

AULA 10

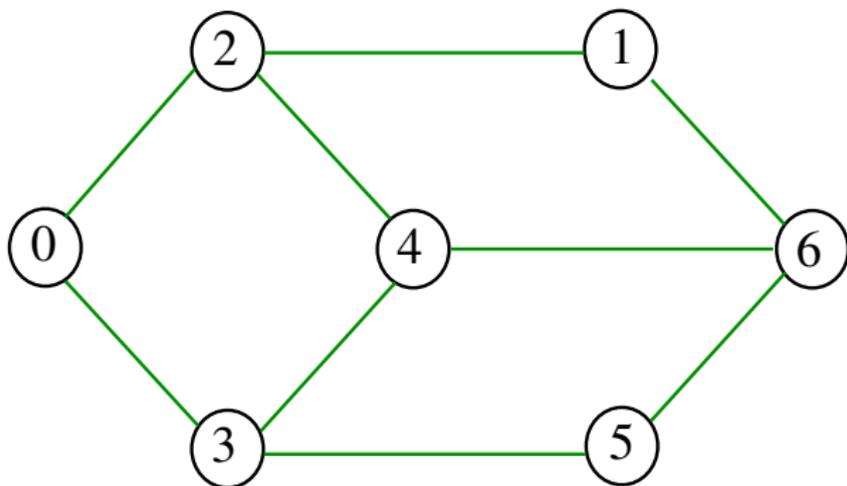
Grafos bipartidos e ciclos ímpares

S 18.5

Bipartição

Um grafo é **bipartido** (= *bipartite*) se existe uma bipartição do seu conjunto de vértices tal que cada aresta tem uma ponta em uma das partes da bipartição e a outra ponta na outra parte

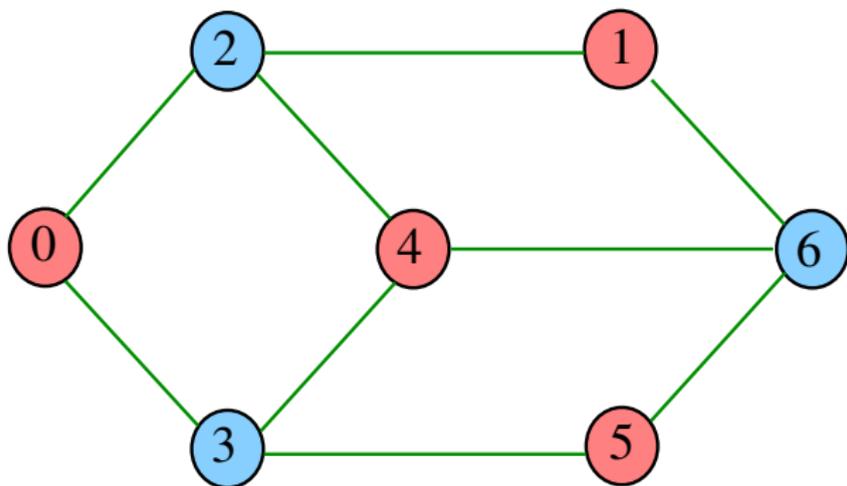
Exemplo:



Bipartição

Um grafo é **bipartido** (= *bipartite*) se existe uma bipartição do seu conjunto de vértices tal que cada aresta tem uma ponta em uma das partes da bipartição e a outra ponta na outra parte

