Caminhos em Grafos

- Caminho simples
  - Dados dois vértices num grafo, saber se estão ligados por um caminho;
  - Determinar se o caminho existe ou calculá-lo explicitamente;
- Caminho de Hamilton
  - Dados dois vértices num grafo, saber se existe um caminho que visite todos os vértices do grafo exactamente uma vez;
  - Determinar se o caminho existe ou calculá-lo explicitamente;
- Caminho de Euler
  - Dados dois vértices num grafo, saber se existe um caminho simples que use todas as arestas do grafo exactamente uma vez;
  - Determinar se o caminho existe ou calculá-lo explicitamente.

Grafos - Caminho Simples (1)

(Cliente para M. de Adj.)

```
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;
 }
```

Grafos - Caminho Simples (2)

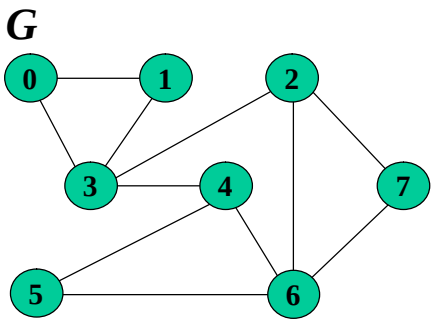
(Cliente para M. de Adj.)

```
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);
 }
```


Grafos - Descrição do algoritmo

- Suporta-se num esquema de procura em *profundidade primeiro* – "depth first search".
- A partir de *v*, encontrando um primeiro vértice adjacente, *t*, chama-se recursivamente tentando encontrar *w* a partir de *t*.
- O vector "visited" serve para garantir uma só visita a cada vértice.

Grafos - Exemplo de execução (1)



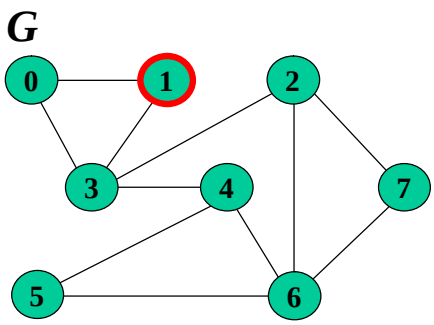
- Chamada à função para determinar a existência de caminho entre os vértices 1 e 4

  `GRAPHpath(G, 1, 4)`

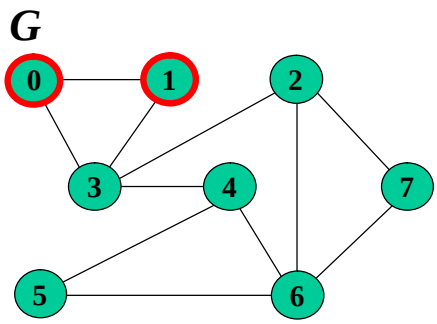
Grafos - Exemplo de execução (2)

- Sequência de chamadas a **"`pathR`"**

  pathR(G, 1, 4)



Grafos - Exemplo de execução (2)



- Sequência de chamadas a **"pathR"**

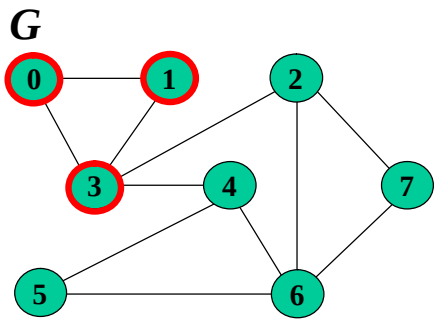
  pathR(G, 1, 4)

  1-0 pathR(G, 0, 4)

  0-0;1 (0 e 1 já visitados)

  0-2 (não adjacentes)

Grafos - Exemplo de execução (2)



- Sequência de chamadas a **"pathR"**

  pathR(G, 1, 4)

  1-0 pathR(G, 0, 4)

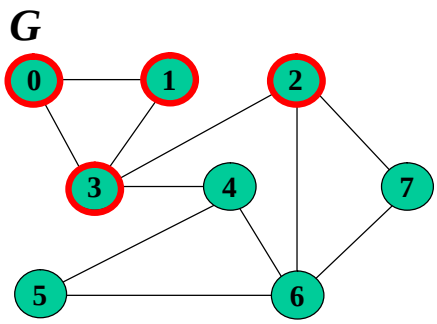
  0-0;1 (0 e 1 já visitados)

  0-2 (não adjacentes)

  0-3 pathR(G, 3, 4)

  3-0;1 (0 e 1 já visitados)

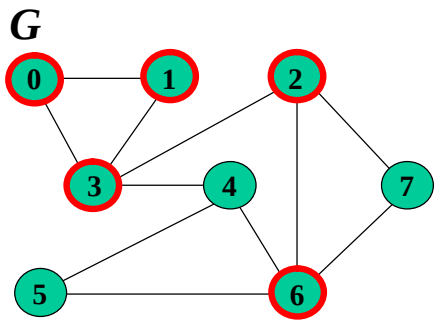
Grafos - Exemplo de execução (2)



- Sequência de chamadas a **"pathR"**

```
pathR(G, 1, 4)
  1-0 pathR(G, 0, 4)
    0-0;1  (0 e 1 já visitados)
    0-2  (não adjacentes)
    0-3 pathR(G, 3, 4)
      3-0;1  (0 e 1 já visitados)
      3-2 pathR(G, 2, 4)
        2-0;1 (não adjacentes)
        2-2 (já visitado)
        ...
```

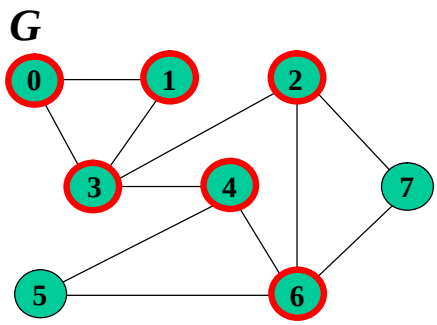

Grafos - Exemplo de execução (3)



- Sequência de chamadas a **"pathR"**

```
...
2-3  (3 já visitado)
2-4;5  (não adjacentes)
2-6 pathR(G, 6, 4)
```

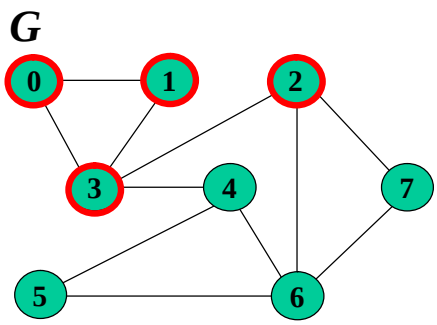
Grafos - Exemplo de execução (3)



- Sequência de chamadas a **"pathR"**

  ...
  2-3 (3 já visitado)
  2-4;5 (não adjacentes)
  2-6 pathR(G, 6, 4)
    6-0;1 (não adjacentes)
    6-2 (2 já visitado)
    6-3 (não adjacentes)
    6-4 pathR(G, 4, 4)

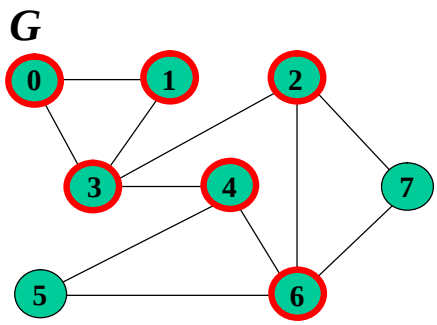
Grafos - Exemplo de execução (2)



- Sequência de chamadas a **"pathR"**

  pathR(G, 1, 4)
    1-0 pathR(G, 0, 4)
      0-0;1 (0 e 1 já visitados)
      0-2 (não adjacentes)
      0-3 pathR(G, 3, 4)
        3-0;1 (0 e 1 já visitados)
        3-2 pathR(G, 2, 4)
          2-0;1 (não adjacentes)
          2-2 (já visitado)
          ...

