Caixeiro Viajante - Simulated Annealing

Nome: Arthur Pontes Nader

Matrícula: 2019022294

Bibliotecas

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
```

Funções

In [ ]:
```
def gerar_cidades(N):

  coordenadas_x = np.random.random(N)
  coordenadas_y = np.random.random(N)

  cidades = np.vstack((coordenadas_x, coordenadas_y))

  return cidades.T
```

In [ ]:
```
def calcular_distancia(cidade1, cidade2):

    return np.sqrt(np.square(cidade1[0] - cidade2[0]) + np.square(cidade1[1] - cidade2[1
```

In [ ]:
```
def gerar_matriz_de_distancias(cidades):

  num_cidades = len(cidades)
  distancias = np.zeros((num_cidades, num_cidades))

  for i in range(num_cidades):
      for k in range(i):
          d = calcular_distancia(cidades[i], cidades[k])
          distancias[i][k] = d
          distancias[k][i] = d

  return distancias
```

In [ ]:
```
def calcular_distancia_do_caminho(caminho, matriz_de_distancias):

  distancia = 0
  for i in range(len(caminho) - 1):
      distancia += matriz_de_distancias[caminho[i],caminho[i+1]]
  distancia += matriz_de_distancias[caminho[0],caminho[N-1]]

  return distancia
```

In [ ]:
```
def gerar_novo_caminho(caminho_atual, distancia_atual, matriz_de_distancias, N):

    novo_caminho = np.zeros(N,dtype=np.int16)
    i=np.random.randint(N)
    j=i

    while j==i: # escolhe j de forma que j ≠ i
        j=np.random.randint(N)
```

```
    if i>j:
        inicio = j
        fim = i
    else:
        inicio = i
        fim = j

    for k in range(N):
        if k >= inicio and k <= fim:
            novo_caminho[k] = caminho_atual[fim-k+inicio]
        else:
            novo_caminho[k] = caminho_atual[k]

    esq = inicio - 1
    if esq < 0:
      esq = N - 1

    dir = fim + 1
    if dir > N - 1:
      dir = 0

    diferenca_distancia = -matriz_de_distancias[caminho_atual[esq], caminho_atual[inicio
                          -matriz_de_distancias[caminho_atual[dir], caminho_atual[fim]]\
                          +matriz_de_distancias[novo_caminho[esq], novo_caminho[inicio]]
                          +matriz_de_distancias[novo_caminho[dir], novo_caminho[fim]]

    return novo_caminho, diferenca_distancia
```

In [ ]:
```
def avaliar_troca_de_caminho(distancia_atual, diferenca, temperatura):

    trocar = False

    if diferenca < 0 or np.random.uniform(0, 1) < np.exp(-diferenca/temperatura):
        trocar = True

    return trocar
```

In [ ]:
```
def caixeiro_viajante(N, Ti = 8, Tf = 0.0005, dt = 0.85):

    cidades = gerar_cidades(N)
    matriz_distancias = gerar_matriz_de_distancias(cidades)
    caminho_atual = np.arange(N)
    distancia_atual = calcular_distancia_do_caminho(caminho_atual, matriz_distancias)

    Tatual = Ti
    caminhos = [caminho_atual]
    distancias = [distancia_atual]

    while Tatual > Tf:
      for i in range(100):
         novo_caminho, diferenca = gerar_novo_caminho(caminho_atual, distancia_atual, ma
         if avaliar_troca_de_caminho(distancia_atual, diferenca, Tatual):
           caminho_atual = novo_caminho
           distancia_atual = distancia_atual + diferenca
         distancias.append(distancia_atual)

      caminhos.append(caminho_atual)
      Tatual = Tatual*dt

    return cidades, caminhos, distancias
```

Funções para exibição dos resultados

In [ ]:
```
def plotar_grafico_distancias(distancias, N, temperatura = 8):

  plt.figure(figsize=(16, 8))
  plt.plot(distancias)
  plt.xlabel("Iteração", fontsize = 16)
  plt.ylabel("Distância total", fontsize = 16)
  plt.title("Evolução da distância para " + str(N) + " cidades e Temperatura Inicial = "
  plt.show()
```

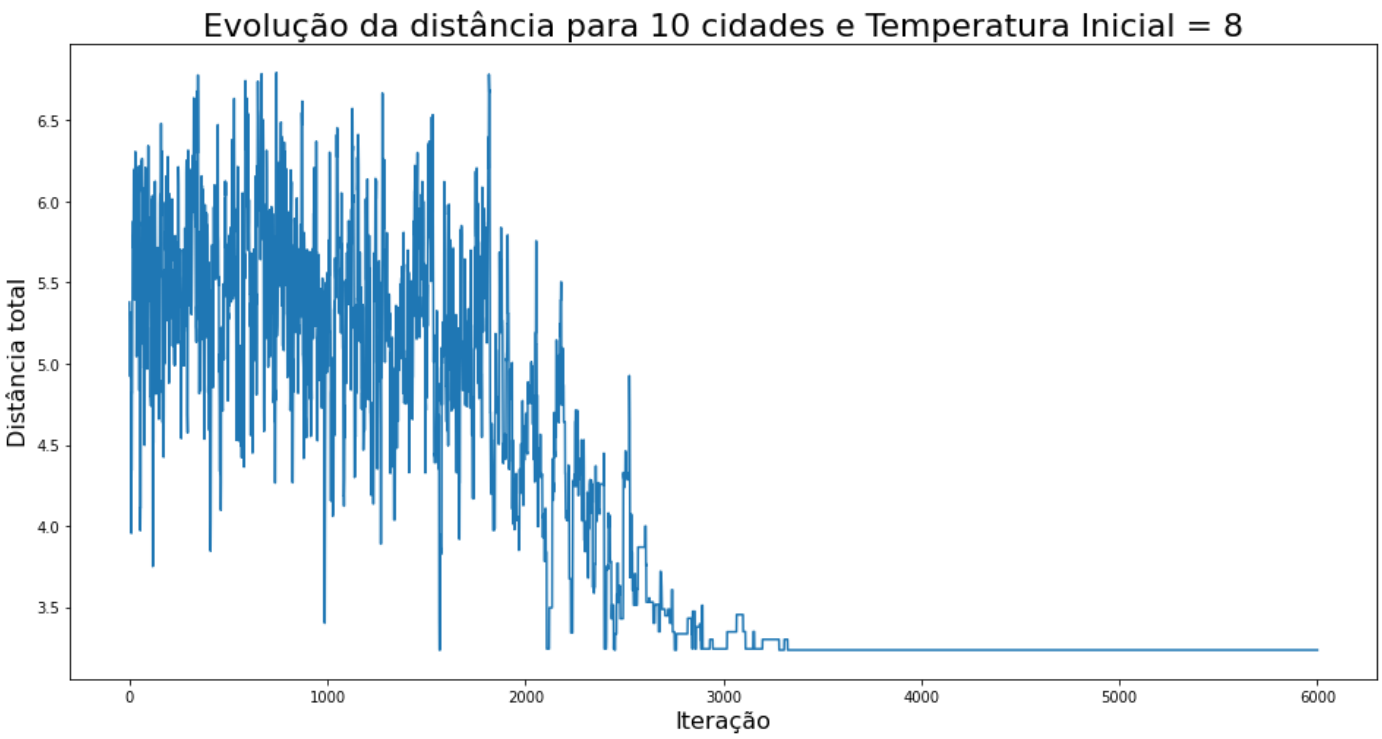
In [ ]:
```
def gerar_caminhos_plot(caminhos):

  caminhos_plot = []
  intervalo_caminhos = len(caminhos)//4

  for i in range(0,len(caminhos), intervalo_caminhos):
    caminhos_plot.append(caminhos[i])
  caminhos_plot.append(caminhos[-1])

  return caminhos_plot
```

In [ ]:
```
def plotar_caminho(cidades, caminho):



  plt.figure(figsize=(16, 8))
  plt.scatter(cidades.T[0], cidades.T[1])
  N = len(caminho)

  for i in range(N-1):
      plt.plot([cidades[caminho[i]][0], cidades[caminho[i+1]][0]], [cidades[caminho[i]][
  plt.plot([cidades[caminho[N-1]][0], cidades[caminho[0]][0]], [cidades[caminho[N-1]][1]

