TRABALHO FINAL DE GRAFOS

(O Carteiro Chinês)

Grupo:
Alan Evangelista de Sousa
Leônidas Pereira de Abreu

1 - INTRODUÇÃO

O trabalho final do Carteiro Chinês consistirá em implementar o algoritmo para resolver o problema do carteiro chinês em grafos não-orientados, utilizando uma heurística para resolver o problema do *1_emparelhamento*, obrigatoriamente aplicando-o a um caso prático;

2 – O CARTEIRO CHINÊS

Problemas de carteiro chinês pertencem a classe de problemas de roteamento em arcos, e são definidos em grafos orientados e não-orientados. Em problemas de roteamento em arcos, a meta é determinar a travessia de custo mínimo de um conjunto específico de arcos (ou arestas) de um grafo, com ou sem restrições.

Este problema, conhecido como problema do carteiro chinês, foi proposto pelo matemático Guan em 1962, quando trabalhava em um correio durante a revolução cultural chinesa. Guan enunciou este problema da seguinte maneira:

• “Um carteiro tem de cobrir seu segmento designado antes de retornar ao posto do correio. O problema é encontrar o caminho mais curto para o carteiro”

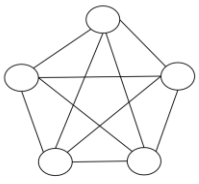
Kwan Mei-Ko foi o primeiro a relatar esse problema em uma publicação datada de 1962 na chinese mathematics, e por esse motivo, o problema foi denominado carteiro chinês.

Caso o grafo não seja Euleriano, ou seja, se existirem vértices de grau impar, será necessário percorrer algumas arestas mais de uma vez para que o ciclo seja possível.

• Guan observou que a adição de arestas ao vértices de grau impar, isto é, replicando arestas com o mesmo custo, gera um grafo em que todos os vértices tem grau par.

O PCC consiste em determinar quais arestas devem ser duplicadas de forma a obter um ciclo Euleriano de custo mínimo.

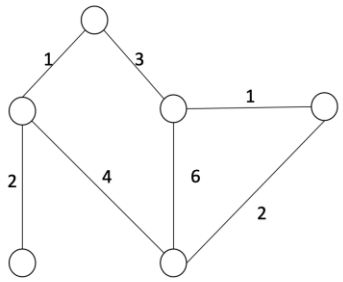
• Exemplo:



• Grafo Euleriano

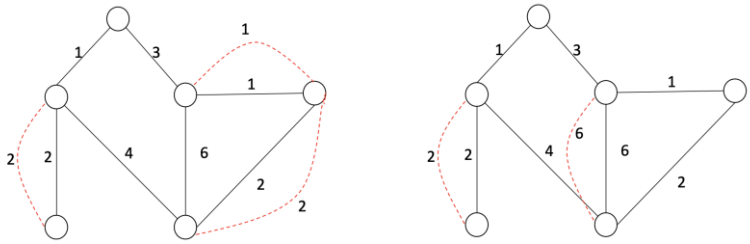
Ciclo euleriano

O PCC consiste em determinar quais arestas devem ser duplicadas de forma a obter um ciclo Euleriano de custo mínimo.



• Custo = 19, Grafo não Euleriano

O PCC consiste em determinar quais arestas devem ser duplicadas de forma a obter um ciclo Euleriano de custo mínimo.



• custo = 24 custo = 27

• Soluções para o PCC obtidos pela duplicação de arestas

2.1 - Aplicações

1. Entrega e coleta de correspondência;
2. Coleta de Lixo;
3. Roteamento de ônibus escolar;
4. Patrulhamento de ruas pela polícia.

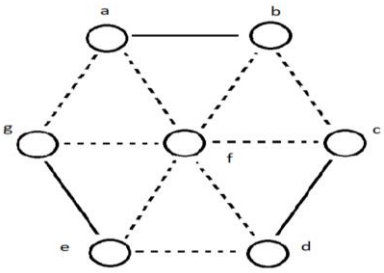
A modelagem matemática do Problemas do Carteiro Chinês é feita utilizando Técnicas de Pro ramação Inteira.