Grafos - Exemplo de execução (3)



- Sequência de chamadas a **"pathR"**

  ...
  2-3 (3 já visitado)
  2-4;5 (não adjacentes)
  2-6 pathR(G, 6, 4)
    6-0;1 (não adjacentes)
    6-2 (2 já visitado)
    6-3 (não adjacentes)
    6-4 pathR(G, 4, 4)

Grafos - Análise de Complexidade (1)

- Número de execuções de **"if (G->adj[v][t]==1)"** para *v*?

  "for (t = 0; t < G->V; t++)"

  Ou seja, *V* vezes no pior caso para cada *v*.

- Número de execuções de **"if (visited[t]==0)"** para *v*?

  ```
  "for (t = 0; t < G->V; t++)
    if (G->adj[v][t] == 1)"
  ```

  Ou seja, sempre que *v* e *t* forem adjacentes. No pior caso, *V* vezes.

Grafos - Análise de Complexidade (2)

- Quantas chamadas a **"pathR"** para cada *v*?

```
"for (t = 0; t < G->V; t++)
  if (G->adj[v][t] == 1)
   if (visited[t] == 0)"
```

  Ou seja, para adjacentes de *v* ainda não visitados. No pior caso, *V* vezes.

- Número total de chamadas a **"pathR"** não é superior a *V*.
  – Porquê?

Grafos - Análise de Complexidade (2)

- Discussão anterior centrou-se na representação por matriz de adjacências.

- Para listas de adjacências, o teste **"if (G->adj[v][t] == 1)"** é substituído pelo teste de fim de lista de adjacentes de *v*
  – executado, no pior caso, grau(*v*) vezes;
  – grau(*v*) ≤ *V*;
  – Para todos os vértices, no pior caso, é executado Σ_v grau(*v*) = *2E*.

- Tudo o mais permanece.

Grafos - Análise de Complexidade (3)

- Na representação por matrizes de adjacência, o algoritmo tem um tempo de execução da ordem de V^2.
  - Porquê?

- Na representação por listas de adjacências, o algoritmo tem um tempo de execução da ordem de E.
  - Porquê?

- Em síntese:
  - ***Pode-se determinar um caminho ligando dois vértices dados de um grafo em tempo linear.***

Grafos - Análise de Complexidade (4)

- Tempo linear significa:
  - tempo proporcional a V^2 para matrizes de adjacência;
  - tempo proporcional a E para listas de adjacências.
- Que representação preferir para este problema?
  - Se o grafo for denso, preferir a solução por matriz de adjacências.
  - Se o grafo for esparso, preferir a solução por listas de adjacências.

Grafos – Síntese da Aula 7

- Primeiros problemas em grafos
  - Caminhos Simples
  - Caminhos de Hamilton
  - Caminhos de Euler
- Caminhos Simples
  - Implementação
  - Descrição da implementação
  - Exemplo de execução
  - Análise de complexidade

Grafos - Caminho de Hamilton

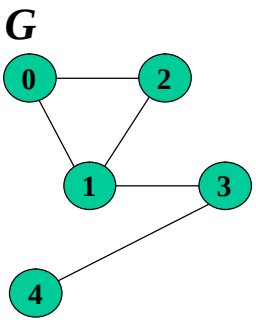
Cliente para M. de Adj.

Caminho de Hamilton:

```
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;}
```

```
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
  );
 }
```

Grafos - Exemplo de execução (1)



- Chamada à função para determinar a existência de caminho de Hamilton entre os vértices 0 e 4

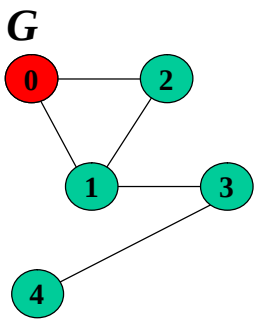
  `GRAPHpathH(G, 0, 4)`

Grafos - Exemplo de execução (2)

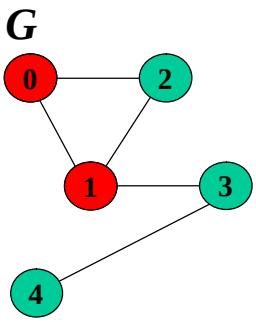
- Sequência de chamadas a **"`pathR`"**

  `pathR(G,0,4,4)`

  0-0 (0 visitado)

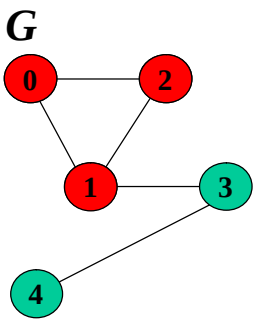


Grafos - Exemplo de execução (2)



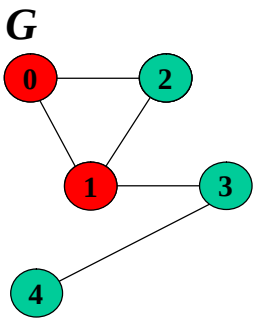
- Sequência de chamadas a **"pathR"**
  `pathR(G,0,4,4)`
  0-0 (0 visitado)
  0-1 **`pathR(G,1,4,3)`**
  1-0;1 (0 e 1 visitados)

Grafos - Exemplo de execução (2)



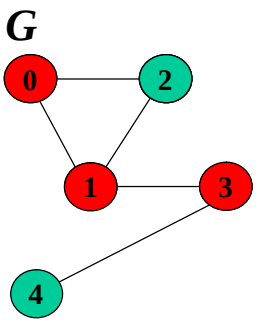
- Sequência de chamadas a **"pathR"**
  `pathR(G,0,4,4)`
  0-0 (0 visitado)
  0-1 **`pathR(G,1,4,3)`**
  1-0;1 (0 e 1 visitados)
  1-2 **`pathR(G,2,4,2)`**
  2-0;1;2 (0, 1, e 2 visitados)
  2-3;4 (não adjacentes)

Grafos - Exemplo de execução (2)



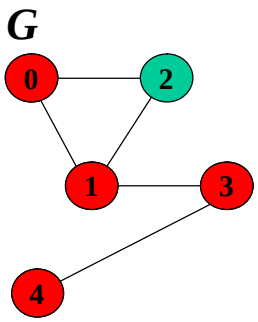
- Sequência de chamadas a **"pathR"**
  - **pathR(G,0,4,4)**
    - 0-0 (0 visitado)
    - 0-1 **pathR(G,1,4,3)**
      - 1-0;1 (0 e 1 visitados)
      - 1-2 **pathR(G,2,4,2)**
        - 2-0;1;2 (0, 1, e 2 visitados)
        - 2-3;4 (não adjacentes)
        - (sai limpando 2)

Grafos - Exemplo de execução (2)



- Sequência de chamadas a **"pathR"**
  - **pathR(G,0,4,4)**
    - 0-0 (0 visitado)
    - 0-1 **pathR(G,1,4,3)**
      - 1-0;1 (0 e 1 visitados)
      - 1-2 **pathR(G,2,4,2)**
        - 2-0;1;2 (0, 1, e 2 visitados)
        - 2-3;4 (não adjacentes)
        - (sai limpando 2)
      - 1-3 **pathR(G,3,4,2)**
        - 3-0 (não adjacentes)
        - (...)

