Três problemas envolvendo caminhos em grafos

Três problemas envolvendo caminhos em grafos

1. Existência de um v–w caminho

Três problemas envolvendo caminhos em grafos

1. Existência de um v–w caminho
2. Caminhos hamiltonianos/circuitos hamiltonianos

Três problemas envolvendo caminhos em grafos

1. Existência de um v–w caminho
2. Caminhos hamiltonianos/circuitos hamiltonianos
3. Trilhas eulerianas (trilhas e trilhas fechadas)

Passeios, trilhas, e caminhos

Passeios, trilhas, e caminhos

$v_0 v_1 \ldots v_\ell$: seqüência de vértices de G, com $v_{i-1} v_i$ aresta em G

1. Passeio (sem restrição adicional)

Passeios, trilhas, e caminhos

$v_0 v_1 \ldots v_\ell$: seqüência de vértices de G, com $v_{i-1} v_i$ aresta em G

1. Passeio (sem restrição adicional)
2. Trilhas: todas as arestas $v_{i-1} v_i$ ($1 \leq i \leq \ell$) distintas

Passeios, trilhas, e caminhos

$v_0 v_1 \ldots v_\ell$: seqüência de vértices de G, com $v_{i-1} v_i$ aresta em G

1. Passeio (sem restrição adicional)
2. Trilhas: todas as arestas $v_{i-1} v_i$ ($1 \leq i \leq \ell$) distintas
3. Caminho: todos os vértices v_i ($0 \leq i \leq \ell$) distintos

Passeios, trilhas, e caminhos

$v_0 v_1 \ldots v_\ell$: seqüência de vértices de G, com $v_{i-1} v_i$ aresta em G

1. Passeio (sem restrição adicional)
2. Trilhas: todas as arestas $v_{i-1} v_i$ ($1 \leq i \leq \ell$) distintas
3. Caminho: todos os vértices v_i ($0 \leq i \leq \ell$) distintos

Fechado: $v_0 = v_\ell$

Algoritmos

Algoritmos

- v–w caminho: busca em profundidade

Algoritmos

- v–w caminho: busca em profundidade
- Hamilton: busca exaustiva

Algoritmos

- v–w caminho: busca em profundidade
- Hamilton: busca exaustiva
- Euler: decomposição em trilhas fechadas

v–w caminho (matriz de adjacências); `prog17.11`

v–w caminho (matriz de adjacências); `prog17.11`

```
static int visited[maxV];
int pathR(Graph G, int v, int w)
  { int t;
    if (v == w) return 1;
    visited[v] = 1;
    for (t = 0; t < G->V; t++)
      if (G->adj[v][t] == 1)
        if (visited[t] == 0)
          if (pathR(G, t, w)) return 1;
    return 0;
  }
```

v–w caminho (matriz de adjacências); `prog17.11`

```
int GRAPHpath(Graph G, int v, int w)
  { int t;
    for (t = 0; t < G->V; t++) visited[t] = 0;
    return pathR(G, v, w);
  }
```

v–w caminho (listas de adjacências)

v–w caminho (listas de adjacências)

`Exercício!`

v–w caminho (listas de adjacências)

`Exercício!`

- Alguns exercícios disponíveis na 6a. feira próxima

v–w caminho (listas de adjacências)

`Exercício!`

- Alguns exercícios disponíveis na 6a. feira próxima
- Experimentem as implementações discutidas em sala

Caminhos hamiltonianos (matriz de adjacências); `prog17.12`

Caminhos hamiltonianos (matriz de adjacências); `prog17.12`

```
static int visited[maxV];
int pathR(Graph G, int v, int w, int d)
  { int t;
    if (v == w)
      { if (d == 0) return 1; else return 0; }
    visited[v] = 1;
    for (t = 0; t < G->V; t++)
      if (G->adj[v][t] == 1)
        if (visited[t] == 0)
          if (pathR(G, t, w, d-1)) return 1;
    visited[v] = 0;
    return 0;
  }
```

Caminhos hamiltonianos (matriz de adjacências); `prog17.12`

```
int GRAPHpathH(Graph G, int v, int w)
  { int t;
    for (t = 0; t < G->V; t++) visited[t] = 0;
    return pathR(G, v, w, G->V-1);
  }
```

Trilhas eulerianas (existência)

Trilhas eulerianas (existência)

```
/* prog17.13 (modificado) */
int GRAPHpathE(Graph G, int v, int w)
  { int t;
    if ((GRAPHdeg(G, v) % 2 == 0)||(GRAPHdeg(G, w) % 2 == 0))
      return 0;
    for (t = 0; t < G->V; t++)
      if ((t != v) && (t != w))
        if ((GRAPHdeg(G, t) % 2) != 0) return 0;
    return 1;
  }
```

Trilhas eulerianas; `prog17.14` (modificado)

Trilhas eulerianas; `prog17.14` (modificado)

```
#include "STACK.h"
int path(Graph G, int v)
  { int w;
    for (; G->adj[v] != NULL; v = w)
      {
        STACKpush(v);
        w = G->adj[v]->v;
        GRAPHremoveE(G, EDGE(v, w));
      }
    return 1;
  }
```

Trilhas eulerianas; `prog17.14` (modificado)

Trilhas eulerianas; `prog17.14` (modificado)

```
void pathEshow(Graph G, int v)
  {
    STACKinit(G->E);
    printf("%d", v);
    while (path(G, v) && !STACKempty())
      { v = STACKpop(); printf("-%d", v); }
    printf("\n");
  }
```

Trilhas eulerianas; (Ex. 17.96/97; para meditar)

Trilhas eulerianas; (Ex. 17.96/97; para meditar)

```
#define GRAPHiso(G, v) (GRAPHdeg(G, v) == 0)
void pathEshow(Graph G, int v, int w)
  { int t;
    if ((v == w) && (G->E == 0)) return;
    for (t = 0; t < G->V; t++)
      if (G->adj[v][t] != 0)
        <examine a aresta {v,t}>
  }
```

Trilhas eulerianas; (Ex. 17.96/97; para meditar)

Trilhas eulerianas; (Ex. 17.96/97; para meditar)

```
<examine a aresta {v,t}> =
        {
          GRAPHremoveE(G, EDGE(v, t));
          if (GRAPHiso(G, v) || GRAPHpath(G, t, v))
            {
              printf("%d-%d\n", v, t);
              pathEshow(G, t, w);
              GRAPHinsertE(G, EDGE(v, t));
              return;
            }
          GRAPHinsertE(G, EDGE(v, t));
        }
```