2.2 - Emparelhamento ou Matching

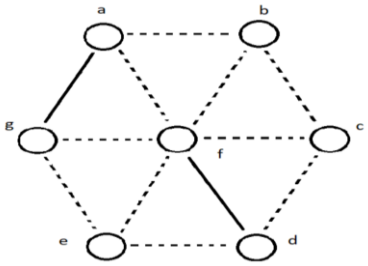
- Dado um grafo G (V,E), um emparelhamento em G – ou matching em G, é definido pelo conjunto
- M* de arestas tal que nenhum vértice de G seja incidente em mais de uma aresta de M*.
- O conjunto de todos os emparelhamentos de G será denotado por M*(G).
- A cardinalidade do maior emparelhamento (máximo) em G é denotada α (G).
- Se um emparelhamento envolver todos os vértices do grafo, ele é chamado um em
- É evidente que apenas um grafo com ordem par poderá ter (ou não) um emparelhamento perfeito.

Exemplos de Emparelhamento

- A cardinalidade deste emparelhamento é igual a 3. Ou seja, α (G) = 3

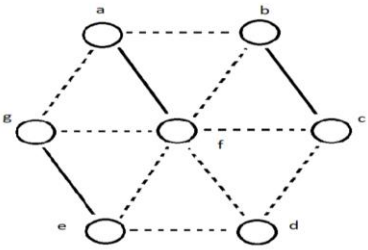


- α (G) = 2



Em um emparelhamento $d_m(i) \leq 1$, $\forall i \in V$, onde $d_m(i)$ é o grau do vértice i no emparelhamento.

A cardinalidade deste emparelhamento é igual a 3.

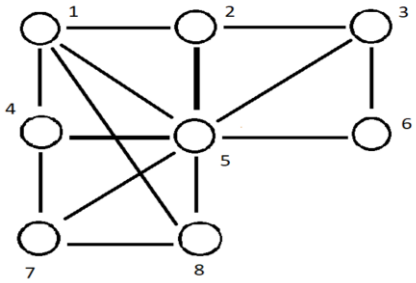


- Alguns autores denominam essa estrutura arestas de **1-emparelhamento**.
- É um emparelhamento de cardinalidade máxima.

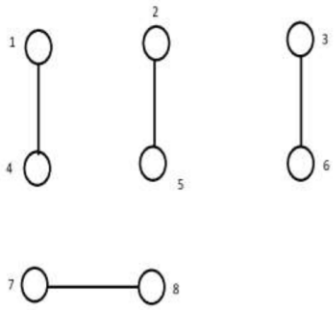
O problema de emparelhamento com cardinalidade máxima

- Pela definição de 1-emparelhamento cada vértice pode ser incidente em no máximo uma aresta. Logo, o conjunto vazio de arestas é um emparelhamento trivial.
- Dessa forma, temos que definir um objetivo para o problema.
- Em problemas de emparelhamento (1-emparelhamento, 2-emparelhamento ..)
- podemos definir diversos objetivos a serem otimizados.
- Para o nosso estudo e interesse

Exemplos de Emparelhamento



- Emparelhamento de cardinalidade = 4.



- Neste exemplo temos um **emparelhamento perfeito**. Ou seja, *é um emparelhamento de cardinalidade máxima.*
- É evidente que apenas um grafo com ordem par poderá ter (ou não) um emparelhamento perfeito.

O Problema de Emparelhamento de Custo Mínimo

- Se o grafo é ponderado, define-se emparelhamento de custo mínimo como o emparelhamento de cardinalidade máxima cuja soma do valor das arestas é mínimo.
- É esse problema que vai nos interessar. Porque?
- No problema do carteiro chinês temos que duplicar algumas arestas de modo a construir um
- hipergrafo G_i que seja Euleriano.
- A questão é que devemos buscar o melhor hipergrafo Gi . Ou seja, um hipergrafo de G cujo ciclo Euleriano tenha custo mínimo.

