

```
In [ ]: import math
import random
import numpy as np
import heapq
from collections import deque
```

## Questão 1

```
In [ ]: numeros = list(range(21))

media = sum(numeros) / len(numeros)

variancia = sum((x - media) ** 2 for x in numeros) / len(numeros)

desvio_padrao = math.sqrt(variancia)

print(f"Sequência: {numeros}")
print(f"Média: {media}")
print(f"Variância: {variancia}")
print(f"Desvio Padrão: {desvio_padrao}")
```

Sequência: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]  
Média: 10.0  
Variância: 36.666666666666664  
Desvio Padrão: 6.0553007081949835

## Questão 2

```
In [ ]: calcular_media = sum(numeros) / len(numeros)

calcular_variancia = sum((x - media) ** 2 for x in numeros) / len(numeros)

calcular_desvio_padrao = math.sqrt(variancia)

entrada = input("Digite uma sequência de números separados por espaço: ")
numeros = list(map(float, entrada.split()))

print(f"Média: {calcular_media:.2f}")
print(f"Variância: {calcular_variancia:.2f}")
print(f"Desvio Padrão: {calcular_desvio_padrao:.2f}")
```

Digite uma sequência de números separados por espaço: 1 2 3  
Média: 10.00  
Variância: 36.67  
Desvio Padrão: 6.06

## Questão 3

```
In [ ]: numeros = [random.randint(0, 1000) for _ in range(100)]

media = sum(numeros) / len(numeros)
```

```

variancia = sum((x - media) ** 2 for x in numeros) / len(numeros)

desvio_padrao = math.sqrt(variancia)

print("Números gerados:", numeros)
print(f"Média: {media:.2f}")
print(f"Variância: {variancia:.2f}")
print(f"Desvio Padrão: {desvio_padrao:.2f}")

```

Números gerados: [426, 529, 532, 949, 583, 85, 626, 90, 530, 887, 178, 326, 93, 8  
43, 264, 349, 698, 594, 507, 907, 235, 668, 951, 702, 633, 425, 11, 441, 695, 42  
1, 840, 175, 696, 141, 520, 442, 624, 862, 703, 575, 717, 262, 269, 57, 858, 971,  
80, 659, 319, 781, 399, 313, 462, 494, 130, 685, 305, 404, 264, 247, 864, 639, 64  
0, 359, 549, 407, 343, 972, 830, 629, 789, 864, 487, 316, 193, 149, 917, 203, 99  
7, 25, 103, 213, 282, 922, 150, 428, 704, 885, 786, 543, 722, 989, 676, 128, 963,  
943, 45, 571, 501, 565]  
Média: 517.23  
Variância: 77713.32  
Desvio Padrão: 278.77

## Questao 4

X	Y	$X^2$	$Y^2$	X.Y
2	1	4	1	2
3	2	9	4	6
4	3	16	9	12
5	-3	25	9	-15
6	4	36	16	24

$$r_{xy} = \frac{5 * 29 - 20 * 7}{\sqrt{5 * 90 - 20^2} * \sqrt{5 * 39 - 7^2}}$$

$$r_{xy} = \frac{145 - 140}{\sqrt{50} * \sqrt{146}}$$

$$r_{xy} = \frac{5}{85,44003745317531}$$

$$r_{xy} = 0,0585205735980653$$

## Questão 5

X	Y	X <sup>2</sup>	Y <sup>2</sup>	X.Y
4	7	12	49	28
5	2	25	4	10
8	3	64	9	24
7	8	49	64	56
6	6	36	36	36

$$r_{xy} = \frac{5 * 154 - 24 * 26}{\sqrt{5 * 186 - 24^2} * \sqrt{5 * 162 - 26^2}}$$