  plt.xlabel("Coordenada x", fontsize = 16)
  plt.ylabel("Coordenada y", fontsize = 16)
  plt.title("Caminho entre as cidades", fontsize = 22)
  plt.show()
```

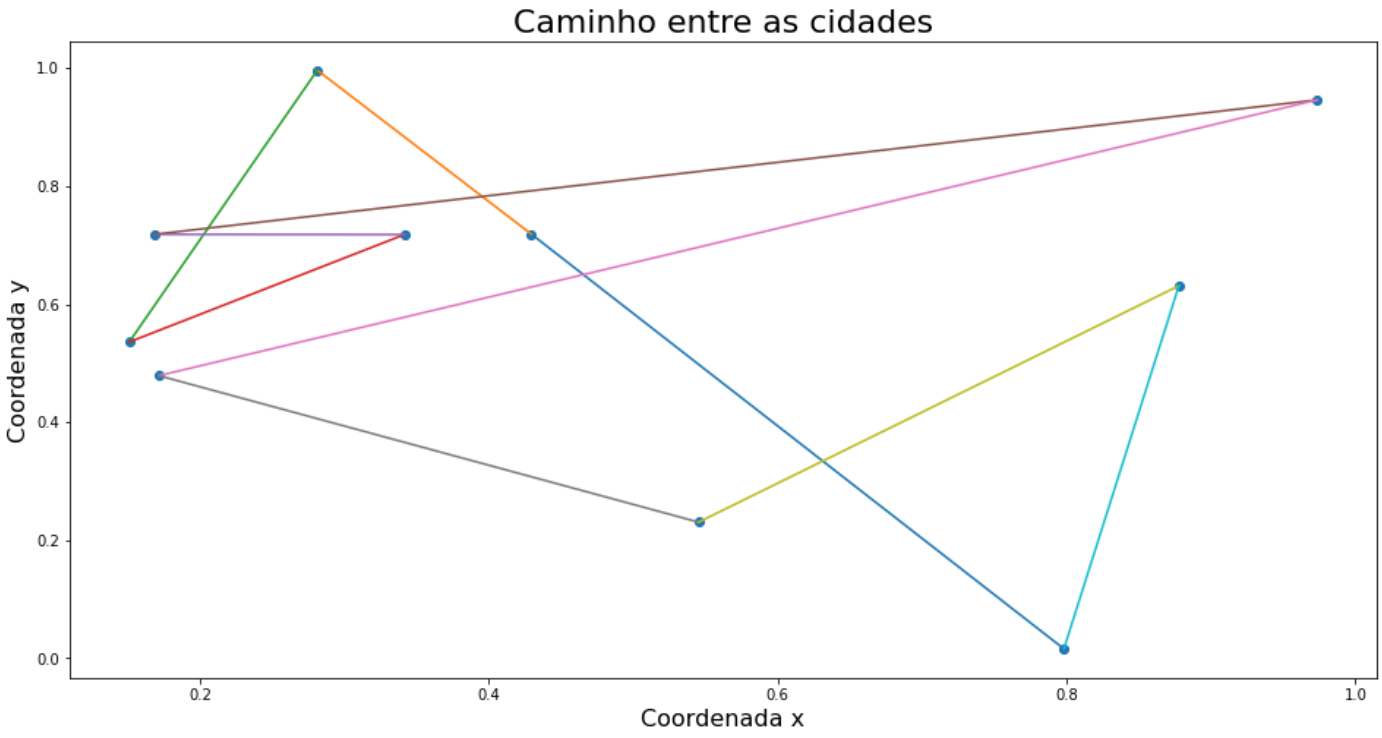
Execuções variando o número de cidades

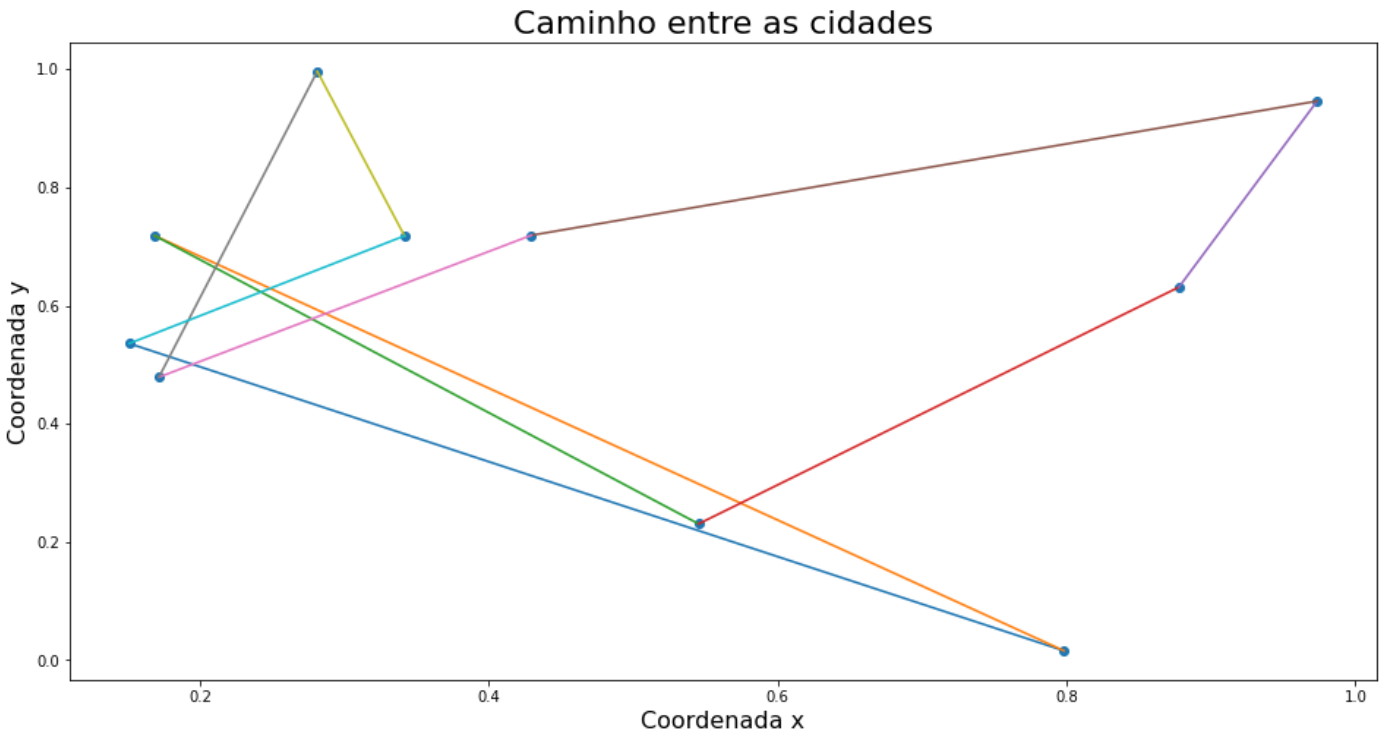
N = 10

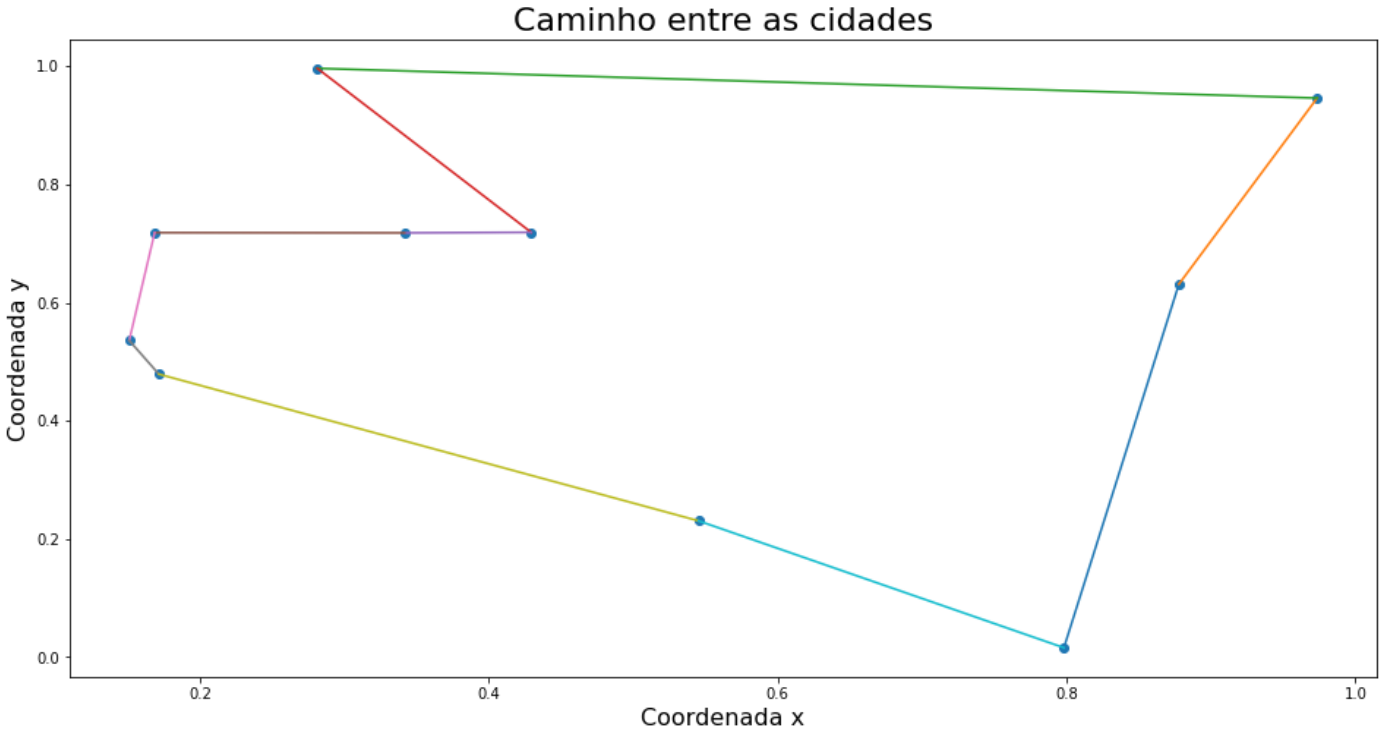
In [ ]:
```
N = 10
cidades, caminhos, distancias = caixeiro_viajante(N)
plotar_grafico_distancias(distancias, N)
```

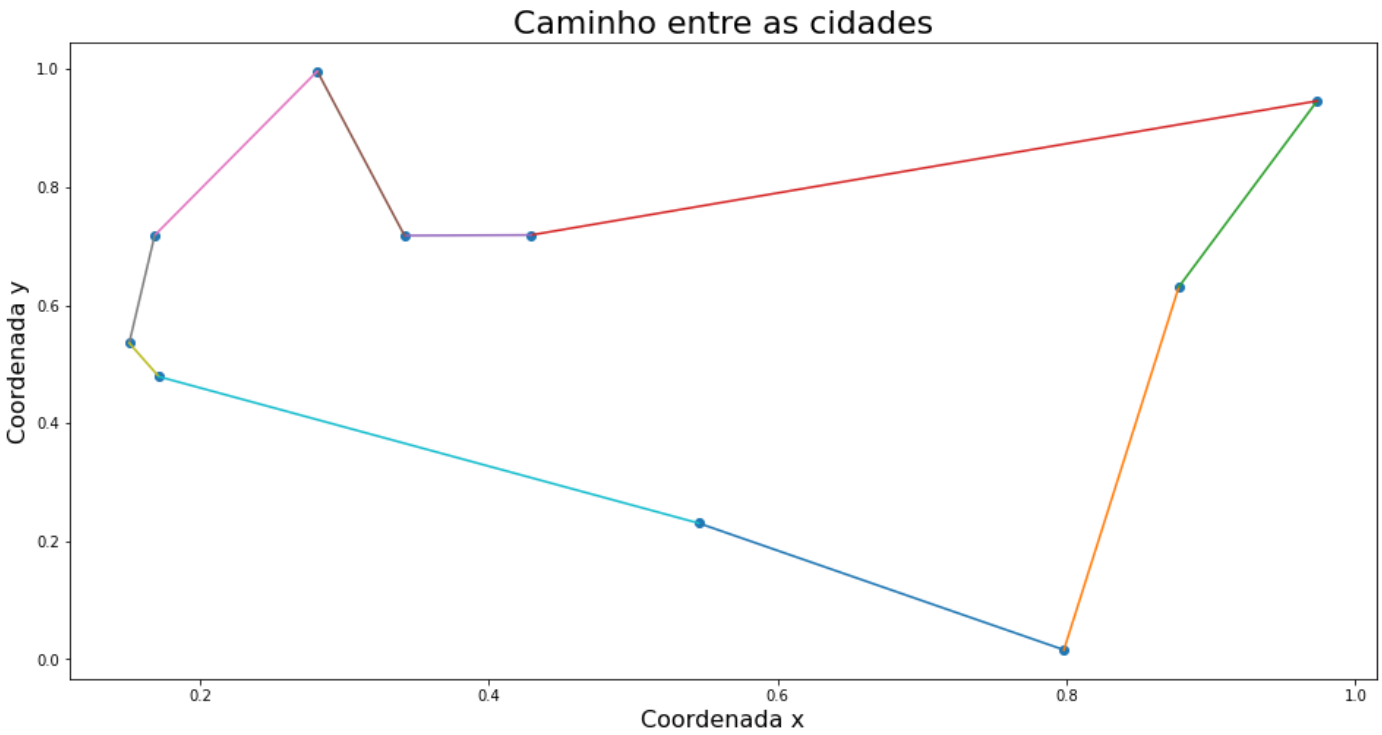


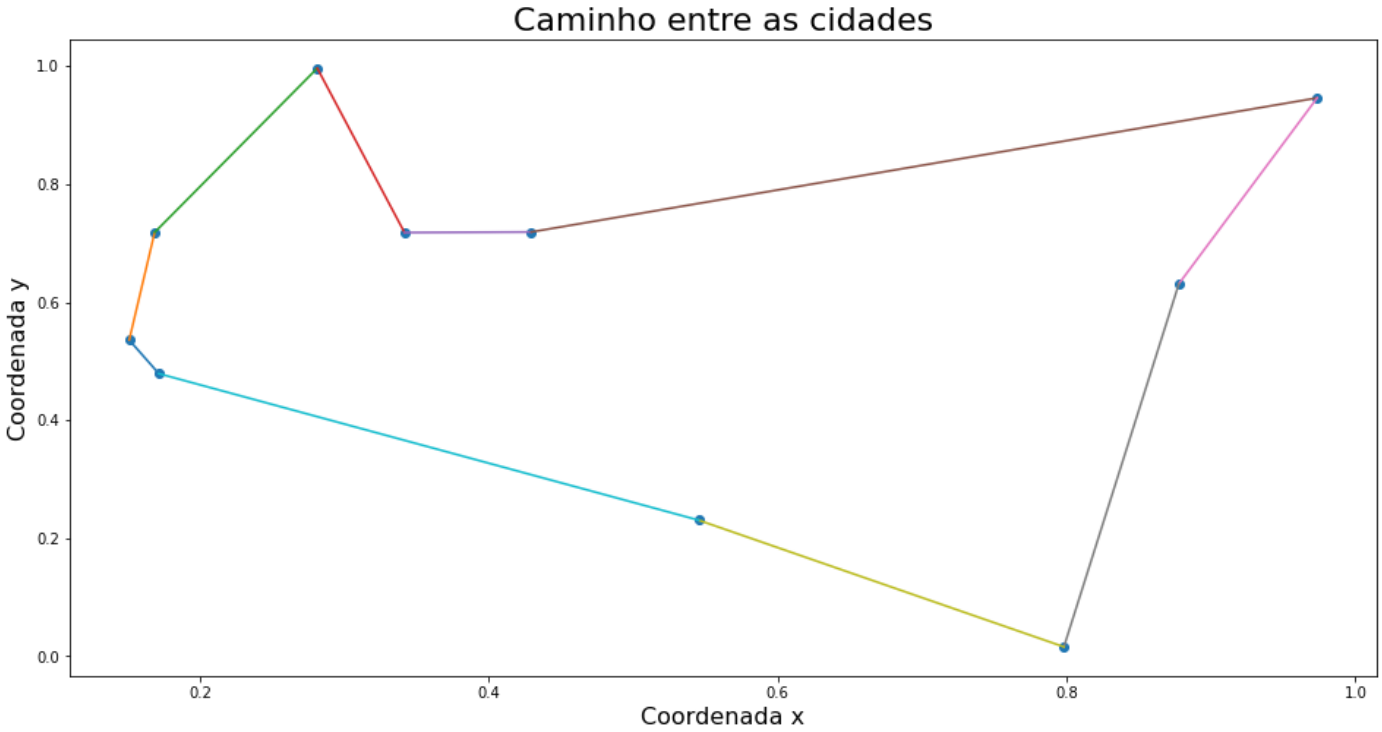
In [ ]:
```
caminhos_plot = gerar_caminhos_plot(caminhos)
for caminho in caminhos_plot:
  plotar_caminho(cidades, caminho)
```

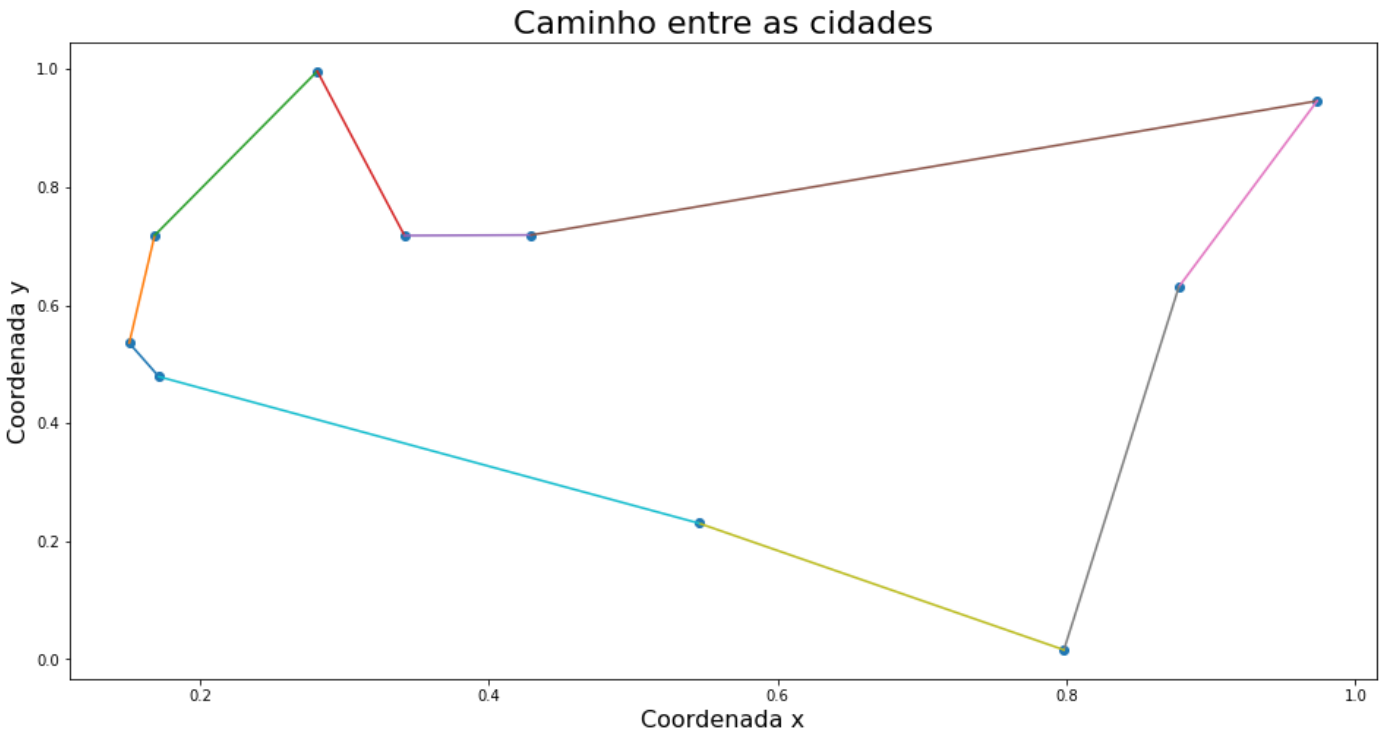



Caminho entre as cidades
Coordenada y
Coordenada x
0.0
0.2
0.4
0.6
0.8
1.0


Caminho entre as cidades
Coordenada y
Coordenada x
0.0
0.2
0.4
0.6
0.8
1.0

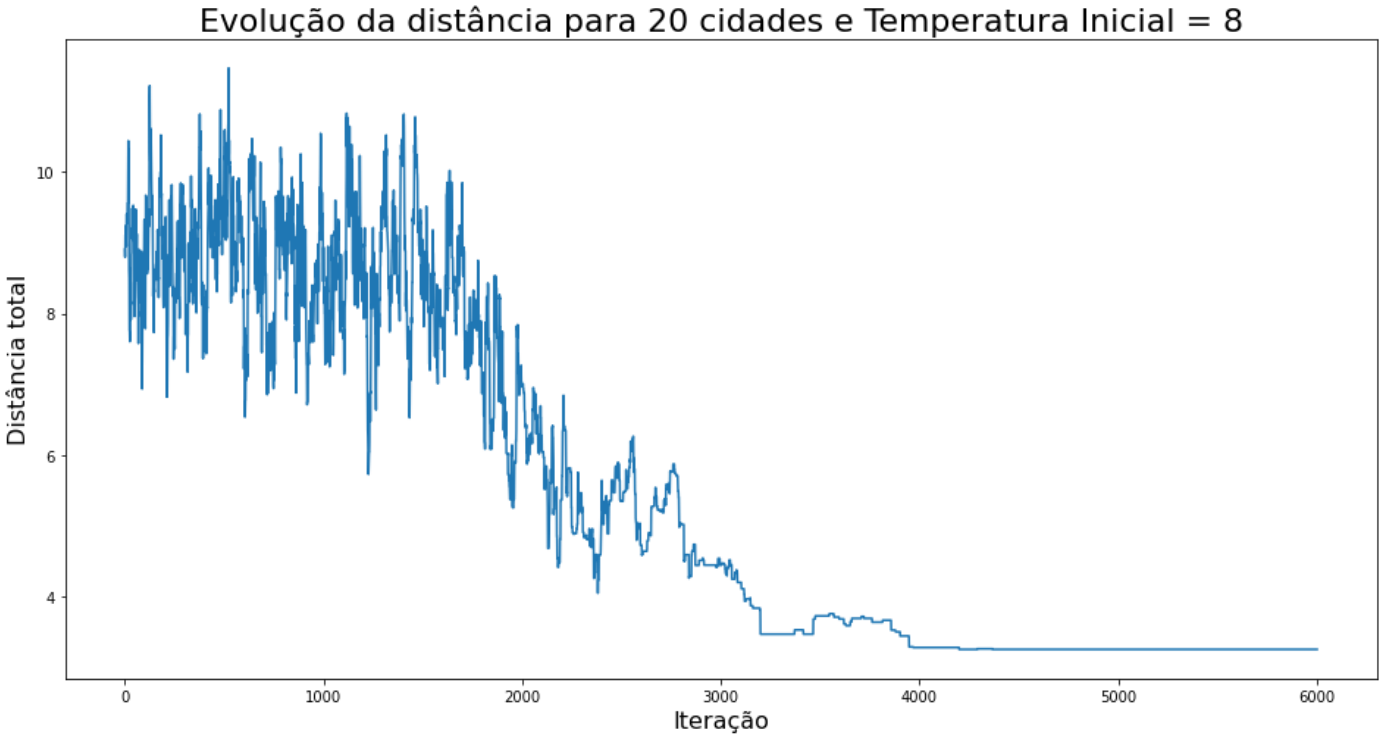

Caminho entre as cidades
Coordenada y
Coordenada x
0.0
0.2
0.4
0.6
0.8
1.0


Caminho entre as cidades
Coordenada y
Coordenada x
0.0
0.2
0.4
0.6
0.8
1.0

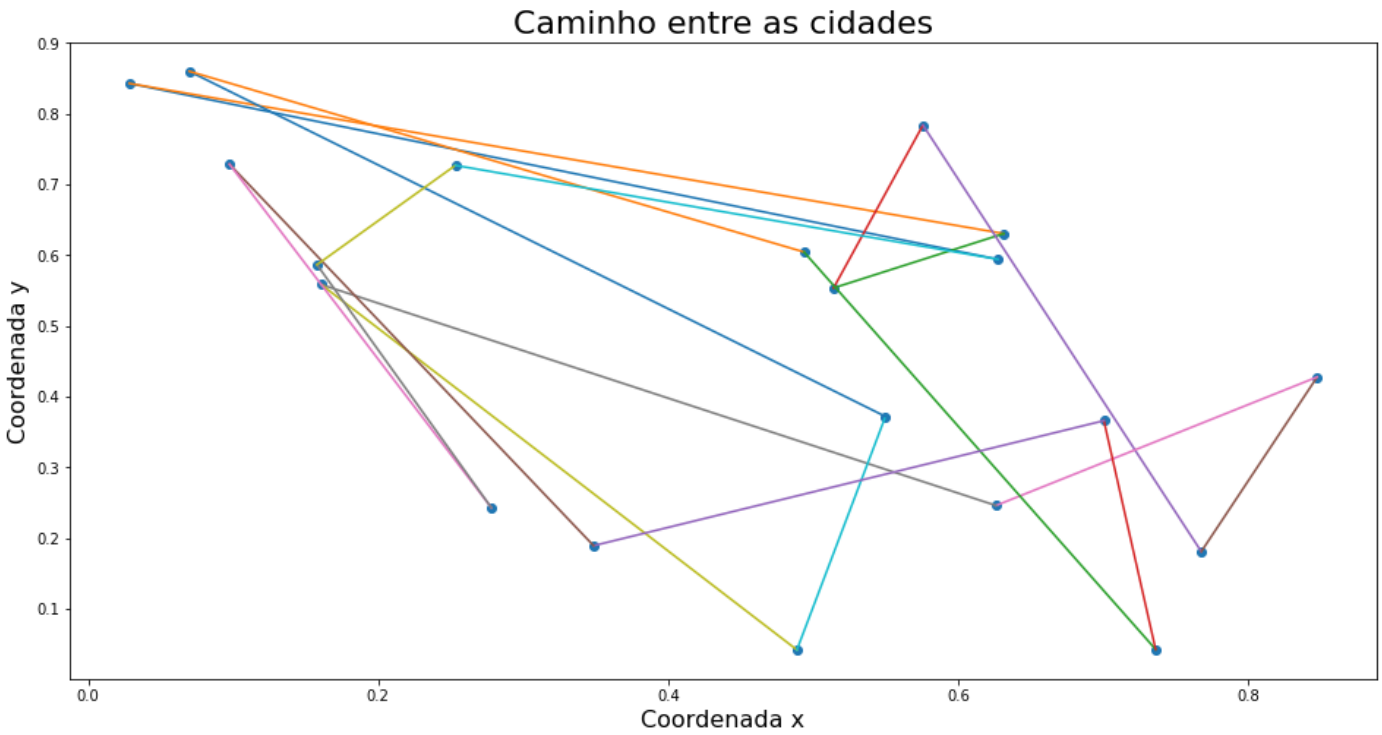


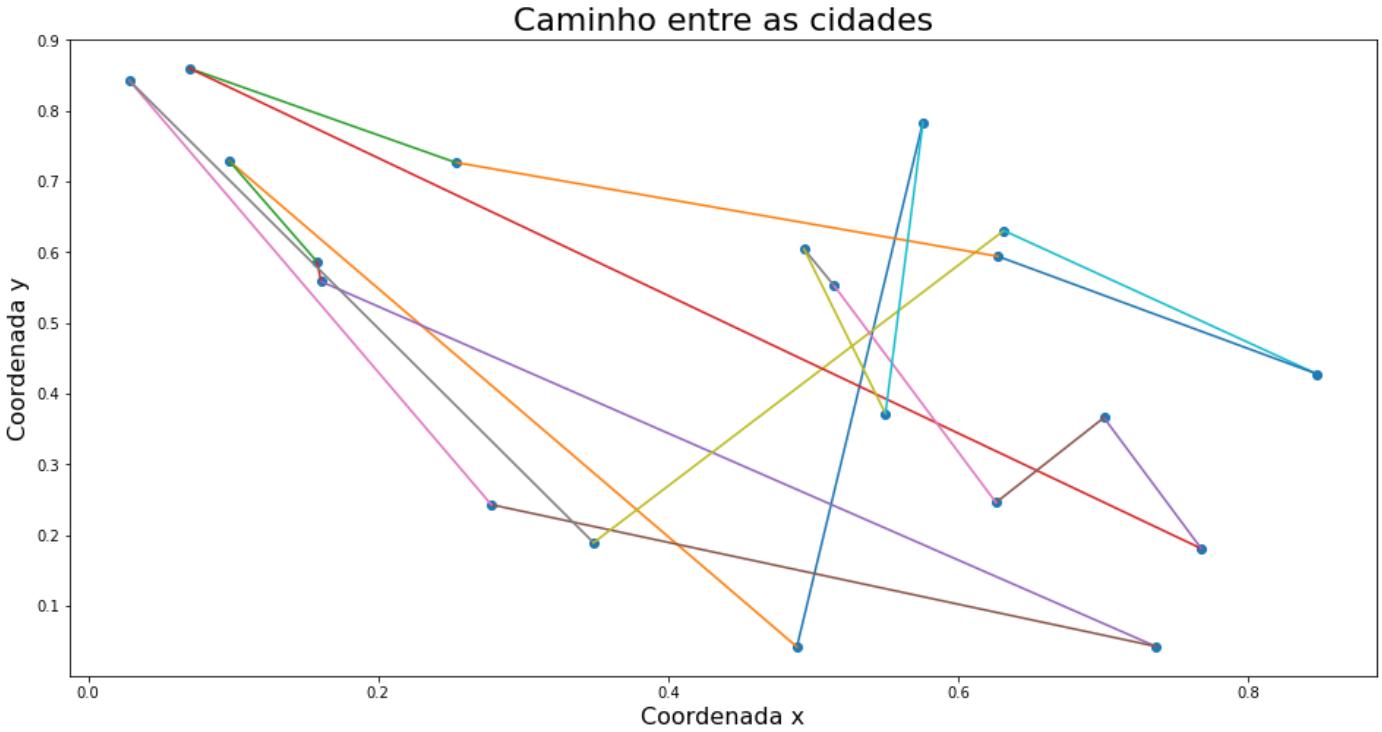
N = 20