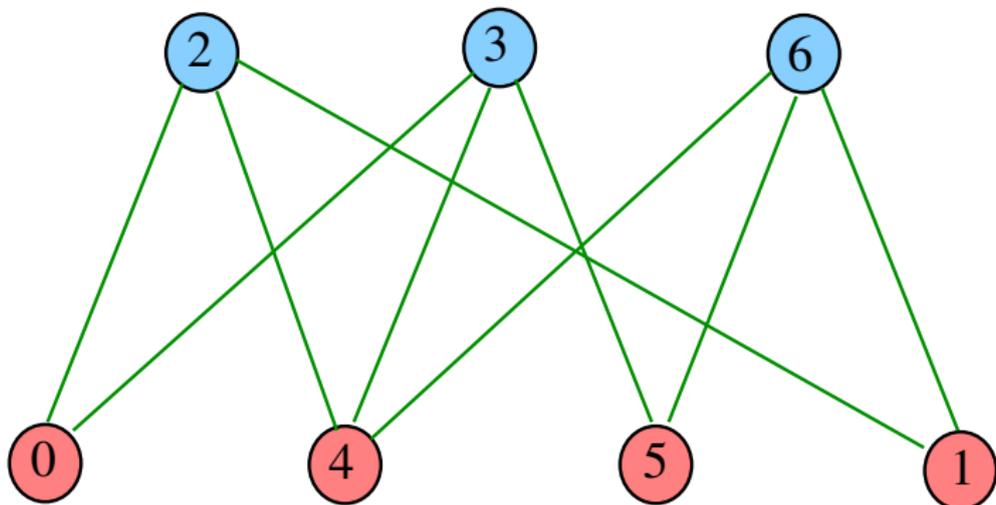
Exemplo:



Bipartição

Um grafo é **bipartido** (= *bipartite*) se existe uma bipartição do seu conjunto de vértices tal que cada aresta tem uma ponta em uma das partes da bipartição e a outra ponta na outra parte

Exemplo:



GRAPHtwocolor

Supomos que nossos grafos têm no máximo maxV vértices

```
int color[maxV];
```

A função devolve **1** se o grafo **G** é bipartido e devolve **0** em caso contrário

Se **G** é **bipartido**, a função atribui uma "cor" a cada vértice de **G** de tal forma que toda aresta tenha **pontas de cores diferentes**

As cores dos vértices, **0** e **1**, são registradas no vetor **color** indexado pelos vértices

```
int GRAPHtwocolor (Graph G);
```

GRAPHtwocolor

```
int GRAPHtwocolor (Graph G) {  
    Vertex v;  
1   for (v = 0; v < G->V; v++)  
2       color[v] = -1;  
3   for (v = 0; v < G->V; v++)  
4       if (color[v] == -1)  
5           if (dfsRclr(G, v, 0) == 0)  
6               return 0;  
7   return 1;  
}
```

dfsRclr

```
int dfsRclr(Graph G, Vertex v, int c){
    link p;
1   color[v] = 1-c;
2   for (p=G->adj[v];p!=NULL;p=p->next) {
3       Vertex w = p->w;
4       if (color[w]==-1)
5           if (dfsRclr(G,w,1-c) == 0)
6               return 0;
7       else if (color[w]==1-c) return 0;
    }
8   return 1;
}
```

Consumo de tempo

O consumo de tempo da função `GRAPHtwocolor` para **vetor de listas de adjacência** é $O(V + A)$.

Conclusão

Para todo grafo G , vale uma e apenas uma das seguintes afirmações:

