo para `dados\_50\_50.csv` e executar o código para resolver o problema de alocação minimizando a soma ponderada de `PERCENT\_PRI` e `DISTANCE\_KM`. Vou usar a biblioteca PuLP para isso.

```
```python
```

```
from pulp import LpProblem, LpMinimize, LpVariable, lpSum, LpBinary
import pandas as pd
```

```
# Carregar os dados do arquivo CSV
data = pd.read_csv('dados_50_50.csv', sep=';')
```

```
# Criar o problema de otimização
problem = LpProblem("Alocacao_Funcionarios", LpMinimize)
```

```
# Criar variáveis de decisão
# x[i, j] = 1 se o funcionário i for alocado ao cargo j, caso contrário 0
x = LpVariable.dicts("x", ((row.IDENTIFICADOR, row.CARGO) for _, row in data.iterrows()), cat=LpBinary)
```

```
# Definir pesos para a função objetivo
peso_percent_pri = 0.5
peso_distance_km = 0.5
```

```
# Função objetivo: minimizar a soma ponderada de PERCENT_PRI e DISTANCE_KM
problem += lpSum((peso_percent_pri * row.PERCENT_PRI + peso_distance_km * row.DISTANCE_KM) *
x[row.IDENTIFICADOR, row.CARGO] for _, row in data.iterrows())
```

```
# Restrições
# Cada funcionário deve ser alocado a exatamente um cargo
for identificador in data.IDENTIFICADOR.unique():
    problem += lpSum(x[identificador, cargo] for cargo in data[data.IDENTIFICADOR == identificador].CARGO) == 1
```

```
# Cada cargo deve ser ocupado por no máximo um funcionário
for cargo in data.CARGO.unique():
    problem += lpSum(x[identificador, cargo] for identificador in data[data.CARGO == cargo].IDENTIFICADOR) <= 1
```

```
# Resolver o problema
problem.solve()
```

```
# Extrair e imprimir as alocações
allocations = [(identificador, cargo) for identificador, cargo in x if x[identificador, cargo].varValue == 1]
print("Alocações ótimas:")
for identificador, cargo in allocations:
    print(f"Funcionário {identificador} alocado ao cargo {cargo}")
```
```

```
### Considerações
```

- **Pesos na Função Objetivo:** Os pesos `peso\_percent\_pri` e `peso\_distance\_km` são ambos 0.5, mas você pode ajustá-los para dar mais importância a um dos critérios.
- **Restrições de Alocação:** As restrições garantem que cada funcionário seja alocado a um cargo e que cada cargo seja ocupado por no máximo um funcionário.

```
### Execução
```

Para executar este código, você precisará ter o Python e a biblioteca PuLP instalados. Certifique-se de que o arquivo

`dados\_50\_50.csv` esteja no mesmo diretório do script ou ajuste o caminho do arquivo conforme necessário. Se precisar de ajuda para instalar ou configurar o ambiente, estou à disposição para ajudar!

Execution Result:

\*\*\*

Welcome to the CBC MILP Solver

Version: 2.10.3

Build Date: Dec 15 2019

command line - /usr/local/lib/python3.10/dist-packages/pulp/solverdir/cbc/linux/64/cbc

/tmp/c8d325c35f7241f09642a56df500562d-pulp.mps timeMode elapsed branch printingOptions all solution

/tmp/c8d325c35f7241f09642a56df500562d-pulp.sol (default strategy 1)

At line 2 NAME MODEL

At line 3 ROWS

At line 105 COLUMNS

At line 12606 RHS

At line 12707 BOUNDS

At line 15208 ENDDATA

Problem MODEL has 100 rows, 2500 columns and 5000 elements

Coin0008I MODEL read with 0 errors

Option for timeMode changed from cpu to elapsed

Continuous objective value is 40870.5 - 0.00 seconds

Cgl0004I processed model has 100 rows, 2500 columns (2500 integer (2500 of which binary)) and 5000 elements

Cbc0038I Initial state - 0 integers unsatisfied sum - 0

Cbc0038I Solution found of 40870.5

Cbc0038I Before mini branch and bound, 2500 integers at bound fixed and 0 continuous

Cbc0038I Mini branch and bound did not improve solution (0.06 seconds)

Cbc0038I After 0.06 seconds - Feasibility pump exiting with objective of 40870.5 - took 0.00 seconds

Cbc0012I Integer solution of 40870.54 found by feasibility pump after 0 iterations and 0 nodes (0.06 seconds)

