Aula 17 - Grafos Eulerianos e o problema do Carteiro Chinês

Luís Felipe

UFF

02 de Junho de 2026

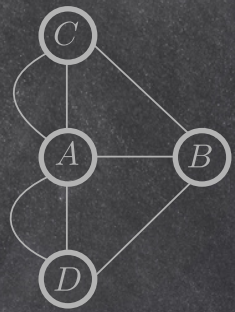
Luís Felipe

02/06/26

Grafos Eulerianos

Um circuito de Euler é um passeio fechado que visita cada aresta do grafo exatamente uma vez.

Lembram do Problema das 7 pontes de Konigsberg?



Existe circuito de Euler no grafo associado a este mapa? Não!

Luís Felipe

02/06/26

Grafos Eulerianos - Caracterização

Teorema: Um grafo conexo não trivial é Euleriano se, e somente se, o grau de cada um dos vértices é par.

Prova: ($\rightarrow$) Suponha G um grafo Euleriano conexo e não trivial (diferente de K_1).
Seja C um passeio de Euler com início e fim no vértice u. Cada vez que um vértice v aparece no passeio de C, duas arestas incidentes a v são contadas (uma vindo para v e outra saindo de v).
Como um passeio de Euleriano passa cada aresta exatamente uma vez, então para todo $v \neq u$, $d(v)$ é par.
Similarmente, $d(u)$ é par, pois C começa e termina em u. Outra forma de verificar que $d(u)$ é par: Há número par de vértices de grau ímpar (Consequência do Lema do Aperto de Mãos) e todo vértice $v \neq u$ possui grau par, então u só pode possuir grau par.

Luís Felipe

02/06/26

Grafos Eulerianos - Continuação da prova

($\leftarrow$) Suponha G é conexo não trivial onde todo vértice tem grau par. Vamos construir um passeio de Euler em duas etapas:

Epata 1: Decompor as arestas de G em ciclos;

Etapa 2: Compor o passeio de Euler a partir dos ciclos da Etapa 1.

Na Etapa 1, começamos por identificar um ciclo C_1 no grafo (não necessariamente gerador). Este ciclo C_1 existe porque todos os vértices possuem grau par. Podemos então repetir a operação em relação a uma componente conexa não trivial deste grafo. Obtemos assim a decomposição.

Luís Felipe

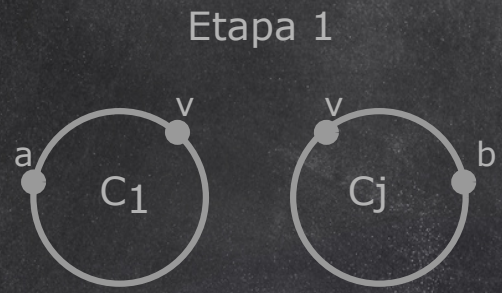
02/06/26

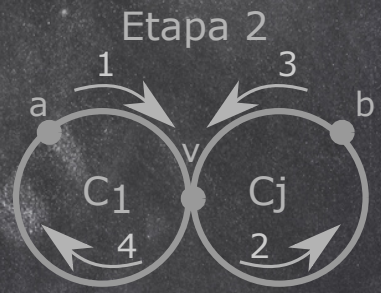
Grafos Eulerianos - Conclusão da prova

Etapa 2: Compor o passeio de Euler a partir dos ciclos da Etapa 1.

Na Etapa 2, começando com o ciclo C_1, compomos C_1 com um ciclo C_j que tenha vértice em comum com C_1. O ciclo C_j existe, porque o grafo é conexo.

A composição é realizada enquanto houver ciclos da Etapa 1 não presente nesta etapa.





Tomando, portanto, vértice a e v em C_1, tal que v também pertença a C_j, que por sua vez possua um outro vértice b. Percorramos entre C_1 e C_j tal como na figura acima. Note que este é um passeio de Euler associado a C_1 e C_j.

Após isto, caso exista outro ciclo C_k, façamos o mesmo procedimento compondo C_k ao C_1C_j.