$$r_{xy} = \frac{770 - 624}{\sqrt{354} * \sqrt{134}}$$

$$r_{xy} = \frac{146}{198}$$

$$r_{xy} = 0,73$$

## Questao 6

```
In [ ]: grafo = {
    'POA': [('FLN', 376), ('CWB', 547)],
    'FLN': [('POA', 376), ('CWB', 251), ('SAO', 489)],
    'CWB': [('POA', 547), ('FLN', 251), ('SAO', 408), ('BSB', 1214)],
    'SAO': [('FLN', 489), ('CWB', 408), ('RIO', 357), ('BH', 586)],
    'RIO': [('SAO', 357), ('BH', 434), ('BSB', 1148)],
    'BH': [('SAO', 586), ('RIO', 434), ('BSB', 624), ('CGB', 1594)],
    'BSB': [('CWB', 1214), ('RIO', 1148), ('BH', 624), ('CGB', 1132), ('SSA', 14),
    'CGB': [('BSB', 1132), ('BH', 1594), ('SSA', 1932), ('MAO', 1931)],
    'SSA': [('BSB', 1442), ('CGB', 1932), ('FOR', 1381)],
    'FOR': [('SSA', 1381), ('MAO', 2384)],
    'MAO': [('CGB', 1931), ('FOR', 2384)]}

heuristica = {
    'POA': 376, 'FLN': 0, 'CWB': 251, 'SAO': 489, 'RIO': 726,
    'BH': 963, 'BSB': 1214, 'CGB': 1347, 'SSA': 1932, 'FOR': 2176, 'MAO': 2716
}

def calcula_custo(caminho, grafo):
    return sum(peso for vizinho, peso in grafo[caminho[i]] if vizinho == caminho[i] for i in range(len(caminho) - 1))

def busca_em_largura(grafo, origem, destino):
    fila = deque([[origem]])
```

```

visitados = set()

while fila:
    caminho = fila.popleft()
    no_atual = caminho[-1]

    if no_atual == destino:
        return caminho, calcula_custo(caminho, grafo)

    if no_atual not in visitados:
        visitados.add(no_atual)
        for vizinho, _ in grafo[no_atual]:
            novo_caminho = list(caminho)
            novo_caminho.append(vizinho)
            fila.append(novo_caminho)

return None, float("inf")

def busca_em_profundidade(grafo, origem, destino):
    pilha = [[origem]]
    visitados = set()

    while pilha:
        caminho = pilha.pop()
        no_atual = caminho[-1]

        if no_atual == destino:
            return caminho, calcula_custo(caminho, grafo)

        if no_atual not in visitados:
            visitados.add(no_atual)
            for vizinho, _ in grafo[no_atual]:
                novo_caminho = list(caminho)
                novo_caminho.append(vizinho)
                pilha.append(novo_caminho)

    return None, float("inf")

def busca_bidirecional(grafo, origem, destino):
    frente = {origem: [origem]}
    traseira = {destino: [destino]}
    visitados = set()

    while frente and traseira:
        novo_frente = {}
        for no_atual, caminho in frente.items():
            if no_atual in traseira:
                caminho_completo = caminho + traseira[no_atual][:-1][1:]
                return caminho_completo, calcula_custo(caminho_completo, grafo)

            visitados.add(no_atual)
            for vizinho, _ in grafo[no_atual]:
                if vizinho not in visitados:
                    novo_frente[vizinho] = caminho + [vizinho]

        frente = novo_frente

        novo_traseira = {}
        for no_atual, caminho in traseira.items():
            if no_atual in frente:

