Aula 18 - Grafos Hamiltonianos e o problema do Caixeiro Viajante

Luís Felipe

UFF

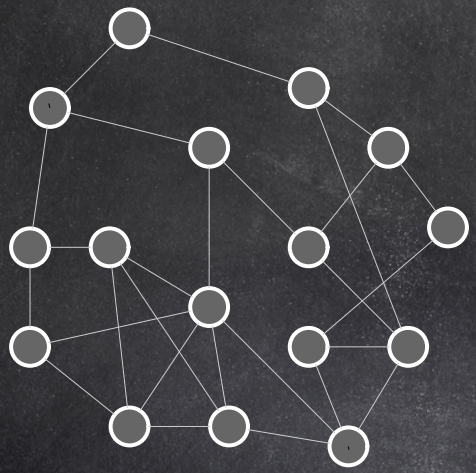
09 de Junho de 2026

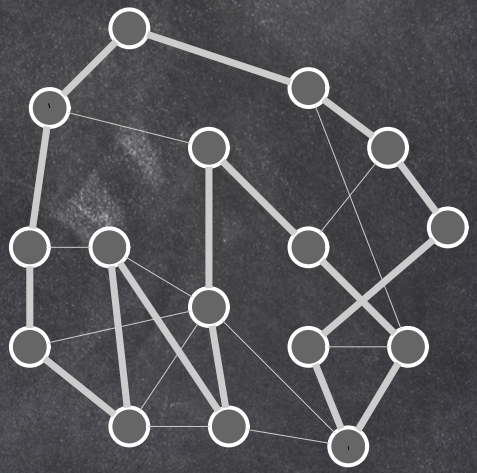
Luís Felipe

09/06/26

Grafos Hamiltonianos

- Um ciclo de Hamilton é um ciclo gerador para o grafo. Ou seja, é um ciclo que contém todos os vértices do grafo.
- Um grafo que admite um ciclo de Hamilton é dito Hamiltoniano.





Luís Felipe

09/06/26

Grafos Hamiltonianos

- Até hoje, não existe um teorema correspondente ao de Euler que caracterize grafos Hamiltonianos. Na verdade, decidir se um grafo é Hamiltoniano é um problema NP-completo (visto em ASA/APA).

- Uma forma de trabalhar com esse assunto é determinar condições necessárias e condições suficientes para tais grafos. Algumas condições:
  - Condições necessárias: ser conexo; ser biconexo; remoção de subconjuntos de vértices S próprios não vazios possuem tornam o grafo com no máximo $|S|$ componentes conexas.
  - Condições suficientes: ser completo; possuir grau mínimo $\delta(G) \geq \frac{n}{2}$.

  Propriedades provadas em Teoria dos Grafos

Luís Felipe

09/06/26

Problema do Caixeiro Viajante

O Problema do Caixeiro Viajante, ou TSP, é um problema de otimização sobre ciclos Hamiltonianos.

- Temos um conjunto de cidades.
- Para cada par de cidades, há um custo de deslocamento.
- O objetivo é sair de uma cidade, visitar todas as outras exatamente uma vez e retornar à cidade inicial.
- Em termos de grafos, queremos encontrar um ciclo Hamiltoniano de menor custo.

Se $G = (V, E)$ é um grafo com custo $c(e)$ em cada aresta, queremos minimizar:

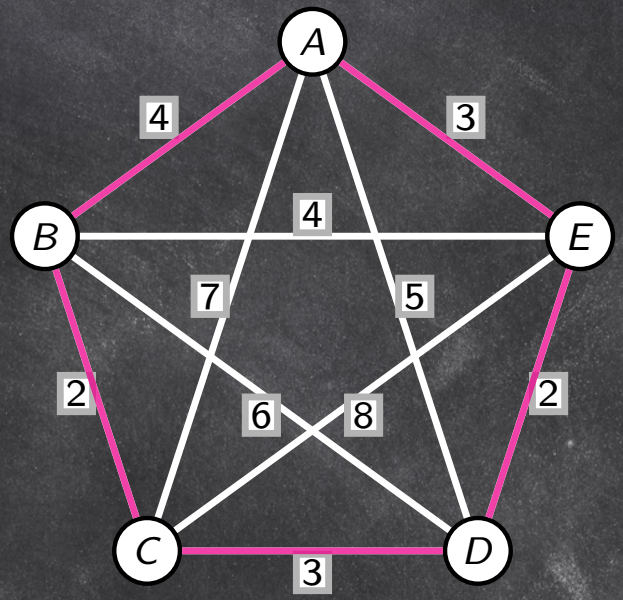
$$c(C) = \sum_{e \in E(C)} c(e),$$

onde C é um ciclo Hamiltoniano de G.

Luís Felipe

09/06/26

Problema do Caixeiro Viajante



Um candidato de solução é o ciclo:

$$A, B, C, D, E, A.$$

Seu custo é:

$$4 + 2 + 3 + 2 + 3 = 14.$$

O problema é encontrar, entre todos os ciclos Hamiltonianos, aquele de menor custo.

Luís Felipe

09/06/26

Caixeiro Viajante Métrico

No Caixeiro Viajante Métrico, trabalhamos com uma versão especial do TSP.

1. O grafo é completo.
2. Cada aresta uv possui um custo $c(u, v) \geq 0$.
3. Os custos satisfazem a desigualdade triangular:

$$c(u, v) \leq c(u, x) + c(x, v),$$

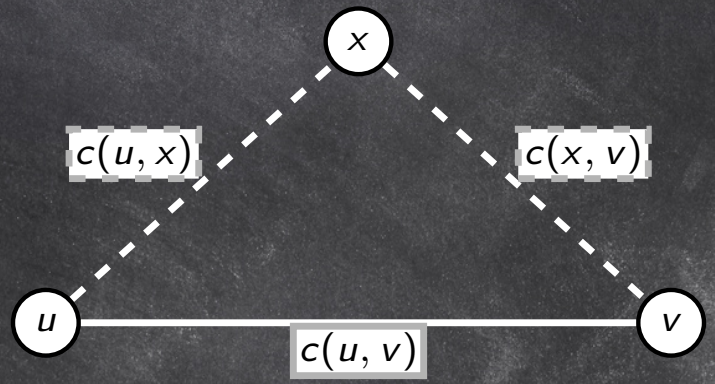
   para quaisquer vértices u, v, x.
4. O objetivo continua sendo encontrar um ciclo Hamiltoniano de menor custo.

A desigualdade triangular significa que ir diretamente de u para v nunca é mais caro do que ir de u para v passando por um terceiro vértice x.

Luís Felipe

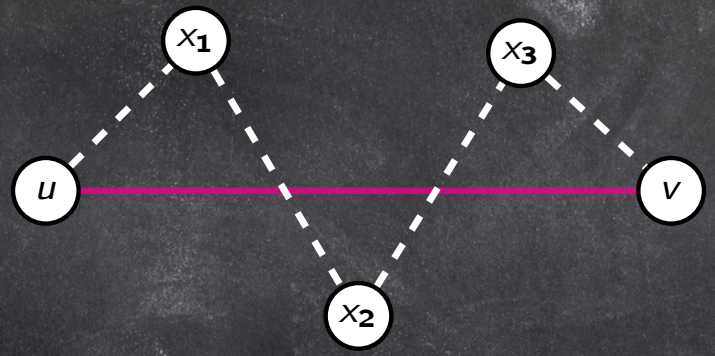
09/06/26

Desigualdade triangular



A desigualdade triangular diz que:

$$c(u, v) \leq c(u, x) + c(x, v).$$



De forma geral, trocar um caminho por uma aresta direta não aumenta o custo. $c(u, v) \leq c(u, x_1) + c(x_1, x_2) + c(x_2, x_3) + c(x_3, v)$.

Luís Felipe

09/06/26

Caixeiro Viajante Métrico

- Entrada: um grafo completo $G = (V, E)$ com custos não negativos nas arestas.
- Hipótese métrica: para quaisquer $u, v, x \in V$,

$$c(u, v) \leq c(u, x) + c(x, v).$$

- Solução viável: um ciclo Hamiltoniano de G.
- Objetivo: minimizar o custo total do ciclo:

$$\min \sum_{e \in E(C)} c(e).$$

- A hipótese métrica será essencial para transformar passeios com vértices repetidos em ciclos Hamiltonianos sem aumentar o custo.

Luís Felipe

09/06/26

Algoritmo RSL – Rosenkrantz, Stearns e Lewis

- Obtém-se uma árvore geradora mínima
- Duplicam-se as arestas da árvore, obtendo um multigrafo.
  - Consequência: todo vértice tem grau par
  - Logo, o grafo resultante é Euleriano
- Toma-se um circuito Euleriano
  - A partir dele, constrói-se um ciclo Hamiltoniano substituindo as repetições de vértices por caminhos "diretos"
  - Assim, arestas que não estão no circuito Euleriano serão inseridas.
  - Note que elas sempre existem, pois o grafo original é completo.

Luís Felipe

09/06/26

RSL em pseudocódigo

```
1  funcao RSL(G, c):
2
3      T <- arvore geradora minima de G
4
5      H <- multigrafo obtido duplicando cada aresta de T
6
7      W <- circuito euleriano de H
8
9      C <- ciclo obtido de W por atalhos
10
11     retornar C
```

Atalho: ao percorrer o circuito euleriano, sempre que um vértice já visitado apareceria novamente, seguimos diretamente para o próximo vértice ainda não visitado.

A desigualdade triangular garante que esse atalho não aumenta o custo.