Luís Felipe

02/06/26

Trilha e circuito de Euler

- Uma trilha de Euler aberta é uma trilha aberta que usa cada aresta exatamente uma vez.
- Em um circuito de Euler, o vértice inicial e o final coincidem.
- Em uma trilha de Euler aberta, o vértice inicial e o final são distintos.
- O teorema de Euler para circuitos diz:

$$G \text{ conexo é Euleriano} \iff \text{todo vértice de } G \text{ tem grau par.}$$

- Para trilhas abertas, a condição muda para exatamente dois vértices de grau ímpar.

Luís Felipe

02/06/26

Trilha de Euler aberta

Teorema: Um grafo conexo G possui uma trilha de Euler aberta se, e somente se, G possui exatamente dois vértices de grau ímpar.

Ideia da prova:

- Se existe uma trilha aberta de Euler de u até v, então u e v são os únicos vértices onde o número de entradas e saídas não se equilibra.
- Logo, u e v têm grau ímpar.
- Todo outro vértice é visitado entrando por uma aresta e saindo por outra, portanto tem grau par.
- Para a volta, sejam u e v os dois vértices de grau ímpar.
- Adicione uma nova aresta uv. Agora todos os vértices têm grau par.
- O grafo resultante possui circuito de Euler. Removendo a aresta uv, obtemos uma trilha de Euler aberta em G.

Luís Felipe

02/06/26

Como encontrar uma trilha de Euler?

- A caracterização diz quando a trilha existe.
- Agora queremos um algoritmo que, dado G, encontre uma trilha ou circuito de Euler.
- A prova já sugere uma ideia: decompor o grafo em ciclos e depois concatenar os ciclos.
- Um algoritmo muito usado é o algoritmo de Hierholzer.
- Ideia: Ele constrói uma trilha fechada inicial e, enquanto houver arestas não usadas incidentes a algum vértice da trilha, insere um novo ciclo naquele ponto.

Luís Felipe

02/06/26

Algoritmo de Hierholzer: ideia

Entrada: grafo conexo G em que todo vértice tem grau par.
Saída: circuito de Euler de G.

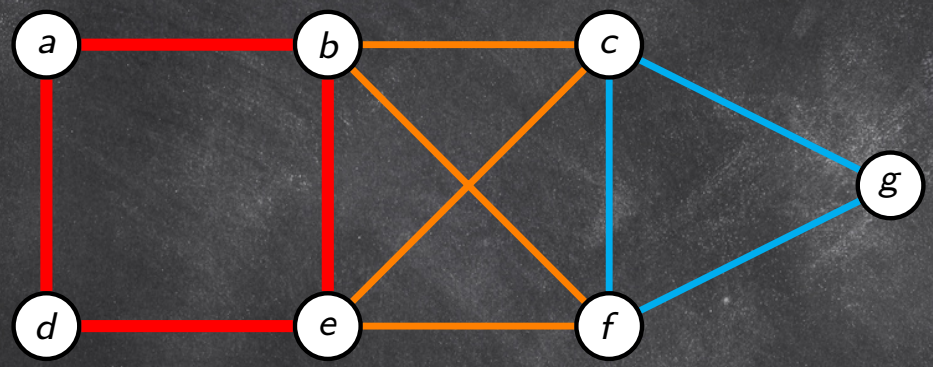
- Passo 1: ciclo inicial
  - Escolha um vértice inicial v.
  - Percorra arestas não usadas até retornar a v.
  - Seja C o ciclo obtido.
- Passo 2: expansão do ciclo
  - Enquanto existir aresta não usada:
    - escolha um vértice x de C incidente a alguma aresta não usada;
    - construa, a partir de x, um novo ciclo C_x usando apenas arestas não usadas;
    - insira C_x em C no vértice x.
- Saída
  - Quando todas as arestas forem usadas, C é um circuito de Euler.

Usamos uma pilha para percorrer os vértices dos ciclos e montar o circuito.