```

```

        caminho_completo = frente[no_atual] + caminho[::-1][1:]
        return caminho_completo, calcula_custo(caminho_completo, grafo)

    visitados.add(no_atual)
    for vizinho, _ in grafo[no_atual]:
        if vizinho not in visitados:
            novo_traseira[vizinho] = caminho + [vizinho]

    traseira = novo_traseira

    return None, float("inf")

def busca_gulosa(grafo, heuristica, origem, destino):
    fila_prioridade = [(heuristica[origem], [origem])]
    visitados = set()

    while fila_prioridade:
        _, caminho = heapq.heappop(fila_prioridade)
        no_atual = caminho[-1]

        if no_atual == destino:
            return caminho, calcula_custo(caminho, grafo)

        if no_atual not in visitados:
            visitados.add(no_atual)
            for vizinho, _ in grafo[no_atual]:
                novo_caminho = list(caminho)
                novo_caminho.append(vizinho)
                heapq.heappush(fila_prioridade, (heuristica[vizinho], novo_caminho))

    return None, float("inf")

def busca_a_estrela(grafo, heuristica, origem, destino):
    fila_prioridade = [(heuristica[origem], 0, [origem])]
    visitados = set()

    while fila_prioridade:
        _, custo_atual, caminho = heapq.heappop(fila_prioridade)
        no_atual = caminho[-1]

        if no_atual == destino:
            return caminho, custo_atual

        if no_atual not in visitados:
            visitados.add(no_atual)
            for vizinho, peso in grafo[no_atual]:
                novo_caminho = list(caminho)
                novo_caminho.append(vizinho)
                novo_custo = custo_atual + peso
                heapq.heappush(fila_prioridade, (novo_custo + heuristica[vizinho], novo_caminho))

    return None, float("inf")

resultado_largura = busca_em_largura(grafo, "CGB", "FLN")
resultado_profundidade = busca_em_profundidade(grafo, "CGB", "FLN")
resultado_bidirecional = busca_bidirecional(grafo, "CGB", "FLN")
resultado_gulosa = busca_gulosa(grafo, heuristica, "CGB", "FLN")
resultado_a_estrela = busca_a_estrela(grafo, heuristica, "CGB", "FLN")

resultados = {

```

```

    "Busca em Largura": resultado_largura,
    "Busca em Profundidade": resultado_profundidade,
    "Busca Bidirecional": resultado_bidirecional,
    "Busca Gulosa": resultado_gulosa,
    "Busca A*": resultado_a_estrela,
}

for metodo, (caminho, custo) in resultados.items():
    print(f'{metodo}: Caminho = {caminho}, Custo = {custo}')

melhor = min(resultados.items(), key=lambda x: x[1][1])
print("\nMelhor Algoritmo:", melhor[0], "com custo", melhor[1][1])

```

Busca em Largura: Caminho = CGB -> BSB -> CWB -> FLN, Custo = 2597  
 Busca em Profundidade: Caminho = CGB -> MAO -> FOR -> SSA -> BSB -> BH -> RIO ->  
 SAO -> CWB -> FLN, Custo = 9212  
 Busca Bidirecional: Caminho = CGB -> BSB -> CWB -> FLN, Custo = 2597  
 Busca Gulosa: Caminho = CGB -> BH -> SAO -> FLN, Custo = 2669  
 Busca A\*: Caminho = CGB -> BSB -> CWB -> FLN, Custo = 2597

Melhor Algoritmo: Busca em Largura com custo 2597

## Questão 7

```

In [ ]: grafo = {
    "POA": {"FLN": 376, "CWB": 547, "SP": 812},
    "FLN": {"POA": 376, "CWB": 251},
    "CWB": {"POA": 547, "FLN": 251, "SP": 408, "CGB": 1679},
    "SP": {"POA": 812, "CWB": 408, "BH": 586, "RJ": 429},
    "RJ": {"SP": 429, "BH": 434, "SSA": 1196},
    "BH": {"SP": 586, "RJ": 434, "BSB": 624},
    "BSB": {"BH": 624, "CGB": 1131, "SSA": 1444},
    "CGB": {"BSB": 1131, "CWB": 1679},
    "SSA": {"BSB": 1444, "RJ": 1196, "FOR": 1203},
    "FOR": {"SSA": 1203, "MAO": 2384},
    "MAO": {"FOR": 2384}
}

heuristica = {
    "POA": 1000, "FLN": 800, "CWB": 600, "SP": 500, "RJ": 400,
    "BH": 300, "BSB": 200, "CGB": 100, "SSA": 400, "FOR": 600, "MAO": 800
}

def calcula_custo(caminho, grafo):
    return sum(grafo[caminho[i]][caminho[i + 1]] for i in range(len(caminho) - 1))

def busca_em_largura(grafo, origem, destino):
    fila = deque([[origem]])
    visitados = set()

    while fila:
        caminho = fila.popleft()
        no_atual = caminho[-1]

        if no_atual == destino:

```

```

        return caminho, calcula_custo(caminho, grafo)

    if no_atual not in visitados:
        visitados.add(no_atual)
        for vizinho in grafo[no_atual]:
            novo_caminho = list(caminho)
            novo_caminho.append(vizinho)
            fila.append(novo_caminho)

    return None, float("inf")

def busca_em_profundidade(grafo, origem, destino):
    pilha = [[origem]]
    visitados = set()

    while pilha:
        caminho = pilha.pop()
        no_atual = caminho[-1]

        if no_atual == destino:
            return caminho, calcula_custo(caminho, grafo)

        if no_atual not in visitados:
            visitados.add(no_atual)
            for vizinho in grafo[no_atual]:
                novo_caminho = list(caminho)
                novo_caminho.append(vizinho)
                pilha.append(novo_caminho)

    return None, float("inf")

def busca_bidirecional(grafo, origem, destino):
    frente = {origem: [origem]}
    traseira = {destino: [destino]}
    visitados = set()

    while frente and traseira:
        novo_frente = {}
        for no_atual, caminho in frente.items():
            if no_atual in traseira:
                caminho_completo = caminho + traseira[no_atual][:-1][1:]
                return caminho_completo, calcula_custo(caminho_completo, grafo)

            visitados.add(no_atual)
            for vizinho in grafo[no_atual]:
                if vizinho not in visitados:
                    novo_frente[vizinho] = caminho + [vizinho]

        frente = novo_frente

        novo_traseira = {}
        for no_atual, caminho in traseira.items():
            if no_atual in frente:
                caminho_completo = frente[no_atual] + caminho[:-1][1:]
                return caminho_completo, calcula_custo(caminho_completo, grafo)

            visitados.add(no_atual)
            for vizinho in grafo[no_atual]:
                if vizinho not in visitados:
                    novo_traseira[vizinho] = caminho + [vizinho]

```

```

        traseira = novo_traseira

    return None, float("inf")

def busca_gulosa(grafo, heuristica, origem, destino):
    fila_prioridade = [(heuristica[origem], [origem])]
    visitados = set()

    while fila_prioridade:
        _, caminho = heapq.heappop(fila_prioridade)
        no_atual = caminho[-1]

        if no_atual == destino:
            return caminho, calcula_custo(caminho, grafo)

        if no_atual not in visitados:
            visitados.add(no_atual)
            for vizinho in grafo[no_atual]:
                novo_caminho = list(caminho)
                novo_caminho.append(vizinho)
                heapq.heappush(fila_prioridade, (heuristica[vizinho], novo_caminho))

    return None, float("inf")

def busca_a_estrela(grafo, heuristica, origem, destino):
    fila_prioridade = [(heuristica[origem], 0, [origem])]
    visitados = set()

    while fila_prioridade:
        _, custo_atual, caminho = heapq.heappop(fila_prioridade)
        no_atual = caminho[-1]

        if no_atual == destino:
            return caminho, custo_atual

        if no_atual not in visitados:
            visitados.add(no_atual)
            for vizinho in grafo[no_atual]:
                novo_caminho = list(caminho)
                novo_caminho.append(vizinho)
                novo_custo = custo_atual + grafo[no_atual][vizinho]
                heapq.heappush(fila_prioridade, (novo_custo + heuristica[vizinho], novo_caminho))

    return None, float("inf")

origem = input("Digite a cidade de origem (Ex: POA): ").strip().upper()
destino = input("Digite a cidade de destino (Ex: MAO): ").strip().upper()

if origem not in grafo or destino not in grafo:
    print("Origem ou destino inválidos! Tente novamente.")
else:

    resultados = {
        "Busca em Largura": busca_em_largura(grafo, origem, destino),
        "Busca em Profundidade": busca_em_profundidade(grafo, origem, destino),
        "Busca Bidirecional": busca_bidirecional(grafo, origem, destino),
        "Busca Gulosa": busca_gulosa(grafo, heuristica, origem, destino),
        "Busca A*": busca_a_estrela(grafo, heuristica, origem, destino),
    }