Luís Felipe

09/06/26

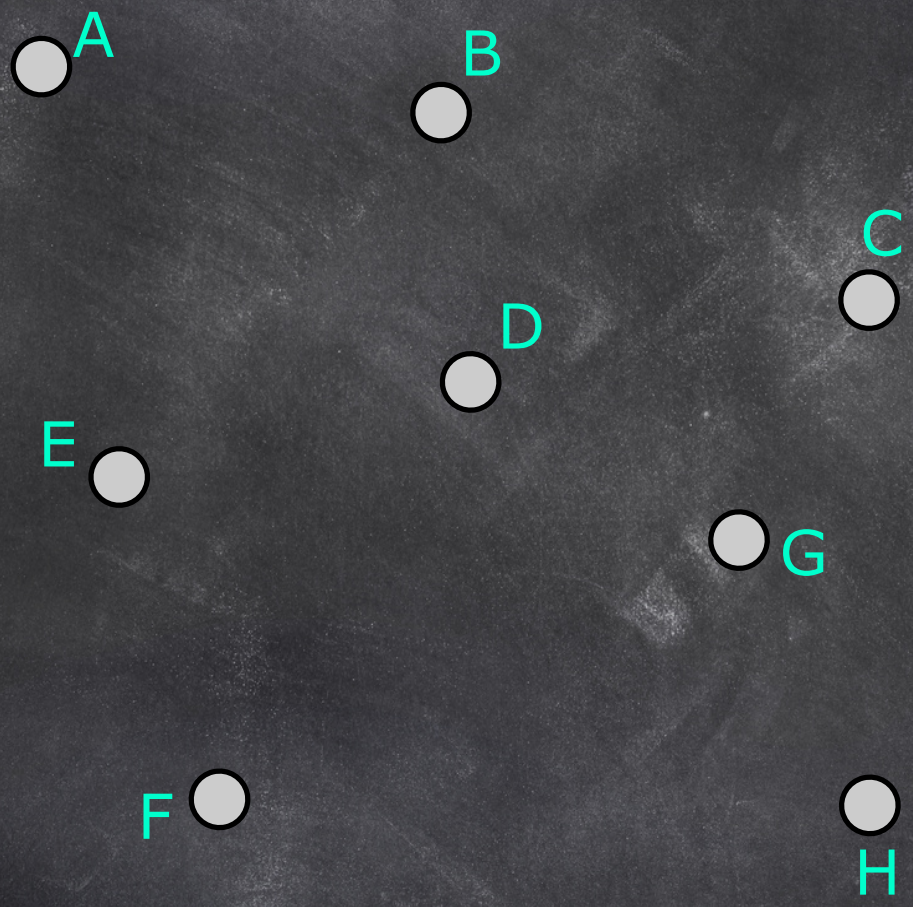
Subrotinas utilizadas

- MST: usaremos Prim para obter uma árvore geradora mínima.
- Circuito Euleriano: usaremos Hierholzer, visto na aula anterior.
- Atalhos: removeremos repetições de vértices usando a desigualdade triangular.