Luís Felipe

02/06/26

Exemplo: ciclos a serem concatenados



Este grafo pode ser visto como a união dos ciclos:

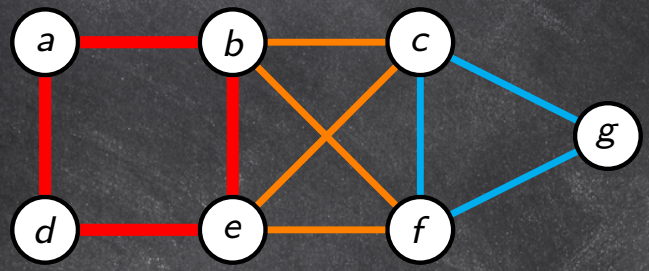
$$C_1 = a, b, e, d, a, \qquad C_2 = b, c, e, f, b, \qquad C_3 = c, g, f, c.$$

Como eles compartilham vértices, podemos concatená-los para obter um único circuito de Euler.

Luís Felipe

02/06/26

Execução do Hierholzer no exemplo



Começamos em a e seguimos por arestas ainda não usadas: a, b, e, d, a.
Nesse momento, a pilha é:

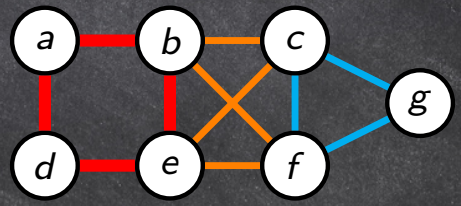
$$[a, b, e, d, a].$$

Como o topo a não possui mais arestas não usadas, removemos a da pilha e adicionamos a ao "circuito" (lista que está sendo montada).
O novo topo é d. Como d também não possui mais arestas não usadas, removemos d e adicionamos d ao circuito.
Agora o topo é e, temos que e ainda possui arestas não usadas. Então a busca continua a partir de e.

Luís Felipe

02/06/26

Execução do Hierholzer no exemplo



Até agora: pilha $= [a, b, e]$ circuito $= [a, d]$.

O topo da pilha é e, que ainda possui arestas não usadas. Avançamos para c. Em c, também há mais de uma aresta disponível. Suponha que escolhemos primeiro cg, então percorremos: c, g, f, c.
Ao retornar a c, ainda resta a aresta cb. Continuamos então por: c, b, f, e
Portanto, a pilha tona-se: $[a, b, e, c, g, f, c, b, f, e]$.
Depois:
A partir desse momento, não há mais arestas não usadas. Assim, os vértices são removidos da pilha e adicionados à saída invertida na ordem: $e, f, b, c, f, g, c, e, b, a$

Como já tínhamos $[a, d]$, obtemos: $[a, d, e, f, b, c, f, g, c, e, b, a]$.
Invertendo: $[a, b, e, c, g, f, c, b, f, e, d, a]$.

Luís Felipe

02/06/26

Observação sobre a pilha

Na versão com pilha, o circuito não é construído enquanto avançamos.

Ele é construído quando não há mais como avançar a partir do topo da pilha.

Por isso, os vértices entram no circuito durante o desempilhamento.

Ao final, invertemos a lista obtida, se quisermos recuperar o circuito na orientação correspondente à exploração realizada.

$$\text{circuito final} = \text{reverse}(\text{ordem de desempilhamento}).$$

A concatenação de ciclos acontece implicitamente pela pilha.

Luís Felipe

02/06/26

Hierholzer em pseudocódigo

```
1   funcao hierholzer(G):
2
3       escolha um vertice inicial s com grau positivo
4
5       pilha <- [s]
6       circuito <- lista vazia
7
8       enquanto pilha nao esta vazia faca
9
10          v <- topo da pilha
11
12          se v possui aresta nao usada vw entao
13
14              marque vw como usada
15              empilhe w
16
17          senao
18              // v nao consegue mais continuar no grafo restante
19              desempilhe v
20              adicione v ao final de circuito
21
22      retornar circuito invertido
```

A pilha guarda o passeio que está sendo explorado. Quando o topo da pilha não tem mais arestas disponíveis, ele é fechado e colocado no circuito.