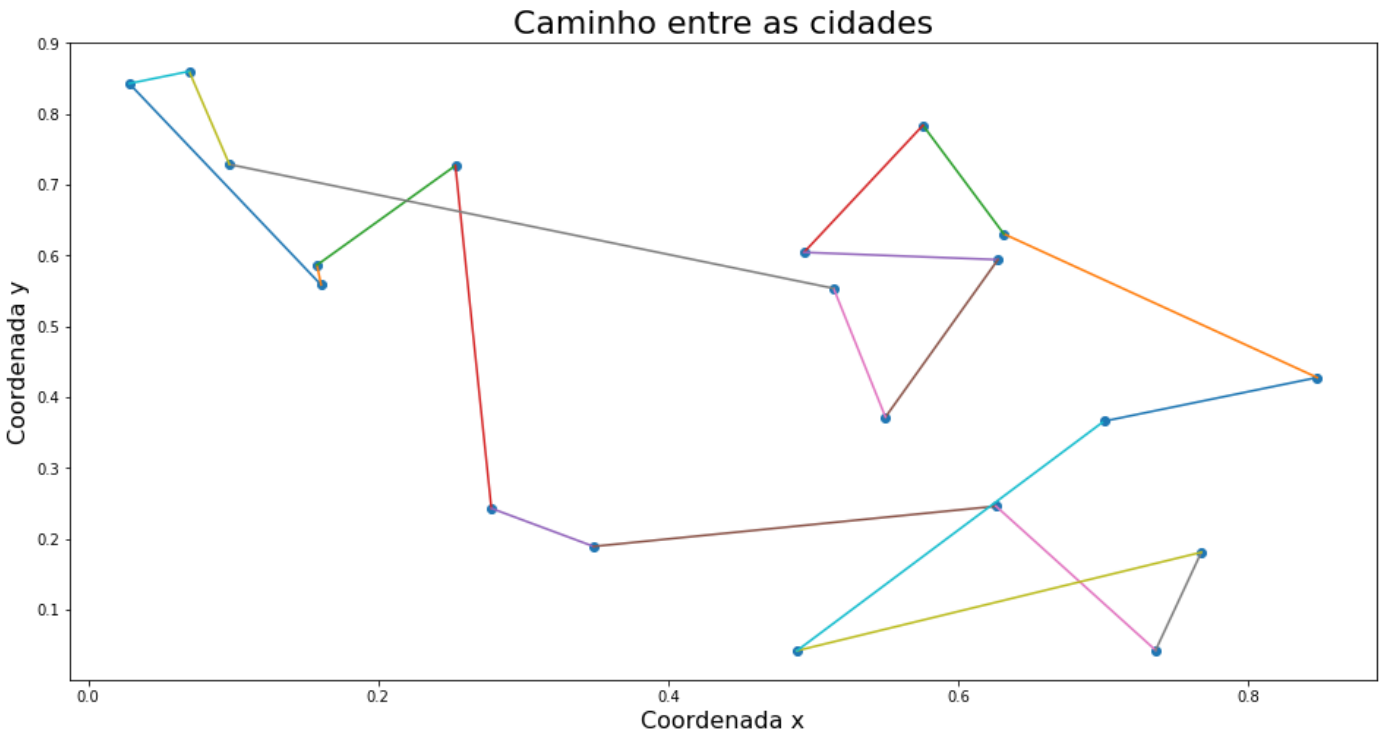
In [ ]:
```
N = 20
cidades, caminhos, distancias = caixeiro_viajante(N)
plotar_grafico_distancias(distancias, N)
```

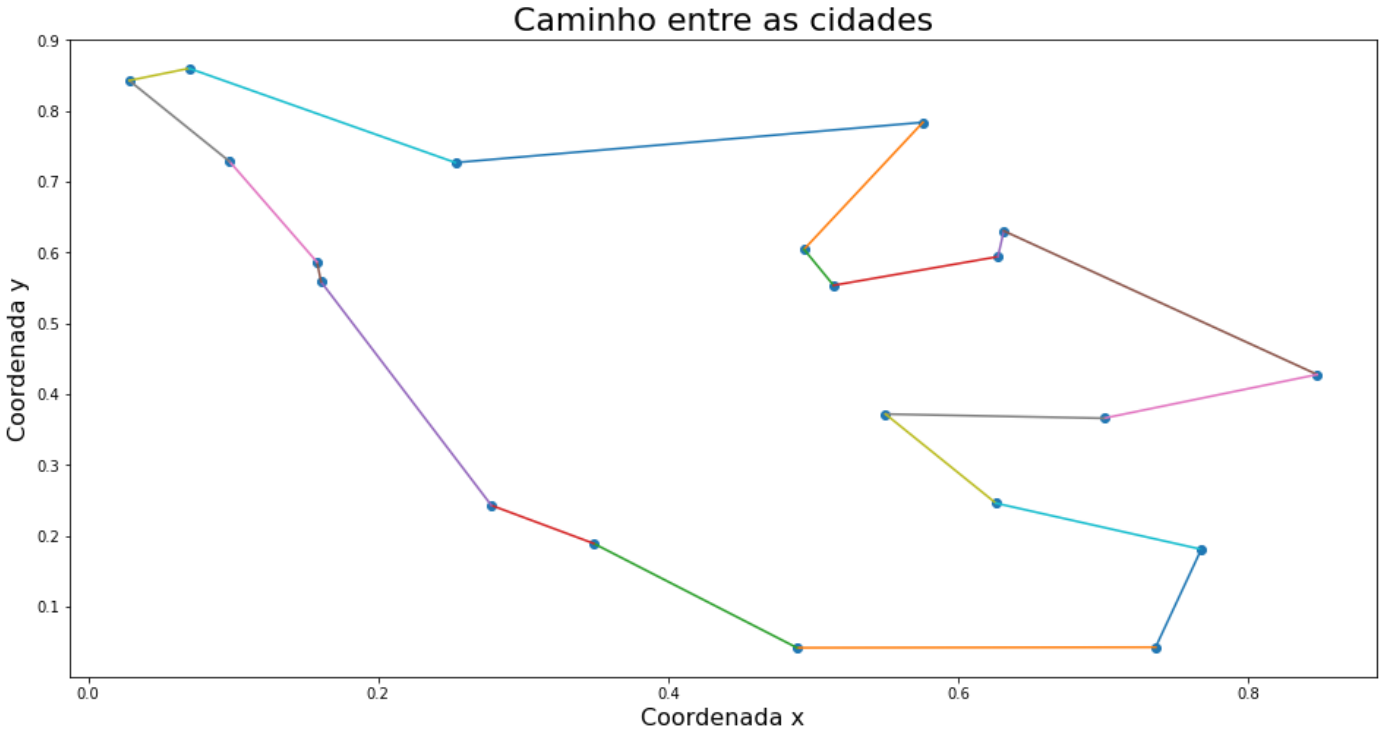


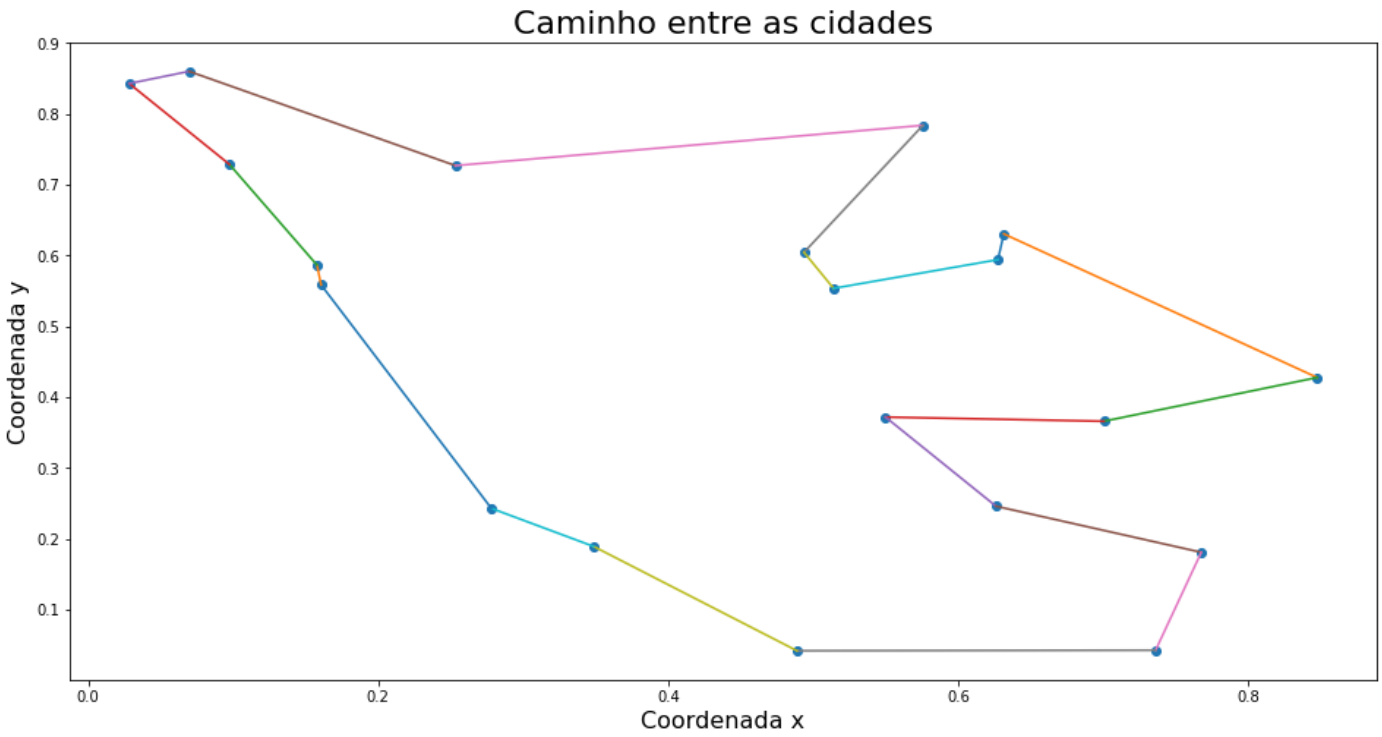
In [ ]:
```
caminhos_plot = gerar_caminhos_plot(caminhos)
for caminho in caminhos_plot:
  plotar_caminho(cidades, caminho)
```

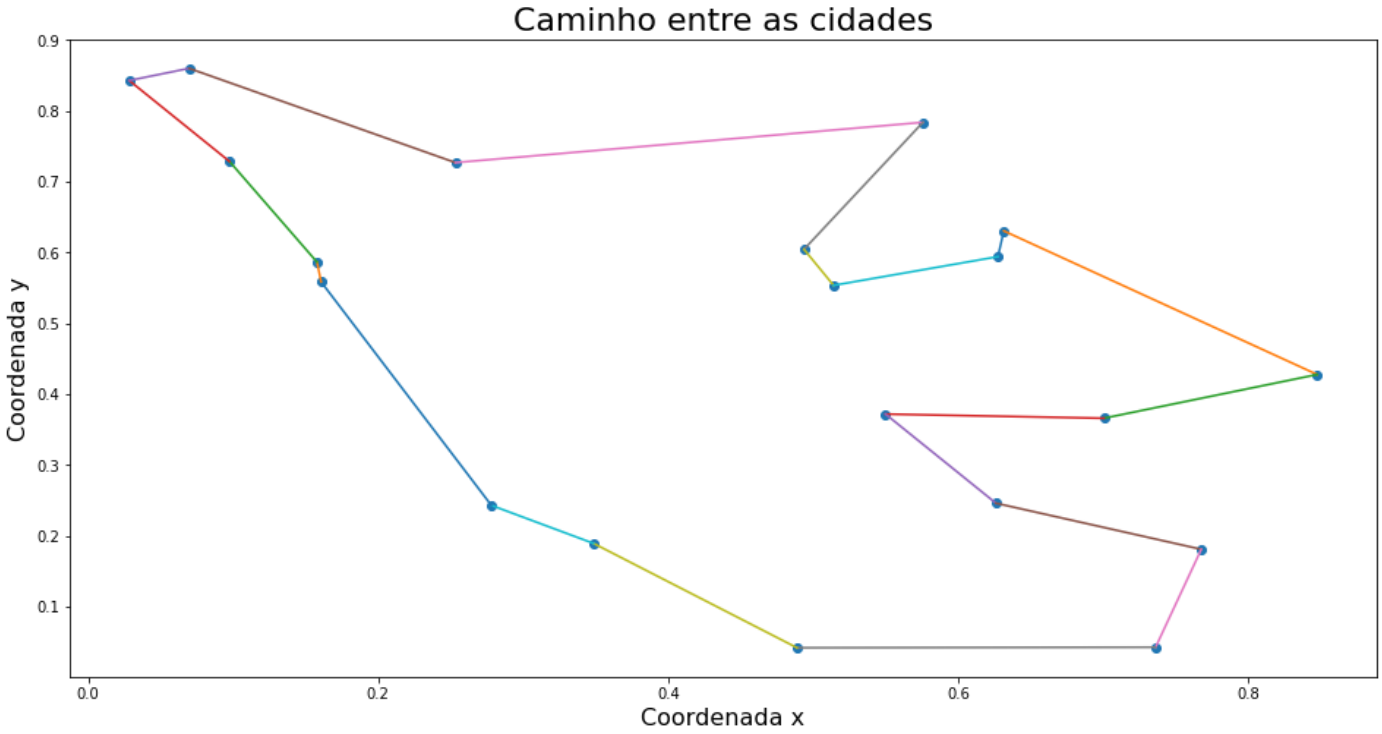

Caminho entre as cidades
Coordenada y
Coordenada x
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.2
0.4
0.6
0.8


Caminho entre as cidades
Coordenada y
Coordenada x
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.2
0.4
0.6
0.8


Caminho entre as cidades
Coordenada y
Coordenada x
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.2
0.4
0.6
0.8

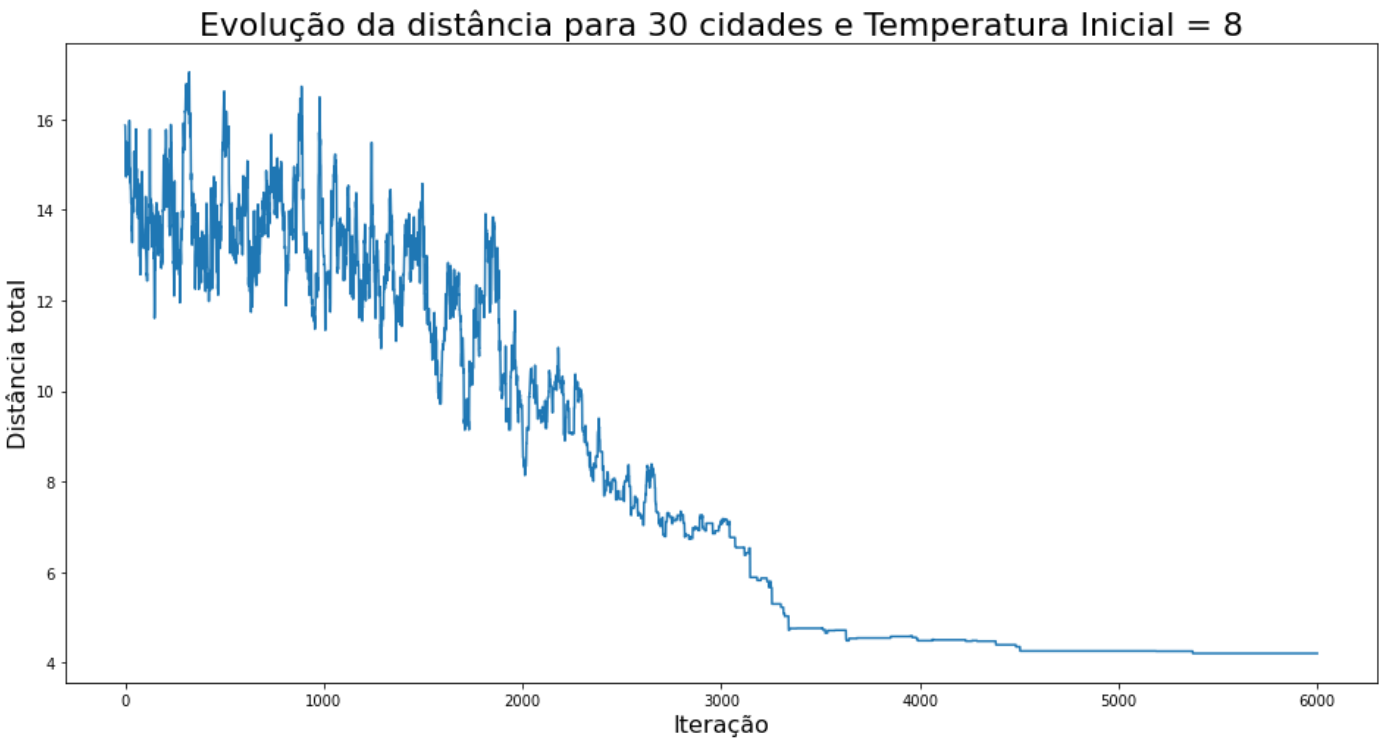

Caminho entre as cidades
Coordenada y
Coordenada x
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.2
0.4
0.6
0.8



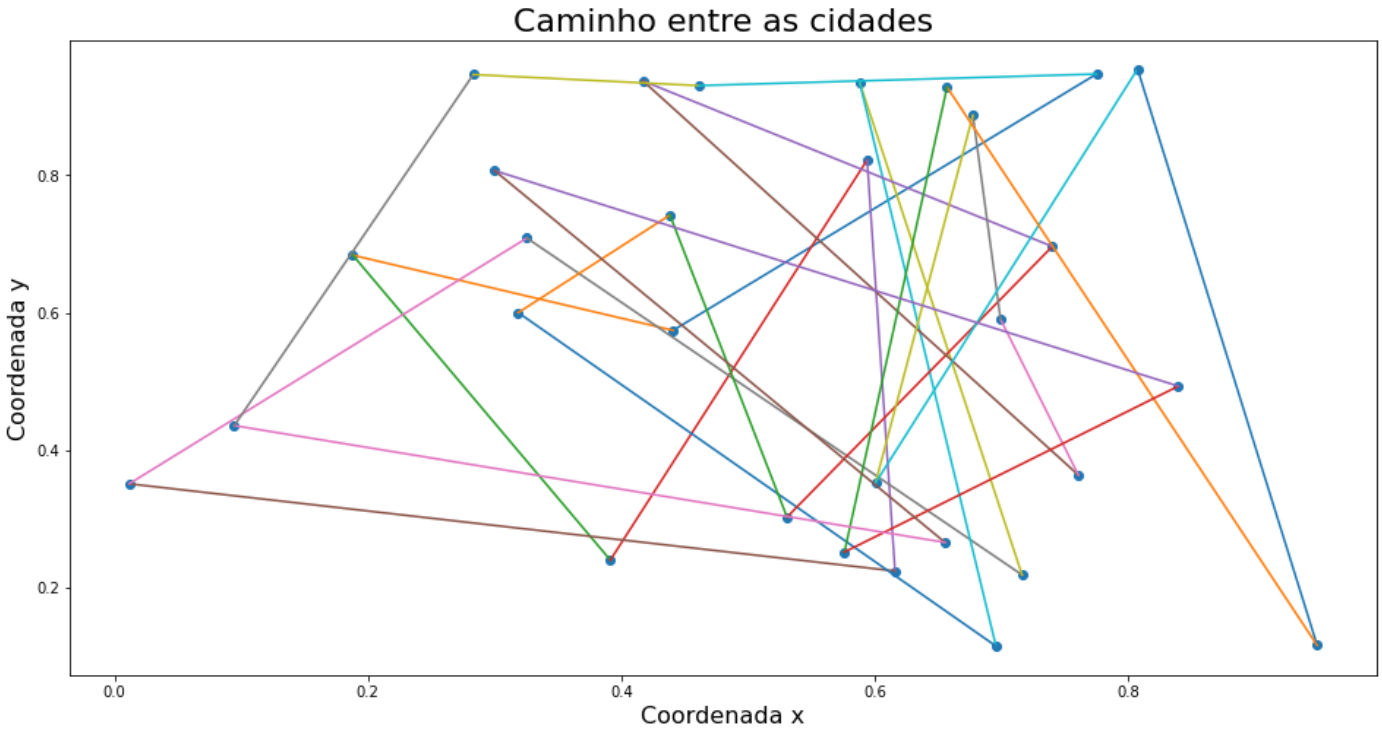


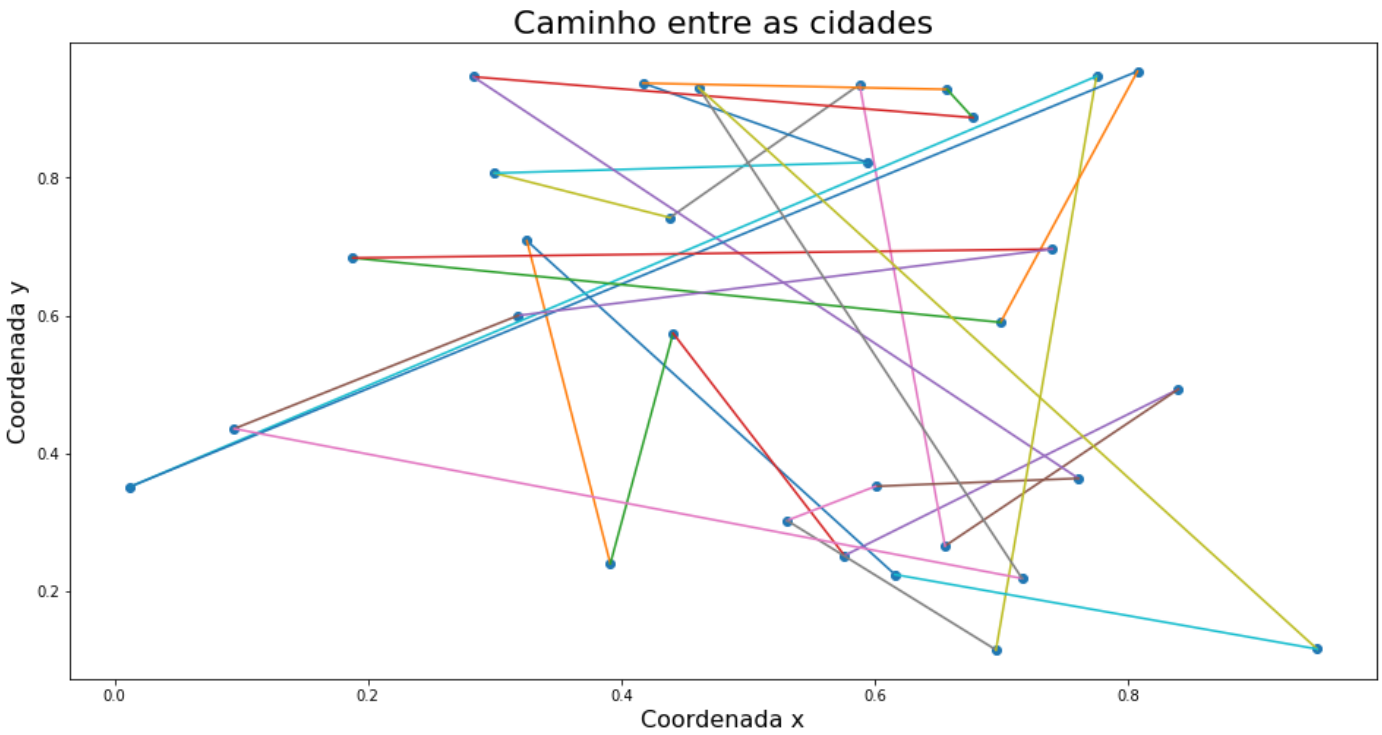
N = 30

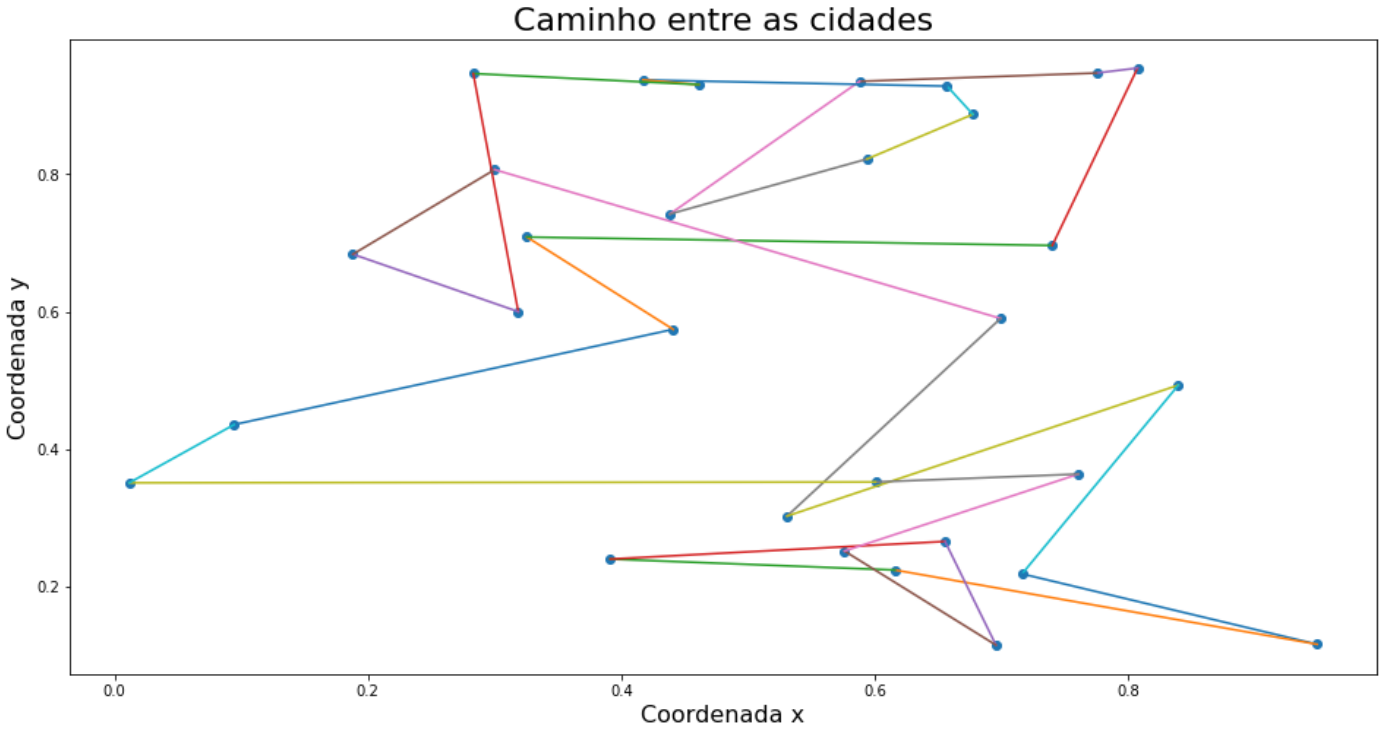
In [ ]:
```
N = 30
cidades, caminhos, distancias = caixeiro_viajante(N)