Luís Felipe

09/06/26

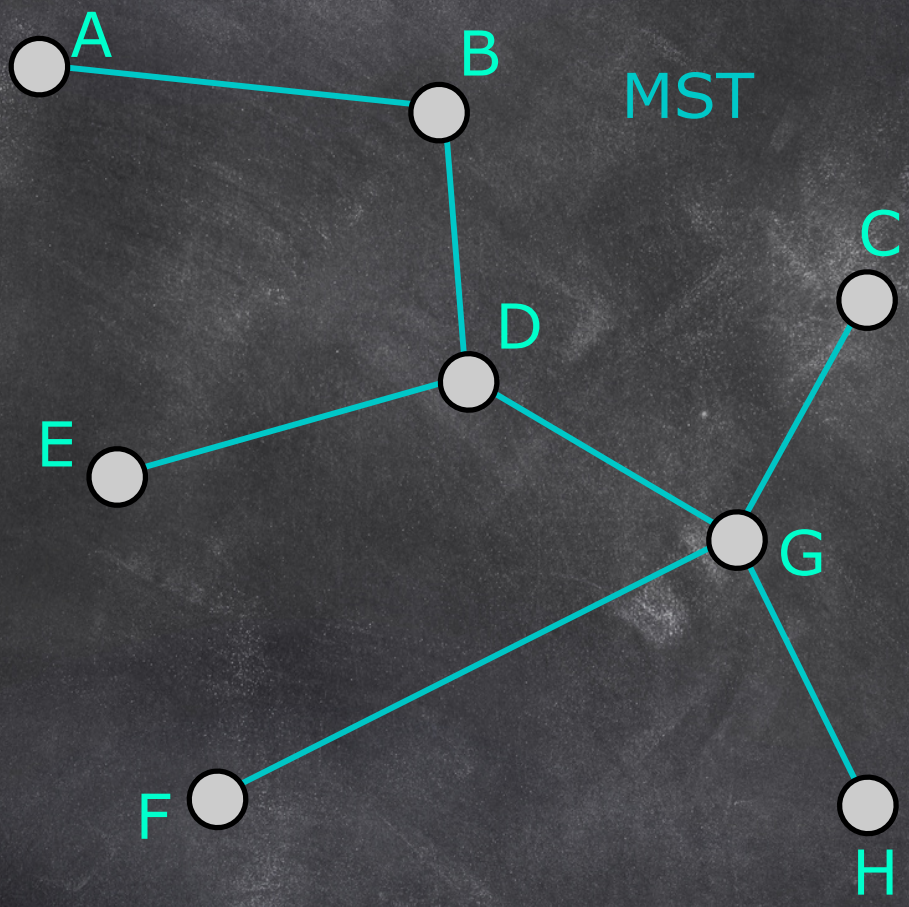
Exemplo de execução – RSL



Luís Felipe

09/06/26

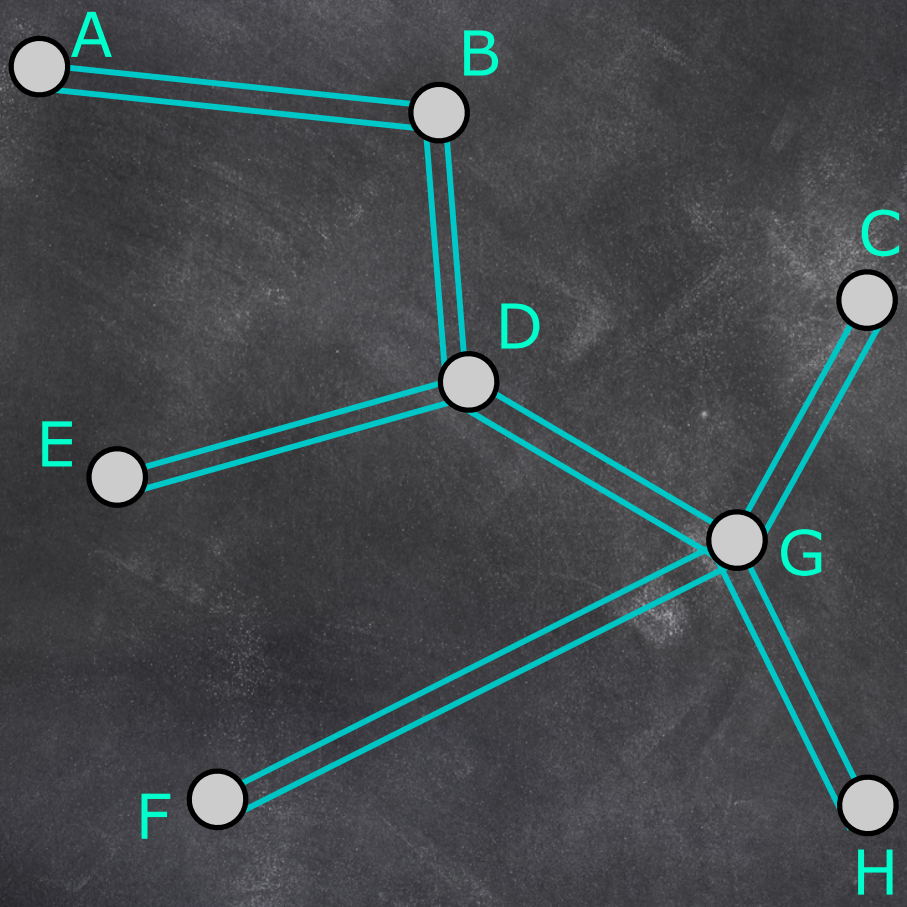
Exemplo de execução – RSL



Luís Felipe

09/06/26

Exemplo de execução – RSL

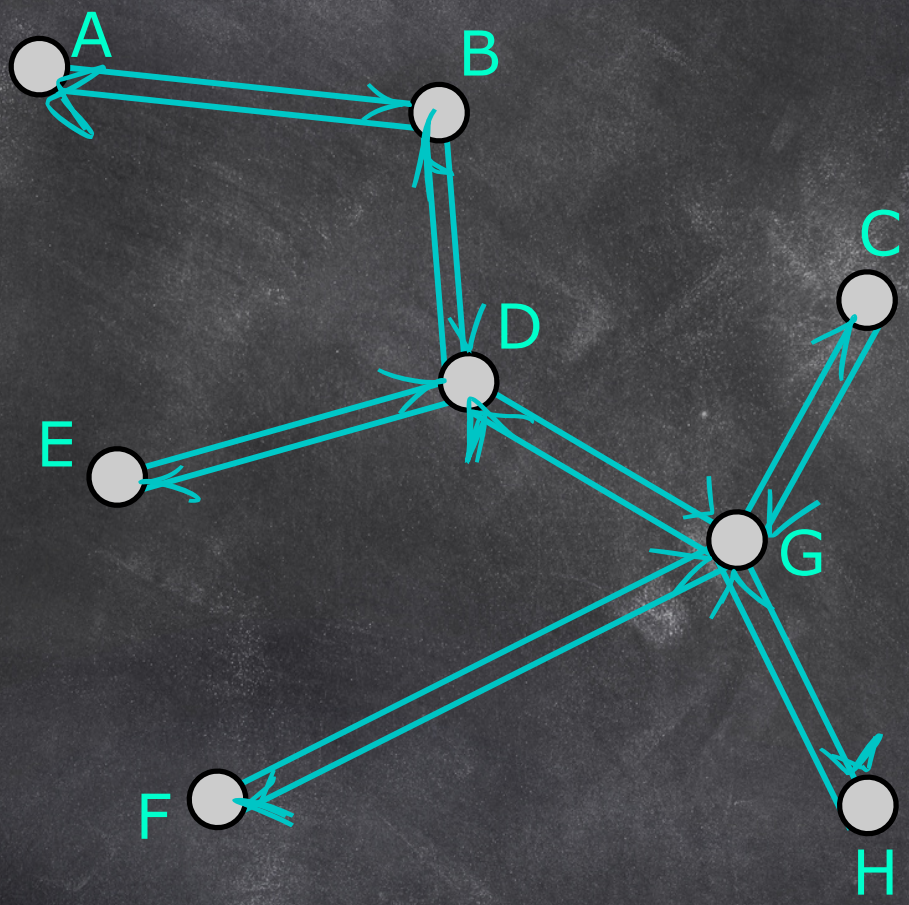


Duplicam-se as arestas da MST

Luís Felipe

09/06/26

Exemplo de execução – RSL

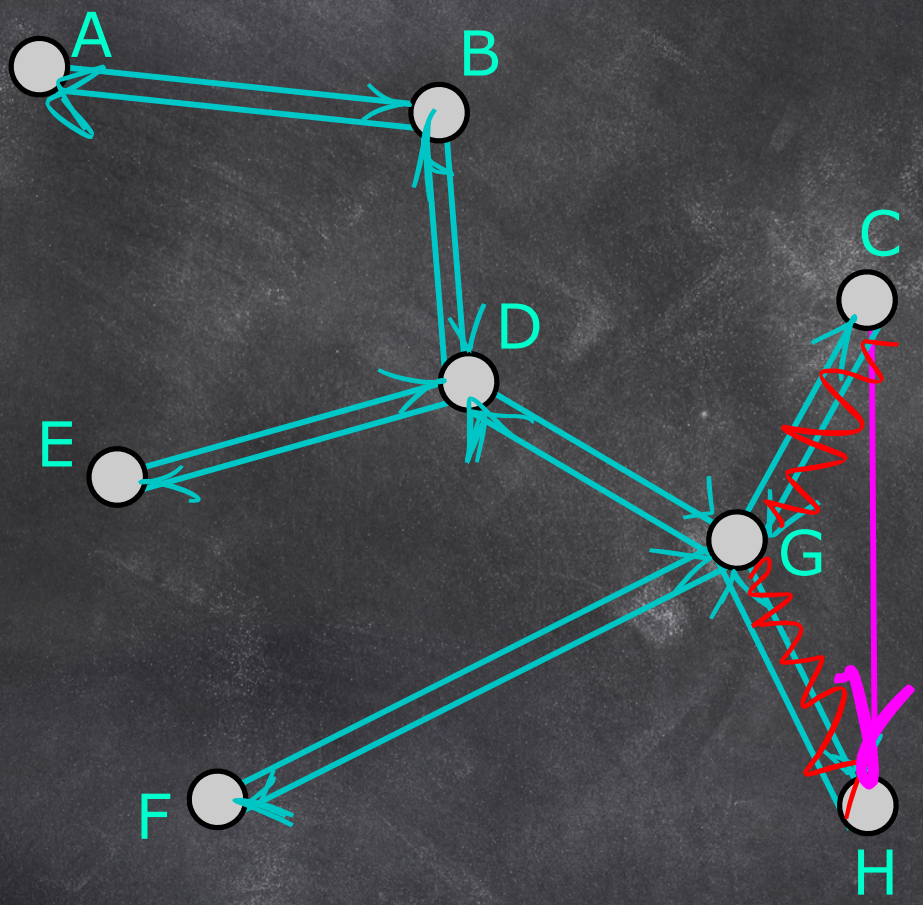


Obtem um circuito Euleriano:
A, B, D, G, C, G, H, G, F, G, D, E, D, B, A

Luís Felipe

09/06/26

Exemplo de execução – RSL



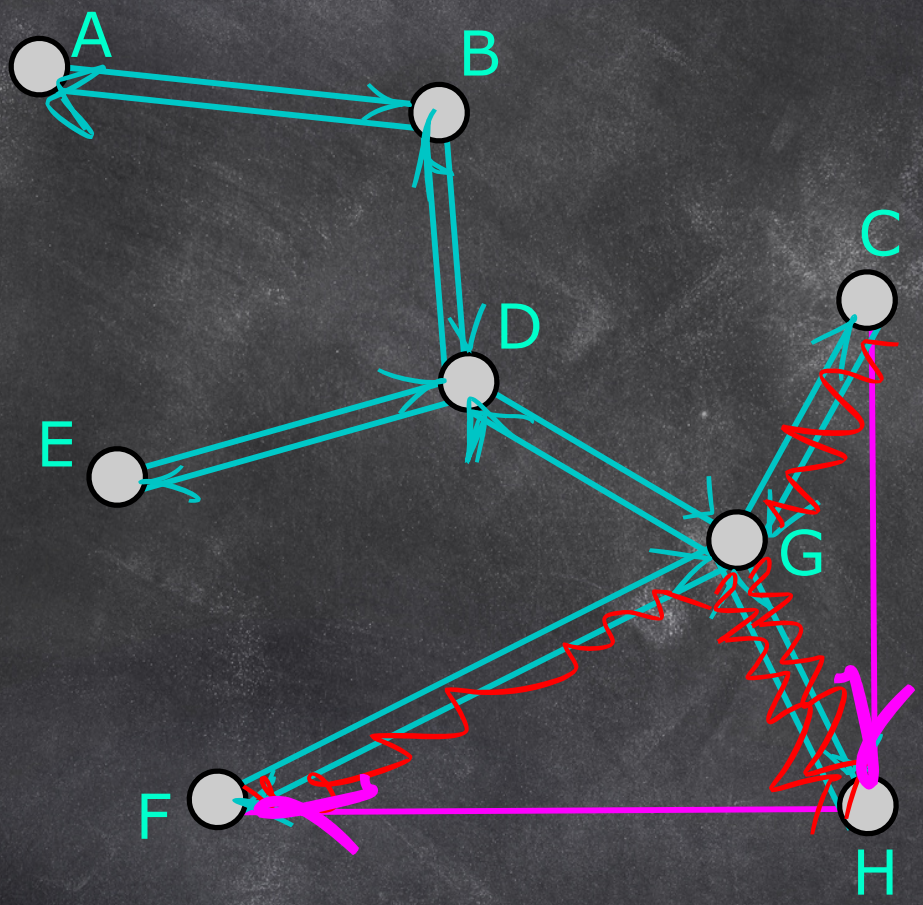
Obtém ciclo Hamiltoniano

A, **B**, **D**, **G**, **C**, G, **H**, G, F, G, D, E, D, B, A

Luís Felipe

09/06/26

Exemplo de execução – RSL



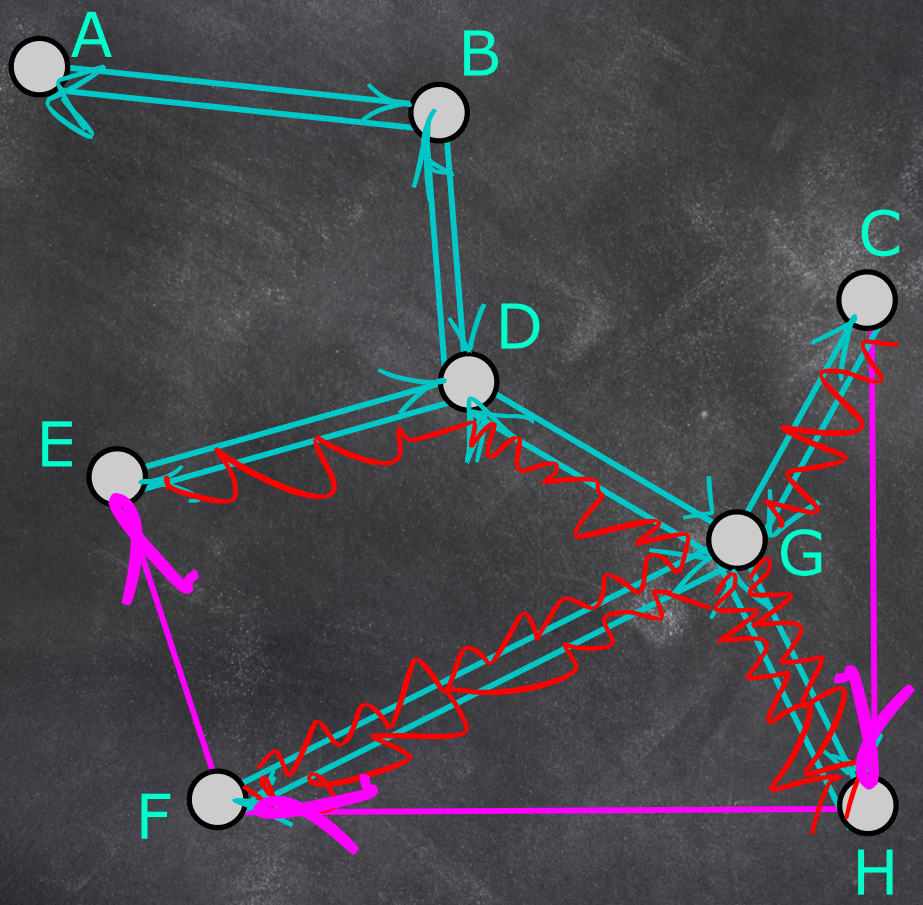
Obtém ciclo Hamiltoniano

A, **B**, **D**, **G**, **C**, G, **H**, G, **F**, G, D, E, D, B, A

Luís Felipe

09/06/26

Exemplo de execução – RSL



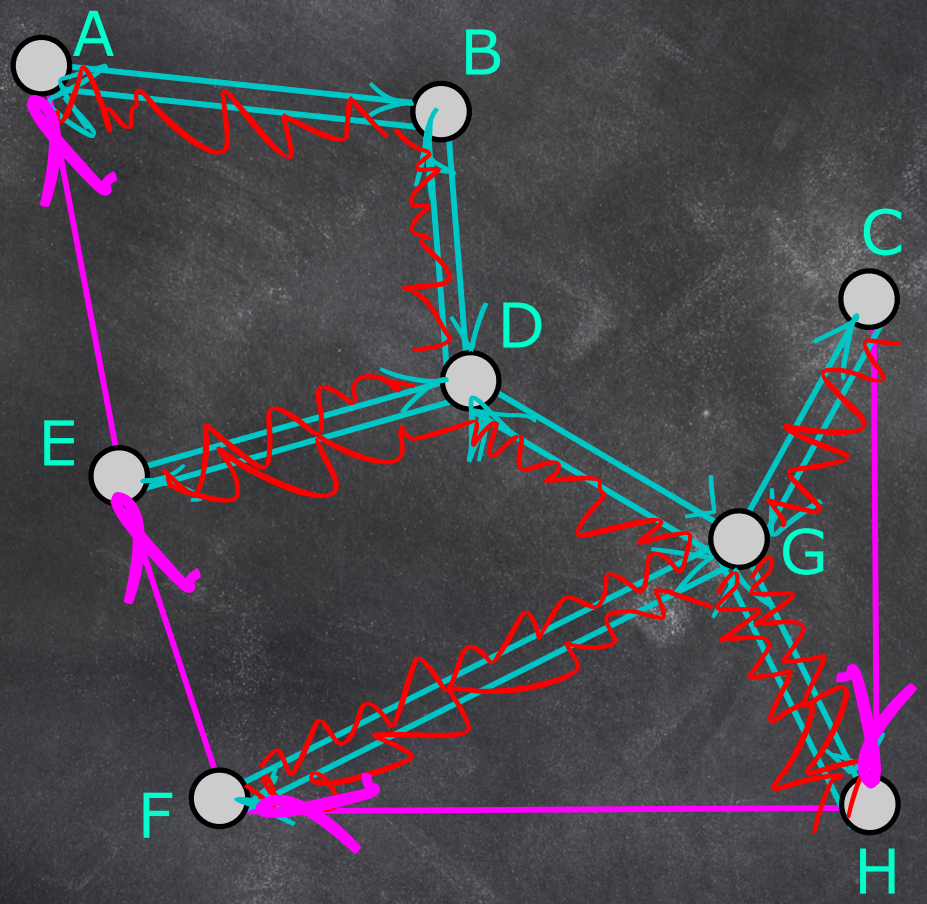
Obtém ciclo Hamiltoniano

A, **B**, **D**, **G**, **C**, G, **H**, G, **F**, G, D, **E**, D, B, A

Luís Felipe

09/06/26

Exemplo de execução – RSL