- ▶ G possui um ciclo ímpar
- ▶ G é bipartido

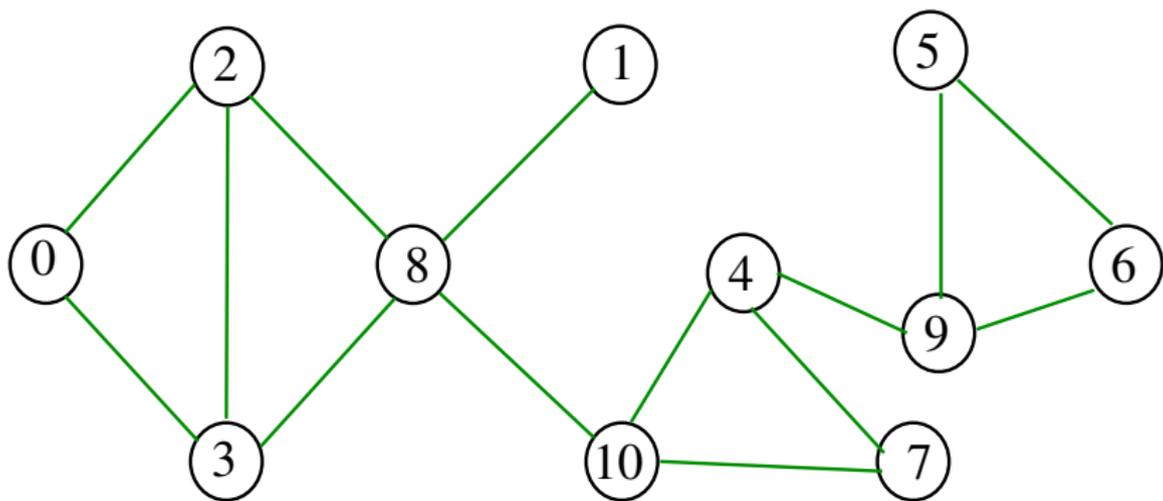
Pontes em grafos e aresta-biconexão

S 18.6

Pontes em grafos

Uma aresta de um grafo é uma **ponte** (= *bridge* = *separation edge*) se ela é a única aresta que atravessa algum corte do grafo.

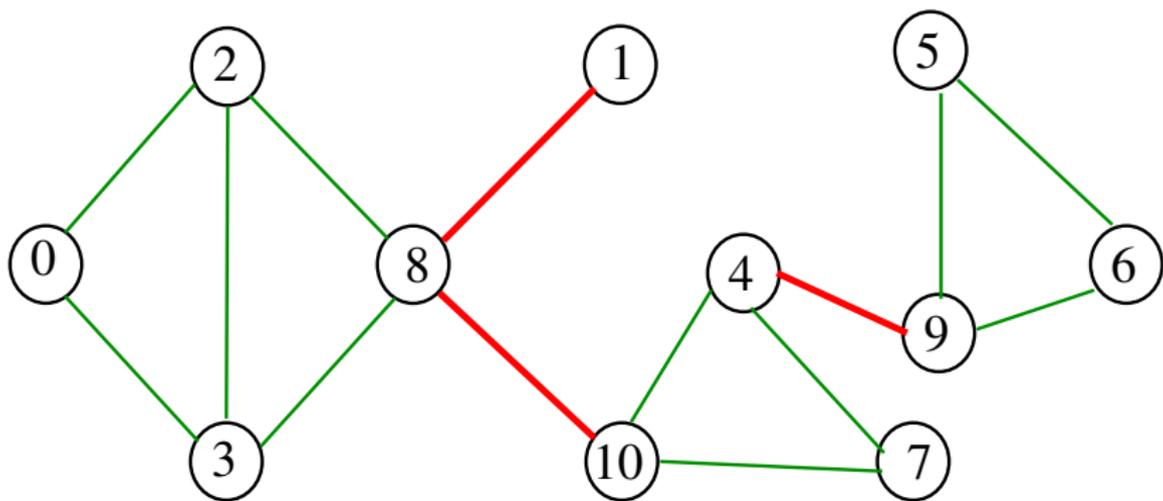
Exemplo:



Pontes em grafos

Uma aresta de um grafo é uma **ponte** (= *bridge* = *separation edge*) se ela é a única aresta que atravessa algum corte do grafo.