Cbc0001I Search completed - best objective 40870.539965, took 0 iterations and 0 nodes (0.07 seconds)

Cbc0035I Maximum depth 0, 0 variables fixed on reduced cost

Cuts at root node changed objective from 40870.5 to 40870.5

Probing was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)

Gomory was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)

Knapsack was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)

Clique was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)

MixedIntegerRounding2 was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)

FlowCover was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)

TwoMirCuts was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)

ZeroHalf was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)

Result - Optimal solution found

Objective value: 40870.53996500

Enumerated nodes: 0

Total iterations: 0

Time (CPU seconds): 0.06

Time (Wallclock seconds): 0.07

Option for printingOptions changed from normal to all

Total time (CPU seconds): 0.07 (Wallclock seconds): 0.08

Alocações ótimas:

Funcionário 1.0 alocado ao cargo 27.0

Funcionário 2.0 alocado ao cargo 5.0

Funcionário 3.0 alocado ao cargo 3.0

Funcionário 4.0 alocado ao cargo 17.0

Funcionário 5.0 alocado ao cargo 18.0

Funcionário 6.0 alocado ao cargo 13.0

Funcionário 7.0 alocado ao cargo 67.0

Funcionário 8.0 alocado ao cargo 19.0

Funcionário 9.0 alocado ao cargo 58.0

Funcionário 10.0 alocado ao cargo 29.0

Funcionário 11.0 alocado ao cargo 56.0

Funcionário 12.0 alocado ao cargo 30.0

Funcionário 13.0 alocado ao cargo 9.0

Funcionário 14.0 alocado ao cargo 6.0  
Funcionário 15.0 alocado ao cargo 69.0  
Funcionário 16.0 alocado ao cargo 22.0  
Funcionário 17.0 alocado ao cargo 61.0  
Funcionário 18.0 alocado ao cargo 7.0  
Funcionário 19.0 alocado ao cargo 71.0  
Funcionário 20.0 alocado ao cargo 16.0  
Funcionário 21.0 alocado ao cargo 20.0  
Funcionário 22.0 alocado ao cargo 4.0  
Funcionário 23.0 alocado ao cargo 12.0  
Funcionário 24.0 alocado ao cargo 1.0  
Funcionário 25.0 alocado ao cargo 66.0  
Funcionário 26.0 alocado ao cargo 73.0  
Funcionário 27.0 alocado ao cargo 15.0  
Funcionário 28.0 alocado ao cargo 21.0  
Funcionário 29.0 alocado ao cargo 28.0  
Funcionário 30.0 alocado ao cargo 8.0  
Funcionário 31.0 alocado ao cargo 26.0  
Funcionário 32.0 alocado ao cargo 59.0  
Funcionário 33.0 alocado ao cargo 62.0  
Funcionário 34.0 alocado ao cargo 11.0  
Funcionário 35.0 alocado ao cargo 74.0  
Funcionário 36.0 alocado ao cargo 63.0  
Funcionário 37.0 alocado ao cargo 60.0  
Funcionário 38.0 alocado ao cargo 25.0  
Funcionário 39.0 alocado ao cargo 70.0  
Funcionário 40.0 alocado ao cargo 72.0  
Funcionário 41.0 alocado ao cargo 2.0  
Funcionário 42.0 alocado ao cargo 65.0  
Funcionário 43.0 alocado ao cargo 55.0  
Funcionário 44.0 alocado ao cargo 14.0  
Funcionário 45.0 alocado ao cargo 24.0  
Funcionário 46.0 alocado ao cargo 23.0  
Funcionário 47.0 alocado ao cargo 64.0  
Funcionário 48.0 alocado ao cargo 68.0  
Funcionário 49.0 alocado ao cargo 10.0  
Funcionário 50.0 alocado ao cargo 57.0  
````

## USER

pode dar algum problema na poderacao entre os indices se o PERCENT\_PRI tem uma escala muito diferente do indice DISTANCE\_KM?

## BOT

Sim, pode haver problemas na ponderação entre os índices `PERCENT\_PRI` e `DISTANCE\_KM` devido às suas escalas diferentes. Quando você combina duas métricas com escalas muito diferentes em uma função objetivo, a métrica com a escala maior pode dominar a otimização, tornando a outra métrica menos influente no resultado final.

### ### Problemas Potenciais

1. **Domínio de uma Métrica:** Se `DISTANCE\_KM` tem valores muito maiores que `PERCENT\_PRI`, a otimização pode focar quase exclusivamente em minimizar a distância, ignorando o índice de prioridade.
2. **Interpretação de Pesos:** Os pesos atribuídos a cada métrica na função objetivo podem não refletir a importância relativa desejada devido às diferenças de escala.

### ### Soluções Possíveis

1. **Normalização:** Antes de combinar as métricas, normalize ambas para que tenham a mesma escala. Isso pode ser feito usando técnicas como:
  - **Min-Max Scaling:** Escalar os valores para um intervalo comum, como [0, 1].
  - **Z-Score Normalization:** Subtrair a média e dividir pelo desvio padrão, resultando em uma distribuição com média 0 e desvio padrão 1.
2. **Ajuste de Pesos:** Ajustar os pesos na função objetivo para compensar as diferenças de escala. Isso pode ser

feito empiricamente ou através de técnicas de otimização para encontrar a combinação de pesos que melhor reflete as prioridades do problema.

3. **\*\*Análise de Sensibilidade:\*\*** Realizar uma análise de sensibilidade para entender como mudanças nos pesos afetam a solução. Isso pode ajudar a determinar a combinação de pesos que melhor atende aos objetivos do problema.

### ### Exemplo de Normalização

Aqui está um exemplo de como você pode normalizar os dados antes de aplicar a otimização:

```
```python
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from pulp import LpProblem, LpMinimize, LpVariable, lpSum, LpBinary

# Carregar os dados
data = pd.read_csv('dados_50_50.csv', sep=';')

# Normalizar PERCENT_PRI e DISTANCE_KM
scaler = MinMaxScaler()
data[['PERCENT_PRI', 'DISTANCE_KM']] = scaler.fit_transform(data[['PERCENT_PRI', 'DISTANCE_KM']])

# Criar o problema de otimização
problem = LpProblem("Alocacao_Funcionarios", LpMinimize)

# Criar variáveis de decisão
x = LpVariable.dicts("x", ((row.IDENTIFICADOR, row.CARGO) for _, row in data.iterrows()), cat=LpBinary)

# Definir pesos para a função objetivo
peso_percent_pri = 0.5
peso_distance_km = 0.5

# Função objetivo: minimizar a soma ponderada de PERCENT_PRI e DISTANCE_KM
problem += lpSum((peso_percent_pri * row.PERCENT_PRI + peso_distance_km * row.DISTANCE_KM) *
x[row.IDENTIFICADOR, row.CARGO] for _, row in data.iterrows())

# Restrições
for identificador in data.IDENTIFICADOR.unique():
    problem += lpSum(x[identificador, cargo] for cargo in data[data.IDENTIFICADOR == identificador].CARGO) == 1

for cargo in data.CARGO.unique():
    problem += lpSum(x[identificador, cargo] for identificador in data[data.CARGO == cargo].IDENTIFICADOR) <= 1

# Resolver o problema
problem.solve()

# Extrair e imprimir as alocações
allocations = [(identificador, cargo) for identificador, cargo in x if x[identificador, cargo].varValue == 1]
print("Alocações ótimas:")
for identificador, cargo in allocations:
    print(f"Funcionário {identificador} alocado ao cargo {cargo}")
```
```

### ### Conclusão

Normalizar os dados ou ajustar os pesos são passos importantes para garantir que ambas as métricas sejam consideradas adequadamente na otimização. Isso ajuda a evitar que uma métrica domine a outra devido a diferenças de escala.

Execution Result:

```
```
Welcome to the CBC MILP Solver
Version: 2.10.3
Build Date: Dec 15 2019

command line - /usr/local/lib/python3.10/dist-packages/pulp/solverdir/cbc/linux/64/cbc
```
```