2.3 - Algoritmo Carteiro Chinês

- **Ler** o grafo não-orientado G = (N, A)

- **Se** todos os vértices de G possuem grau par

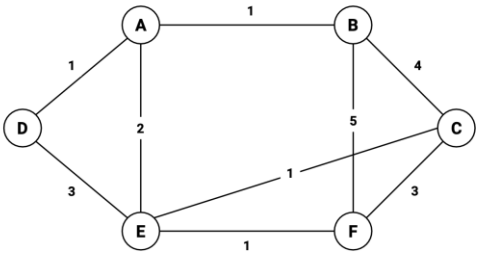
  então d**etermine u**m ciclo Euleriano em G e Fim.

- **Organize** um grafo K_n da seguinte forma:

Reúna todos os vértices de grau ímpar no grafo Kn e ass**ocie a** cada par de vértices i e j no grafo, uma aresta (i,j) com peso igual ao caminho mínimo que liga i a j no grafo G.

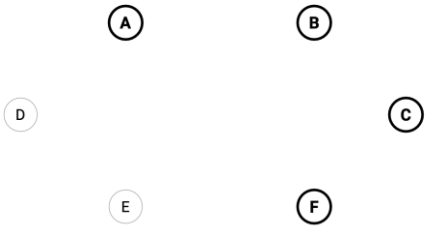
- **Determine** o 1-emparelhamento perfeito de custo mínimo em Kn, M*. **Para cada aresta** pertencente a M* **associe** uma nova aresta em G no caminho mínimo que ela representa, obtendo um hipergrafo Gi.
- **Determine** a solução do carteiro chinês que é representada por um ciclo Euleriano em Gi

Exemplo 1: Problema do Carteiro Chinês



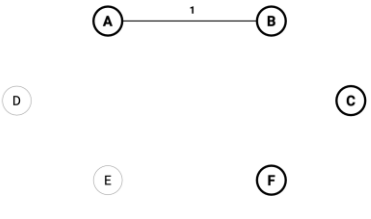
Grafo Original

Exemplo 1



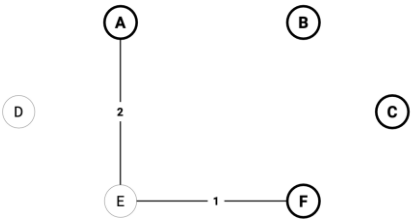
Vértices de Grau Ímpar

Exemplo 1



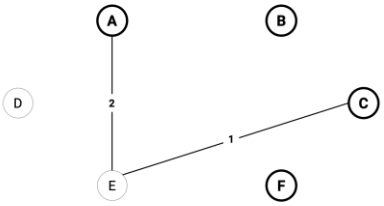
Caminho mais curto A - B

Exemplo 1



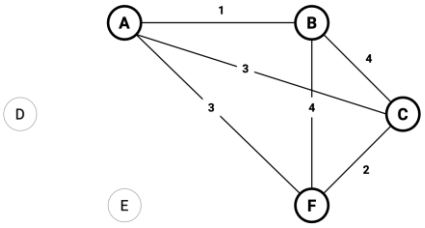
Caminho mais curto A - F

Exemplo 1



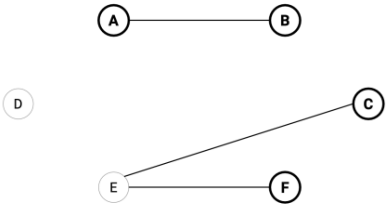
Caminho mais curto A-C

Exemplo 1 (A,B);(C,F); (A,C);(B,F); (A,F);(B,C)



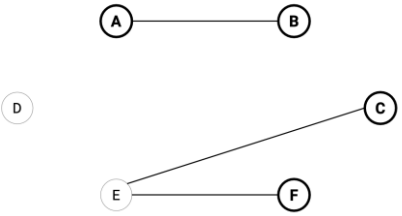
K_4

Exemplo 1



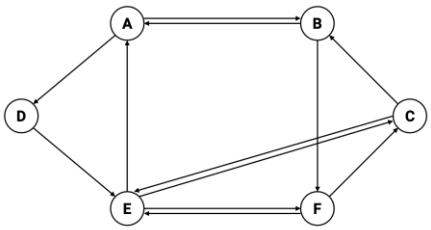
Caminhos Associados – arestas a serem duplicadas