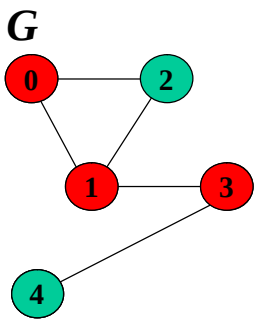
Grafos - Exemplo de execução (3)



- Sequência de chamadas a **"pathR"**

  (...)
  3-1 (1 visitado)
  3-2 (não adjacentes)
  3-3 (3 visitado)
  3-4 **pathR(G,4,4,1)**

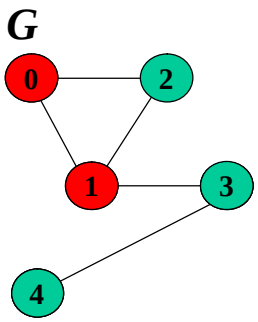
Grafos - Exemplo de execução (3)



- Sequência de chamadas a **"pathR"**

  (...)
  3-1 (1 visitado)
  3-2 (não adjacentes)
  3-3 (3 visitado)
  3-4 **pathR(G,4,4,1)**
  (d > 0, devolve 0 e limpa 4)

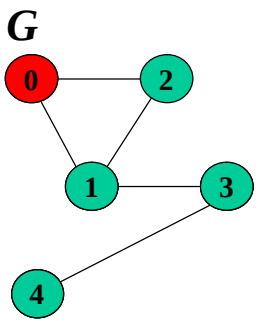
Grafos - Exemplo de execução (3)



- Sequência de chamadas a **"pathR"**

  (...)
  3-1 (1 visitado)
  3-2 (não adjacentes)
  3-3 (3 visitado)
  3-4 **pathR(G,4,4,1)**
  (d > 0, devolve 0 e limpa 4)
  (sai limpando 3)
  1-4 (não adjacentes)

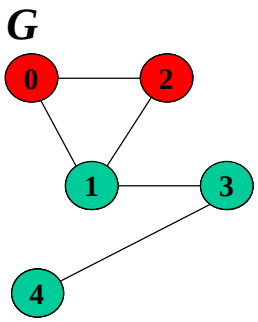
Grafos - Exemplo de execução (3)



- Sequência de chamadas a **"pathR"**

  (...)
  3-1 (1 visitado)
  3-2 (não adjacentes)
  3-3 (3 visitado)
  3-4 **pathR(G,4,4,1)**
  (d > 0, devolve 0 e limpa 4)
  (sai limpando 3)
  1-4 (não adjacentes)
  (sai limpando 1)

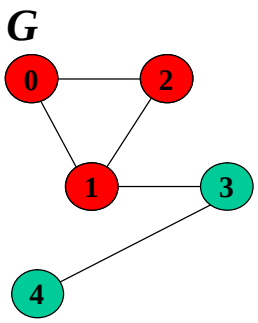
Grafos - Exemplo de execução (3)



- Sequência de chamadas a **"pathR"**

  (...)
  3-1 (1 visitado)
  3-2 (não adjacentes)
  3-3 (3 visitado)
  3-4 **pathR(G,4,4,1)**
  (d > 0, devolve 0 e limpa 4)
  (sai limpando 3)
  1-4 (não adjacentes)
  (sai limpando 1)
  0-2 **pathR(G,2,4,3)**
  (...)

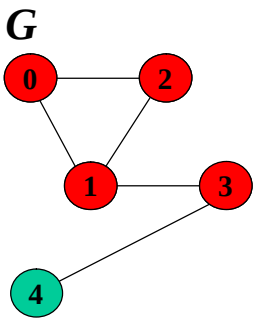
Grafos - Exemplo de execução (4)



- Sequência de chamadas a **"pathR"**

  (...)
  2-0 (0 visitado)
  2-1 **pathR(G,1,4,2)**
  1-0;1;2 (0, 1 e 2 visitados)

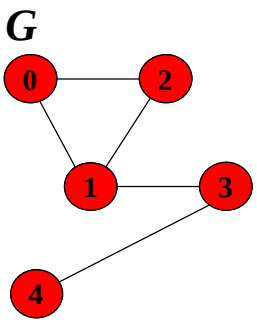
Grafos - Exemplo de execução (4)



- Sequência de chamadas a **"pathR"**

  (...)
  2-0 (0 visitado)
  2-1 **pathR(G,1,4,2)**
  1-0;1;2 (0, 1 e 2 visitados)
  1-3 **pathR(G,3,4,1)**
  3-0 (não adjacentes)
  3-1;2;3 (1, 2 e 3 visitados)

Grafos - Exemplo de execução (4)

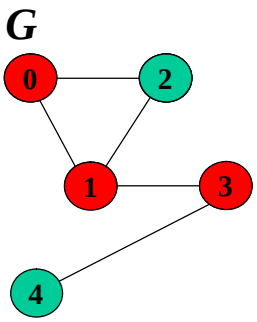


- Sequência de chamadas a **"pathR"**

  (...)
  2-0 (0 visitado)
  2-1 **pathR(G,1,4,2)**
  1-0;1;2 (0, 1 e 2 visitados)
  1-3 **pathR(G,3,4,1)**
  3-0 (não adjacentes)
  3-1;2;3 (1, 2 e 3 visitados)
  3-4 **pathR(G,4,4,0)**
  (**d == 0**, caminho encontrado)
  (...)

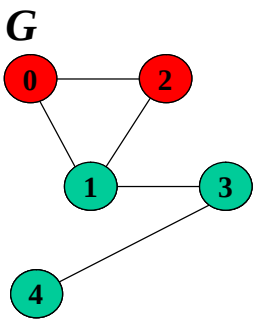
  (Caminho Encontrado!)

Grafos - Exemplo de execução (2)



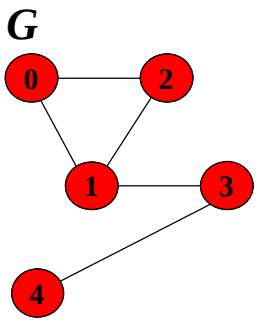
- Sequência de chamadas a **"pathR"**
  - **pathR(G,0,4,4)**
    - 0-0 (0 visitado)
    - 0-1 **pathR(G,1,4,3)**
      - 1-0;1 (0 e 1 visitados)
      - 1-2 **pathR(G,2,4,2)**
        - 2-0;1;2 (0, 1, e 2 visitados)
        - 2-3;4 (não adjacentes)
        - (sai limpando 2)
      - 1-3 **pathR(G,3,4,2)**
        - 3-0 (não adjacentes)
        - (...)

Grafos - Exemplo de execução (3)



- Sequência de chamadas a **"pathR"**
  - (...)
  - 3-1 (1 visitado)
  - 3-2 (não adjacentes)
  - 3-3 (3 visitado)
  - 3-4 **pathR(G,4,4,1)**
    - (d > 0, devolve 0 e limpa 4)
  - (sai limpando 3)
- 1-4 (não adjacentes)
- (sai limpando 1)
- 0-2 **pathR(G,2,4,3)**
  - (...)