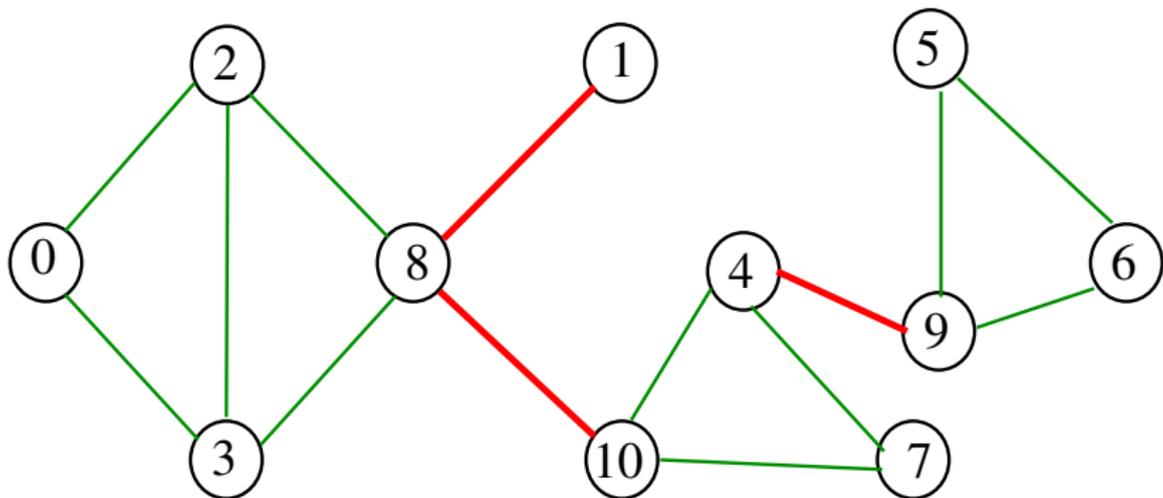
Exemplo: as arestas em **vermelho** são pontes



Procurando pontes

Problema: encontrar as pontes de um grafo dado

Exemplo: as arestas em **vermelho** são pontes



all_bridges1

Recebe um grafo G e calcula o número `bcnt` de pontes do grafo G e imprime todas as pontes.

```
void all_bridges1 (Graph G);
```

Primeiro algoritmo

```
void all_bridges1 (Graph G) {
    Vertex v, w; link p; int output;
1   for (v = 0; v < G->V; v++)
2       for(p=G->adj[v];p!=NULL;p=p->next){
3           w = p->w;
4           if (v < w) {
5               GRAPHremoveA(G,w,v);
6               output = DIGRAPHpath(G,w,v);
7               GRAPHinsertA(G,w,v);
8               if (output == 0) {
9                   bcnt++;
10                  printf("%d-%d\n", v, w);
                    }
                }
            }
}
```

Consumo de tempo

O consumo de tempo da função `all_bridges1` é $A/2$ vezes o consumo de tempo da função `DIGRAPHpath`.

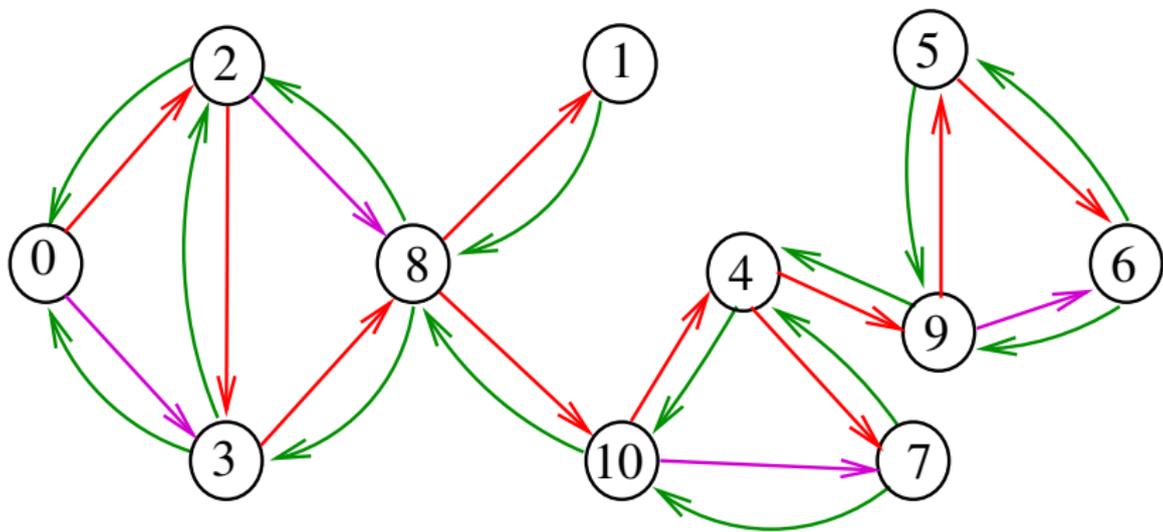
O consumo de tempo da função `all_bridges1` para **vetor de listas de adjacência** é $O(A(V + A))$.