Exemplo 1



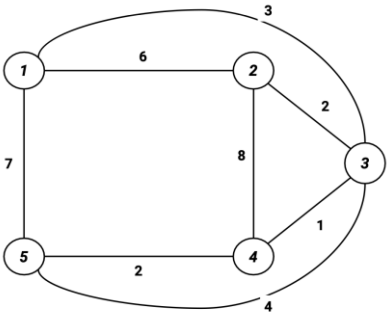
Caminhos Associados – arestas a serem duplicadas

Exemplo 1

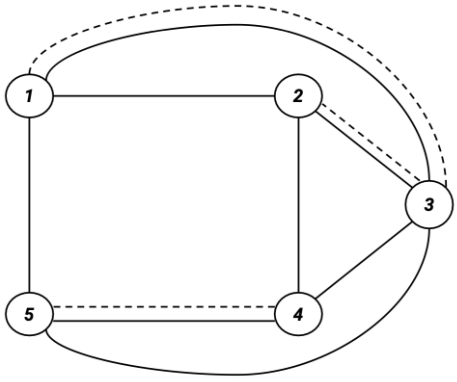


Solução Final – hipergrafo G_1
Aplica-se um método para encontrar
um ciclo Euleriano

Problema do Carteiro Chinês



- Os vértices 1, 2, 4 e 5 possuem grau ímpar.
- Calcular o caminho mínimo entre todos os pares de vértices 1-2; 1-4; 1-5; 2-4; 2-5; e 4-5.
- Possíveis 1-emparelhamento (1-matching)
    - (1-2) e (4,5);
    - (1-4) e (2,5);
    - (1,5) e (2,4);