Grafos - Exemplo de execução (4)



- Sequência de chamadas a **"pathR"**

(...)
2-0 (0 visitado)
2-1 **pathR(G,1,4,2)**
1-0;1;2 (0, 1 e 2 visitados)
1-3 **pathR(G,3,4,1)**
3-0 (não adjacentes)
3-1;2;3 (1, 2 e 3 visitados)
3-4 **pathR(G,4,4,0)**
(**d == 0**, caminho encontrado)
(...)
(Caminho Encontrado!)

Grafos - Análise de Complexidade

- Prop: *A procura recursiva por um caminho de Hamilton pode levar tempo exponencial.*
- Esboço de Demonstração
  - Considere-se um grafo em que o vértice *V-1* está isolado e os restantes *V-1* vértices formam um sub-grafo completo.
  - O programa nunca conseguirá encontrar um caminho de Hamilton (Porquê?).
  - No entanto, por indução pode-se estabelecer que examinará todos os ***(V-1)!*** caminhos no sub-grafo completo e em todos eles executará *V-1* chamadas recursivas a "graphR".
  - Logo, o número total de chamadas recursivas é da ordem de ***V!***, ou da ordem de $(V/e)^V$, o que é superior a qualquer polinómio em *V*.

Grafos – Discussão (1)

- Como é que dois problemas tão próximos no seu enunciado e dois programas tão próximos na sua estrutura, possuem complexidades tão distintas?

  – Quando se procura um caminho simples sabemos que, se existe um caminho entre os vértices *v* e *w*, ele é encontrado a partir de algum *t* adjacente a v.

  – O mesmo é verdade para caminhos de Hamilton.

  – Se não encontrarmos um caminho simples entre *t* e *w*, sabemos que não existe um caminho simples entre *v* e *w* que passe por *t*.

Grafos – Discussão (2)

  – Para caminhos de Hamilton, pode não existir um caminho que comece por *v*-*t*, mas pode existir um caminho que comece por *v*-*x*-*t*, para algum outro vértice *x* (daí termos de desmarcar vértices visitados).

  – Logo, teremos que executar uma chamada recursiva a partir de *t* para todos os caminhos possíveis originados em *v* e que passam em *t*.

  – O que é o mesmo que dizer que poderemos ter que investigar todos os caminhos possíveis no grafo.

Grafos - Caminhos e Ciclos de Euler (1)

- Prop: *Um grafo ligado possui um caminho de Euler entre o vértice v e o vértice w se e só se v e w têm grau ímpar e todos os outros vértices têm grau par.*
- Demonstração
  - Prova-se por indução no número de arestas.
  - Claramente, o resultado é verdadeiro para um grafo com dois vértices e uma só aresta ligando-os.
  - Seja G um grafo ligado com mais que uma aresta.
  - Existe um caminho de Euler entre *v* e *w* se e só se existir um vértice *t*, adjacente de *v*, para o qual exista um caminho de Euler até *w* no grafo G^* (G^* obtém-se de G removendo a aresta que liga *v* a *t*);

Grafos - Caminhos e Ciclos de Euler (2)

- Demonstração (continuação)
  - Pela hipótese de indução, existe um caminho de Euler em G^* se e só se todos os vértices têm grau par excepto *t* e *w*, que têm grau ímpar.
  - Se existir um caminho de Euler entre *t* e *w* em G^*, adicionando uma aresta entre *v* e *t* faz com que *t* tenha grau par e com que *v* tenha grau ímpar, sem afectar o grau dos restantes vértices.
  - Logo, o resultado fica estabelecido.

Grafos - Caminhos e Ciclos de Euler (3)

- Demonstração (continuação)
  - Se não existir um caminho de Euler entre *t* e *w* em G^*, então é porque
    - *t* tem grau par - (de grau ímpar em G), ou
    - *w* tem grau par - (também de grau par em G), ou
    - *v* tem grau ímpar - (de grau par em G), ou
    - outro vértice tem grau ímpar - (também de grau ímpar em G).
  - Em qualquer dos casos, violando a condição enunciada.

Grafos - Caminhos e Ciclos de Euler (4)

- Prop: *Um grafo ligado possui um ciclo de Euler se e só se todos os seus vértices tiverem grau par.*

- Demonstração
  - Consequência imediata da propriedade anterior e do argumento utilizado na sua demonstração.

- Grau dos vértices
  - Tempo proporcional a *E* para listas de adjacências.
  - Tempo proporcional a V^2 para matrizes de adjacências.
  - Ou mantém-se uma tabela indexada pelos vértices com o grau de cada vértice.

Grafos - Caminhos e Ciclos de Euler (5)

- Neste último caso, determinar se existe ou não um caminho ou ciclo de Euler pode ser feito em tempo proporcional a V.

- Sendo fácil saber se existe, será que é fácil calculá-lo como nos caminhos simples ou será que é tão difícil como com os caminhos de Hamilton?

- Examinando a demonstração
  - Qualquer aresta cuja remoção mantenha o grafo ligado pode ser a primeira aresta de um caminho de Euler.
  - Encontrando uma primeira aresta nestas condições, aplica-se o conceito recursivamente.

Grafos – Síntese da Aula 8

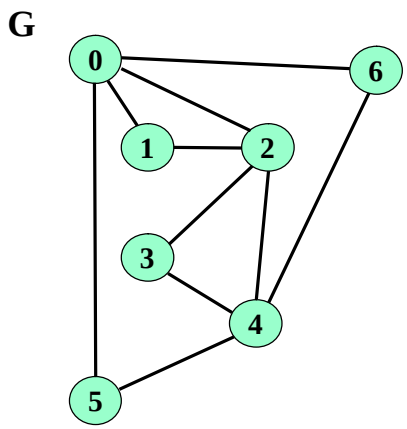
- Caminhos de Hamilton
  - Implementação
  - Exemplo de execução
  - Análise de complexidade
- Caminhos simples vs. caminhos de Hamilton
  - Discussão de diferenças
- Caminhos e ciclos de Euler
  - Caracterização dos grafos que possuem caminhos de Euler
  - Caracterização dos grafos que possuem ciclos de Euler
  - Discussão especulativa da complexidade do problema

Grafos - Caminho de Euler

Cliente para M. de Adj.

```
int GRAPHpathE(Graph G, int v, int w) {
  int t;
  if ((v == w)  && G->E == 0) return 1;
  for (t = 0; t < G->V; t++)
    if (G->adj[v][t] != 0){
      GRAPHremoveE(G, EDGE(v, t));
      if (GRAPHiso(G, v) || GRAPHpath(G, v, t))
        if (GRAPHpathE(G, t, w)){
          printf("%d-%d\n", v, t); return 1;
        }
      GRAPHinsertE(G, EDGE(v, t));
    }
  return 0;}
```


Grafos - Exemplo de execução (1)

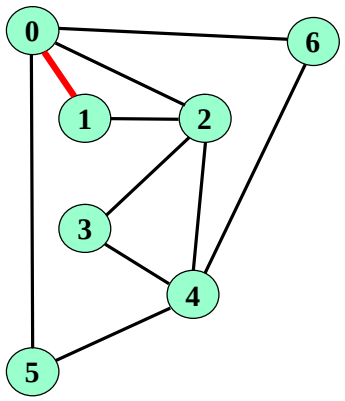


- Chamada à função para calcular um ciclo de Euler com início no vértice 1

  `GRAPHpathE(G, 1, 1)`

Grafos - Exemplo de execução (2)