Se algum vértice anterior da pilha ainda tiver arestas não usadas, a busca continua a partir dele. Isso corresponde a inserir um novo ciclo naquele vértice.

Luís Felipe

02/06/26

Por que o algoritmo funciona?

- Enquanto seguimos por arestas não usadas, nunca repetimos uma aresta.
- Se chegamos a um vértice e não há mais arestas não usadas saindo dele, ele pode ser finalizado no circuito.
- Em um grafo Euleriano, não ficamos presos antes de fechar uma parte do passeio, pois todos os graus são pares.
- A pilha permite fazer exatamente a concatenação de ciclos da prova do Teorema de Euler.
- No final, cada aresta foi usada exatamente uma vez.
- Portanto, o algoritmo retorna um circuito de Euler.

Luís Felipe

02/06/26

Como escolher o vértice inicial?

- Se todos os vértices têm grau par, queremos um circuito de Euler.
- Nesse caso, podemos começar em qualquer vértice com grau positivo.

- Se existem exatamente dois vértices de grau ímpar, queremos uma trilha de Euler aberta.
- Nesse caso, devemos começar em um dos vértices de grau ímpar.
- A trilha terminará no outro vértice de grau ímpar.

- Se o número de vértices de grau ímpar é diferente de 0 e de 2, não existe trilha de Euler.

Luís Felipe

02/06/26

Teste de existência

```
1  funcao existe_trilha_euleriana(G):
2
3      se os vertices de grau positivo nao estao na mesma componente entao
4          retornar falso
5
6      impares <- vertices de grau impar
7
8      se |impares| = 0 entao
9          retornar "existe circuito de Euler"
10
11     se |impares| = 2 entao
12         retornar "existe trilha de Euler aberta"
13
14     retornar falso
```

Obs.: vértices isolados não atrapalham. A conectividade relevante é na parte do grafo que possui arestas.

Luís Felipe

02/06/26

Implementação em Python: preparação

```python
# G e uma lista de adjacencias: G[v] contem os vizinhos de v

def graus(G):
    return {v: len(G[v]) for v in G}


def vertices_impares(G):
    d = graus(G)
    impares = []

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

    return impares


def vertice_inicial(G):
    impares = vertices_impares(G)

    if len(impares) == 2:
        return impares[0]          # trilha aberta

    if len(impares) == 0:
        for v in G:
            if len(G[v]) > 0:
                return v           # circuito
        return None                # grafo sem arestas

    return None                    # nao ha trilha euleriana
```

Luís Felipe

02/06/26

Implementação em Python: Hierholzer

```python
def trilha_euleriana(arestas):
    # arestas: lista de pares (u, v)
    # permite multiarestas

    G = {}
    id_aresta = 0

    for u, v in arestas:
        if u not in G:
            G[u] = []

        if v not in G:
            G[v] = []

        G[u].append((v, id_aresta))
        G[v].append((u, id_aresta))

        id_aresta = id_aresta + 1

    s = vertice_inicial(G)

    if s is None:
        return None

    usadas = [False] * len(arestas)
    pilha = [s]
    trilha = []

    while pilha:

        v = pilha[-1]

        while G[v]:

            w, id_aresta = G[v][-1]

            if not usadas[id_aresta]:
                break

            G[v].pop()

        if G[v]:

            w, id_aresta = G[v].pop()
            usadas[id_aresta] = True
            pilha.append(w)

        else:

            trilha.append(pilha.pop())

    if len(trilha) != len(arestas) + 1:
        return None

    trilha.reverse()
    return trilha
```

Luís Felipe

02/06/26

Complexidade

- O teste dos graus custa $O(n+m)$.
- O teste de conectividade na parte não isolada também custa $O(n+m)$.
- No algoritmo de Hierholzer, cada aresta é removida/usada uma única vez.
- Logo, o tempo total é:

$$O(n+m).$$

- A memória usada também é linear:

$$O(n+m).$$

- Existe outro algoritmo clássico, chamado algoritmo de Fleury, mas ele precisa evitar pontes no grafo resultante durante a construção da trilha.
- Por isso, Fleury é mais simples conceitualmente, mas geralmente menos eficiente que Hierholzer.