```

```

}

for algoritmo, resultado in resultados.items():
    caminho, custo = resultado
    print(f"{algoritmo}: Caminho = {' -> '.join(caminho)} | Custo = {custo}")

melhor = min(resultados.items(), key=lambda x: x[1][1])
print(f"\nMelhor Algoritmo: {melhor[0]} com custo {melhor[1][1]}")

```

Digite a cidade de origem (Ex: POA): POA  
 Digite a cidade de destino (Ex: MAO): MAO  
 Busca em Largura: Caminho = POA -> SP -> RJ -> SSA -> FOR -> MAO | Custo = 6024  
 Busca em Profundidade: Caminho = POA -> SP -> RJ -> SSA -> FOR -> MAO | Custo = 6024  
 Busca Bidirecional: Caminho = POA -> SP -> RJ -> SSA -> FOR -> MAO | Custo = 6024  
 Busca Gulosa: Caminho = POA -> SP -> BH -> BSB -> SSA -> FOR -> MAO | Custo = 705  
 3  
 Busca A\*: Caminho = POA -> SP -> RJ -> SSA -> FOR -> MAO | Custo = 6024

Melhor Algoritmo: Busca em Largura com custo 6024

## Questão 8

## Questão 9

```

In [ ]: X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 1, 1, 1])

np.random.seed(42)
weights = np.random.rand(2)
bias = np.random.rand()

learning_rate = 0.1
epochs = 10

def step_function(value):
    return 1 if value >= 0 else 0

for epoch in range(epochs):
    print(f"Época {epoch + 1}")
    for i in range(len(X)):

        weighted_sum = np.dot(X[i], weights) + bias

        y_pred = step_function(weighted_sum)

        error = y[i] - y_pred

        weights += learning_rate * error * X[i]
        bias += learning_rate * error

```

```
print(f" Amostra {i + 1}:")
print(f" Entrada: {X[i]}")
print(f" Soma ponderada: {weighted_sum:.4f}")
print(f" Saída prevista: {y_pred}")
print(f" Erro: {error}")
print(f" Novos pesos: {weights}")
print(f" Novo bias: {bias}")
print("-" * 50)

print("Treinamento concluído!")
print(f"Pesos finais: {weights}")
print(f"Bias final: {bias}")
```

**Época 1****Amostra 1:**

Entrada: [0 0]  
Soma ponderada: 0.7320  
Saída prevista: 1  
Erro: -1  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: 0.6319939418114051

**Amostra 2:**

Entrada: [0 1]  
Soma ponderada: 1.5827  
Saída prevista: 1  
Erro: 0  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: 0.6319939418114051

**Amostra 3:**

Entrada: [1 0]  
Soma ponderada: 1.0065  
Saída prevista: 1  
Erro: 0  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: 0.6319939418114051

**Amostra 4:**

Entrada: [1 1]  
Soma ponderada: 1.9572  
Saída prevista: 1  
Erro: 0  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: 0.6319939418114051

---

**Época 2****Amostra 1:**

Entrada: [0 0]  
Soma ponderada: 0.6320  
Saída prevista: 1  
Erro: -1  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: 0.5319939418114051

**Amostra 2:**

Entrada: [0 1]  
Soma ponderada: 1.4827  
Saída prevista: 1  
Erro: 0  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: 0.5319939418114051

**Amostra 3:**

Entrada: [1 0]  
Soma ponderada: 0.9065  
Saída prevista: 1  
Erro: 0  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: 0.5319939418114051

**Amostra 4:**

Entrada: [1 1]  
Soma ponderada: 1.8572  
Saída prevista: 1  
Erro: 0  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: 0.5319939418114051

---

**Época 3****Amostra 1:**

Entrada: [0 0]  
Soma ponderada: 0.5320  
Saída prevista: 1  
Erro: -1  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: 0.43199394181140516