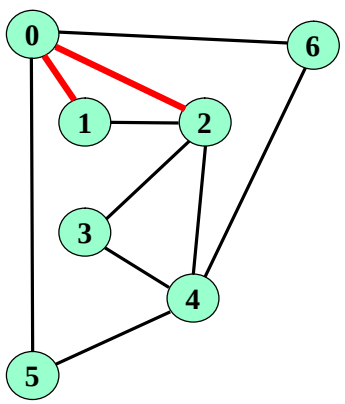
G



- Chamadas a **GRAPHremoveE**
  GRAPHpathE(G,1,1)
  1-0 **GRAPHpathE(G,0,1)**
  0-0;1 (não adjacentes)

Grafos - Exemplo de execução (2)

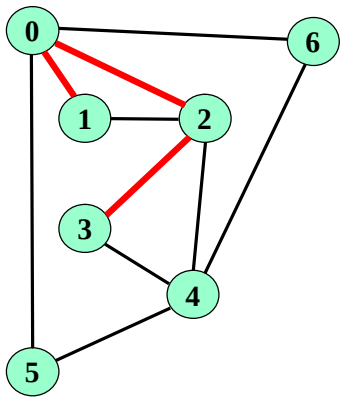
G



- Chamadas a **GRAPHremoveE**
  GRAPHpathE(G,1,1)
  1-0 **GRAPHpathE(G,0,1)**
  0-0;1 (não adjacentes)
  0-2 **GRAPHpathE(G,2,1)**
  2-0 (não adjacentes)
  2-1 (grafo não ligado)
  2-2 (não adjacentes)

Grafos - Exemplo de execução (2)

G



- Chamadas a **GRAPHremoveE**

GRAPHpathE(G,1,1)

1-0 **GRAPHpathE(G,0,1)**

0-0;1 (não adjacentes)

0-2 **GRAPHpathE(G,2,1)**

2-0 (não adjacentes)

2-1 (grafo não ligado)

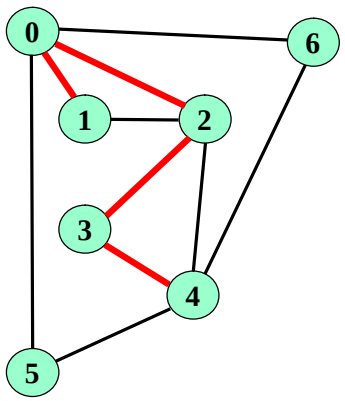
2-2 (não adjacentes)

2-3 **GRAPHpathE(G,3,1)**

3-0;1;2;3 (não adjacentes)

Grafos - Exemplo de execução (2)

G



- Chamadas a **GRAPHremoveE**

GRAPHpathE(G,1,1)

1-0 **GRAPHpathE(G,0,1)**

0-0;1 (não adjacentes)

0-2 **GRAPHpathE(G,2,1)**

2-0 (não adjacentes)

2-1 (grafo não ligado)

2-2 (não adjacentes)

2-3 **GRAPHpathE(G,3,1)**

3-0;1;2;3 (não adjacentes)

3-4 **GRAPHpathE(G,4,1)**

4-0;1 (não adjacentes)

Grafos - Exemplo de execução (3)

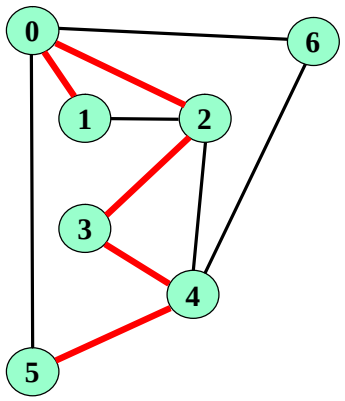
- Chamadas a **`GRAPHremoveE (cont.)`**

  4-2 (grafo não ligado)

  4-3;4 (não adjacentes)

  4-5 **`GRAPHpathE(G,5,1)`**

G



Grafos - Exemplo de execução (3)

- Chamadas a **`GRAPHremoveE (cont.)`**

  4-2 (grafo não ligado)

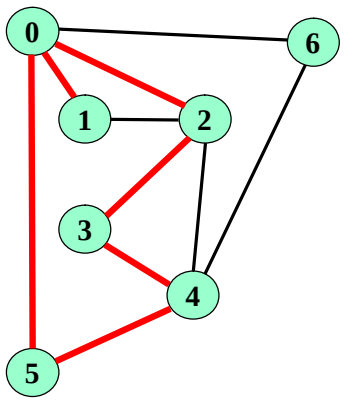
  4-3;4 (não adjacentes)

  4-5 **`GRAPHpathE(G,5,1)`**

  5-0 **`GRAPHpathE(G,0,1)`**

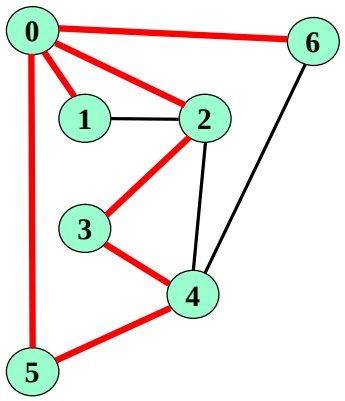
  0-0;1;2;3;4;5 (não adjacentes)

G



Grafos - Exemplo de execução (3)

G



- Chamadas a **`GRAPHremoveE (cont.)`**

  4-2 (grafo não ligado)

  4-3;4 (não adjacentes)

  4-5 **`GRAPHpathE(G,5,1)`**

  5-0 **`GRAPHpathE(G,0,1)`**

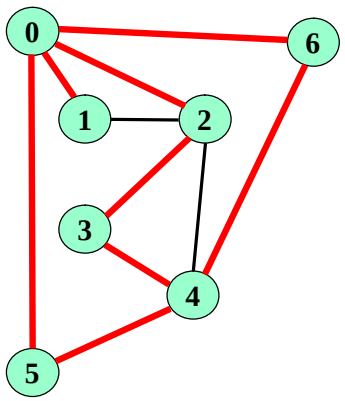
  0-0;1;2;3;4;5 (não adjacentes)

  0-6 **`GRAPHpathE(G,6,1)`**

  6-0;1;2;3 (não adjacentes)

Grafos - Exemplo de execução (3)

G



- Chamadas a **`GRAPHremoveE (cont.)`**

  4-2 (grafo não ligado)

  4-3;4 (não adjacentes)

  4-5 **`GRAPHpathE(G,5,1)`**

  5-0 **`GRAPHpathE(G,0,1)`**

  0-0;1;2;3;4;5 (não adjacentes)

  0-6 **`GRAPHpathE(G,6,1)`**

  6-0;1;2;3 (não adjacentes)

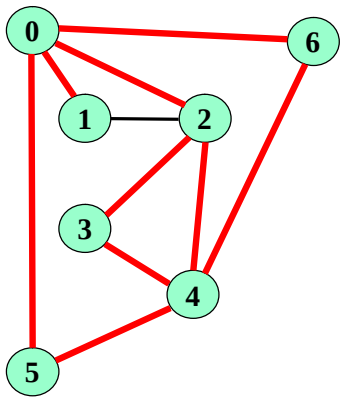
  6-4 **`GRAPHpathE(G,4,1)`**

  4-0;1 (não adjacentes)

Grafos - Exemplo de execução (4)