Luís Felipe

02/06/26

Algoritmo de Fleury

- O algoritmo de Fleury também constrói uma trilha Euleriana.
- A ideia é escolher as arestas uma a uma, removendo cada aresta depois que ela é usada.
- A regra principal é:

  não escolha uma ponte do grafo restante, exceto se for inevitável.

- Aqui, "grafo restante" significa o grafo formado pelas arestas que ainda não foram usadas.
- Mesmo que o grafo original não tenha pontes, o grafo restante pode ter.
- Por isso, Fleury precisa testar pontes durante a execução. No final, terá complexidade $O(m(n+m))$.

Luís Felipe

02/06/26

Do circuito de Euler ao Carteiro Chinês

- Em um grafo Euleriano, existe um circuito que percorre cada aresta exatamente uma vez.
- Mas, em muitos problemas reais, o grafo não é Euleriano.
- Exemplo: ruas de uma cidade que precisam ser percorridas por um carteiro, caminhão de lixo ou leiturista.
- Queremos sair de um ponto, percorrer todas as ruas e voltar ao ponto de partida.
- Como nem sempre existe circuito de Euler, algumas ruas precisarão ser percorridas mais de uma vez.
- Objetivo: Minimizar o custo total dessas repetições.

Luís Felipe

02/06/26

Problema do Carteiro Chinês

Entrada: um grafo conexo G com pesos não negativos nas arestas.
Objetivo: encontrar um passeio fechado de custo mínimo que percorra todas as arestas de G pelo menos uma vez.

Se G já é Euleriano, a solução é simples:

- encontre um circuito de Euler;
- cada aresta é percorrida exatamente uma vez;
- o custo total é:

$$\sum_{e \in E(G)} w(e).$$

O caso interessante ocorre quando há vértices de grau ímpar.

Luís Felipe

02/06/26

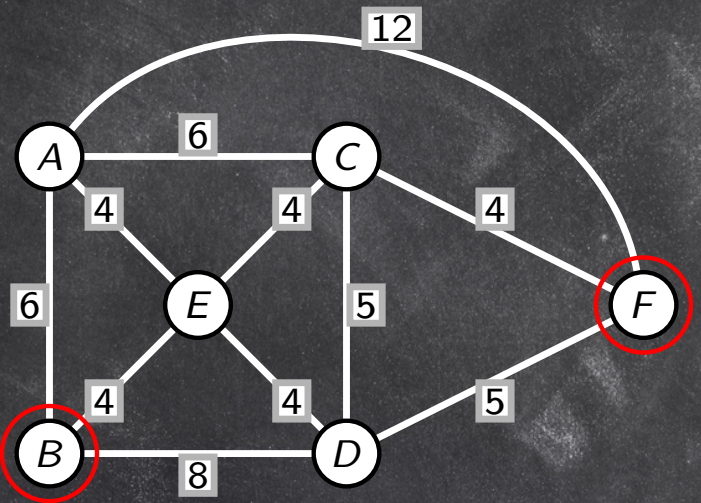
Ideia central do Carteiro Chinês

- Para existir circuito de Euler, todos os vértices precisam ter grau par.
- Se há vértices de grau ímpar, precisamos tornar seus graus pares.
- Percorrer uma aresta mais de uma vez equivale a duplicar essa aresta no grafo.
- Duplicar um caminho entre dois vértices ímpares altera a paridade desses dois extremos.
- Portanto, devemos:
  - parear os vértices ímpares e
  - duplicar caminhos entre as duplas escolhidos.
- Queremos escolher as duplas de modo que o custo adicional seja mínimo.