```

/tmp/169c6464afbd4d5ebcefb20ad1127bc8-pulp.mps timeMode elapsed branch printingOptions all solution
/tmp/169c6464afbd4d5ebcefb20ad1127bc8-pulp.sol (default strategy 1)
At line 2 NAME      MODEL
At line 3 ROWS
At line 105 COLUMNS
At line 12606 RHS
At line 12707 BOUNDS
At line 15208 ENDATA
Problem MODEL has 100 rows, 2500 columns and 5000 elements
Coin0008I MODEL read with 0 errors
Option for timeMode changed from cpu to elapsed
Continuous objective value is 28.8931 - 0.00 seconds
Cgl0004I processed model has 100 rows, 2500 columns (2500 integer (2500 of which binary)) and 5000 elements
Cbc0038I Initial state - 0 integers unsatisfied sum - 0
Cbc0038I Solution found of 28.8931
Cbc0038I Before mini branch and bound, 2500 integers at bound fixed and 0 continuous
Cbc0038I Mini branch and bound did not improve solution (0.06 seconds)
Cbc0038I After 0.06 seconds - Feasibility pump exiting with objective of 28.8931 - took 0.00 seconds
Cbc0012I Integer solution of 28.893106 found by feasibility pump after 0 iterations and 0 nodes (0.06 seconds)
Cbc0001I Search completed - best objective 28.89310555618498, took 0 iterations and 0 nodes (0.06 seconds)
Cbc0035I Maximum depth 0, 0 variables fixed on reduced cost
Cuts at root node changed objective from 28.8931 to 28.8931
Probing was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
Gomory was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
Knapsack was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
Clique was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
MixedIntegerRounding2 was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000
seconds)
FlowCover was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
TwoMirCuts was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
ZeroHalf was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)

```

Result - Optimal solution found

```

Objective value:      28.89310556
Enumerated nodes:     0
Total iterations:     0
Time (CPU seconds):   0.06
Time (Wallclock seconds): 0.06

```

```

Option for printingOptions changed from normal to all
Total time (CPU seconds): 0.07 (Wallclock seconds): 0.07

```

Alocações ótimas:

```

Funcionário 1.0 alocado ao cargo 30.0
Funcionário 2.0 alocado ao cargo 26.0
Funcionário 3.0 alocado ao cargo 22.0
Funcionário 4.0 alocado ao cargo 16.0
Funcionário 5.0 alocado ao cargo 69.0
Funcionário 6.0 alocado ao cargo 3.0
Funcionário 7.0 alocado ao cargo 70.0
Funcionário 8.0 alocado ao cargo 11.0
Funcionário 9.0 alocado ao cargo 57.0
Funcionário 10.0 alocado ao cargo 13.0
Funcionário 11.0 alocado ao cargo 71.0
Funcionário 12.0 alocado ao cargo 10.0
Funcionário 13.0 alocado ao cargo 1.0
Funcionário 14.0 alocado ao cargo 29.0
Funcionário 15.0 alocado ao cargo 56.0
Funcionário 16.0 alocado ao cargo 64.0
Funcionário 17.0 alocado ao cargo 58.0
Funcionário 18.0 alocado ao cargo 6.0
Funcionário 19.0 alocado ao cargo 60.0
Funcionário 20.0 alocado ao cargo 27.0
Funcionário 21.0 alocado ao cargo 72.0
Funcionário 22.0 alocado ao cargo 25.0
Funcionário 23.0 alocado ao cargo 8.0
Funcionário 24.0 alocado ao cargo 17.0

```



Funcionário 25.0 alocado ao cargo 18.0  
Funcionário 26.0 alocado ao cargo 73.0  
Funcionário 27.0 alocado ao cargo 23.0  
Funcionário 28.0 alocado ao cargo 24.0  
Funcionário 29.0 alocado ao cargo 20.0  
Funcionário 30.0 alocado ao cargo 9.0  
Funcionário 31.0 alocado ao cargo 12.0  
Funcionário 32.0 alocado ao cargo 59.0  
Funcionário 33.0 alocado ao cargo 62.0  
Funcionário 34.0 alocado ao cargo 7.0  
Funcionário 35.0 alocado ao cargo 66.0  
Funcionário 36.0 alocado ao cargo 63.0  
Funcionário 37.0 alocado ao cargo 74.0  
Funcionário 38.0 alocado ao cargo 19.0  
Funcionário 39.0 alocado ao cargo 15.0  
Funcionário 40.0 alocado ao cargo 67.0  
Funcionário 41.0 alocado ao cargo 28.0  
Funcionário 42.0 alocado ao cargo 65.0  
Funcionário 43.0 alocado ao cargo 55.0  
Funcionário 44.0 alocado ao cargo 14.0  
Funcionário 45.0 alocado ao cargo 5.0  
Funcionário 46.0 alocado ao cargo 2.0  
Funcionário 47.0 alocado ao cargo 4.0  
Funcionário 48.0 alocado ao cargo 68.0  
Funcionário 49.0 alocado ao cargo 21.0  
Funcionário 50.0 alocado ao cargo 61.0  
``