AULA 10

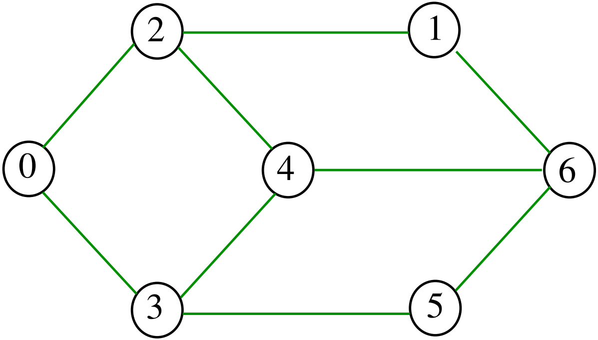
Grafos bipartidos e ciclos ímpares

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

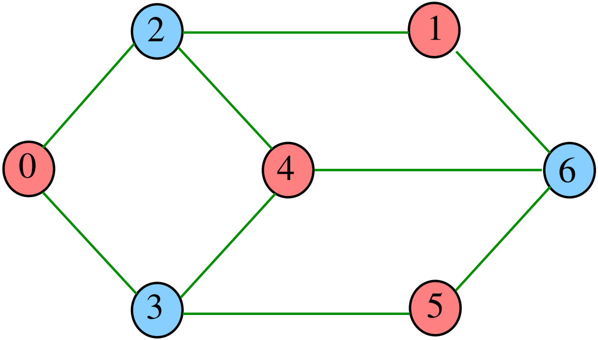
Exemplo:



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

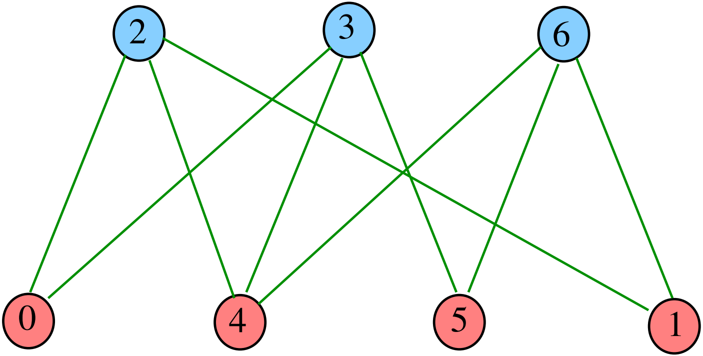
Exemplo:



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;
int c = 0;
for (v = 0; v < G->V; v++)
    color[v] = -1;
for (v = 0; v < G->V; v++)
    if (color[v] == -1)
        if (dfsRclr(G,v,0)==0)
            return 0;
return 1;
}
```

dfsRclr

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

Consumo de tempo

$\boxed{\text{O consumo de tempo da função GRAPHtwocolor para vetor de listas de adjacência é } O(\mathtt{V} + \mathtt{A}).}$

Conclusão

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

- G possui um ciclo ímpar
- G é bipartido

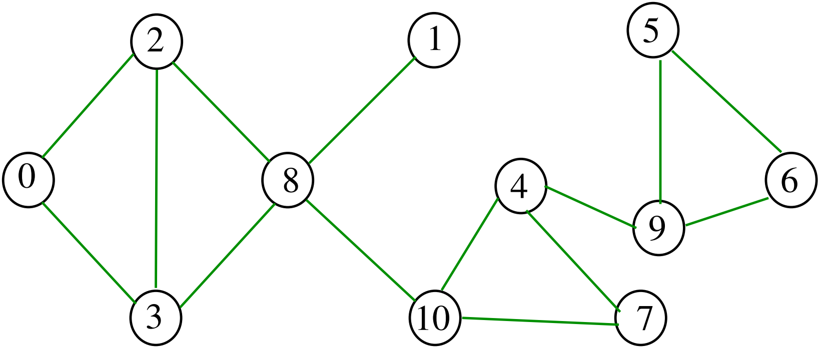
Pontes em grafos e aresta-biconexão

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.

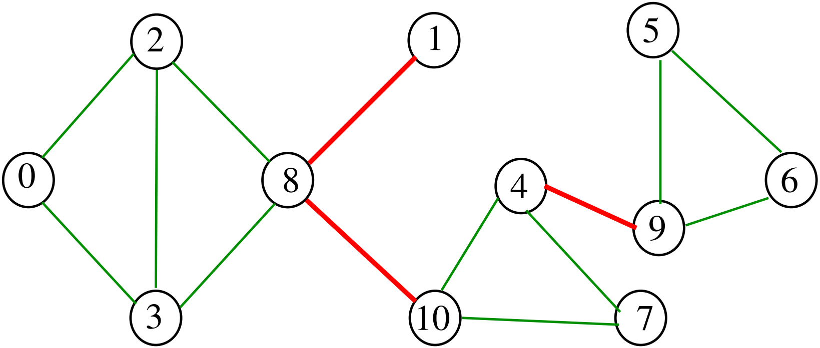
Exemplo:



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.

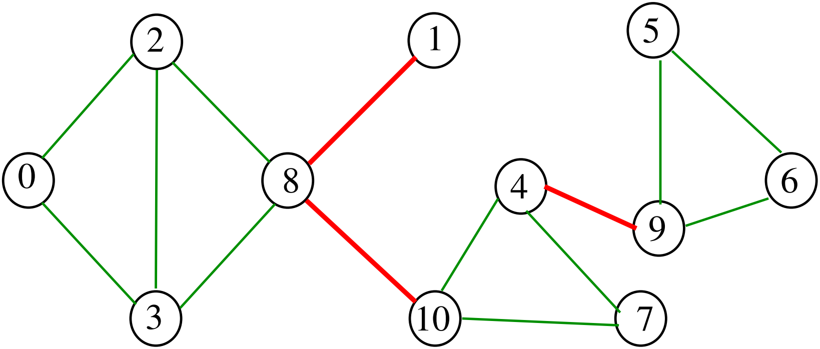
Exemplo: as arestas em vermelho são pontes



Procurando pontes

Problema: encontrar as pontes de um grafo dado

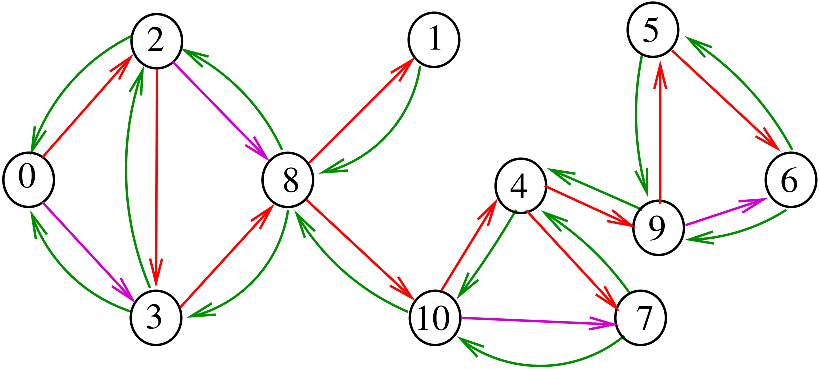
Exemplo: as arestas em vermelho são pontes



Pontes e busca em profundidade

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

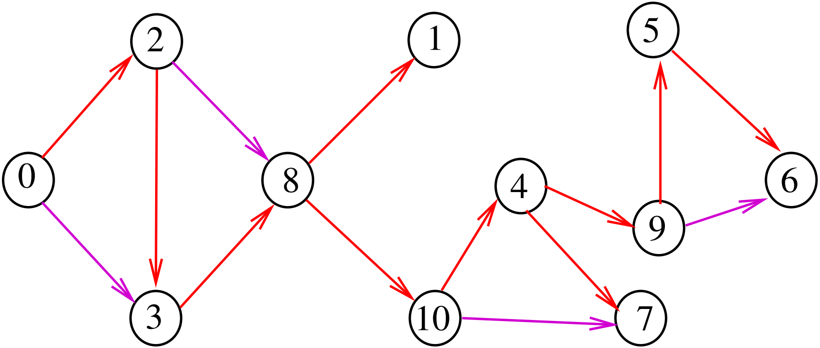
Exemplo: arcos em vermelho são de arborescência



Pontes e busca em profundidade

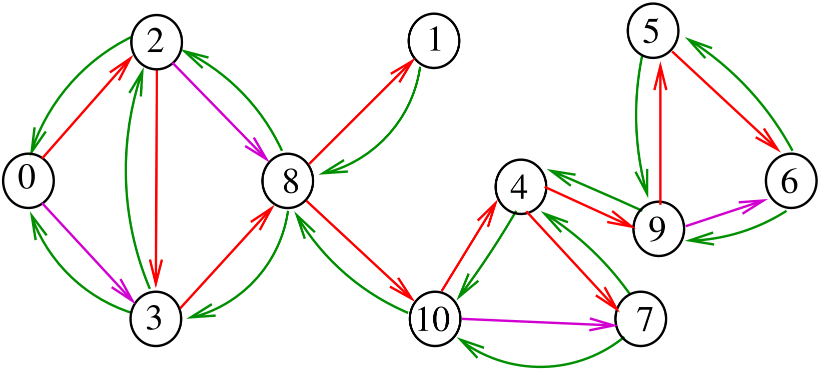
Em uma floresta DFS, um dos dois arcos que de cada ponte será um arco de arborescência

Exemplo: arcos em vermelho são de arborescência



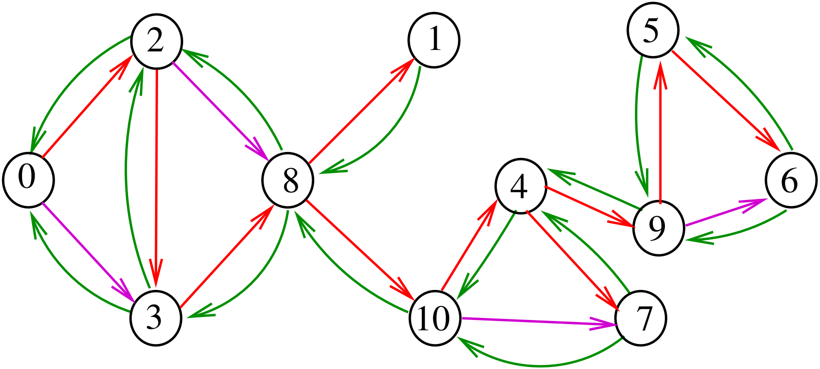