Luís Felipe

02/06/26

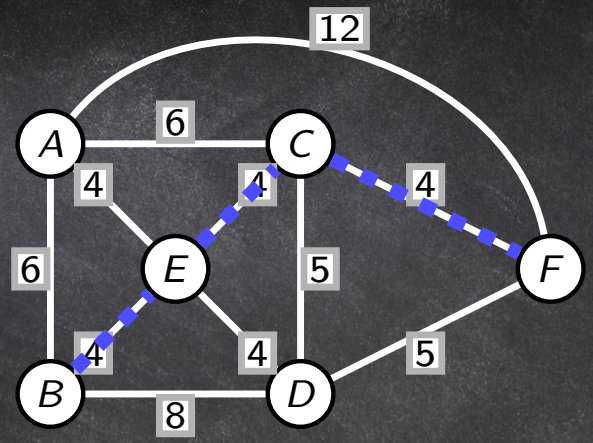
Exemplo pequeno



Neste grafo, os vértices de grau ímpar são B e F.
Para tornar o grafo Euleriano, basta duplicar um caminho mínimo entre B e F.

Luís Felipe

02/06/26



Um caminho mínimo entre B e F é:

$$B, E, C, F,$$

com custo:

$$4 + 4 + 4 = 12.$$

Duplicando esse caminho, todos os graus ficam pares. O multigrafo resultante é Euleriano.
Duplicar esse caminho significa acrescentar uma cópia extra de cada uma das arestas BE, EC e CF.
Assim, os graus de B e F mudam de paridade, enquanto os vértices internos E e C continuam com a mesma paridade.

Luís Felipe

02/06/26

Algoritmo para o Carteiro Chinês

1. Localize os vértices de grau ímpar de G:

$$O = \{v_1, v_2, \ldots, v_k\}.$$

2. Calcule os caminhos mínimos entre todos os pares de vértices de O.
3. Construa um grafo completo auxiliar H sobre os vértices de O.
4. O peso da aresta $v_i v_j$ em H é a distância mínima entre v_i e v_j em G.
5. Encontre em H um emparelhamento perfeito de custo mínimo.
6. Para cada dupla escolhida, duplique em G as arestas do caminho mínimo correspondente.
7. No multigrafo resultante, encontre um circuito de Euler.

Luís Felipe

02/06/26

Carteiro Chinês em pseudocódigo

```
1  funcao carteiro_chines(G, w):
2
3      O <- vertices de grau impar em G
4
5      se O esta vazio entao
6          retornar circuito_euleriano(G)
7
8      para cada u em O faca
9          calcule caminhos minimos de u ate os demais vertices
10
11     construa H completo com V(H) = O
12     peso_H(u,v) <- distancia minima entre u e v em G
13
14     M <- emparelhamento perfeito de custo minimo em H
15
16     para cada dupla uv em M faca
17         P <- caminho minimo entre u e v em G
18         duplique as arestas de P em G
19
20     retornar circuito_euleriano(G aumentado)
```

Luís Felipe

02/06/26

Por que o algoritmo funciona?

- Todo vértice de grau ímpar precisa ter sua paridade alterada.
- Duplicar um caminho altera a paridade exatamente dos seus dois extremos.
- Logo, precisamos parear os vértices ímpares.
- Para cada dupla, o melhor é duplicar um caminho mínimo entre os eles.
- O problema global passa a ser: escolher um pareamento dos vértices ímpares com menor soma de distâncias.
- Isso é exatamente um emparelhamento perfeito de custo mínimo no grafo auxiliar completo.
- Depois das duplicações, todos os graus ficam com graus pares; portanto, existe um circuito de Euler.

Luís Felipe

02/06/26

Custo da solução

Seja G^+ o multigrafo obtido após duplicar os caminhos escolhidos obtidos pelo emparelhamento M do grafo H.

O custo do circuito encontrado em G^+ é:

$$\sum_{e \in E(G)} w(e) + \sum_{uv \in M} \text{dist}_G(u, v).$$

A primeira parcela é inevitável: toda aresta original precisa ser percorrida pelo menos uma vez.
A segunda parcela é o custo adicional das repetições.

Portanto, minimizar o passeio total é equivalente a minimizar o custo dos caminhos duplicados.

Luís Felipe

02/06/26

Complexidade do Carteiro Chinês

Seja k o número de vértices de grau ímpar.