O consumo de tempo da função `all_bridges1` para **matriz de adjacência** é $O(AV^2)$.

Pontes e busca em profundidade

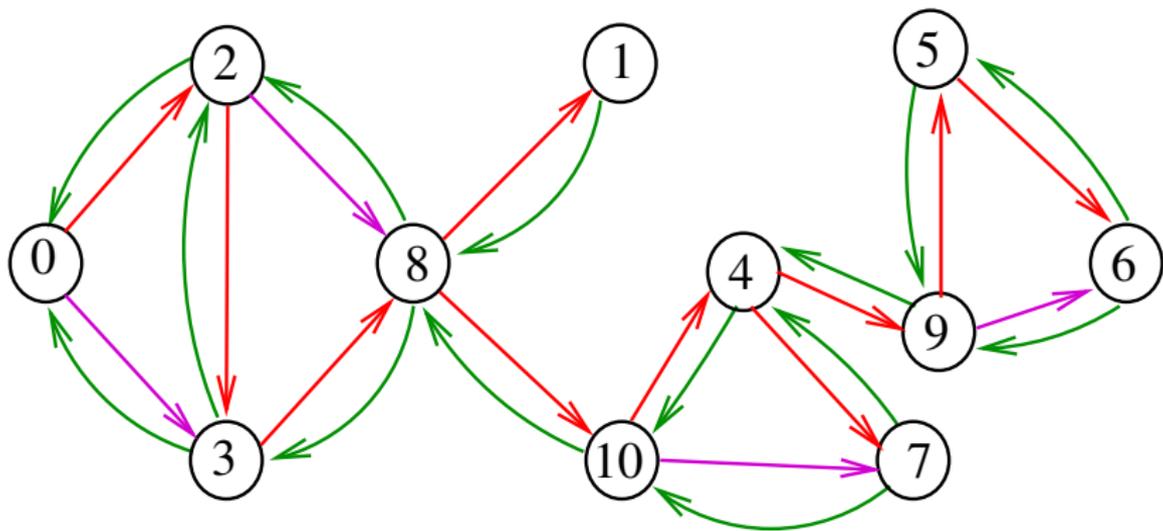
Em uma floresta DFS, um dos dois arcos de cada ponte será um arco da **arborescência**