Propriedade

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`



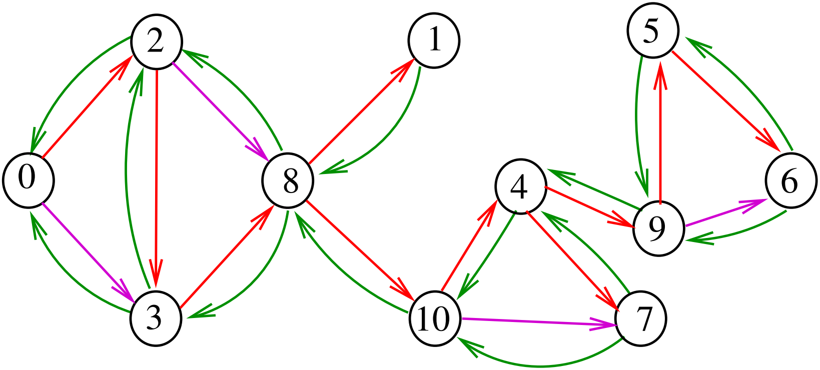
Lowest Preorder Number

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 `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 $\texttt{v}$,

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

Para todo arco $\texttt{v-w}$ do grafo

- se $\texttt{v-w}$ é um arco de **arborescência** então

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

- se $\texttt{v-w}$ é uma arco de **retorno**, então

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

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;
cnt = bcnt = 0;
for (v = 0; v< G->V; v++)
    pre[v] = -1;
for (v = 0; v < G->V; v++)
    if (pre[v] == -1) {
        parnt[v] = v;
        bridgeR(G, v);
}
```

bridgeR

```
void bridgeR (Graph G, Vertex v) {
link p; Vertex w;
 pre[v] = cnt++;
 low[v] = pre[v];
 for (p=G->adj[v];p!=NULL;p=p->next)
      if (pre[w=p->w] == -1) {
           parnt[w] = v;
           bridgeR(G, w);
           if (low[v] > low[w]) low[v]=low[w];
           if (low[w] == pre[w]) {
                bcnt++;
                printf("%d-%d\n", v, w);
           }
      }
      else if (w!=parnt[v] && low[v]>pre[w])
                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** (= `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 pelo menos duas arestas de um grafo aresta-biconexo para que ele deixe de ser conexo