Solução Final – Aplica-se um
método para encontrar um ciclo
Euleriano

3 - CÓDIGO FONTE

```
rom Graph import Graph, Edge, Vertex
import copy

def sum_edges(graph):
w_sum = 0
l = len(graph)
for i in range(l):
for j in range(i,l):
w_sum += graph[i][j]
return w_sum

def dijktra(graph, source, dest):
shortest = [0 for i in range(len(graph))]
selected = [source]
l = len(graph)
#Base case from source
inf = 10000000
min_sel = inf
for i in range(l):
if(i==source):
shortest[source] = 0 #graph[source][source]
else:
if(graph[source][i]==0):
shortest[i] = inf
else:
shortest[i] = graph[source][i]
if(shortest[i] < min_sel):
min_sel = shortest[i]
ind = i
if(source==dest):
return 0
```

```
# Dijktra's in Play
selected.append(ind)
while(ind!=dest):
#print('ind',ind)
for i in range(l):
if i not in selected:
if(graph[ind][i]!=0):
#Check if distance needs to be updated
if((graph[ind][i] + min_sel) < shortest[i]):
shortest[i] = graph[ind][i] + min_sel
temp_min = 1000000
#print('shortest:',shortest)
#print('selected:',selected)
for j in range(l):
if j not in selected:
if(shortest[j] < temp_min):
temp_min = shortest[j]
ind = j
min_sel = temp_min
selected.append(ind)
return shortest[dest]
#Finding odd degree vertices in graph

def get_odd(graph):
degrees = [0 for i in range(len(graph))]
for i in range(len(graph)):
for j in range(len(graph)):
if(graph[i][j]!=0):
degrees[i]+=1
```

```
#print(degrees)
odds = [i for i in range(len(degrees)) if degrees[i]%2!=0]
#print('odds are:',odds)
return odds
#Function to generate unique pairs
def gen_pairs(odds):
pairs = []
for i in range(len(odds)-1):
pairs.append([])
for j in range(i+1,len(odds)):
pairs[i].append([odds[i],odds[j]])
#print('pairs are:',pairs)
#print('\n')
return pairs

#Final Compiled Function
def Chinese_Postman(graph):
odds_vertices = get_odd(graph)
if(len(odds_vertices)==0):
return sum_edges(graph)
vertexes_pairs_list = gen_pairs(odds_vertices)
l = (len(vertexes_pairs_list)+1)//2
pairings_sum = []
def get_pairs(vertexes_pairs, vertexes_done = [], final = []):
if(vertexes_pairs[0][0][0] not in vertexes_done):
vertexes_done.append(vertexes_pairs[0][0][0])
for i in vertexes_pairs[0]:
finalvertex = final[:]
value_vertex = vertexes_done[:]
if(i[1] not in value_vertex):
finalvertex.append(i)
else:
continue
```

```
if(len(finalvertex)==l):
pairings_sum.append(finalvertex)
return
else:

value_vertex.append(i[1])
get_pairs(vertexes_pairs[1:],value_vertex, finalvertex)
else:
get_pairs(vertexes_pairs[1:], vertexes_done, final)
get_pairs(vertexes_pairs_list)
min_sums = []
for i in pairings_sum:
sumtotal = 0
for j in range(len(i)):
sumtotal += dijktra(graph, i[j][0], i[j][1])
min_sums.append(sumtotal)
added_dis = min(min_sums)
chinese_dis = added_dis + sum_edges(graph)
return chinese_dis

class ChinesePostmanAlgorithm:
def __init__(self, graph: Graph):
self.graph: Graph = graph
def solve(self):
# Prepare
## Create path to be returned by algorithm
path = []
r = None
```

```
# Solve
## Check if it is an euler graph
### If so, return the euler path as the answer
success, euler_path = self.graph.findEulerPath(self.graph.edges, self.graph.vertexes,
self.graph.vertexes[0])
if self.graph.isCompleteGraph() and success == True:
return euler_path
### Else
else:
# Eulerize the graph
vertexes = copy.deepcopy(self.graph.vertexes)
edges = copy.deepcopy(self.graph.edges)
matrix = []
for i in range(len(vertexes)):
line = []
matrix.append(line)
for j in range(len(vertexes)):
line.append(0)

id_to_index = {}
for ind, val in enumerate(vertexes):
val: Vertex = val
id_to_index[val.id] = ind
print(id_to_index)


for edge in edges:
edge: Edge = edge
print(edge)




con1 = edge.connection[0].id
```

```
con2 = edge.connection[1].id
pos1 = id_to_index[con1]
pos2 = id_to_index[con2]
matrix[pos1][pos2] = edge.weight
matrix[pos2][pos1] = edge.weight
print(f"ACTION: {pos1}:{pos2} = {edge.weight}")
for l in matrix:
print(l)
r = Chinese_Postman(matrix)
print(r)
# Return the algorithm generated path
return path, r
```

4 - CONCLUSÃO

O Problema do Caixeiro Viajante ou TSP (do inglês Traveling Salesman Problem) é importante pois é um problema de otimização combinatória no qual busca-se determinar a rota de menor custo entre um grupo de pontos, que podem ser representações de cidades, localidades, ou de outros elementos tratados em problemas de designação. Nessa rota, cada ponto deve ser visitado uma, e uma única, vez. O custo, geralmente, é calculado com base em distintos fatores, tais como a distância de deslocamento entre os pontos, o tempo necessário para tal deslocamento, condições de cada trecho, gastos com combustível, ou outros fatores que podem estar, direta ou indiretamente, associados aos trechos de deslocamento. Uma importante generalização do TSP é o Problema do Caixeiro Viajante com Múltiplas Rotas ou MTSP (do inglês Multiple Traveling Salesman Problem), onde são considerados vários caixeiros viajantes, cada um percorrendo a sua própria rota. Assim sendo, no caso de um MTSP, são admitidas soluções com várias rotas partindo da origem, passando por alguns pontos, e retornando a origem. Contudo, assim como no caso do TSP, todos os pontos, com exceção do ponto de origem, devem ser visitados uma única vez, ou seja, devem pertencer a uma única rota.