Exemplo: arcos em **vermelho** são da arborescência



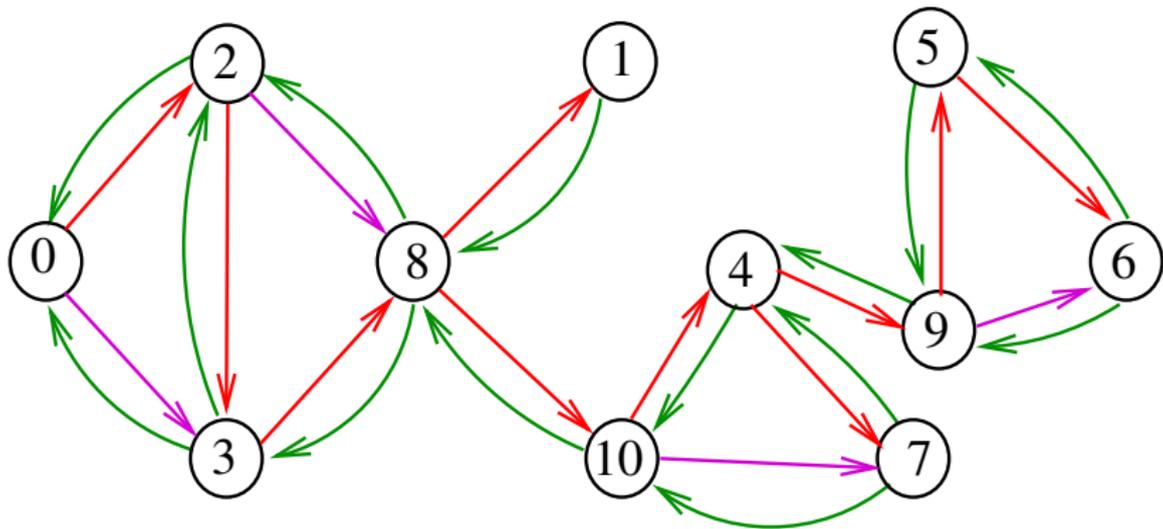
Propriedade

Um arco $v-w$ da **floresta DFS** faz parte (juntamente com $w-v$) de uma ponte se e somente se não existe arco de **retorno** que ligue um descendente de w a um ancestral de v