**Amostra 2:**

Entrada: [0 1]  
Soma ponderada: 1.3827  
Saída prevista: 1  
Erro: 0  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: 0.43199394181140516

**Amostra 3:**

Entrada: [1 0]  
Soma ponderada: 0.8065  
Saída prevista: 1  
Erro: 0  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: 0.43199394181140516

**Amostra 4:**

Entrada: [1 1]  
Soma ponderada: 1.7572  
Saída prevista: 1  
Erro: 0  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: 0.43199394181140516

---

**Época 4****Amostra 1:**

Entrada: [0 0]  
Soma ponderada: 0.4320  
Saída prevista: 1  
Erro: -1  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: 0.3319939418114052

**Amostra 2:**

Entrada: [0 1]  
Soma ponderada: 1.2827  
Saída prevista: 1  
Erro: 0  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: 0.3319939418114052

**Amostra 3:**

Entrada: [1 0]  
Soma ponderada: 0.7065  
Saída prevista: 1  
Erro: 0  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: 0.3319939418114052

**Amostra 4:**

Entrada: [1 1]  
Soma ponderada: 1.6572  
Saída prevista: 1  
Erro: 0  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: 0.3319939418114052

---

**Época 5****Amostra 1:**

Entrada: [0 0]  
Soma ponderada: 0.3320  
Saída prevista: 1  
Erro: -1  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: 0.23199394181140517

**Amostra 2:**

Entrada: [0 1]  
Soma ponderada: 1.1827  
Saída prevista: 1  
Erro: 0  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: 0.23199394181140517

**Amostra 3:**

Entrada: [1 0]  
Soma ponderada: 0.6065  
Saída prevista: 1  
Erro: 0  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: 0.23199394181140517

**Amostra 4:**

Entrada: [1 1]  
Soma ponderada: 1.5572  
Saída prevista: 1  
Erro: 0  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: 0.23199394181140517

---

**Época 6****Amostra 1:**

Entrada: [0 0]  
Soma ponderada: 0.2320  
Saída prevista: 1  
Erro: -1  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: 0.13199394181140517

**Amostra 2:**

Entrada: [0 1]  
Soma ponderada: 1.0827  
Saída prevista: 1  
Erro: 0  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: 0.13199394181140517

**Amostra 3:**

Entrada: [1 0]  
Soma ponderada: 0.5065  
Saída prevista: 1  
Erro: 0  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: 0.13199394181140517

**Amostra 4:**

Entrada: [1 1]  
Soma ponderada: 1.4572  
Saída prevista: 1  
Erro: 0  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: 0.13199394181140517

---

**Época 7****Amostra 1:**

Entrada: [0 0]  
Soma ponderada: 0.1320  
Saída prevista: 1  
Erro: -1  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: 0.03199394181140516

**Amostra 2:**

Entrada: [0 1]  
Soma ponderada: 0.9827  
Saída prevista: 1  
Erro: 0  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: 0.03199394181140516

**Amostra 3:**

Entrada: [1 0]  
Soma ponderada: 0.4065  
Saída prevista: 1  
Erro: 0  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: 0.03199394181140516

**Amostra 4:**

Entrada: [1 1]  
Soma ponderada: 1.3572  
Saída prevista: 1  
Erro: 0  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: 0.03199394181140516

---

**Época 8****Amostra 1:**

Entrada: [0 0]  
Soma ponderada: 0.0320  
Saída prevista: 1  
Erro: -1  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: -0.06800605818859484

**Amostra 2:**

Entrada: [0 1]  
Soma ponderada: 0.8827  
Saída prevista: 1  
Erro: 0  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: -0.06800605818859484

**Amostra 3:**

Entrada: [1 0]  
Soma ponderada: 0.3065  
Saída prevista: 1  
Erro: 0  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: -0.06800605818859484

**Amostra 4:**

Entrada: [1 1]  
Soma ponderada: 1.2572  
Saída prevista: 1  
Erro: 0  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: -0.06800605818859484