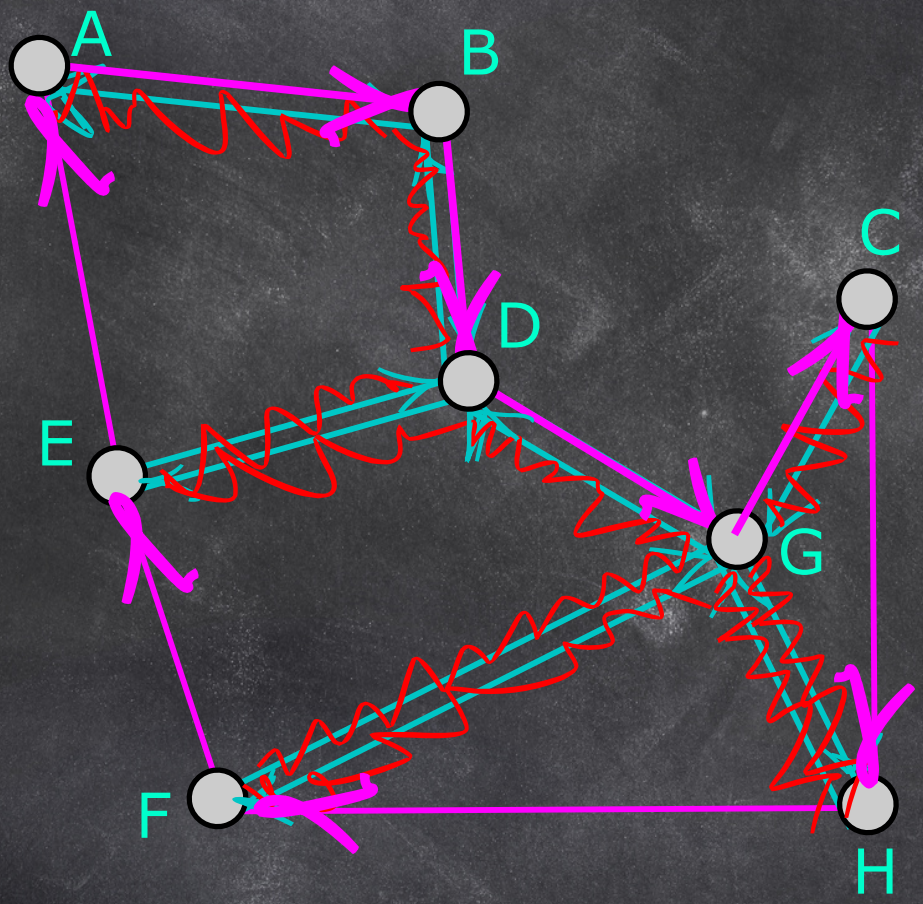
Obtém ciclo Hamiltoniano

A, **B**, **D**, **G**, **C**, G, **H**, G, **F**, G, D, **E**, D, B, **A**

Luís Felipe

09/06/26

Exemplo de execução – RSL



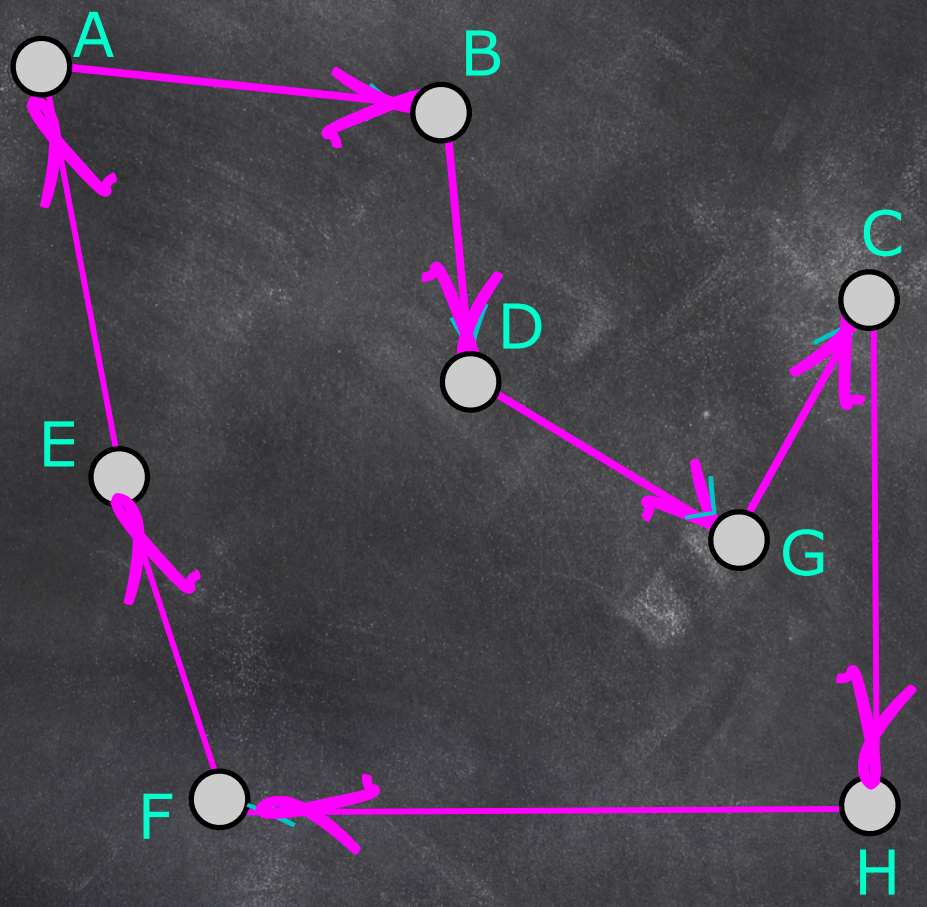
Obtém ciclo Hamiltoniano

A, **B**, **D**, **G**, **C**, G, **H**, G, **F**, G, D, **E**, D, B, **A**

Luís Felipe

09/06/26

Exemplo de execução



Obtém ciclo Hamiltoniano

A, B, D, G, C, H, F, E, A

Luís Felipe

09/06/26

Python: Prim para MST

```python
import heapq


def prim_mst(vertices, c):
    # vertices: lista de vertices
    # c[u][v]: custo da aresta uv

    s = vertices[0]
    visitado = set([s])
    fila = []
    T = []

    for v in vertices:
        if v != s:
            heapq.heappush(fila, (c[s][v], s, v))

    while len(visitado) < len(vertices):
        peso, u, v = heapq.heappop(fila)

        if v in visitado:
            continue

        visitado.add(v)
        T.append((u, v))

        for w in vertices:
            if w not in visitado:
                heapq.heappush(fila, (c[v][w], v, w))

    return T
```