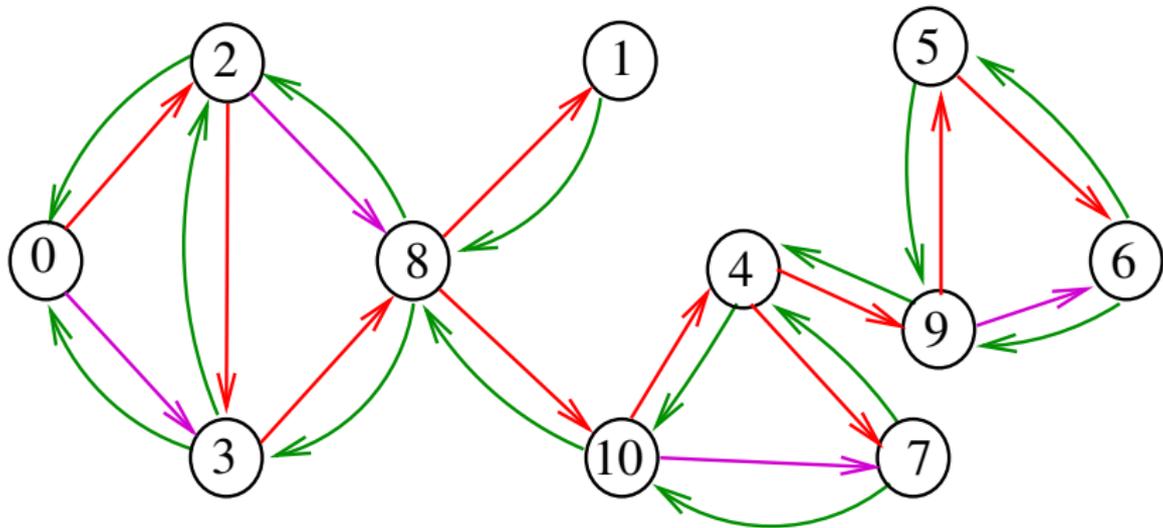
Numeração pré-ordem

v	0	1	2	3	4	5	6	7	8	9	10
pre[v]	0	4	1	2	6	8	9	10	3	7	5



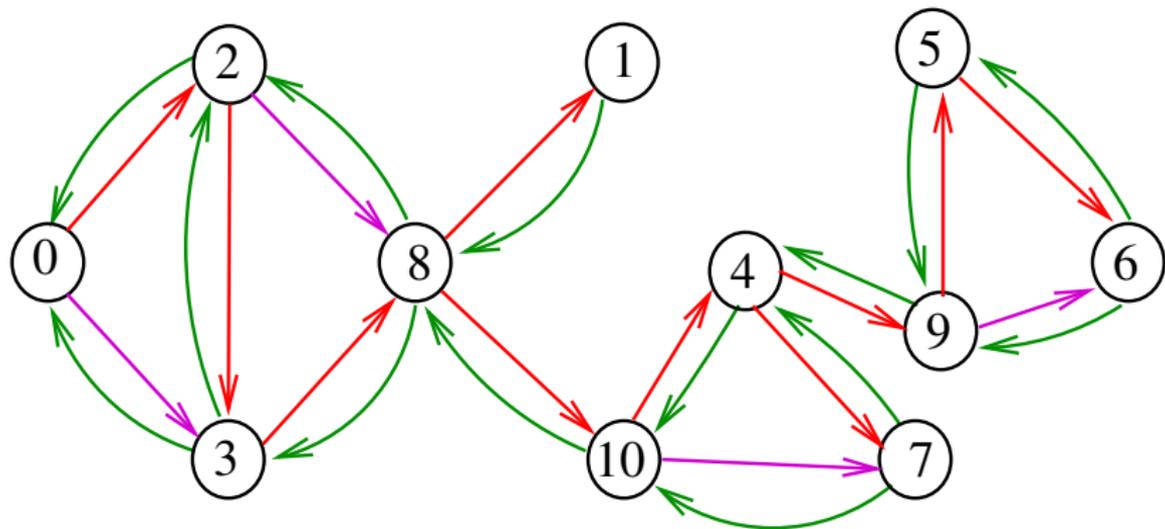
Lowest Preorder Number

O menor **número de pré-ordem** que pode ser alcançado por v utilizando arcos da **arborescência** e **até um** arco de **retorno** (“arco-pai” não vale) será denotado por $\text{low}[v]$



Exemplo

v	0	1	2	3	4	5	6	7	8	9	10
pre[v]	0	4	1	2	6	8	9	10	3	7	5
low[v]	0	4	0	0	5	7	7	5	1	7	5



Observações

Para todo vértice v ,

$$\text{low}[v] \leq \text{pre}[v]$$

Para todo arco $v-w$ do grafo

- ▶ se $v-w$ é um arco de **arborescência** então

$$\text{low}[v] \leq \text{low}[w];$$

- ▶ se $v-w$ é uma arco de **retorno**, então

$$\text{low}[v] \leq \text{pre}[w].$$

Algoritmo das pontes

Em qualquer floresta de busca em profundidade de um grafo, um arco de **arborescência** $v-w$ faz parte de uma ponte se e somente se $low[w] == pre[w]$

```
static int cnt, pre[maxV], bcnt, low[maxV];  
static int parnt[maxV];
```

A função abaixo calcula o número $bcnt$ de pontes do grafo G e imprime todas as pontes

```
void all_bridges (Graph G);
```

all_bridges

```
void all_bridges (Graph G) {  
    Vertex v;  
1    cnt = bcnt = 0;  
2    for (v = 0; v < G->V; v++)  
3        pre[v] = -1;  
4    for (v = 0; v < G->V; v++)  
5        if (pre[v] == -1) {  
6            parnt[v] = v;  
7            bridgeR(G, v);  
        }  
}
```

bridgeR

```
void bridgeR (Graph G, Vertex v) {
link p; Vertex w;
1  pre[v] = cnt++;
2  low[v] = pre[v];
3  for (p=G->adj[v]; p!=NULL; p=p->next)
4      if (pre[w=p->w] == -1) {
5          parnt[w] = v;
6          bridgeR(G, w);
7          if (low[v] > low[w]) low[v]=low[w];
9          if (low[w] == pre[w]) {
10             bcnt++;
11             printf("%d-%d\n", v, w);
           }
       }
12     else if (w!=parnt[v] && low[v]>pre[w])
13         low[v] = pre[w];
}
```

Consumo de tempo

O consumo de tempo da função `all_bridges` é $O(V + A)$.

Aresta-biconexão

Um grafo é **aresta-biconexo** (= 2-edge-connected) ou **2-aresta-conexo** se for conexo e não tiver pontes.

Fato básico importante:

Um grafo é aresta-biconexo se e somente se, para cada par (s, t) de seus vértices, existem (pelo menos) dois caminhos de s a t sem arestas em comum.

Exemplo

É preciso remover **pele menos duas** arestas de um grafo aresta-biconexo para que ele deixe de ser conexo