---

**Época 9****Amostra 1:**

Entrada: [0 0]  
Soma ponderada: -0.0680  
Saída prevista: 0  
Erro: 0  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: -0.06800605818859484

**Amostra 2:**

Entrada: [0 1]  
Soma ponderada: 0.8827  
Saída prevista: 1  
Erro: 0  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: -0.06800605818859484

**Amostra 3:**

Entrada: [1 0]  
Soma ponderada: 0.3065  
Saída prevista: 1  
Erro: 0  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: -0.06800605818859484

**Amostra 4:**

Entrada: [1 1]  
Soma ponderada: 1.2572  
Saída prevista: 1  
Erro: 0  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: -0.06800605818859484

---

**Época 10****Amostra 1:**

Entrada: [0 0]  
Soma ponderada: -0.0680  
Saída prevista: 0  
Erro: 0  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: -0.06800605818859484

**Amostra 2:**

Entrada: [0 1]  
Soma ponderada: 0.8827  
Saída prevista: 1  
Erro: 0  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: -0.06800605818859484

**Amostra 3:**

Entrada: [1 0]  
Soma ponderada: 0.3065  
Saída prevista: 1  
Erro: 0  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: -0.06800605818859484

**Amostra 4:**

Entrada: [1 1]  
Soma ponderada: 1.2572  
Saída prevista: 1  
Erro: 0  
Novos pesos: [0.37454012 0.95071431]  
Novo bias: -0.06800605818859484

---

Treinamento concluído!  
 Pesos finais: [0.37454012 0.95071431]  
 Bias final: -0.06800605818859484

## Questão 10

```
In [ ]: X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[1], [0], [0], [1]])

input_neurons = 2
hidden_neurons = 2
output_neurons = 1
learning_rate = 0.1
epochs = 10000

np.random.seed(42)
weights_input_hidden = np.random.rand(input_neurons, hidden_neurons)
weights_hidden_output = np.random.rand(hidden_neurons, output_neurons)
bias_hidden = np.random.rand(1, hidden_neurons)
bias_output = np.random.rand(1, output_neurons)

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)

for epoch in range(epochs):
    hidden_layer_input = np.dot(X, weights_input_hidden) + bias_hidden
    hidden_layer_output = sigmoid(hidden_layer_input)

    output_layer_input = np.dot(hidden_layer_output, weights_hidden_output) + bias_output
    predicted_output = sigmoid(output_layer_input)

    error = y - predicted_output

    d_predicted_output = error * sigmoid_derivative(predicted_output)
    error_hidden_layer = d_predicted_output.dot(weights_hidden_output.T)
    d_hidden_layer = error_hidden_layer * sigmoid_derivative(hidden_layer_output)

    weights_hidden_output += hidden_layer_output.T.dot(d_predicted_output) * learning_rate
    bias_output += np.sum(d_predicted_output, axis=0, keepdims=True) * learning_rate
    weights_input_hidden += X.T.dot(d_hidden_layer) * learning_rate
    bias_hidden += np.sum(d_hidden_layer, axis=0, keepdims=True) * learning_rate

print("Pesos da camada de entrada para a oculta:")
print(f"{weights_input_hidden}\n")
print("Pesos da camada oculta para a saída:")
print(f"{weights_hidden_output}\n")
print("Bias da camada oculta:")
print(f"{bias_hidden}\n")
print("Bias da camada de saída:")
print(f"{bias_output}\n")

print("\nPrevisão final:")
print(predicted_output)
```

Pesos da camada de entrada para a oculta:

```
[[3.77215957 5.75838716]
 [3.77730823 5.7826105 ]]
```

Pesos da camada oculta para a saída:

```
[[ 8.16931337]
 [-7.5635735 ]]
```

Bias da camada oculta:

```
[[ -5.78169618 -2.41069041]]
```

Bias da camada de saída:

```
[[3.42455728]]
```

Previsão final:

```
[[0.94409812]
 [0.05127717]
 [0.05135274]
 [0.94478705]]
```