- Encontrar os vértices ímpares custa $O(n+m)$.
- Com pesos não negativos, podemos calcular caminhos mínimos a partir de cada vértice ímpar usando Dijkstra: $O(k(m+n\log n))$.
- Construímos o grafo completo auxiliar H com k vértices.
- Para cada par de vértices ímpares u, v, a aresta uv em H recebe como peso a distância mínima entre u e v em G.
- O passo central é encontrar um emparelhamento perfeito de custo mínimo em H.
- Esse passo pode ser resolvido pelo algoritmo de Edmonds para emparelhamento ponderado, uma extensão do algoritmo das flores.
- Uma cota clássica para esse passo é: $O(k^3)$.
- Após duplicar os caminhos escolhidos, obtemos um multigrafo Euleriano, e o circuito de Euler pode ser encontrado por Hierholzer em tempo linear.
- Assim, uma forma de expressar a complexidade total é: $O(k(m+n\log n)+k^3+m')$, onde m' é o número de arestas do multigrafo aumentado.

Luís Felipe

02/06/26

Observação sobre implementação

- Para a teoria, usamos emparelhamento perfeito de custo mínimo em grafo geral.
- Esse é um problema polinomial, mas sua implementação completa é mais sofisticada.
- Para exemplos pequenos, podemos usar programação dinâmica sobre os vértices ímpares.
- Essa versão tem complexidade:

$$O(2^k k^2),$$

  onde k é o número de vértices ímpares.
- Ela é exponencial em k, mas é simples e funciona bem quando há poucos vértices ímpares.

Luís Felipe

02/06/26

Pareamento mínimo por programação dinâmica

Temos um conjunto I de vértices de grau ímpar.

Queremos parear todos os vértices de I minimizando o custo total.

A ideia da DP é:

- escolha um vértice $u \in I$;
- tente parear u com cada outro vértice $v \in I$;
- resolva recursivamente o problema para $I \setminus \{u, v\}$;
- escolha o par uv que dá o menor custo total.

Se $I = \emptyset$, o custo é zero.

$$DP(S) = \min_{v \in S \setminus \{u\}} \{dist(u, v) + DP(S \setminus \{u, v\})\},$$

onde u é um vértice fixo de S.

Luís Felipe

02/06/26

Pareamento mínimo por DP

```python
def pareamento_minimo_dp(impares, dist):
# impares: lista com os vertices de grau impar;
# dist[u][v]: custo de um caminho minimo entre u e v
    memo = {}

    def resolve(restantes):     # restantes: vertices impares ainda nao pareados
        restantes = tuple(sorted(restantes))
        if len(restantes) == 0:
            return 0, []             # custo zero, nenhum par restante
        if restantes in memo:
            return memo[restantes]   # evita recalcular o mesmo caso
        u = restantes[0]             # fixa um vertice impar
        melhor_custo = float("inf")
        melhor_pares = []

        for i in range(1, len(restantes)): # tenta parear u com cada outro vertice restante
            v = restantes[i]
            novos_restantes = []           # remove u e v da lista de vertices restantes

            for j in range(1, len(restantes)):
                if j != i:
                    novos_restantes.append(restantes[j])

            custo_restante, pares_restantes = resolve(novos_restantes)
            custo_total = dist[u][v] + custo_restante

            if custo_total < melhor_custo:
                melhor_custo = custo_total
                melhor_pares = [(u, v)] + pares_restantes

        memo[restantes] = (melhor_custo, melhor_pares)
        return memo[restantes]

    return resolve(impares)
```

Luís Felipe

02/06/26

Carteiro Chinês: aplicações

- entrega de correspondências;
- leitura de água, luz ou gás;
- coleta de lixo residencial;
- inspeção de ruas, trilhos, redes elétricas ou tubulações;
- planejamento de rotas em que as arestas precisam ser cobertas.

Atenção: isso é diferente do Problema do Caixeiro Viajante.
No Carteiro Chinês, queremos cobrir arestas. No Caixeiro Viajante, queremos visitar vértices (Caixeiro Viajante será visto na próxima aula).