Luís Felipe

09/06/26

Python: grafo duplicado e atalho

```python
def duplicar_arestas(arestas):
    # devolve duas copias de cada aresta

    H = []

    for u, v in arestas:
        H.append((u, v))
        H.append((u, v))

    return H


def atalho(passeio):
    # transforma passeio fechado em ciclo Hamiltoniano

    visitados = set()
    ciclo = []

    for v in passeio:
        if v not in visitados:
            visitados.add(v)
            ciclo.append(v)

    ciclo.append(ciclo[0])
    return ciclo
```

Luís Felipe

09/06/26

Python: algoritmo RSL

```python
def rsl_tsp(vertices, c):
    # G e completo e metrico

    T = prim_mst(vertices, c)

    H = duplicar_arestas(T)

    W = trilha_euleriana(H)
    # usa o algoritmo de Hierholzer visto na aula anterior

    C = atalho(W)

    return C


def custo_ciclo(C, c):
    total = 0

    for i in range(len(C) - 1):
        total = total + c[C[i]][C[i + 1]]

    return total
```

Observação: a função `trilha_euleriana` é a implementação de Hierholzer da aula anterior (Aula 17).

Luís Felipe

09/06/26

Fator de aproximação e complexidade

- Sejam:
  $c(T)$ o custo da árvore geradora mínima,
  $c(\text{ótimo})$ o custo ótimo do Caixeiro Viajante e
  $c(A)$ o custo do algoritmo RSL
  - $c(T) \leq c(\text{ótimo})$
  - Note que $c(A) \leq 2c(T)$ pela desigualdade triangular. Basicamente, $c(A)$ na primeira etapa de A é igual a $2c(T)$, pela obtenção do ciclo Hamiltoniano do multigrafo na cópia das arestas de T, depois reduzimos $c(A)$ pelo "corte de caminho" garantido da desigualdade triangular na adição de novas arestas
  - Como $2c(T) \leq 2c(\text{ótimo})$, temos $c(A) \leq 2c(\text{ótimo})$. Portanto, $\frac{c(a)}{c(\text{ótimo})} \leq 2$.
  - O algoritmo RSL é 2-aproximativo
  - Complexidade: $O(n^2)$ após ter obtido a MST.

Luís Felipe

09/06/26

Algoritmo de Christofides

- Obtém-se uma árvore geradora mínima
- Toma-se um emparelhamento perfeito mínimo no grafo sobre o conjunto dos vértices de grau ímpar e adicionam-se tais arestas à arvore
  - Sempre temos um emparelhamento perfeito? Sim, pois o grafo é completo e o número de vértices de grau ímpar em qualquer grafo é par (Teoria dos Grafos)
  - Consequência da adição de arestas: Grafo Euleriano
- Toma-se um circuito Euleriano
  - A partir dele, constrói-se um ciclo Hamiltoniano substituindo as repetições de vértices por caminhos "diretos"
  - Assim, arestas que não estão no circuito Euleriano serão inseridas.
  - Note que elas sempre existem, pois o grafo original é completo.

Luís Felipe

09/06/26

Algoritmo de Christofides

- Obtém-se uma árvore geradora mínima T.
- Seja I o conjunto de vértices que têm grau ímpar em T.
- Encontramos um emparelhamento perfeito de custo mínimo em $G[I]$.
- Adicionamos as arestas desse emparelhamento à árvore T.
- O multigrafo resultante tem todos os graus pares; logo, é Euleriano.
- Tomamos um circuito Euleriano e aplicamos atalhos para obter um ciclo Hamiltoniano.

Luís Felipe

09/06/26

Christofides em pseudocódigo

```
1  funcao Christofides(G, c):
2
3      T <- arvore geradora minima de G
4
5      I <- vertices de grau impar em T
6
7      M <- emparelhamento perfeito de custo minimo em G[I]
8
9      H <- multigrafo com arestas de T mais arestas de M
10
11     W <- circuito euleriano de H
12
13     C <- ciclo obtido de W por atalhos
14
15     retornar C
```

A diferença para o RSL é que não duplicamos toda a árvore. Adicionamos apenas um emparelhamento perfeito sobre os vértices ímpares de T.

Luís Felipe

09/06/26

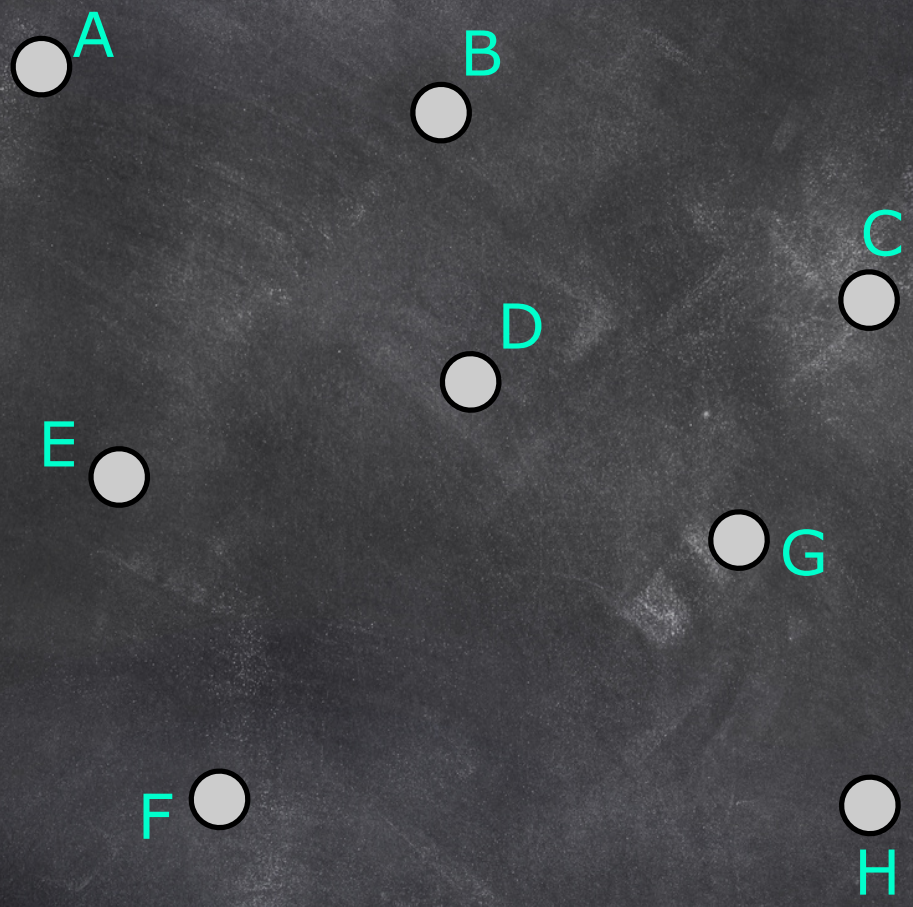
Subrotinas utilizadas

- MST: usaremos Prim para obter uma árvore geradora mínima.
- Circuito Euleriano: usaremos Hierholzer, visto na aula anterior.
- Atalhos: removeremos repetições de vértices usando a desigualdade triangular.
- Emparelhamento perfeito mínimo: Usaremos essa subrotina sobre os vértices de graus ímpares da MST.