plotar_grafico_distancias(distancias, N)
```

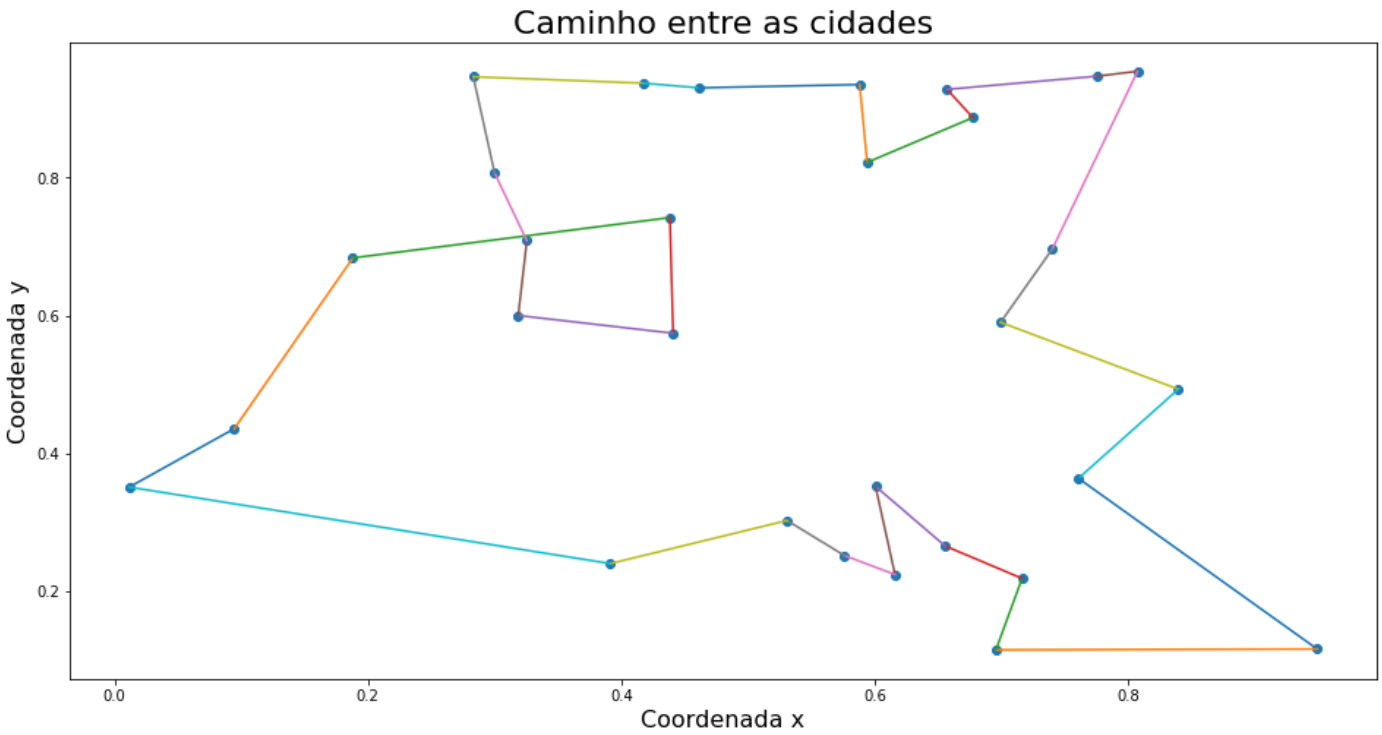


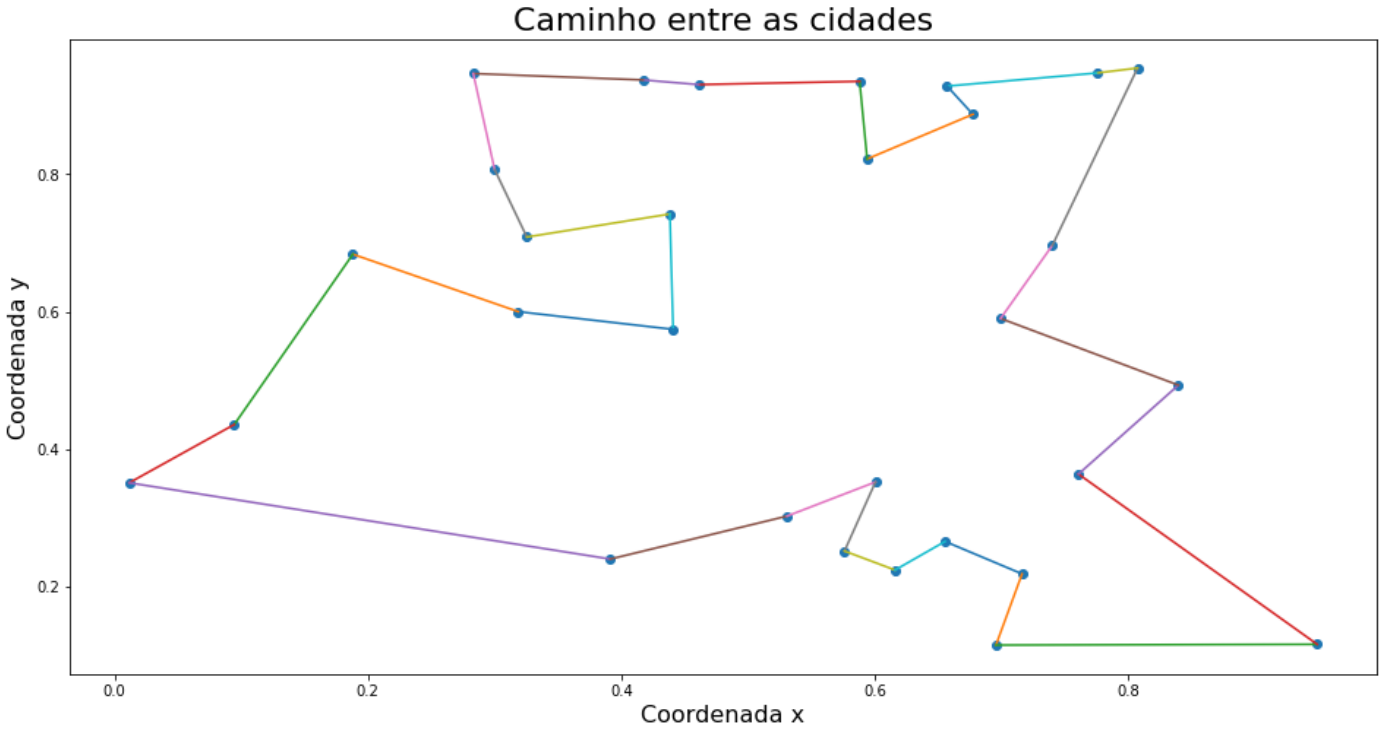
In [ ]:
```
caminhos_plot = gerar_caminhos_plot(caminhos)
for caminho in caminhos_plot:
  plotar_caminho(cidades, caminho)
```

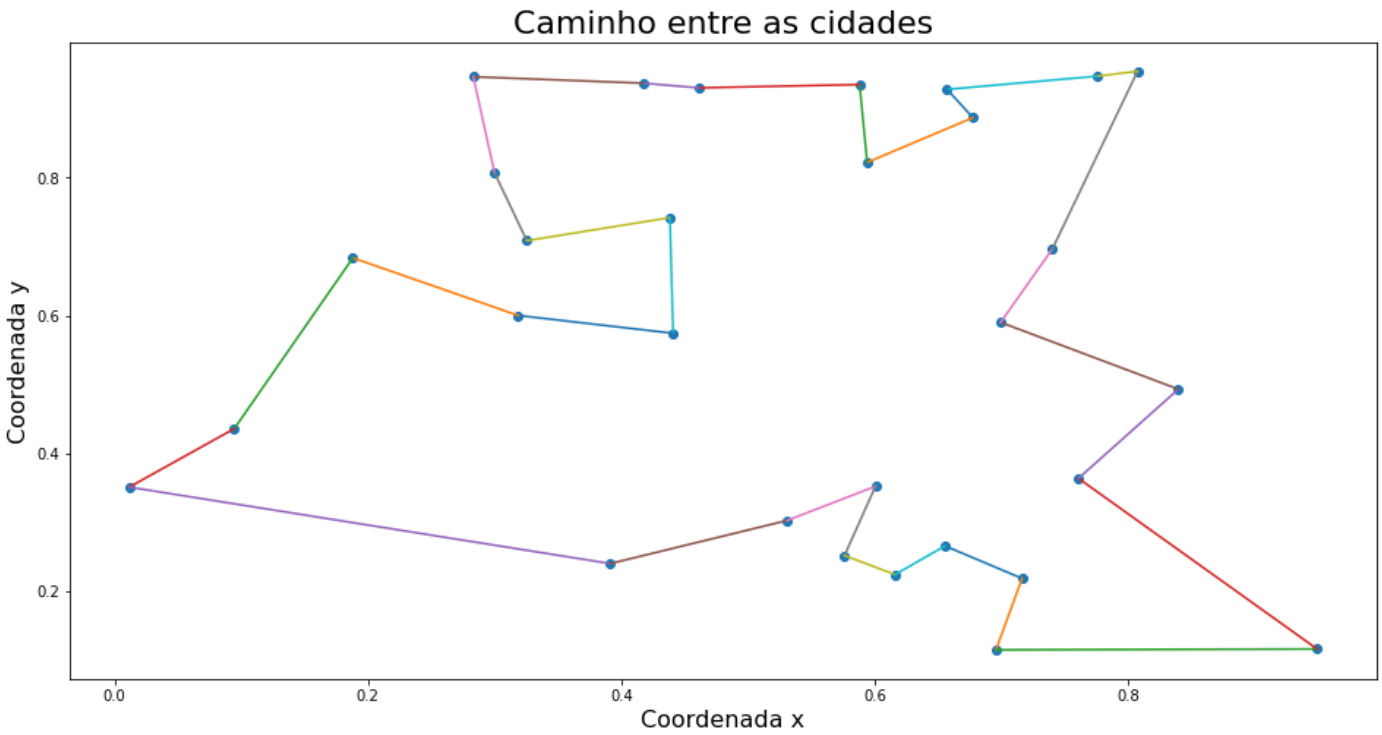



Caminho entre as cidades
Coordenada y
Coordenada x
0.0
0.2
0.4
0.6
0.8
0.2
0.4
0.6
0.8


Caminho entre as cidades
Coordenada y
Coordenada x
0.0
0.2
0.4
0.6
0.8
0.2
0.4
0.6
0.8

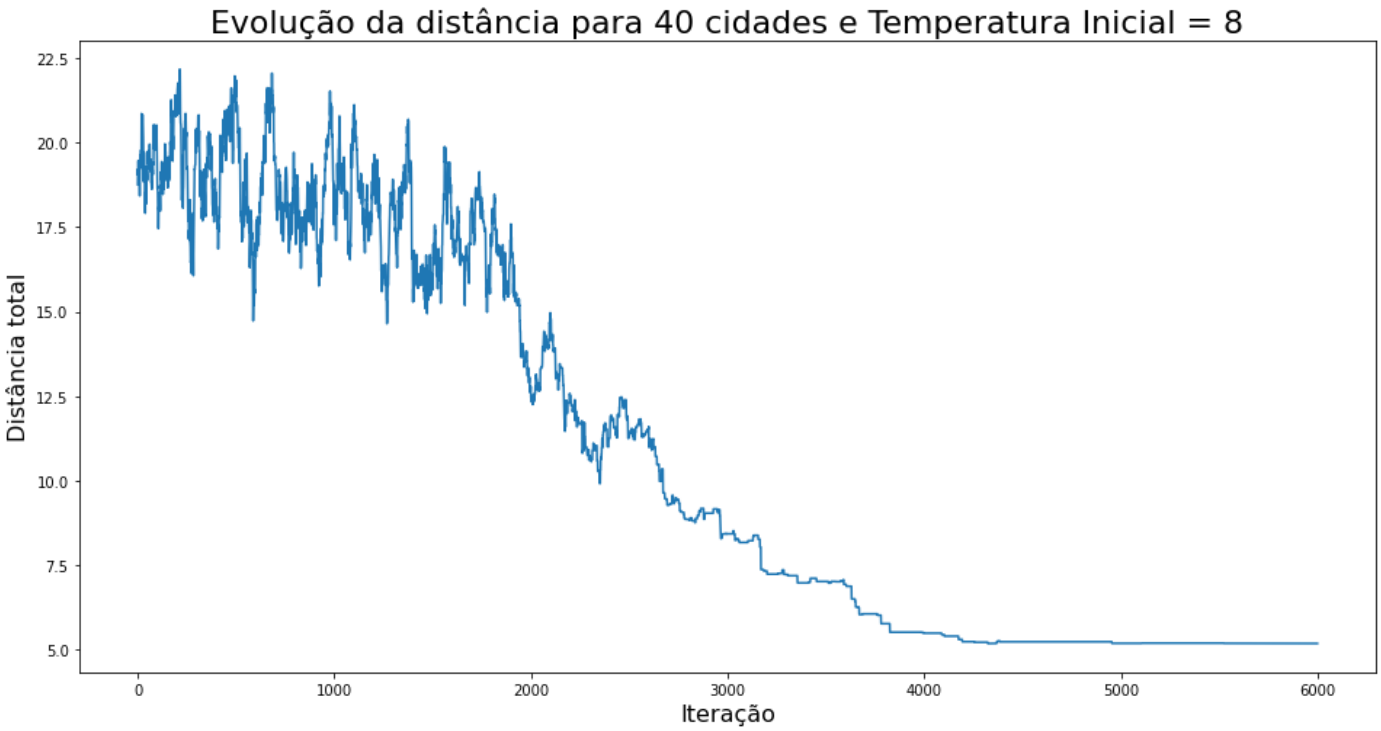

Caminho entre as cidades
Coordenada y
Coordenada x
0.0
0.2
0.4
0.6
0.8


Caminho entre as cidades
Coordenada y
Coordenada x
0.0
0.2
0.4
0.6
0.8

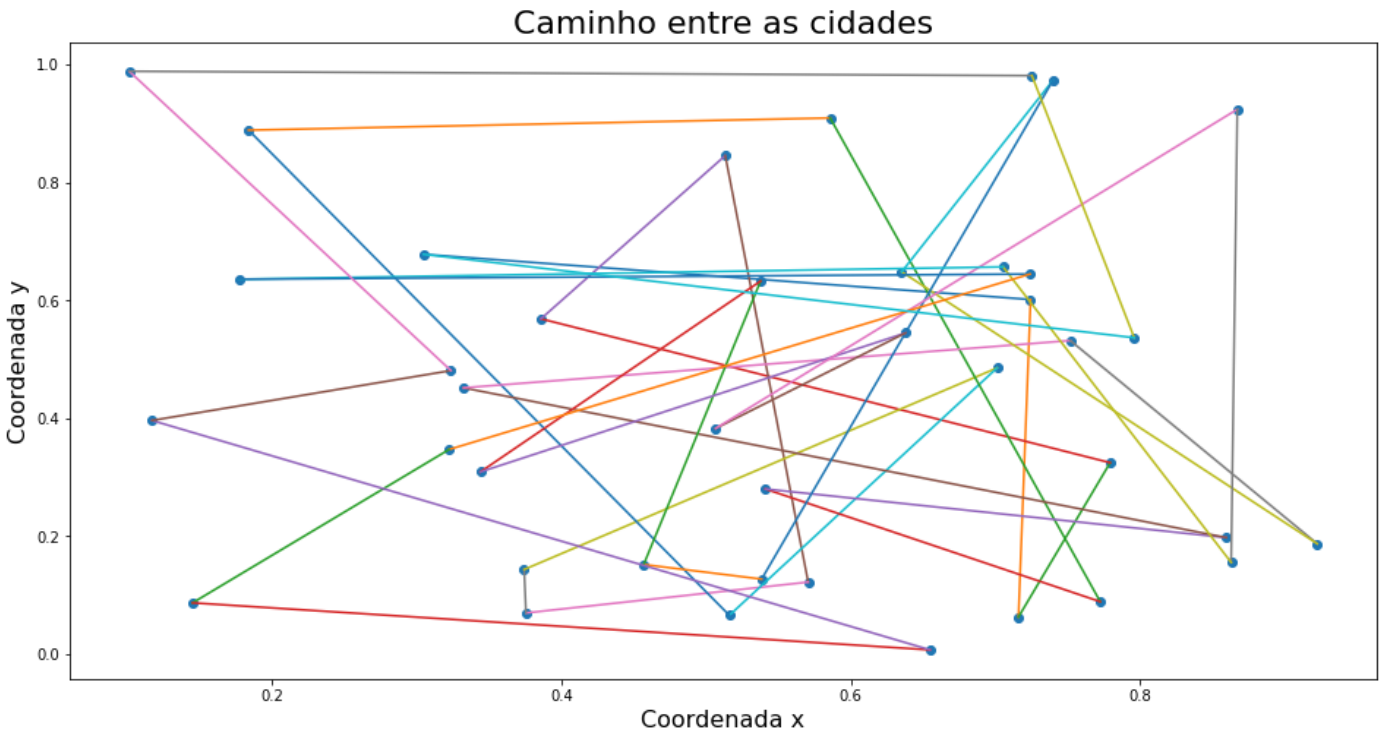


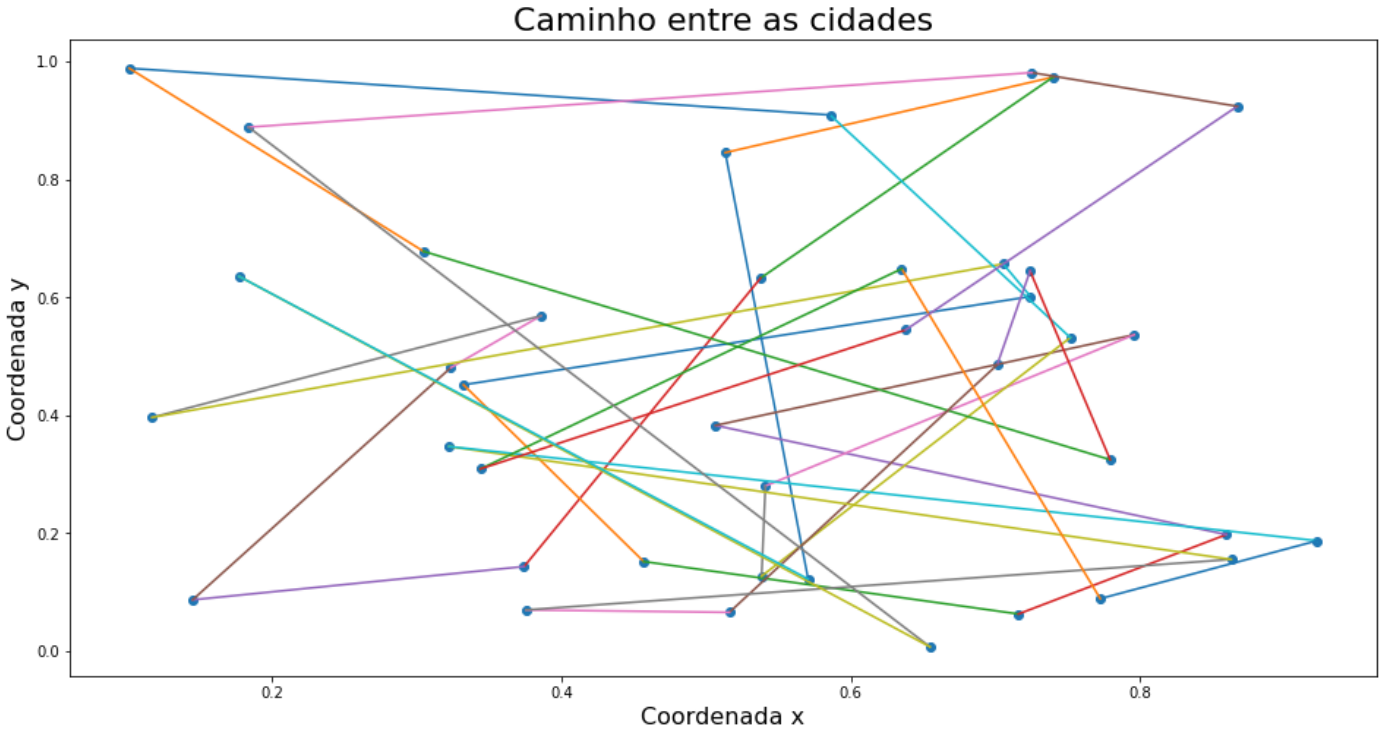
N = 40

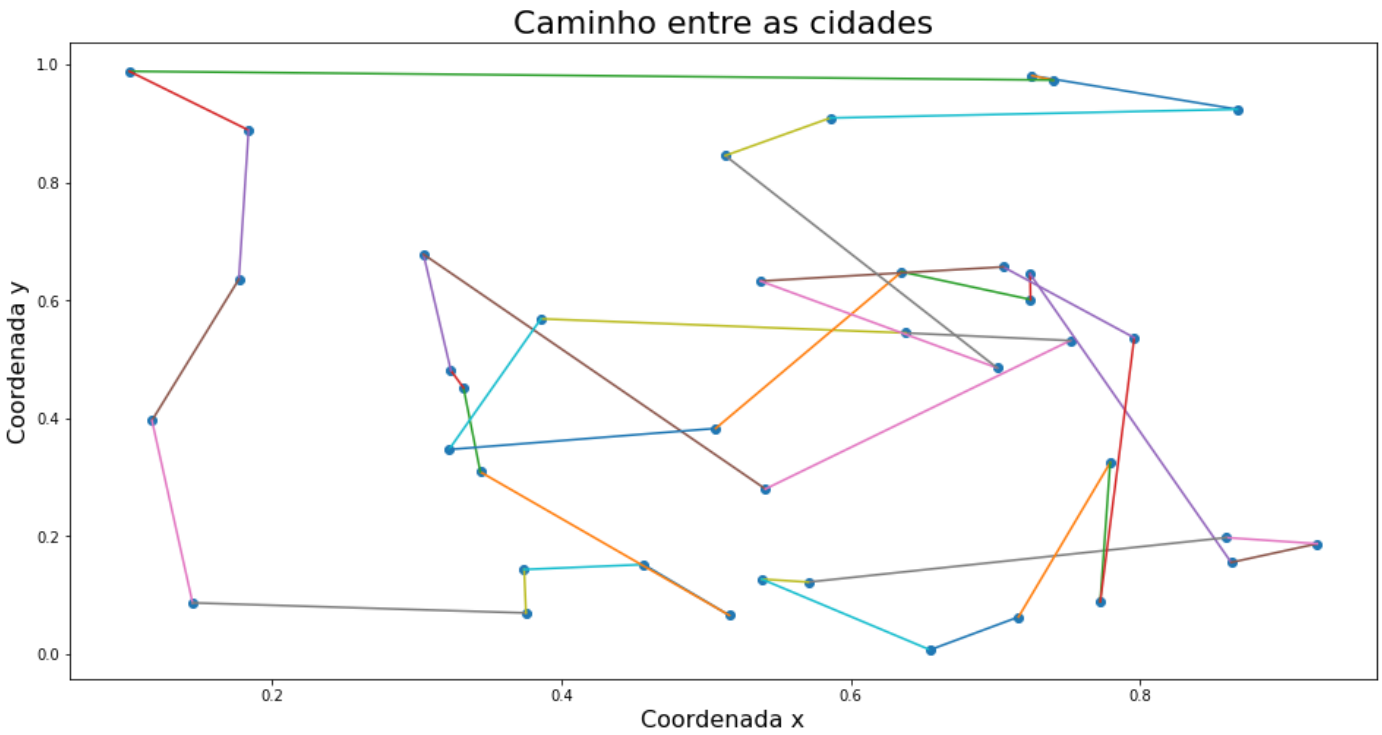
In [ ]:
```
N = 40
cidades, caminhos, distancias = caixeiro_viajante(N)
plotar_grafico_distancias(distancias, N)
```

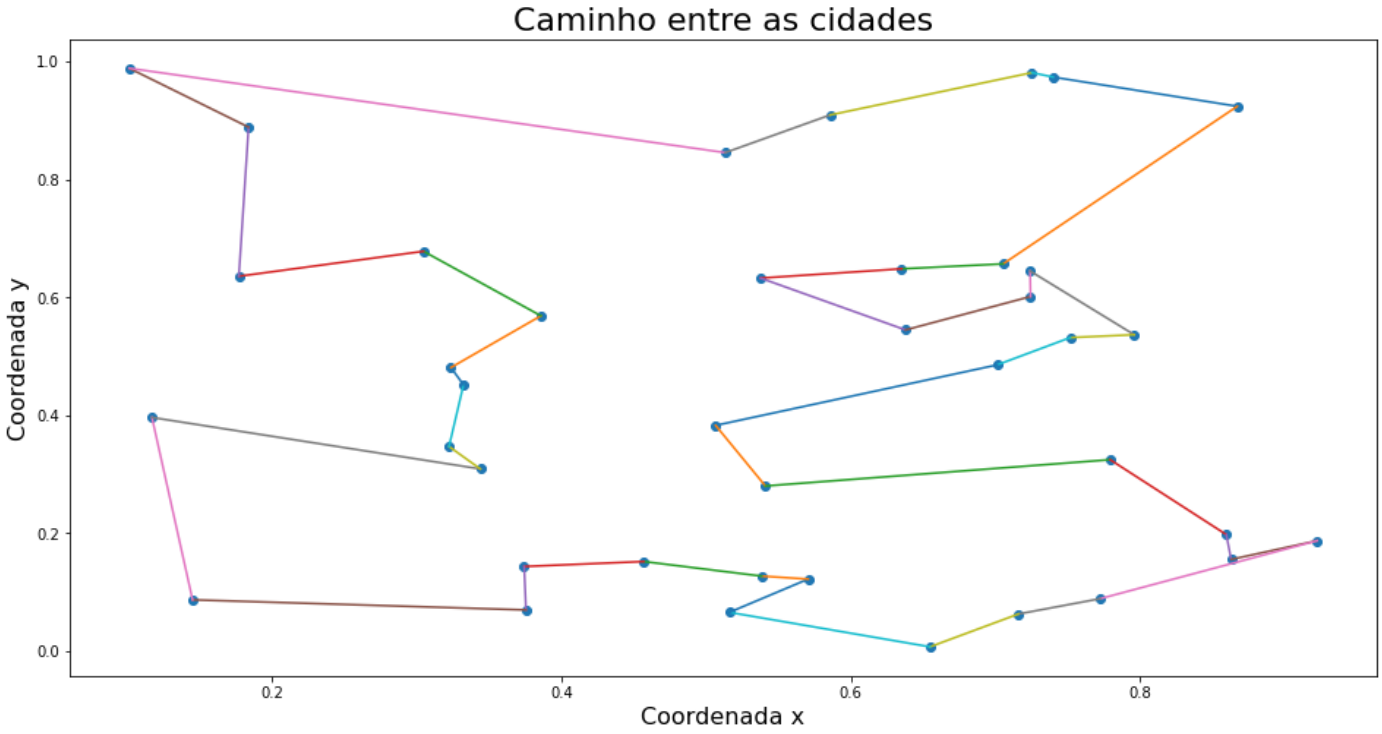


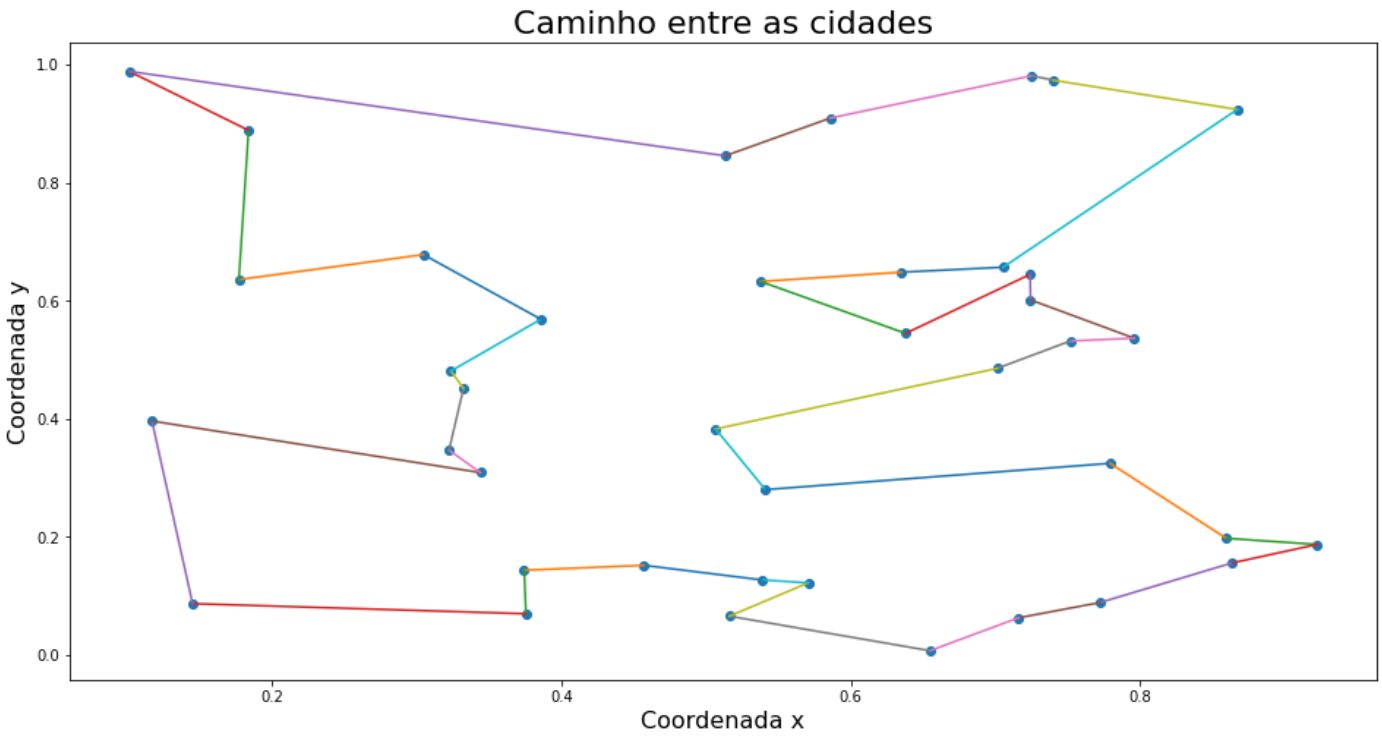
In [ ]:
```
caminhos_plot = gerar_caminhos_plot(caminhos)
for caminho in caminhos_plot:
  plotar_caminho(cidades, caminho)
```

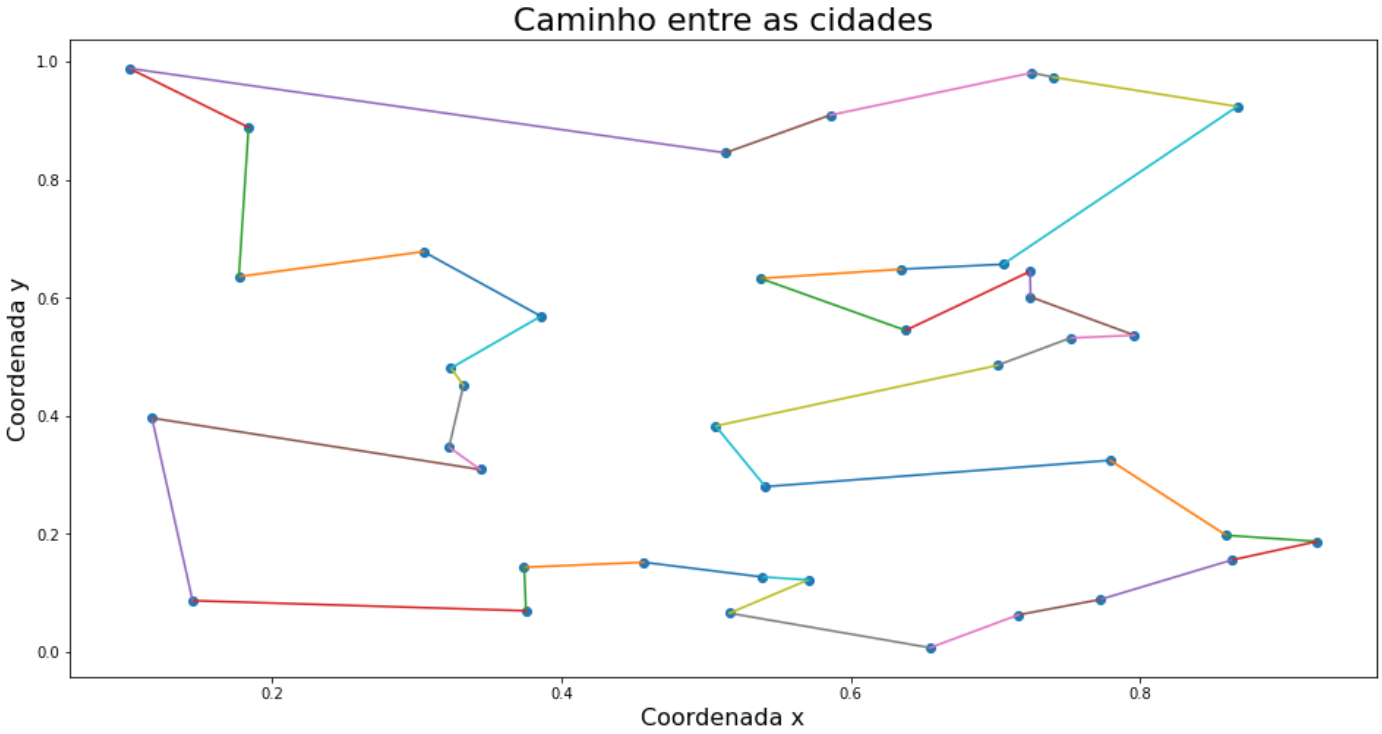