- Chamadas a **`GRAPHremoveE (cont.)`**
  4-2 **`GRAPHpathE(G,2,1)`**
  2-0 (não adjacentes)

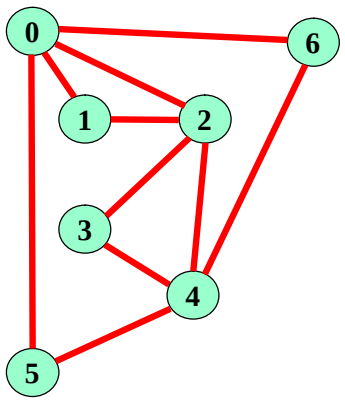
G



Grafos - Exemplo de execução (4)

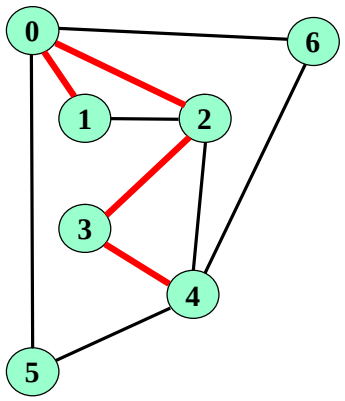
- Chamadas a **`GRAPHremoveE (cont.)`**
  4-2 **`GRAPHpathE(G,2,1)`**
  2-0 (não adjacentes)
  2-1 **`GRAPHpathE(G,1,1)`**
  `FIM com SUCESSO!`

G



Grafos - Exemplo de execução (2)

G



- Chamadas a **GRAPHremoveE**

GRAPHpathE(G,1,1)

1-0 **GRAPHpathE(G,0,1)**

0-0;1 (não adjacentes)

0-2 **GRAPHpathE(G,2,1)**

2-0 (não adjacentes)

2-1 (grafo não ligado)

2-2 (não adjacentes)

2-3 **GRAPHpathE(G,3,1)**

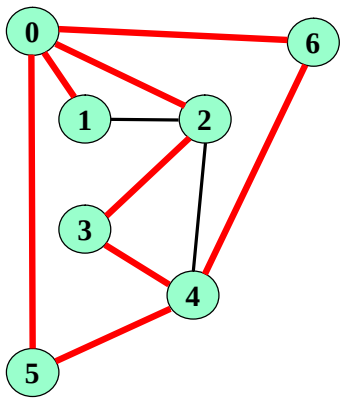
3-0;1;2;3 (não adjacentes)

3-4 **GRAPHpathE(G,4,1)**

4-0;1 (não adjacentes)

Grafos - Exemplo de execução (3)

G



- Chamadas a **GRAPHremoveE (cont.)**

4-2 (grafo não ligado)

4-3;4 (não adjacentes)

4-5 **GRAPHpathE(G,5,1)**

5-0 **GRAPHpathE(G,0,1)**

0-0;1;2;3;4;5 (não adjacentes)

0-6 **GRAPHpathE(G,6,1)**

6-0;1;2;3 (não adjacentes)

6-4 **GRAPHpathE(G,4,1)**

4-0;1 (não adjacentes)

Grafos - Exemplo de execução (4)

- Chamadas a **`GRAPHremoveE (cont.)`**

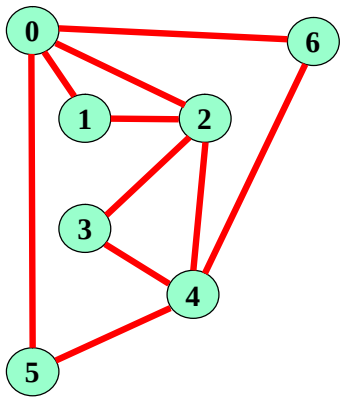
  4-2 **`GRAPHpathE(G,2,1)`**

  2-0 (não adjacentes)

  2-1 **`GRAPHpathE(G,1,1)`**

  `FIM com SUCESSO!`

G



Grafos – Análise de Complexidade

- O programa atrás é muito simples, mas pouco eficiente porque pode gastar um tempo proporcional a E^2, dependendo de como o grafo é representado e se implementa as operações básicas.
  - Testar se um vértice está isolado ou não pode ser feito em tempo constante.
    - Como?
  - Testar se existe um caminho entre dois vértices é feito em tempo linear.
  - Logo, com representação por lista de adjacências o limite de pior caso é de E^2 para grafos esparsos.
- Como resolver o problema de uma forma mais eficiente?

Grafos – Ideia alternativa

- Suponhamos que, começando em qualquer vértice *v*, seguimos e removemos uma qualquer aresta.
- A partir do vértice que atingimos continuamos fazendo o mesmo até que cheguemos a um vértice isolado.
- Este processo tem que terminar, dado que o número de arestas é finito e retiramos uma em cada passo.
- O que é que pode acontecer?
  - Regressa-se a *v* se e só se o grafo possuir um ciclo de Euler.
    - Porquê?
  - Pode acontecer que consigamos gerar assim o ciclo.
  - Caso não, o grafo que sobra possui um ciclo de Euler se o original o possuir.
    - Porquê?

Grafos - Caminho de Euler em tempo linear

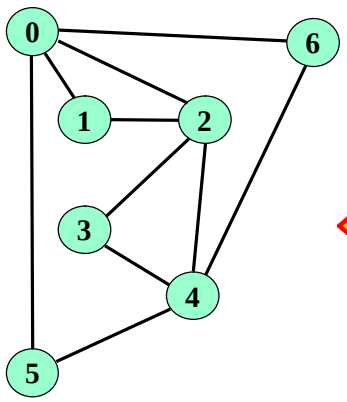
Cliente para L. de Adj.

```
#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 v;
 }
```

```
int GraphpathE(Graph G,int v,int w)
 {
  STACKinit(G->E);
  printf("%d", w);
  while ((path(G,v) == v) &&
         (!STACKempty()) )
    printf("-%d", v = STACKpop());
  printf("\n");
  return (G->E == 0);
 }
```

Grafos - Exemplo de execução (1)

G



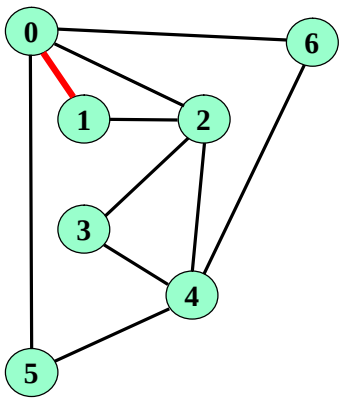
Lista de adjacências
- 0 : 1 - 2 - 5 - 6
- 1 : 0 - 2
- 2 : 0 - 3 - 4 - 1
- 3 : 4 - 2
- 4 : 6 - 5 - 3 - 2
- 5 : 0 - 4
- 6 : 4 - 0

- Chamada à função para calcular um ciclo de Euler com início no vértice 0

  `GRAPHpathE(G, 0, 0)` – *escreve 0*

Grafos - Exemplo de execução (2)

G



- Chamadas a **`GRAPHremoveE`**

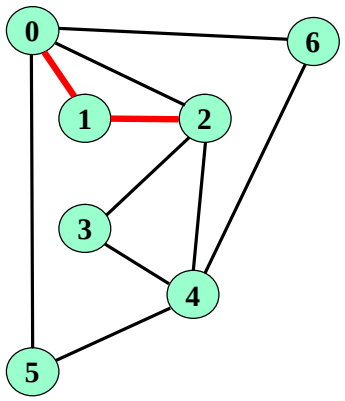
  `path(G, 0)`

  0-1 **`push(0)`**

  1-0 (já não existe)