Luís Felipe

09/06/26

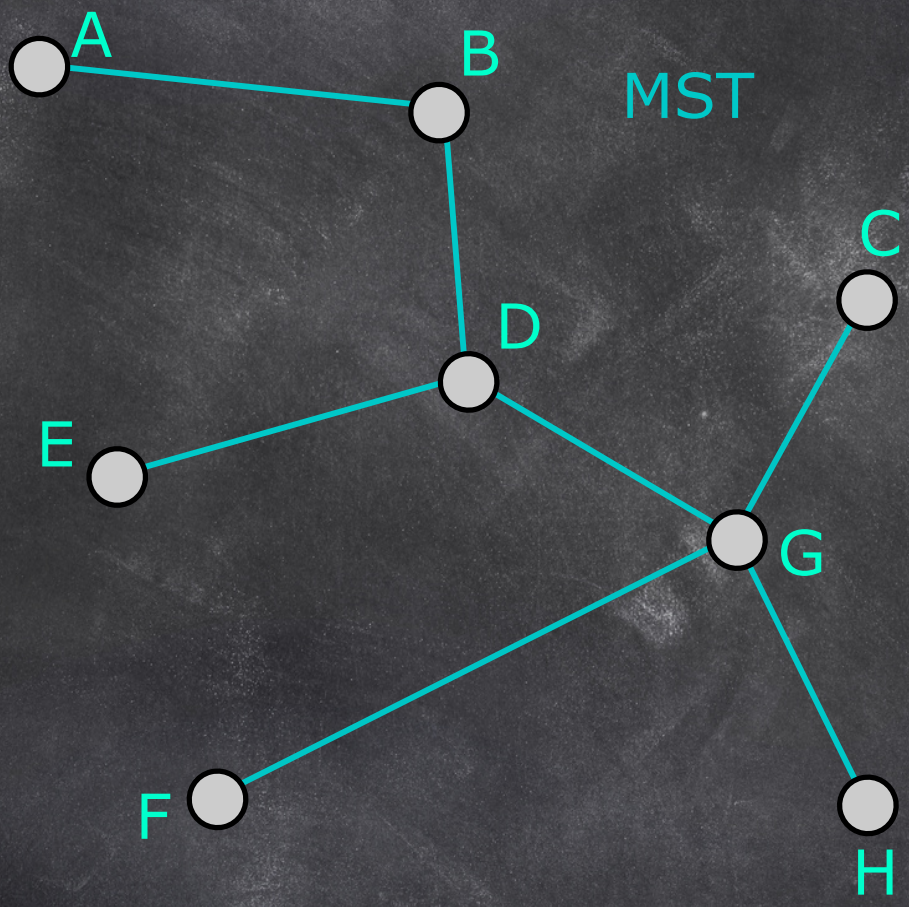
Exemplo de execução – Christofides



Luís Felipe

09/06/26

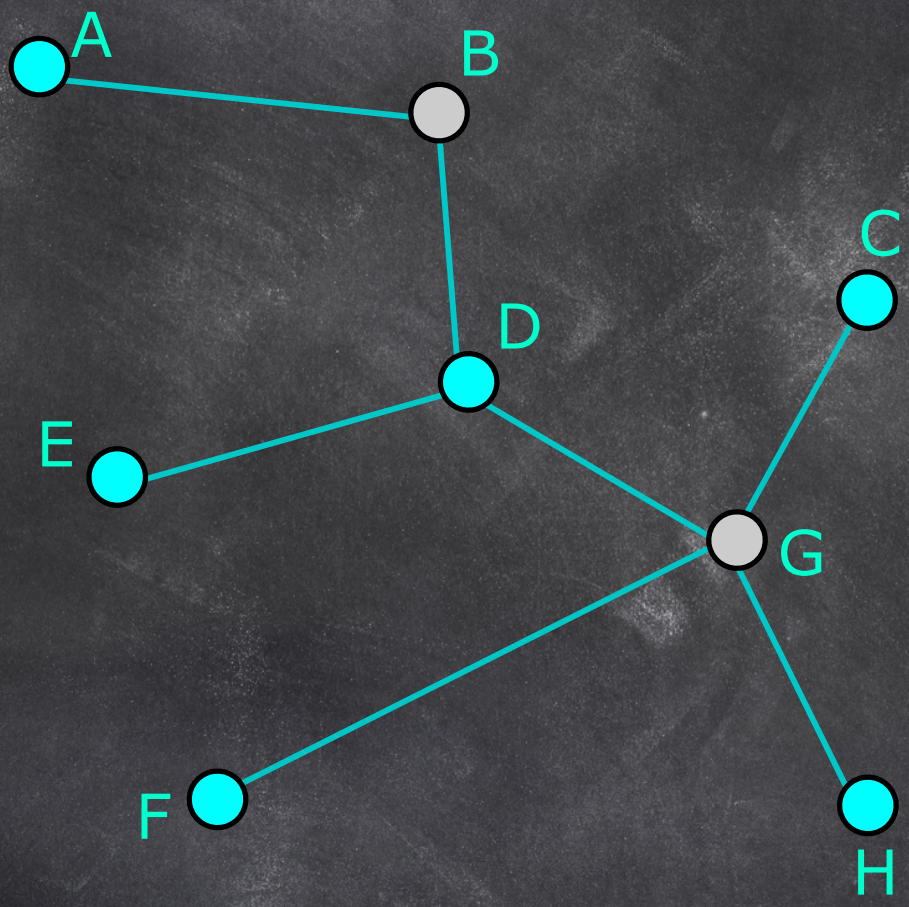
Exemplo de execução – Christofides



Luís Felipe

09/06/26

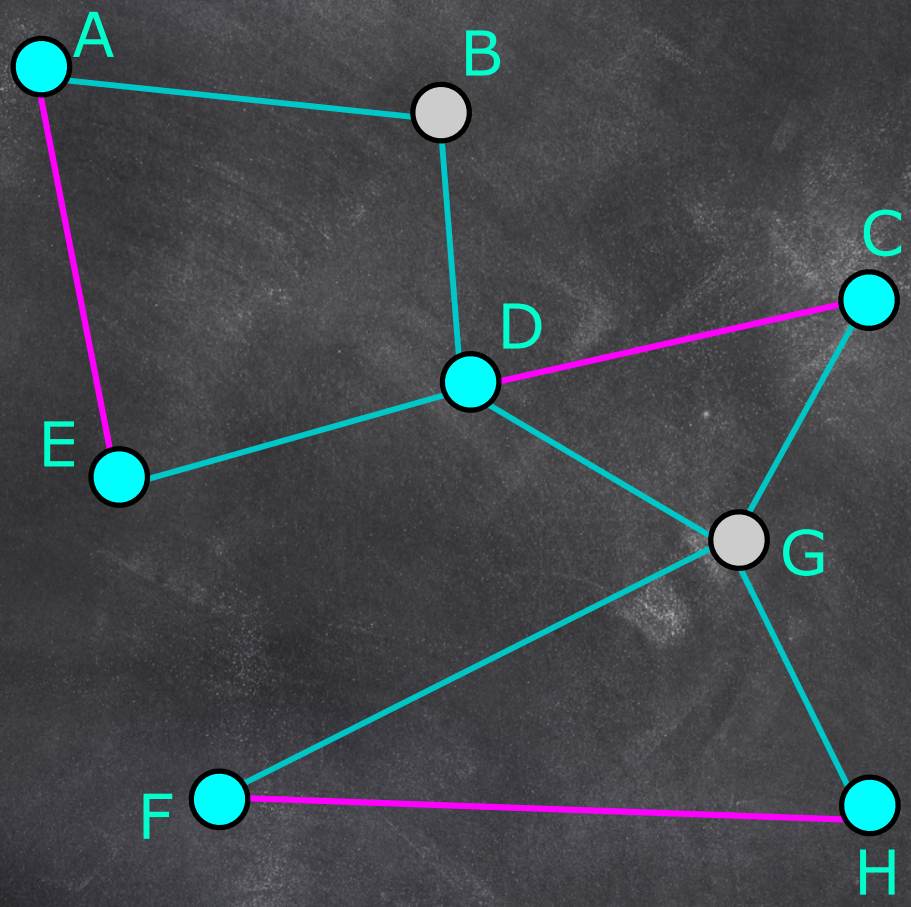
Exemplo de execução – Christofides



Luís Felipe

09/06/26

Exemplo de execução – Christofides

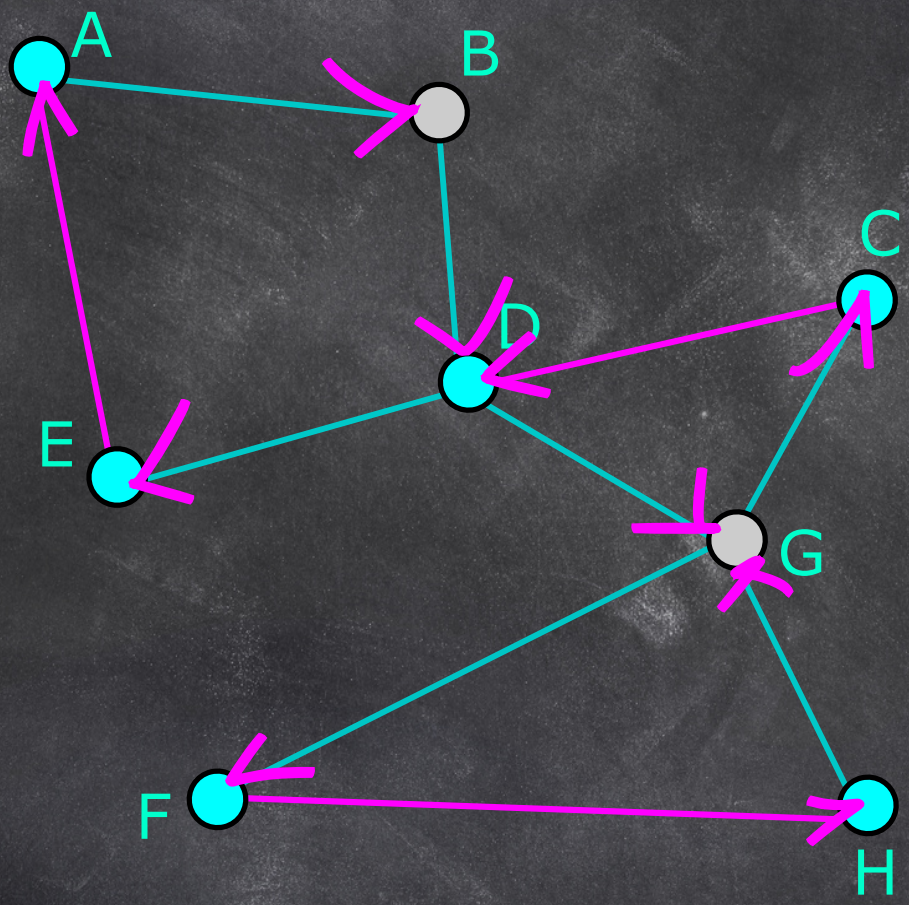


Emparelhamento perfeito de custo mínimo
sobre o subgrafo dos vértices de grau ímpar