Caminho entre as cidades
Coordenada y
Coordenada x
0.0
0.2
0.4
0.6
0.8
1.0
0.2
0.4
0.6
0.8


Caminho entre as cidades
Coordenada y
Coordenada x
0.0
0.2
0.4
0.6
0.8
1.0
0.2
0.4
0.6
0.8


Caminho entre as cidades
Coordenada y
Coordenada x
0.0
0.2
0.4
0.6
0.8
1.0
0.2
0.4
0.6
0.8

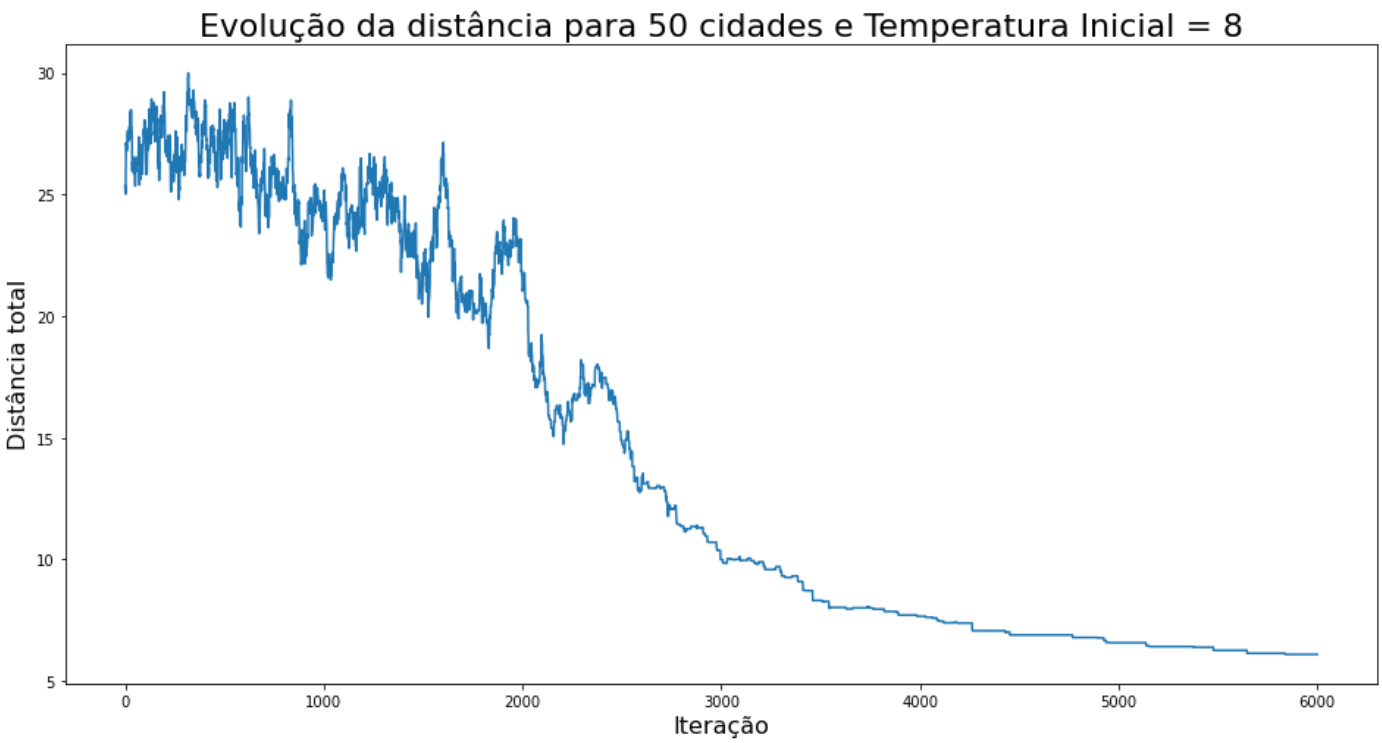

Caminho entre as cidades
Coordenada y
Coordenada x
0.0
0.2
0.4
0.6
0.8
1.0
0.2
0.4
0.6
0.8



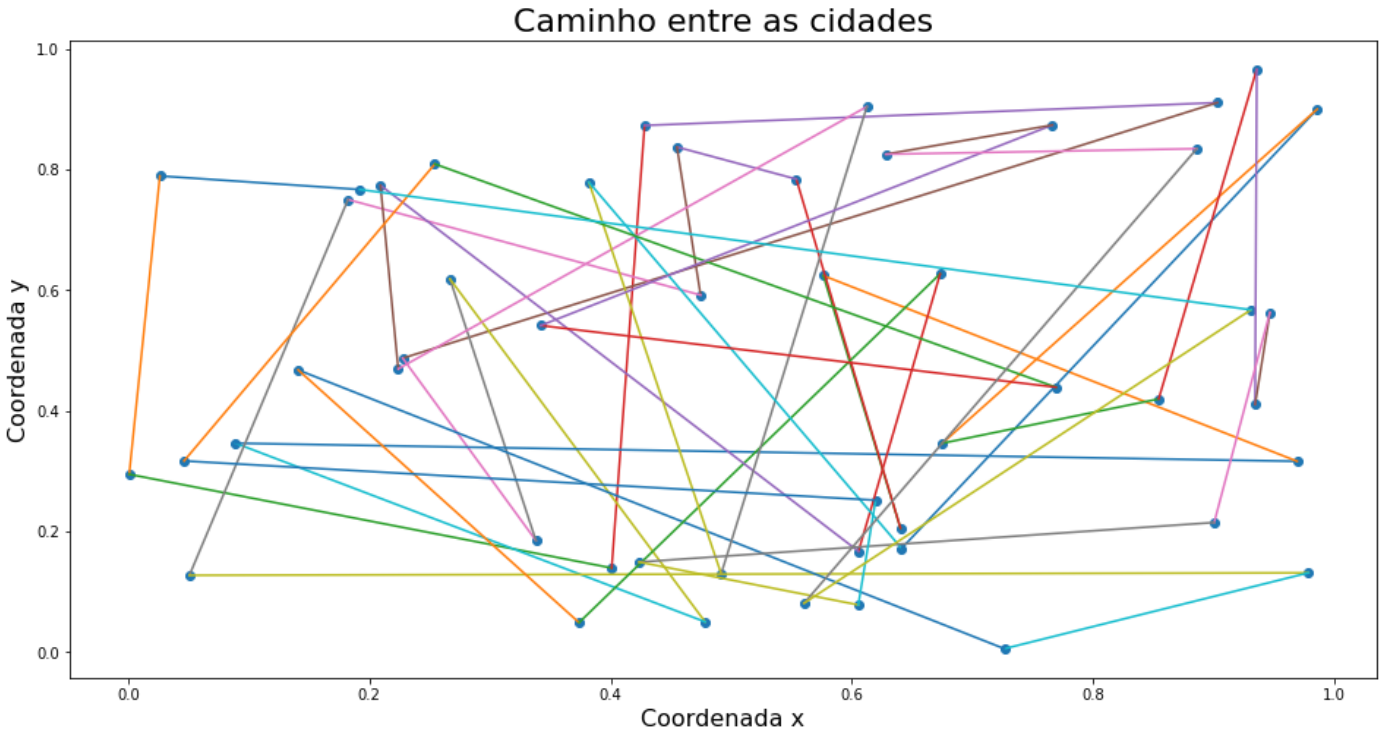


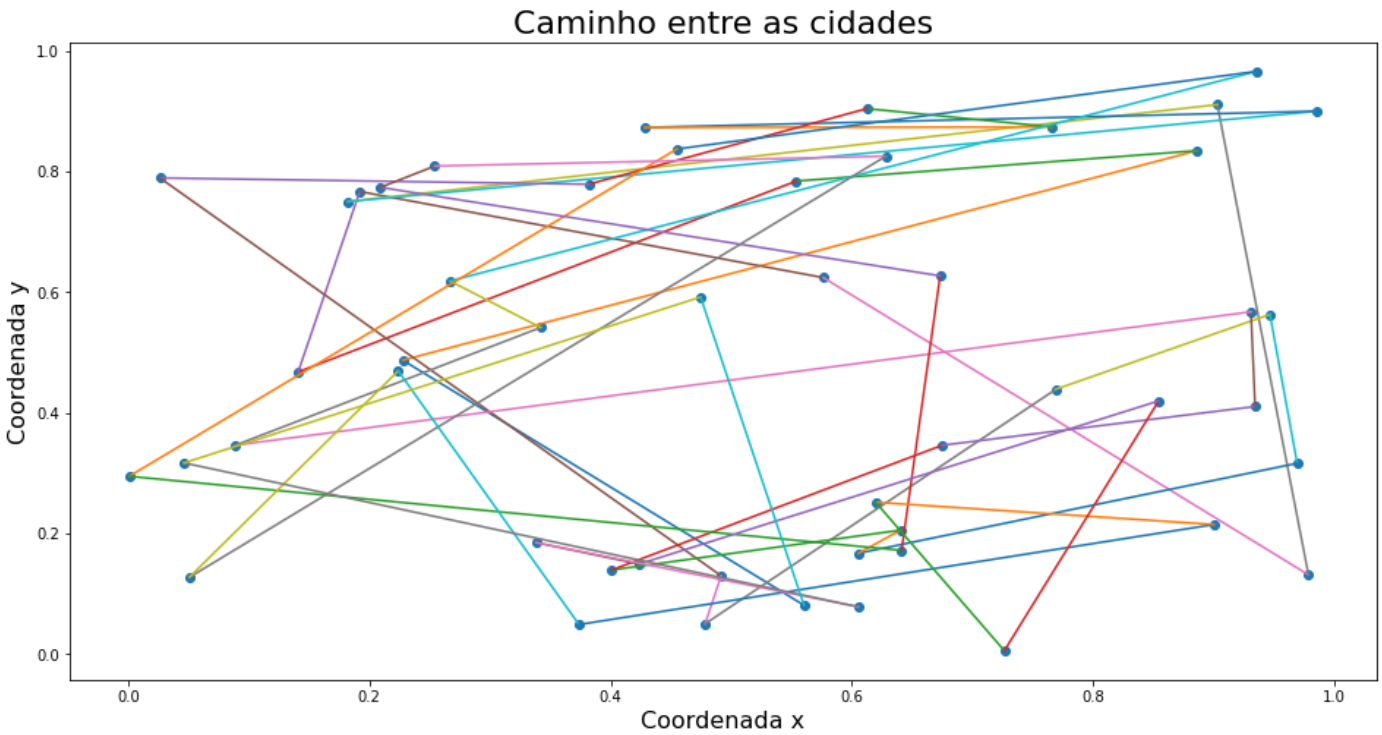
N = 50

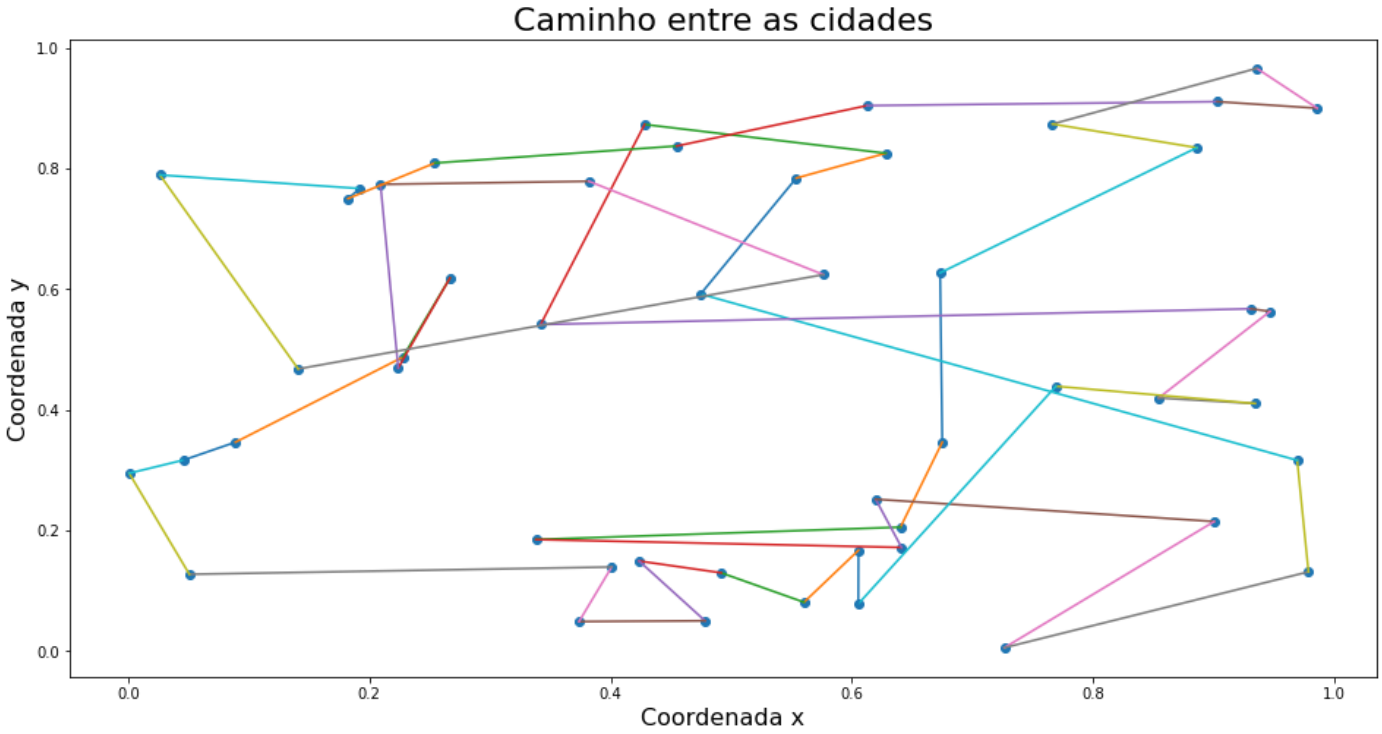
In [ ]:
```
N = 50
cidades, caminhos, distancias = caixeiro_viajante(N)
plotar_grafico_distancias(distancias, N)
```

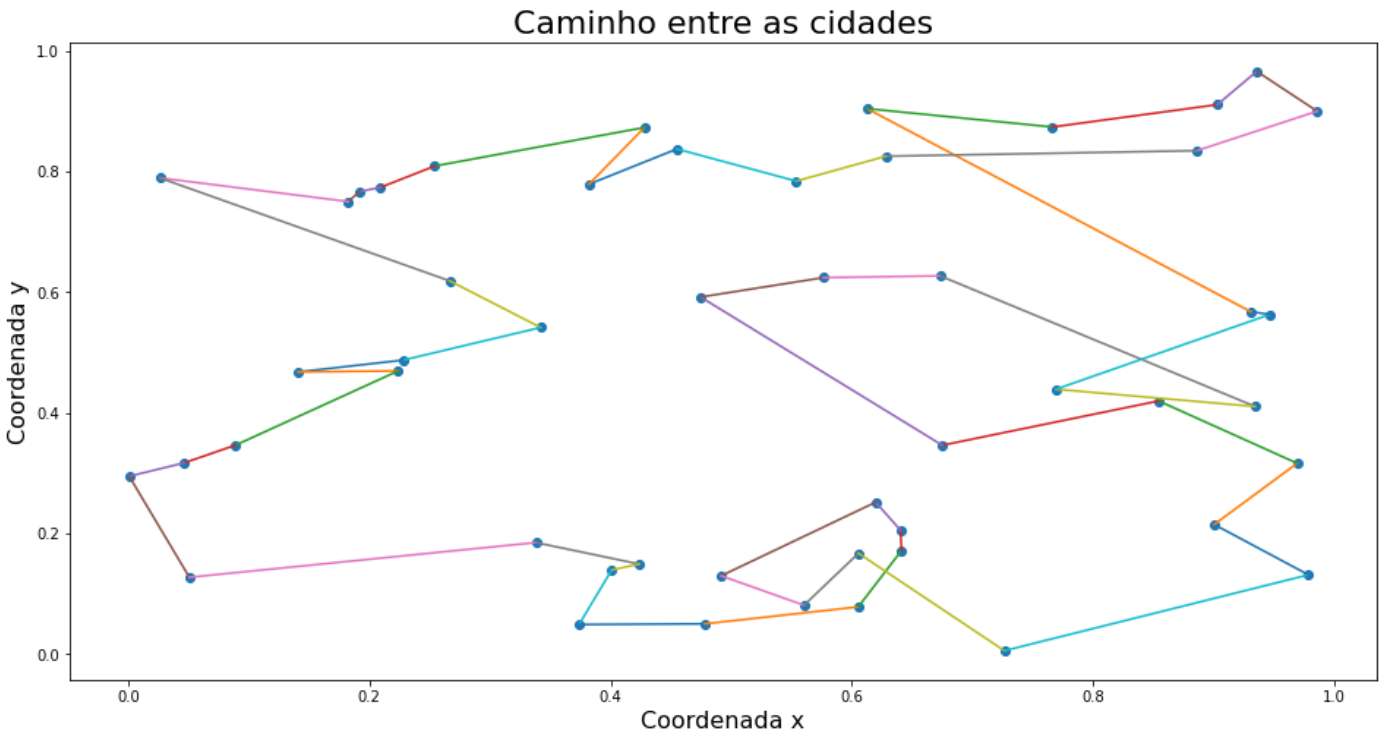


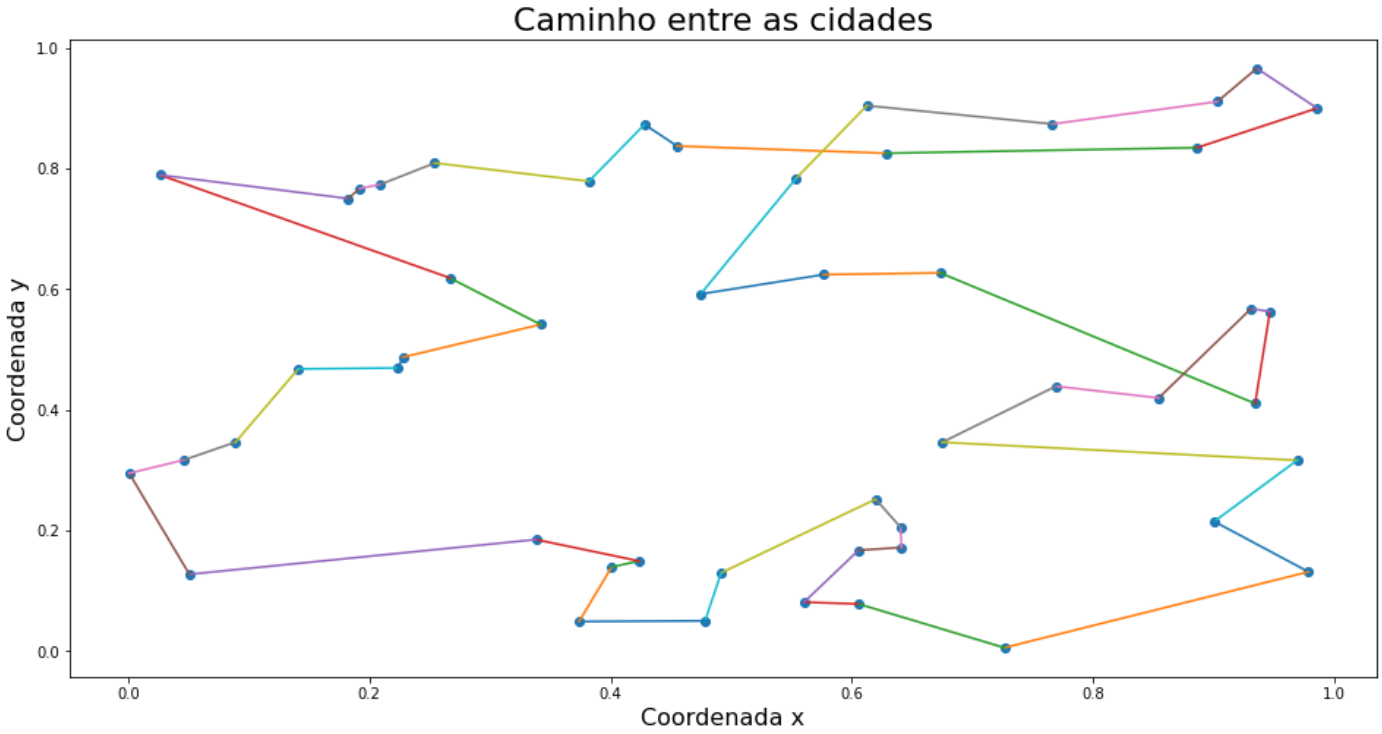
In [ ]:
```
caminhos_plot = gerar_caminhos_plot(caminhos)
for caminho in caminhos_plot:
  plotar_caminho(cidades, caminho)
```

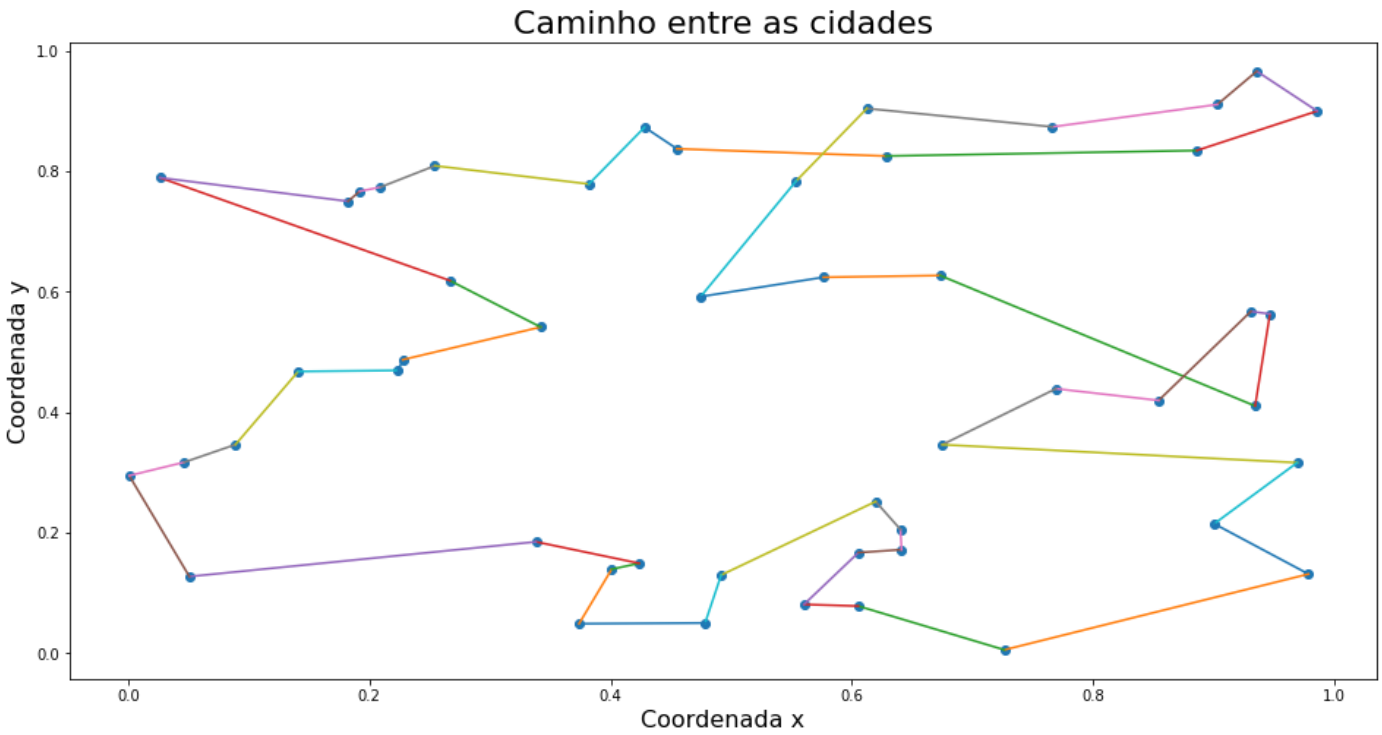



Caminho entre as cidades
Coordenada y
Coordenada x
0.0
0.2
0.4
0.6
0.8
1.0


Caminho entre as cidades
Coordenada y
Coordenada x
0.0
0.2
0.4
0.6
0.8
1.0

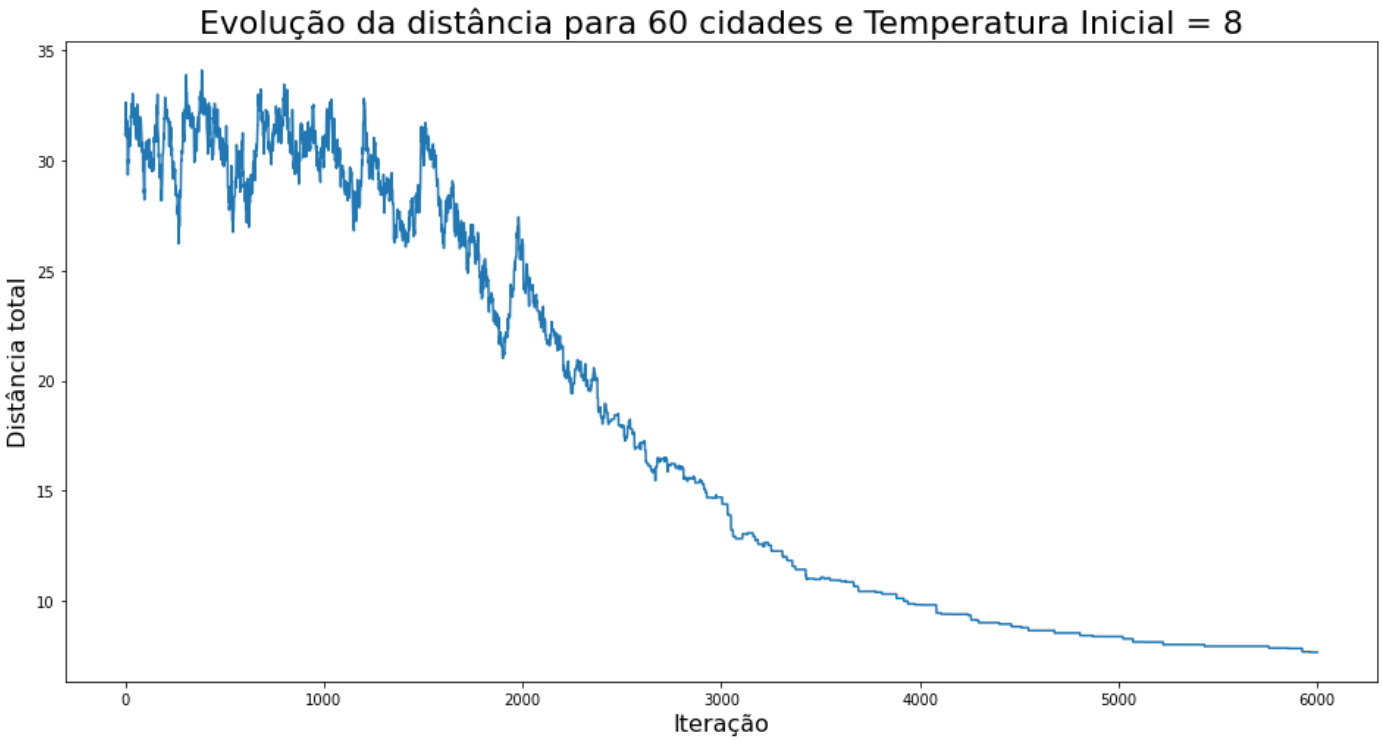

Caminho entre as cidades