Grafos - Exemplo de execução (2)

G



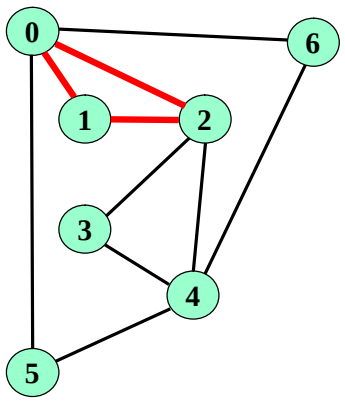
- Chamadas a **`GRAPHremoveE`**

`path(G, 0)`

0-1	**`push(0)`**
1-0 (já não existe)	
1-2	**`push(1)`**

Grafos - Exemplo de execução (2)

G



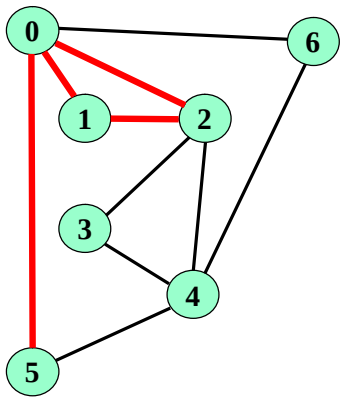
- Chamadas a **`GRAPHremoveE`**

`path(G, 0)`

0-1	**`push(0)`**
1-0 (já não existe)	
1-2	**`push(1)`**
2-0	**`push(2)`**
0-1;2 (já não existe)	

Grafos - Exemplo de execução (2)

G



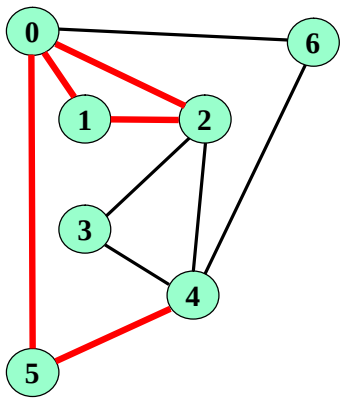
- Chamadas a **GRAPHremoveE**

path(G, 0)

0-1	**push(0)**
1-0 (já não existe)	
1-2	**push(1)**
2-0	**push(2)**
0-1;2 (já não existe)	
0-5	**push(0)**
5-0 (já não existe)	

Grafos - Exemplo de execução (2)

G



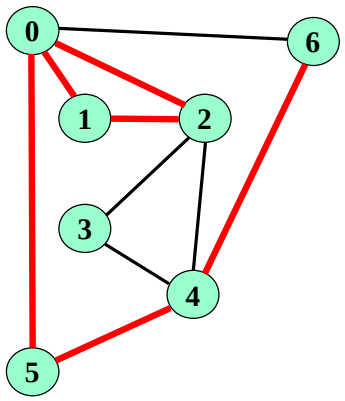
- Chamadas a **GRAPHremoveE**

path(G, 0)

0-1	**push(0)**
1-0 (já não existe)	
1-2	**push(1)**
2-0	**push(2)**
0-1;2 (já não existe)	
0-5	**push(0)**
5-0 (já não existe)	
5-4	**push(5)**

Grafos - Exemplo de execução (2)

G



- Chamadas a **`GRAPHremoveE`**

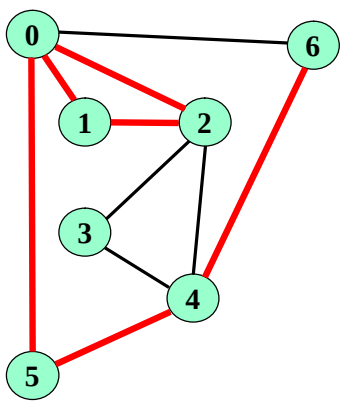
`path(G, 0)`

0-1	**`push(0)`**
1-0 (já não existe)	
1-2	**`push(1)`**
2-0	**`push(2)`**
0-1;2 (já não existe)	
0-5	**`push(0)`**
5-0 (já não existe)	
5-4	**`push(5)`**
4-6*	**`push(4)`**

**** Assumindo que o vértice 6 surge na lista antes de 2 e 3***

Grafos - Exemplo de execução (2)

G



- Chamadas a **`GRAPHremoveE`**

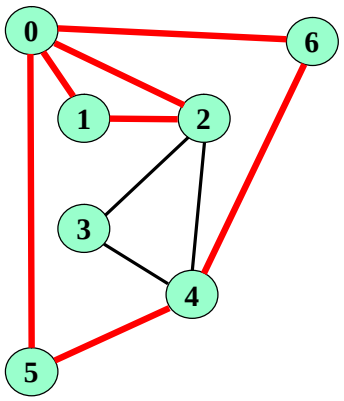
`path(G, 0)`

0-1	**`push(0)`**
1-0 (já não existe)	
1-2	**`push(1)`**
2-0	**`push(2)`**
0-1;2 (já não existe)	
0-5	**`push(0)`**
5-0 (já não existe)	
5-4	**`push(5)`**
4-6*	**`push(4)`**

**** Assumindo que o vértice 6 surge na lista antes de 2 e 3***

Grafos – Exemplo de Execução (3)

G

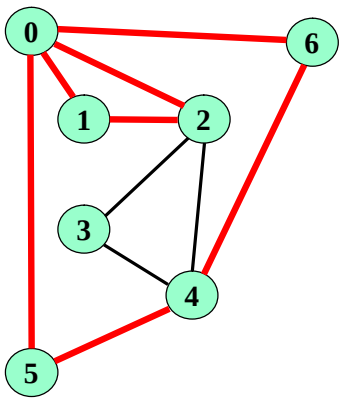


- Chamadas a **`GRAPHremoveE`** (cont.)

6-0	**`push(6)`**
0 – *isolado*	**`return(0)`**

Grafos – Exemplo de Execução (3)

G

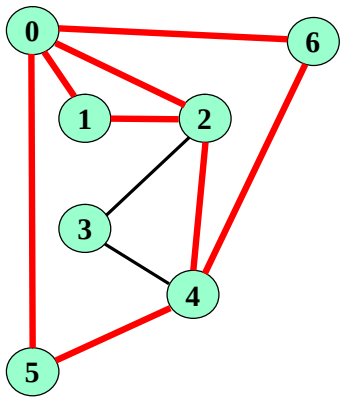


- Chamadas a **`GRAPHremoveE`** (cont.)

6-0	**`push(6)`**
0 – *isolado*	**`return(0)`**
`pop(6)`	***escreve -6***
`path(G, 6)`	
6 – *isolado*	**`return(6)`**
`pop(4)`	***escreve -4***
`path(G, 4)`	

Grafos – Exemplo de Execução (3)

G

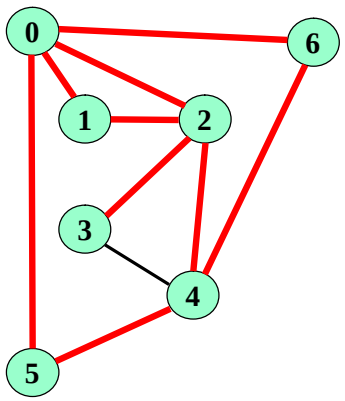


- Chamadas a **GRAPHremoveE** (cont.)