Luís Felipe

09/06/26

Exemplo de execução – Christofides

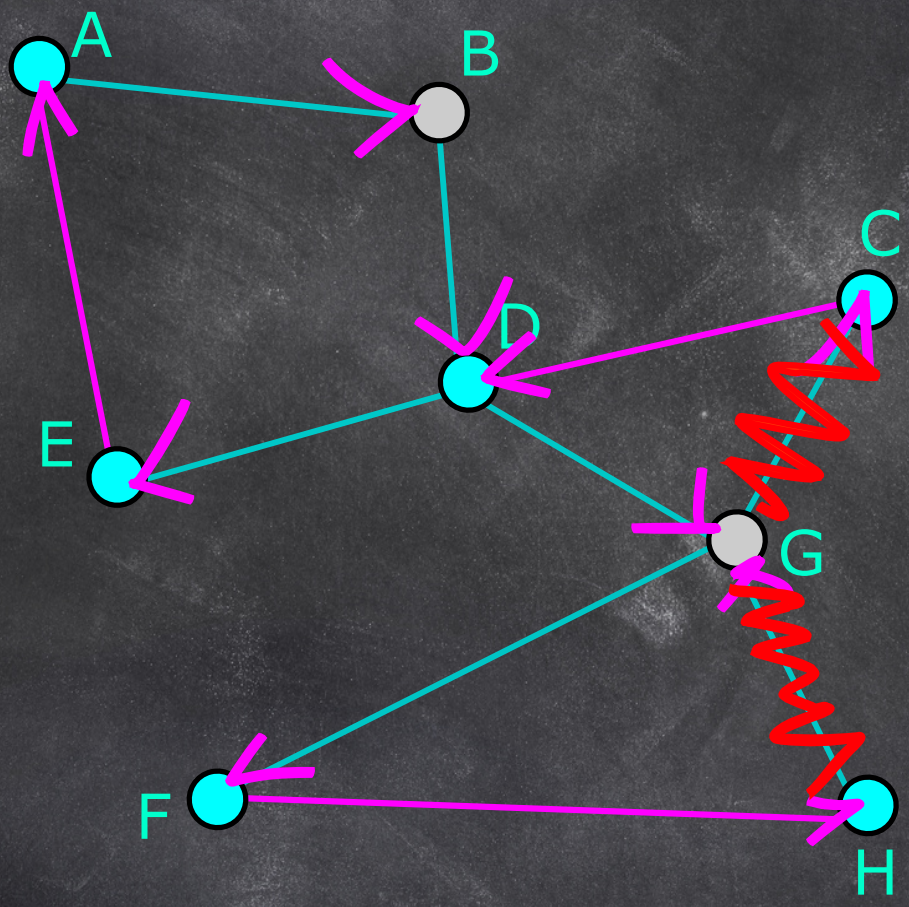


Circuito Euleriano de todo o grafo
A, B, D, G, F, H, G, C, D, E, A

Luís Felipe

09/06/26

Exemplo de execução – Christofides



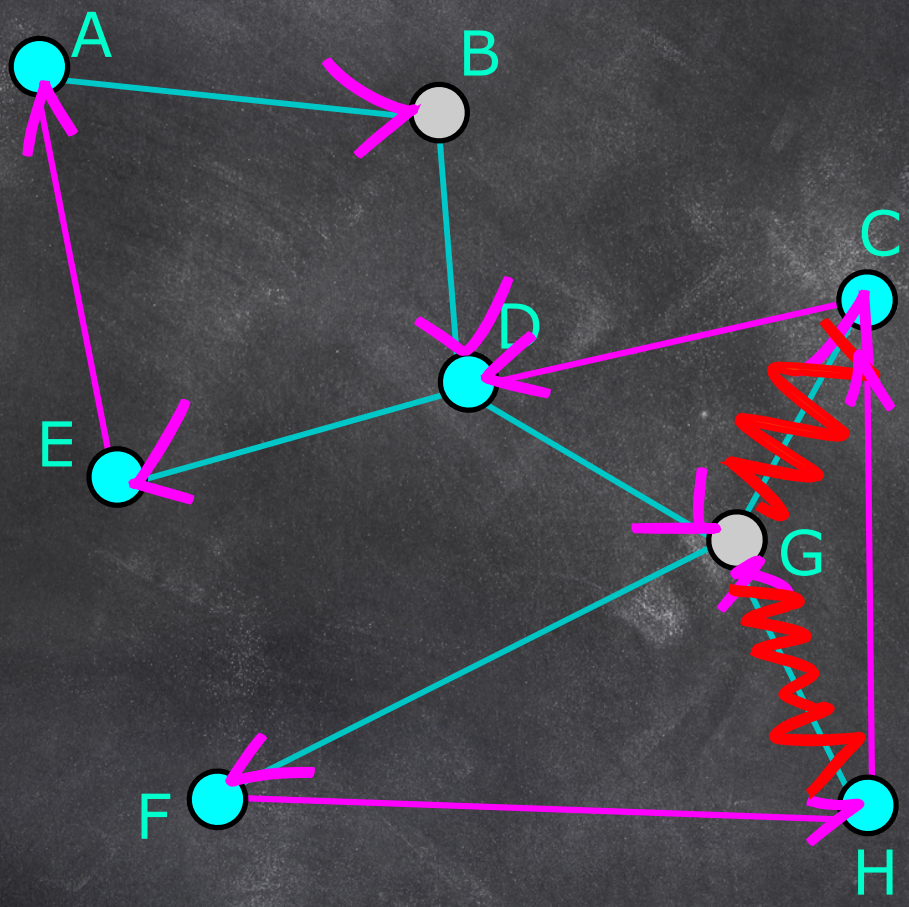
Ciclo Hamiltoniano resultante

A, **B**, **D**, **G**, **F**, **H**, G, C, D, E, A

Luís Felipe

09/06/26

Exemplo de execução – Christofides



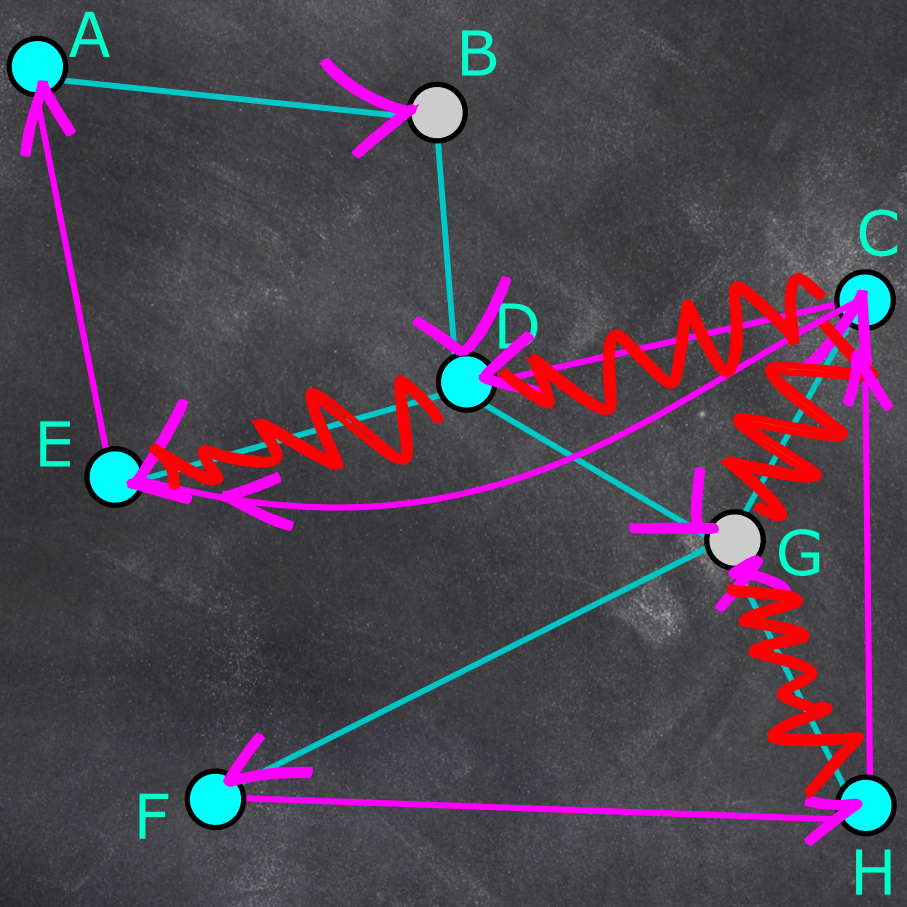
Ciclo Hamiltoniano resultante

A, **B**, **D**, **G**, **F**, **H**, G, **C**, D, E, A

Luís Felipe

09/06/26

Exemplo de execução – Christofides



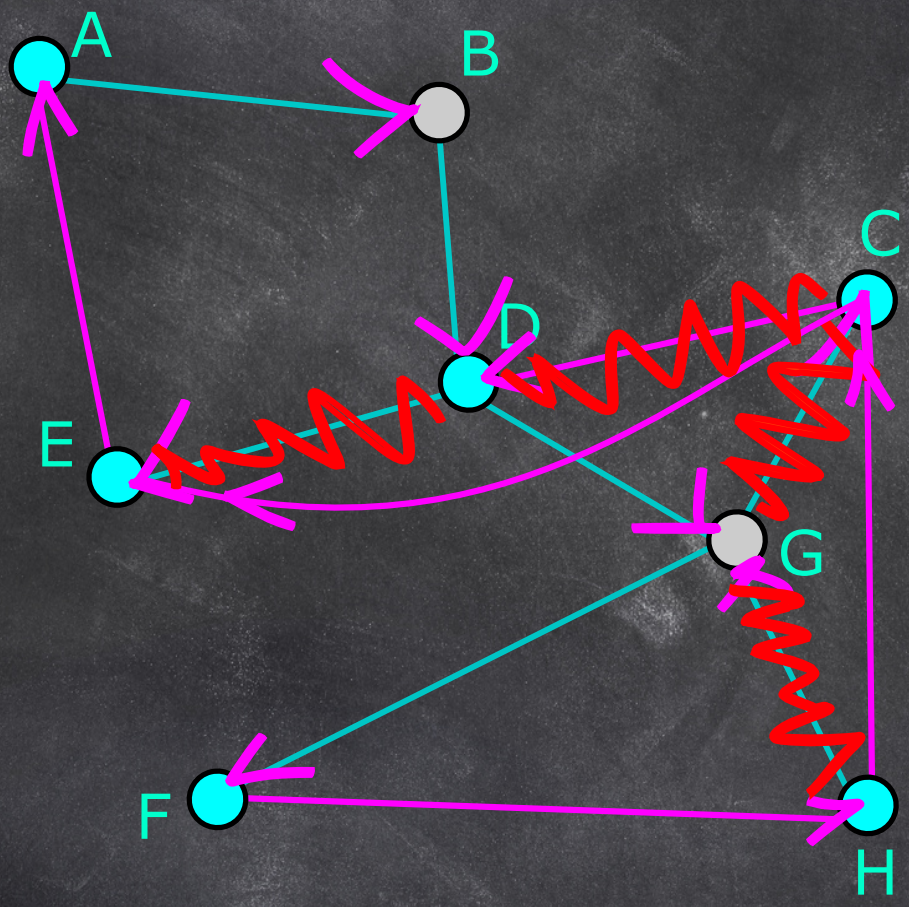
Ciclo Hamiltoniano resultante

A, **B**, **D**, **G**, **F**, **H**, G, **C**, D, **E**, A

Luís Felipe

09/06/26

Exemplo de execução – Christofides



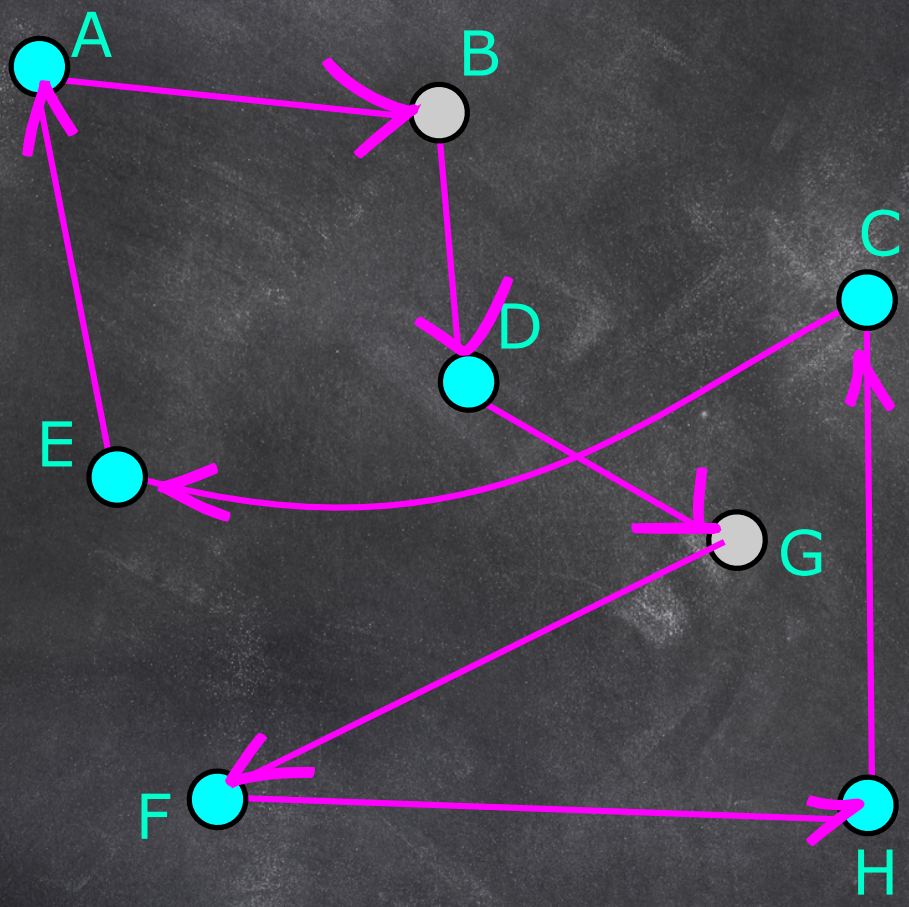
Ciclo Hamiltoniano resultante

A, **B**, **D**, **G**, **F**, **H**, G, **C**, D, **E**, **A**

Luís Felipe

09/06/26

Exemplo de execução – Christofides



Ciclo Hamiltoniano resultante

A, B, D, G, F, H, C, E, A

Luís Felipe

09/06/26

Python: vértices ímpares da MST

```python
def vertices_impares_da_arvore(vertices, T):
    grau = {}

    for v in vertices:
        grau[v] = 0

    for u, v in T:
        grau[u] = grau[u] + 1
        grau[v] = grau[v] + 1

    impares = []

    for v in vertices:
        if grau[v] % 2 == 1:
            impares.append(v)

    return impares
```

Luís Felipe

09/06/26

Python: emparelhamento mínimo por DP

A recorrência implementada é:

$$DP(S) = \begin{cases} 0, & \text{se } S = \emptyset, \\ \min\limits_{v \in S \setminus \{u\}} \{c(u,v) + DP(S \setminus \{u,v\})\}, & \text{caso contrário,} \end{cases}$$

onde u é um vértice fixo de S.

```
18  def emparelhamento_minimo_dp(impares, c): # versao útil para quando há poucos vertices impares
19      memo = {}
20      def resolve(restantes):
21          restantes = tuple(sorted(restantes))
22          if len(restantes) == 0:
23              return 0, []
24          if restantes in memo:
25              return memo[restantes]
26          u = restantes[0]
27          melhor_custo = float("inf")