Coordenada y
Coordenada x
0.0
0.2
0.4
0.6
0.8
1.0


Caminho entre as cidades
Coordenada y
Coordenada x
0.0
0.2
0.4
0.6
0.8
1.0

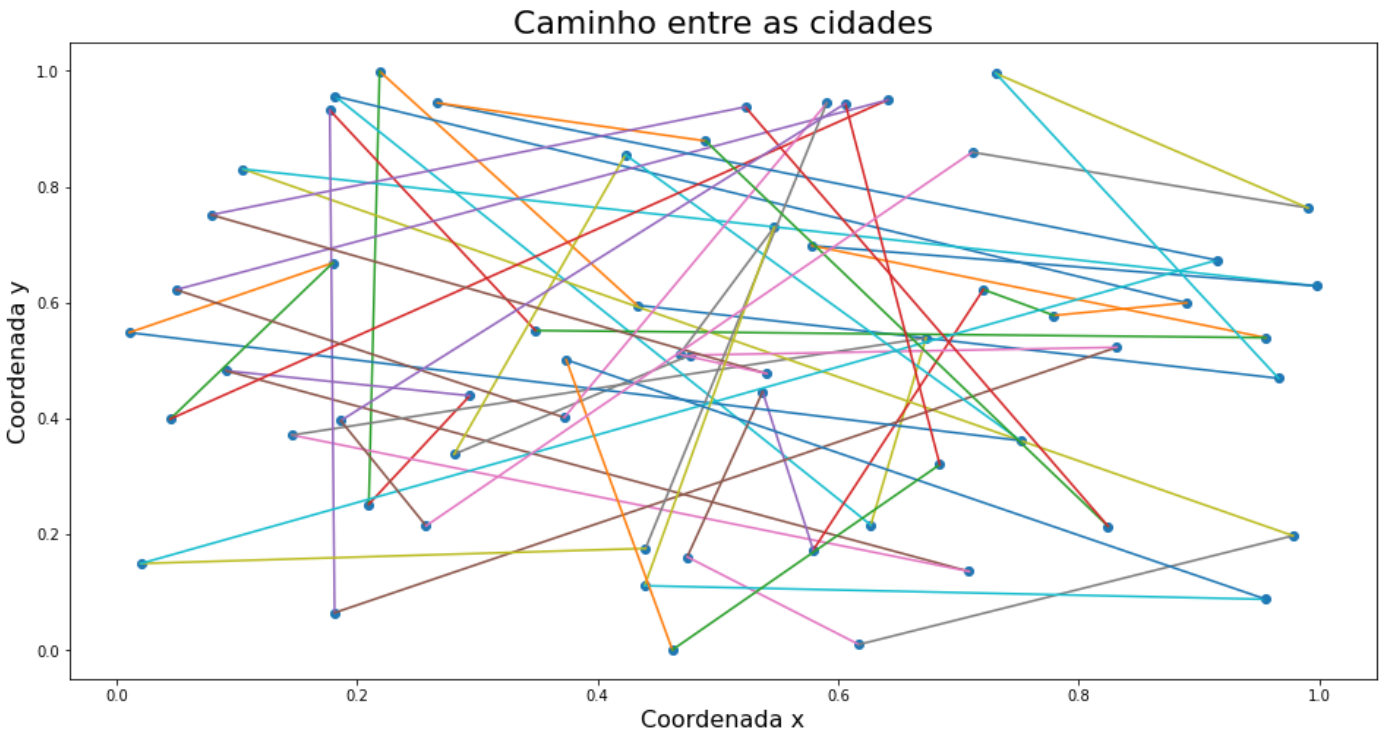


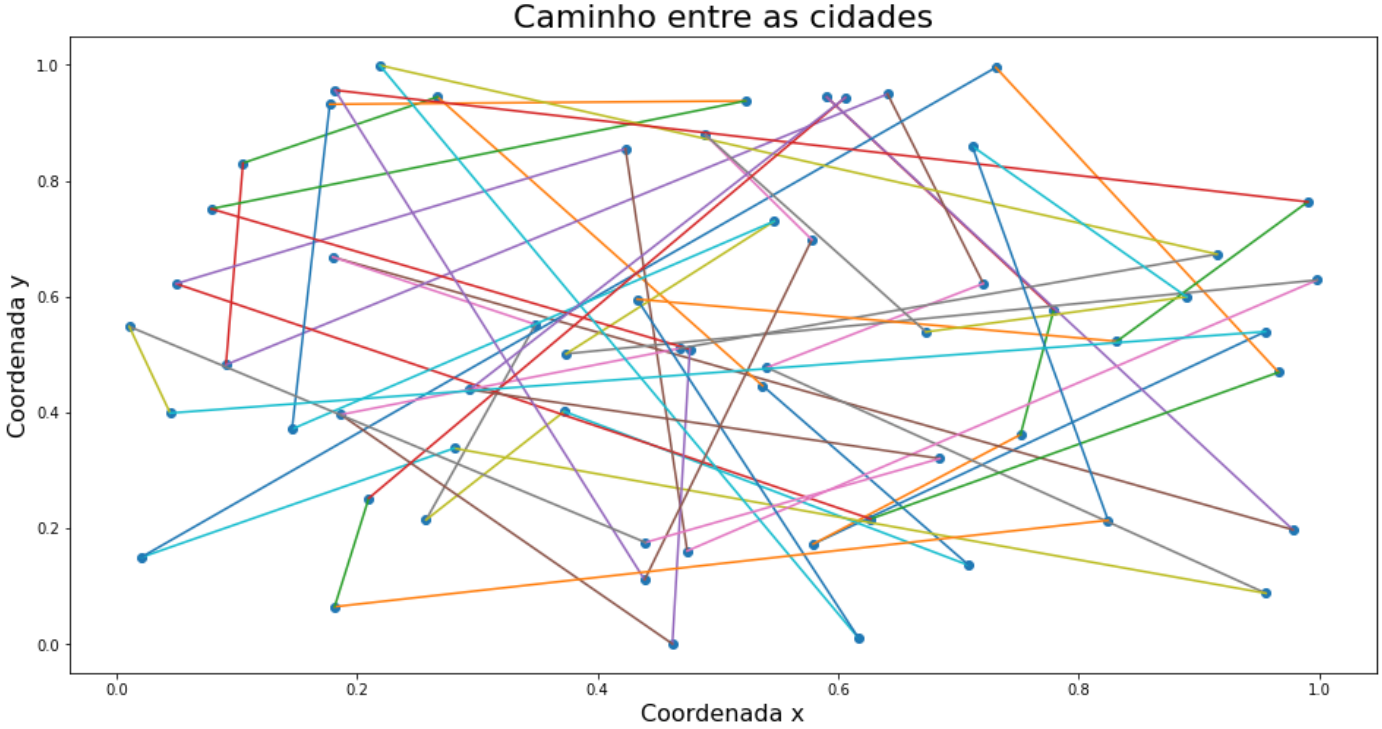
N = 60

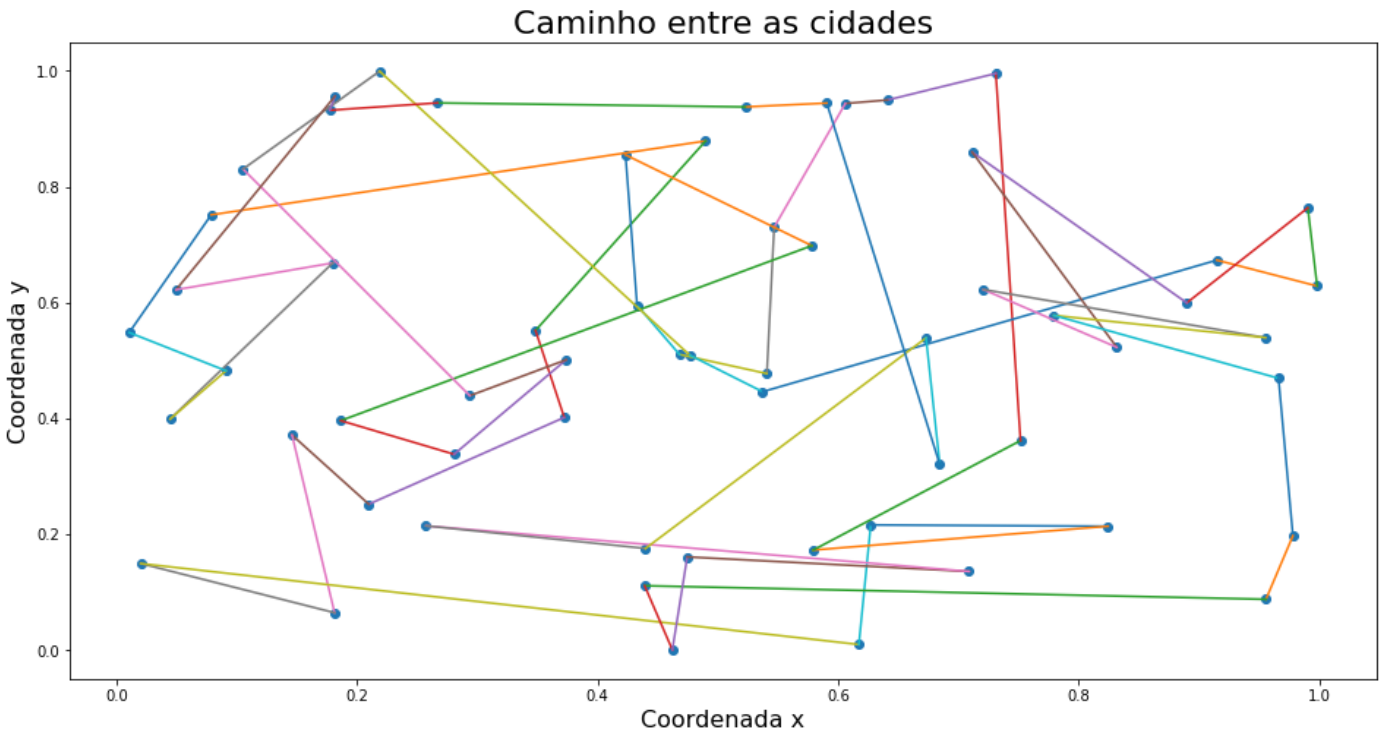
In [ ]:
```
N = 60
cidades, caminhos, distancias = caixeiro_viajante(N)
plotar_grafico_distancias(distancias, N)
```

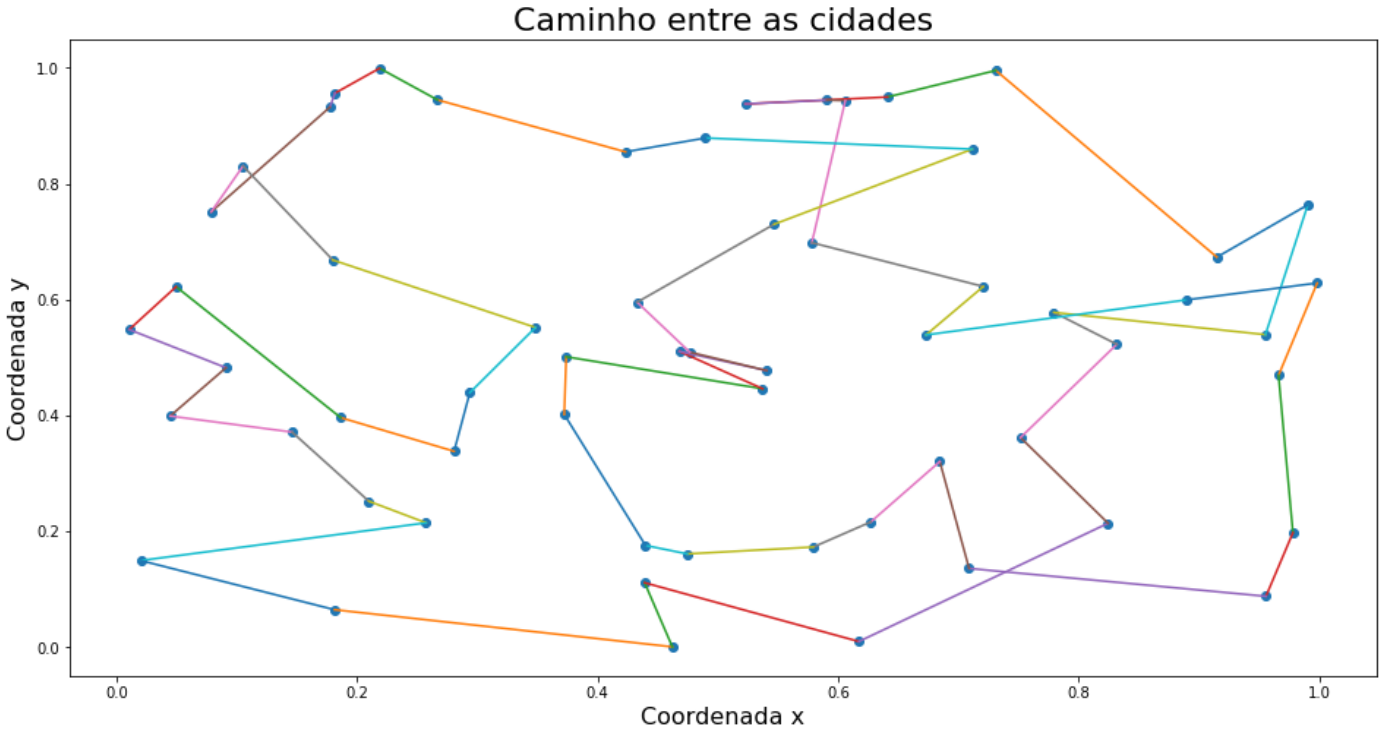


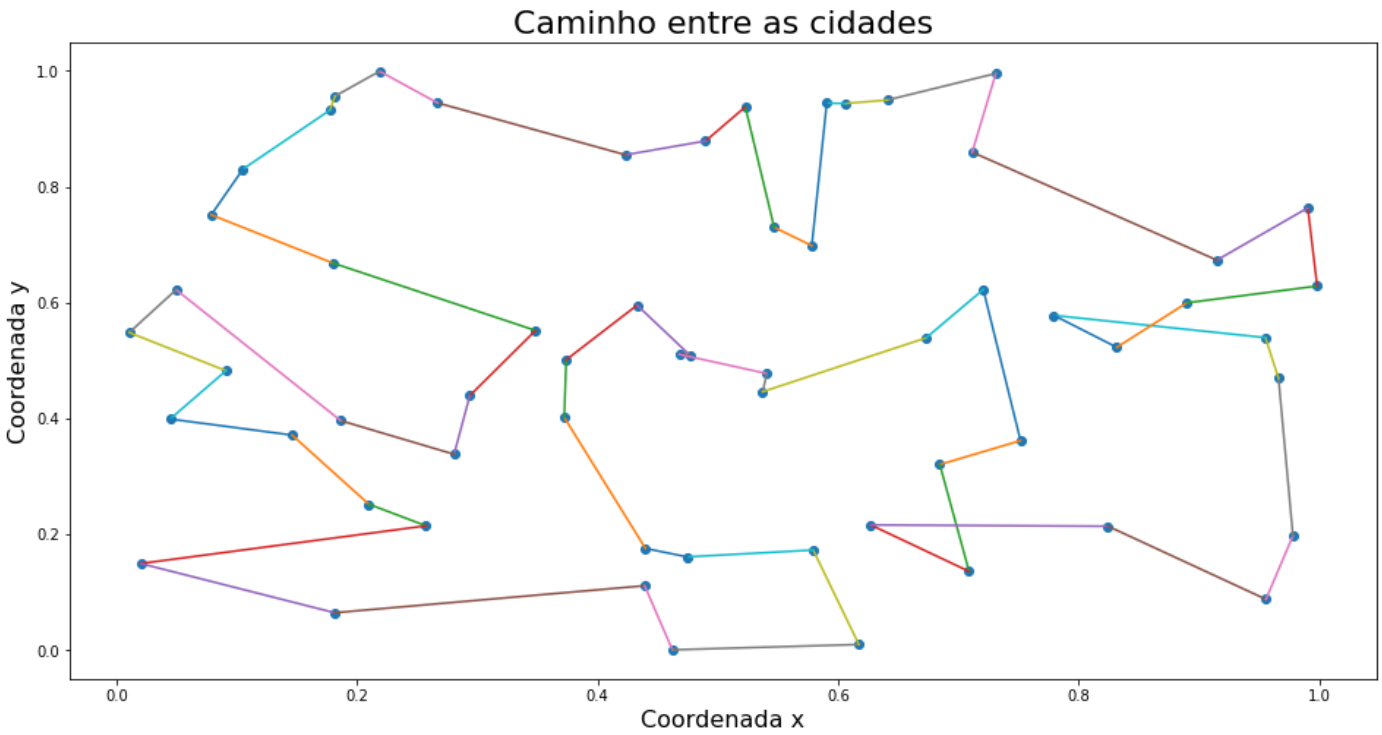
In [ ]:
```
caminhos_plot = gerar_caminhos_plot(caminhos)
for caminho in caminhos_plot:
  plotar_caminho(cidades, caminho)
```

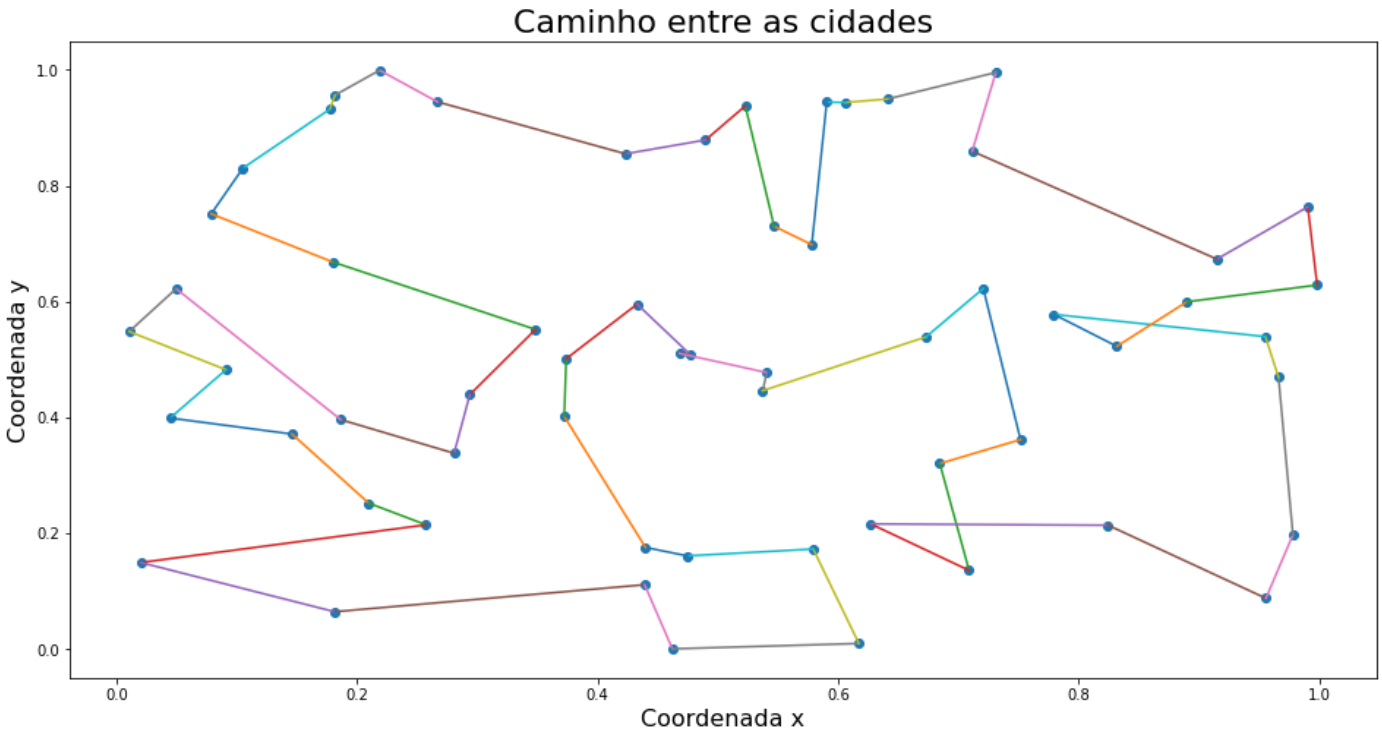

Caminho entre as cidades
Coordenada y
Coordenada x
0.0
0.2
0.4
0.6
0.8
1.0


Caminho entre as cidades
Coordenada y
Coordenada x
0.0
0.2
0.4
0.6
0.8
1.0


Caminho entre as cidades
Coordenada y
Coordenada x
0.0
0.2
0.4
0.6
0.8
1.0


Caminho entre as cidades
Coordenada y
Coordenada x
0.0
0.2
0.4
0.6
0.8
1.0



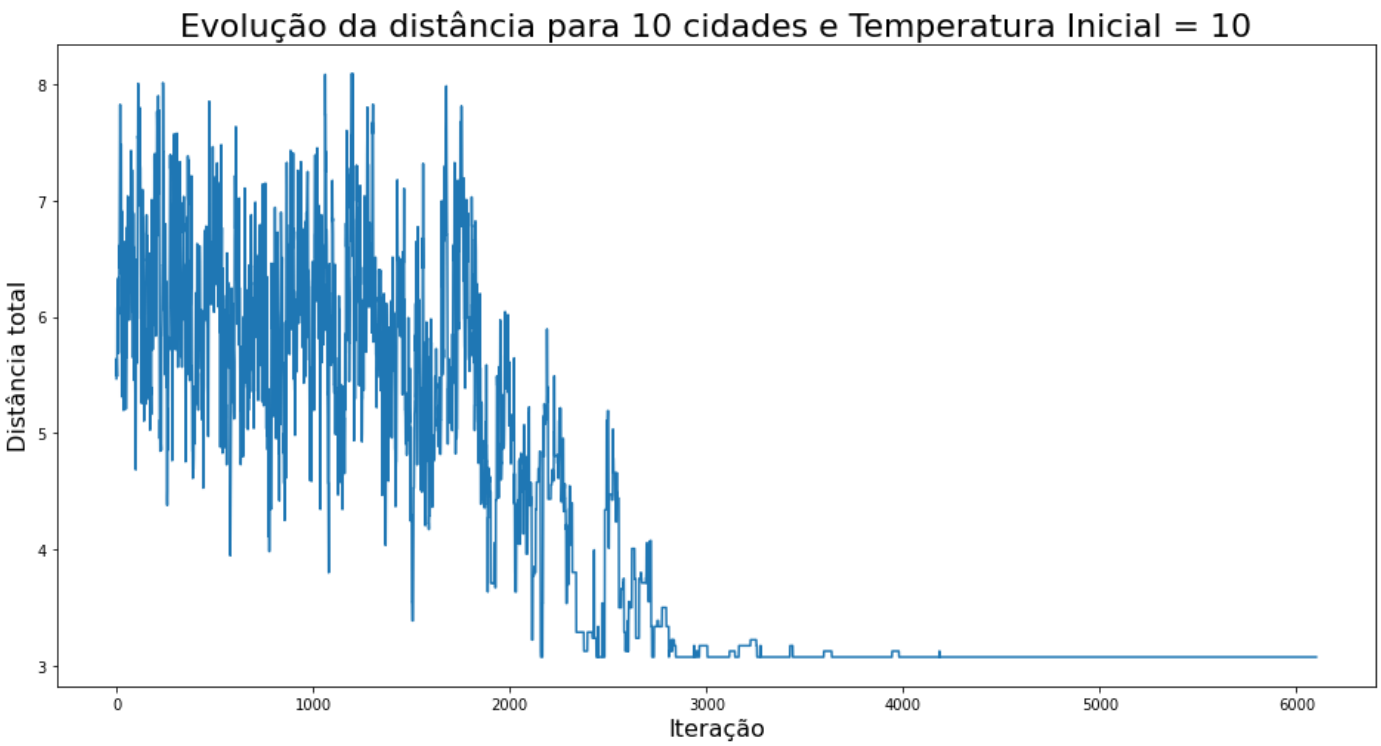


Execuções variando a temperatura inicial

In [ ]:
```
N = 10
```

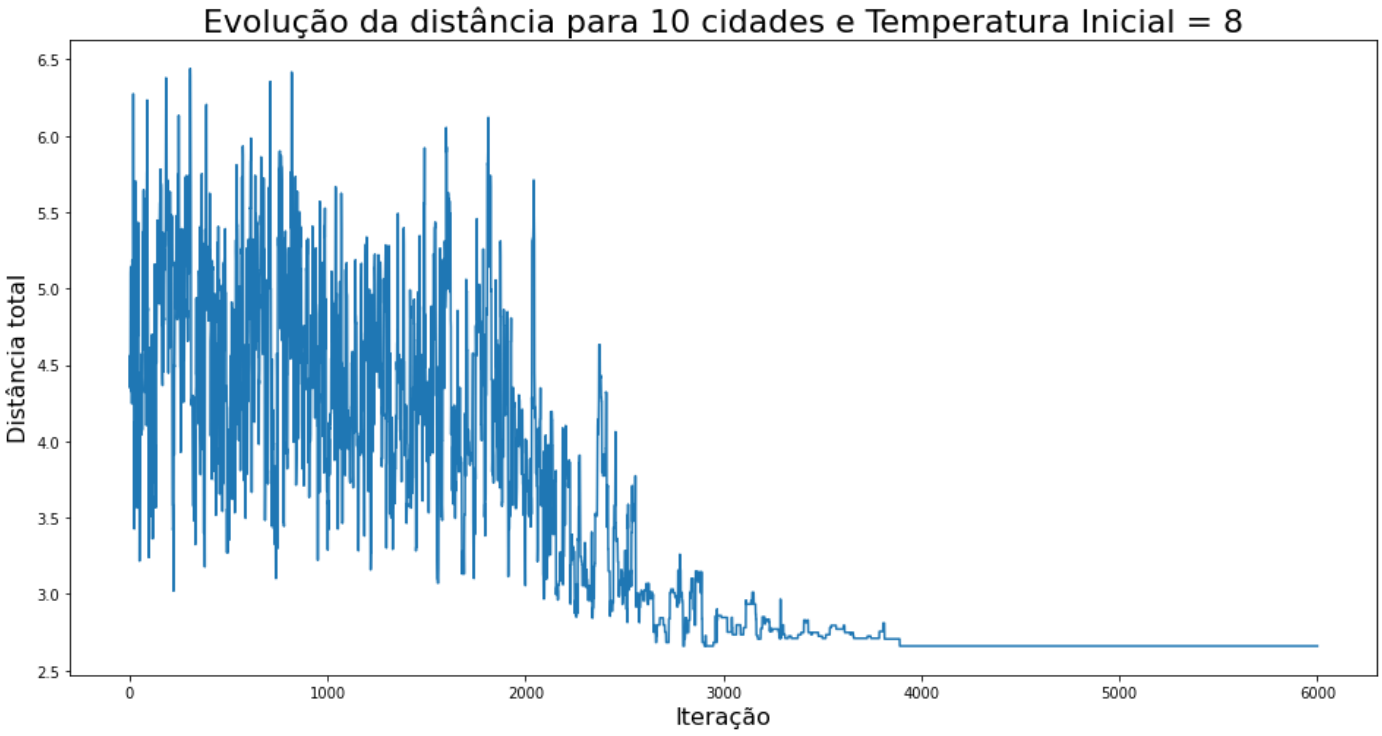
Temperatura inicial = 10

In [ ]:
```