6-0	**push(6)**
0 – *isolado*	**return(0)**
pop(6)	*escreve -6*
path(G, 6)	
6 – *isolado*	**return(6)**
pop(4)	*escreve -4*
path(G, 4)	
4-2	**push(4)**

Grafos – Exemplo de Execução (3)

G

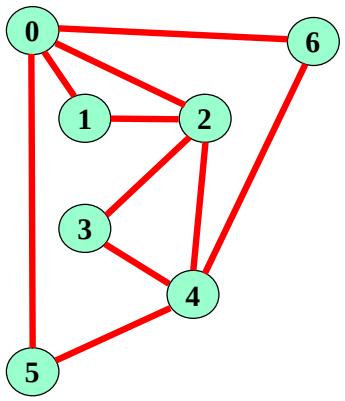


- Chamadas a **GRAPHremoveE** (cont.)

6-0	**push(6)**
0 – *isolado*	**return(0)**
pop(6)	*escreve -6*
path(G, 6)	
6 – *isolado*	**return(6)**
pop(4)	*escreve -4*
path(G, 4)	
4-2	**push(4)**
2-3	**push(2)**

Grafos – Exemplo de Execução (3)

G

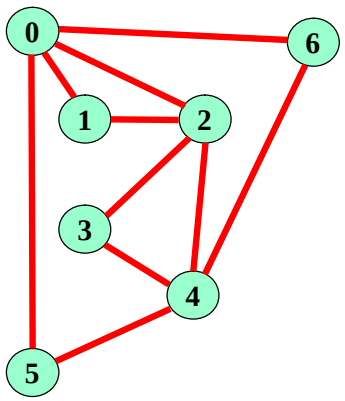


- Chamadas a **GRAPHremoveE** (cont.)

6-0	**push(6)**
0 – *isolado*	**return(0)**
pop(6)	*escreve -6*
path(G, 6)	
6 – *isolado*	**return(6)**
pop(4)	*escreve -4*
path(G, 4)	
4-2	**push(4)**
2-3	**push(2)**
3-4	**push(3)**
4 – *isolado*	**return(4)**
pop(3)	*escreve -3*

Grafos – Exemplo de Execução (3)

G

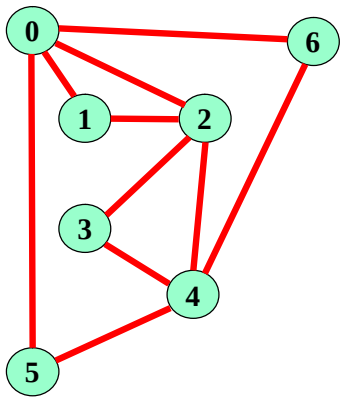


- Chamadas a **GRAPHremoveE** (cont.)

6-0	**push(6)**
0 – *isolado*	**return(0)**
pop(6)	*escreve -6*
path(G, 6)	
6 – *isolado*	**return(6)**
pop(4)	*escreve -4*
path(G, 4)	
4-2	**push(4)**
2-3	**push(2)**
3-4	**push(3)**
4 – *isolado*	**return(4)**
pop(3)	*escreve -3*

Grafos – Exemplo de Execução (4)

G



- Chamadas a **GRAPHremoveE** (cont.)

path(G, 3)
3 – *isolado* **return(3)**
pop(2) *escreve -2*

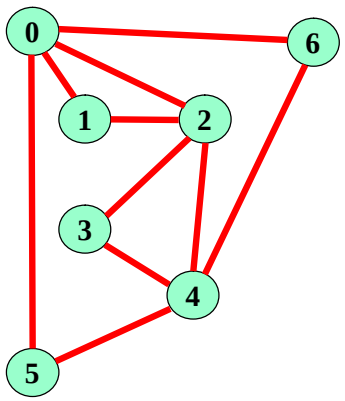
path(G, 2)
2 – *isolado* **return(2)**
pop(4) *escreve -4*

path(G, 4)
4 – *isolado* **return(4)**
pop(5) *escreve -5*

path(G, 5)
5 – *isolado* **return(5)**
pop(0) *escreve -0*

Grafos – Exemplo de Execução (5)

G



- Chamadas a **GRAPHremoveE** (cont.)

path(G, 0)
0 – *isolado* **return(0)**
pop(2) *escreve -2*

path(G, 2)
2 – *isolado* **return(2)**
pop(1) *escreve -1*

path(G, 1)
1 – *isolado* **return(1)**
pop(0) *escreve -0*

Globalmente escreve

0-6-4-3-2-4-5-0-2-1-0

Grafos – Caminho de Euler (1)

- Prop: É possível encontrar um ciclo de Euler num grafo, se um existir, em tempo linear.
- Esboço de demonstração
  - Após a primeira chamada a **`path`** a pilha contém um percurso entre *v* e *w*, e o grafo restante (após remover os vértices isolados) consiste de componentes ligadas de menor dimensão com, pelo menos, um vértice nesse caminho.
  - Cada uma dessas componentes possui um ciclo de Euler.
  - Retiram-se os vértices isolados da pilha e usa-se **`path`** para determinar ciclos de Euler contendo os vértices não isolados, da mesma forma.
  - Cada aresta do grafo é colocada na pilha (e retirada) exactamente uma vez, de tal modo que o tempo de execução é proporcional a *E*.

Grafos – Problemas em grafos

- Estes três exemplos servem para ilustrar a gama de variabilidade que pode existir em problemas com grafos.
  - Problemas aparentemente iguais com diferentes complexidades.
  - Problemas aparentemente complexos com soluções simples.
- Categorias em termos de complexidade
  - Simples, Tratáveis, Intratáveis, Desconhecida.
- Exemplos de problemas (ver discussão no livro)
  - Conectividade simples, conectividade forte em digrafos, fecho transitivo, árvores mínimas de suporte, caminhos mais curtos com uma única origem, planaridade, emparelhamento, alocação, caminho mais comprido, coloração, conjuntos independentes, cliques, isomorfismo, ciclos de tamanho par em digrafos.

Grafos – Complexidade dos Problemas (1)

- Grafos não direccionados
  - Conectividade **S**
  - Conectividade geral **T**
  - Ciclo de Euler **S**
  - Ciclo de Hanilton **I**
  - Emparelhamento em grafos bipartidos **S**
  - Emparelhamento máximo **T**
  - Planaridade **T**
  - Clique máxima **I**
  - Bi-coloração **S**
  - Tri-coloração **I**
  - Caminhos mais curtos **S**
  - Caminhos mais compridos **I**
  - Cobertura de vértices **I**
  - Isomorfismo **?**

Grafos – Complexidade dos Problemas (2)

- Digrafos
  - Fecho transitivo **S**
  - Conectividade forte **S**
  - Ciclos ímpares **S**
  - Ciclos pares **?**
- Grafos ponderados
  - Árvore mínima de suporte **S**
  - Caixeiro viajante **I**
- Redes
  - Caminhos mais curtos **S**
  - Ciclos negativos **I**
  - Fluxo da rede **S**
  - Atribuição **T**
  - Fluxo de custo mínimo **T**

Grafos – Síntese da Aula 9

- Discussão comparativa dos problemas de caminhos simples e de Hamilton.
- Caminhos e ciclos de Euler
  - Propriedades
  - Implementação #1
  - Exemplo de execução
  - Implementação #2
  - Exemplo de execução
  - Análise de complexidade
- Problemas em grafos
  - Síntese de complexidade