28          melhor_pares = []
29
30          for i in range(1, len(restantes)):
31              v = restantes[i]
32              novos = restantes[1:i] + restantes[i + 1:]
33              custo, pares = resolve(novos)
34              custo = custo + c[u][v]
35              if custo < melhor_custo:
36                  melhor_custo = custo
37                  melhor_pares = [(u, v)] + pares
38
39          memo[restantes] = (melhor_custo, melhor_pares)
40          return memo[restantes]
41
42      return resolve(tuple(impares))
```

Luís Felipe

09/06/26

Python: algoritmo de Christofides

```
43  def christofides_tsp(vertices, c):
44      # G eh um grafo completo que satisfaz a restrição métrica
45
46      T = prim_mst(vertices, c)
47
48      impares = vertices_impares_da_arvore(vertices, T)
49
50      custo_M, M = emparelhamento_minimo_dp(impares, c)
51      # para muitos vertices, usar Edmonds ponderado
52
53      H = []
54
55      for e in T:
56          H.append(e)
57
58      for e in M:
59          H.append(e)
60
61      W = trilha_euleriana(H)
62
63      C = atalho(W)
64
65      return C
```

Observação: a DP é exponencial no número de vértices ímpares. Para a complexidade $O(n^3)$, usa-se emparelhamento perfeito de custo mínimo por Edmonds ponderado.

Luís Felipe

09/06/26

Fator de aproximação e complexidade

- Sejam:
  $c(T)$ o custo da árvore geradora mínima,
  $c(M)$ o custo do emparelhamento perfeito,
  $c(\text{ótimo})$ o custo ótimo e
  $c(A)$ o custo do algoritmo de Christofides
  - $c(T) \leq c(\text{ótimo})$
  - $c(M) \leq \frac{c(\text{ótimo})}{2}$, pois usamos no máximo metade das arestas do custo ótimo no emparelhamento, dado que usamos no máximo $\frac{|V|}{2}$ arestas e no ciclo temos $|V|$ arestas
  - $c(A) \leq c(T) + c(M)$, pela desigualdade triangular e pelo fato de usarmos arestas de T e de M
  - Logo, $c(A) \leq c(\text{ótimo}) + \frac{c(\text{ótimo})}{2} = \frac{3}{2} c(\text{ótimo})$
  - O algoritmo de Christofides é 1.5-aproximativo
  - Complexidade: $O(n^3)$ pelo algoritmo para obter emparelhamento perfeito mínimo (Algoritmos de Edmonds)