cidades, caminhos, distancias = caixeiro_viajante(N, 10)
plotar_grafico_distancias(distancias, N, 10)
```



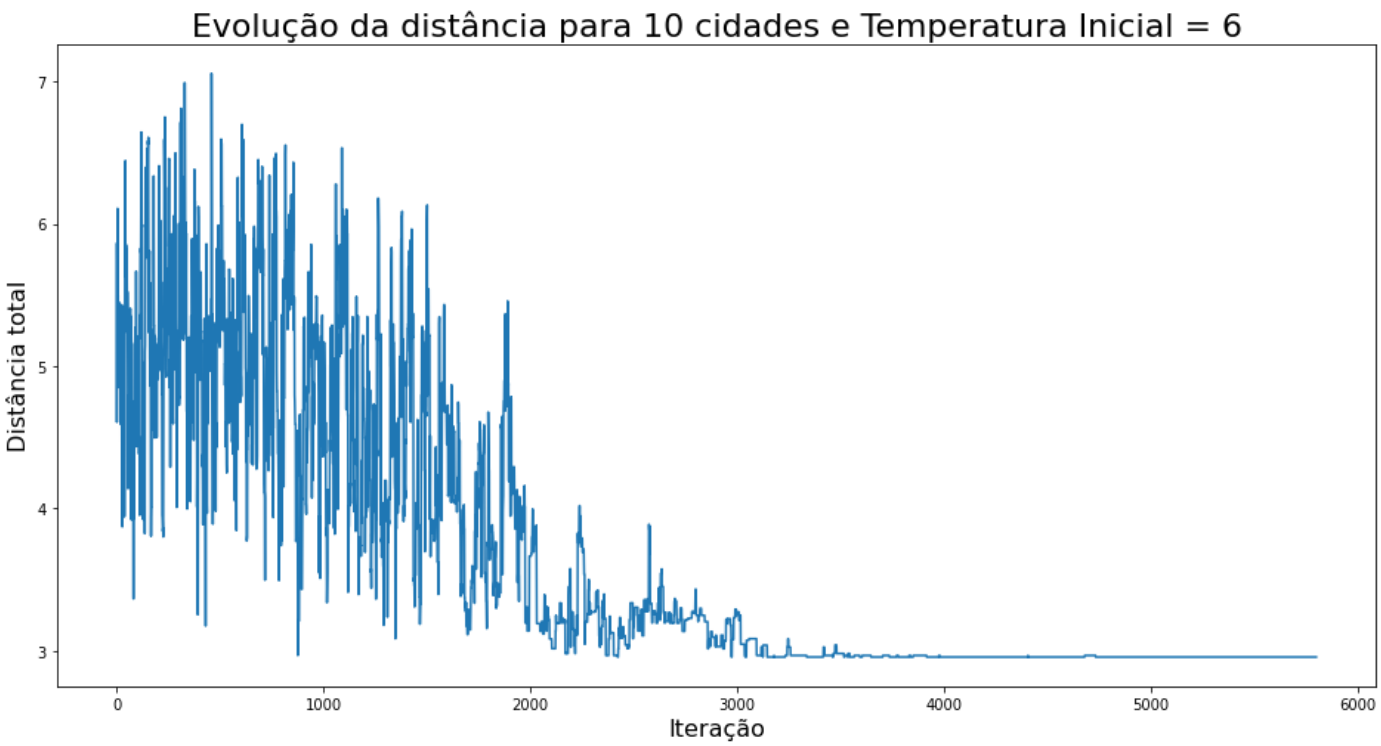
Temperatura inicial = 8

In [ ]:
```
cidades, caminhos, distancias = caixeiro_viajante(N, 8)
plotar_grafico_distancias(distancias, N, 8)
```



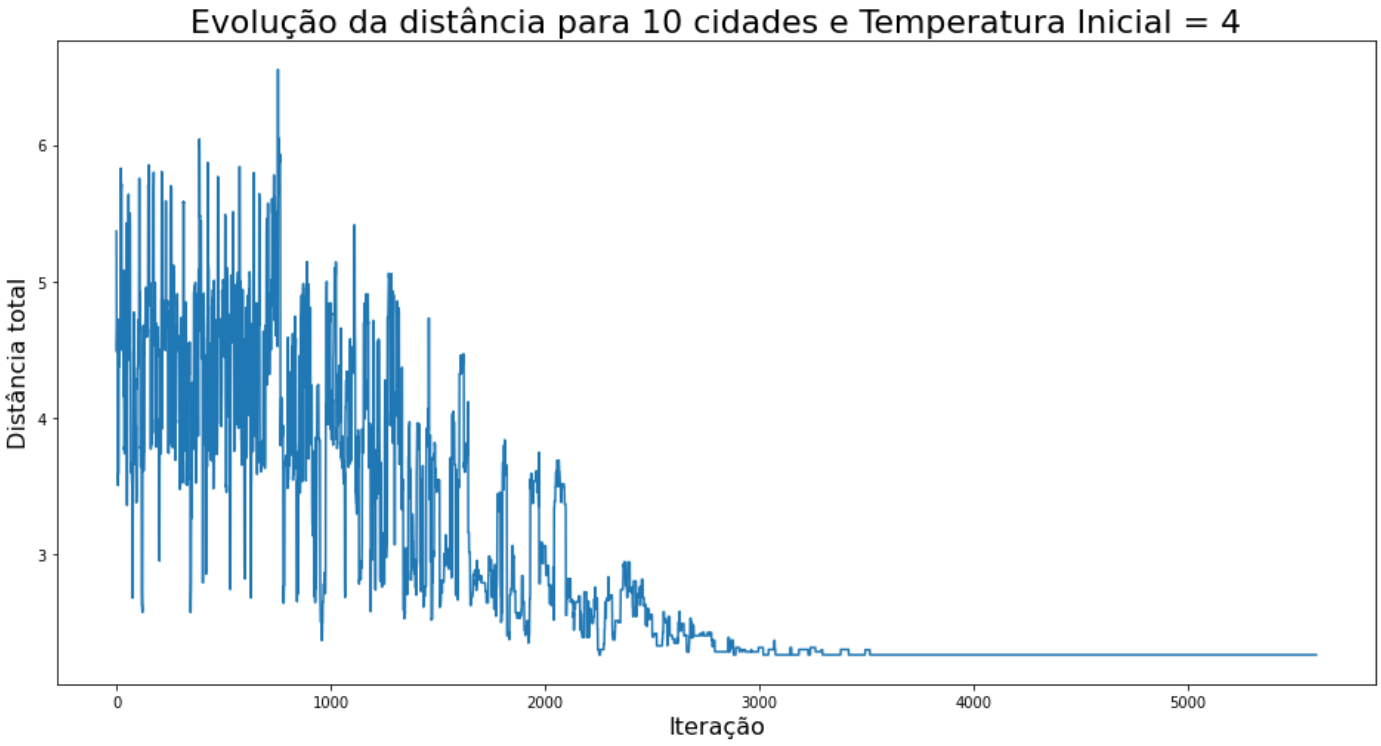
Temperatura inicial = 6

In [ ]:
```
cidades, caminhos, distancias = caixeiro_viajante(N, 6)
plotar_grafico_distancias(distancias, N, 6)
```



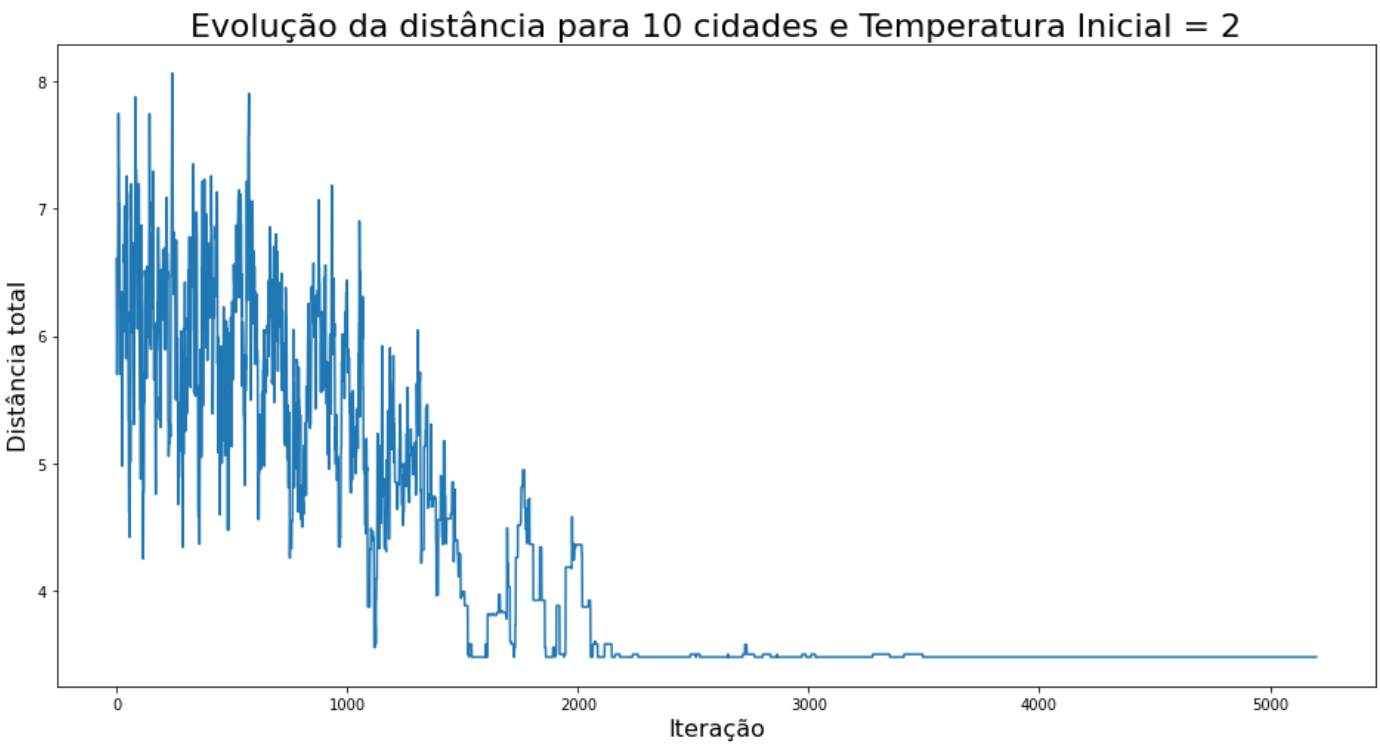
Temperatura inicial = 4

In [ ]:
```
cidades, caminhos, distancias = caixeiro_viajante(N, 4)
plotar_grafico_distancias(distancias, N, 4)
```



Temperatura inicial = 2

In [ ]:
```
cidades, caminhos, distancias = caixeiro_viajante(N, 2)
plotar_grafico_distancias(distancias, N, 2)
```



Análise dos resultados e conclusão

Pelos resultados obtidos, pode-ser perceber que para todos os valores de N executados, o gráfico das distâncias apresentou a tendência de diminuir até um valor que fosse estável, tal como esperado. Esse valor pode ou não ser o ótimo.

Uma fato notado é que quanto menos cidade houver, menor é o número de iterações necessárias para que o sistema atinja um valor estável. Por exemplo, para N = 10 a estabilização ocorre por volta da iteração 3300, enquanto para N = 60, essa estabilização parece ter ocorrido após a iteração 5400. Esses resultados foram obtidos para temperatura inicial igual a 8.

Já nos gráficos que mostram a evolução do caminho a ser percorrido, é possível notar que a medida que o número de iterações sobe, o caminho gerado fica mais organizado. Apesar disso, pode-se perceber que há alguns casos, principalmente a execução para N = 60, ainda há possíveis otimizações que poderiam melhorar os resultados obtidos.

Para os gráficos de evolução da distância mudando-se a temperatura inicial, foi possível perceber que as variações de distâncias se tornam menores à medida que a temperatura inicial fica menor.

Portanto, a solução do problema do Caixeiro Viajante usando o método Simulated Annealing apresentou resultados que visivelmente se aproximam muito do caminho ótimo a ser percorrido. A principal vantagem desse método está no seu menor tempo de execução, sendo uma boa opção para casos em que uma aproximação da solução ótima já seja suficiente.