Representação computacional de grafos

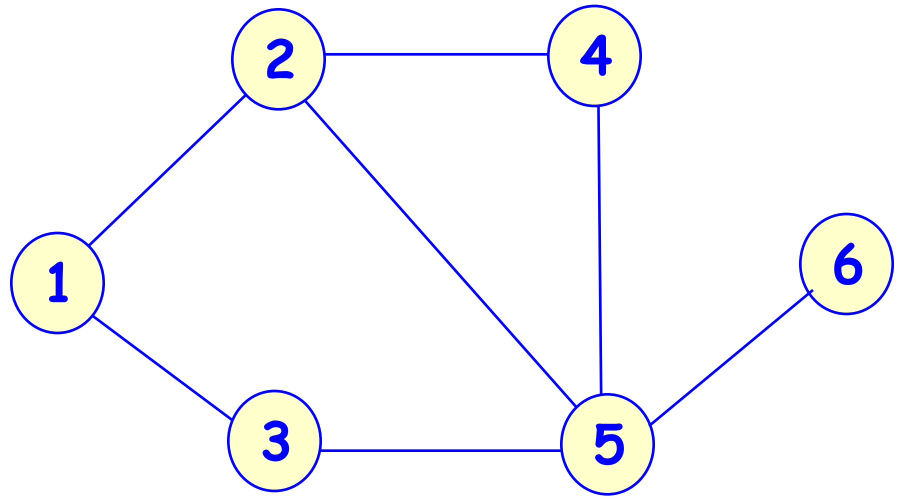
Notas de aula da disciplina IME 04-11312 (Otimização em Grafos)

Paulo Eustáquio Duarte Pinto
(pauloedp at ime.uerj.br)

agosto/2021

Grafos - Representação Computacional

Convenções

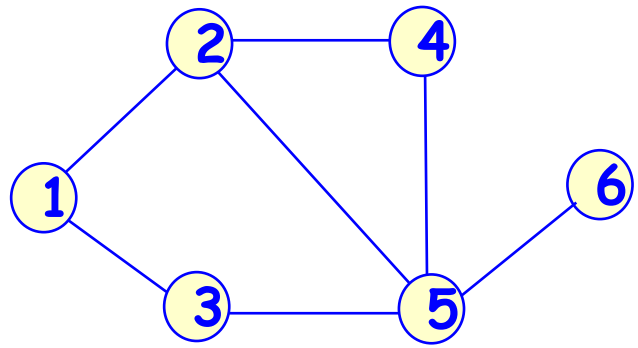


1) Quando não explicitado, supomos que o grafo G(V,E) é grafo simples (sem loops nem multiarestas).

2) Um grafo G(V,E) tem n vértices e m arestas.

3) Os vértices serão numerados de 1 a n.

Grafos - Representação Computacional



Há inúmeras possibilidades de representação computacional de um grafo. Depende dos algoritmos a serem feitos. As principais são:

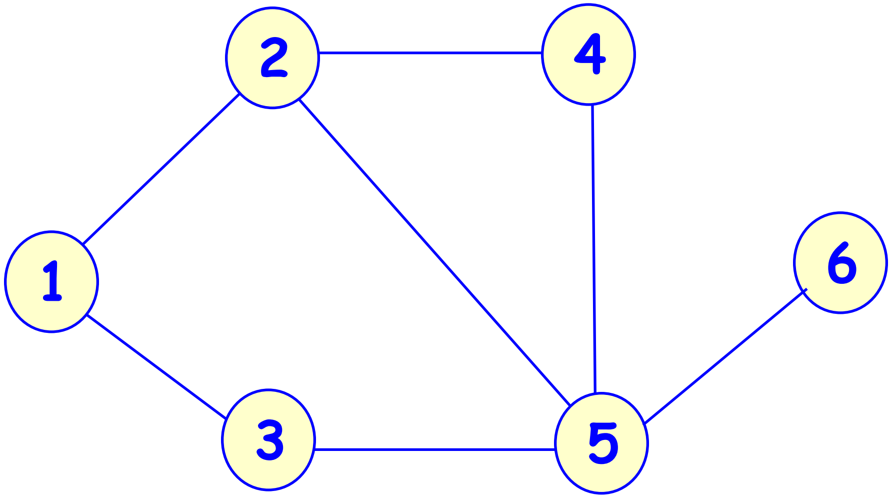
-Matriz de Adjacências
-Listas de Adjacências

Usaremos ainda:
-Up trees
-Vetor de arestas

Mais raramente:
-matriz de incidências
- representação especial

Grafos - Representação Computacional

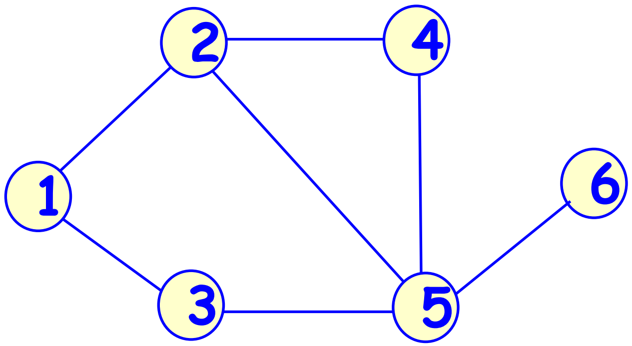
Matriz de adjacências



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

Grafos - Representação Computacional

Matriz de adjacências



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

- Matriz n x n, onde cada valor (u, v) igual a 1 indica que u e v são adjacentes, 0 que não são.

- Matriz simétrica com 2m 1's, pois cada aresta é representada 2 vezes.

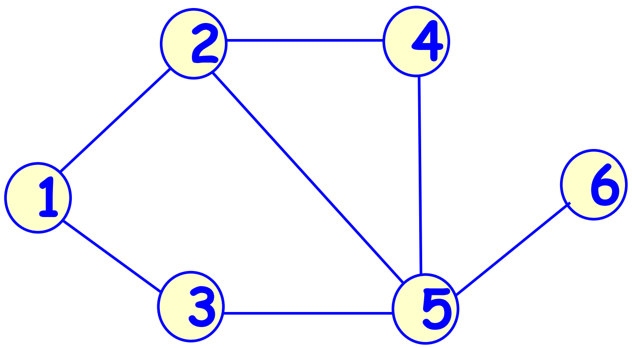
- Diagonal principal zerada.

- Boa para grafos densos. A complexidade de espaço é $O(n^2)$.

- Pode ser estendida para representar parcialmente multigrafos ou grafos simples com pesos nas arestas.

Grafos - Representação Computacional

Leitura de um grafo simples e armazenamento em uma matriz de adjacências

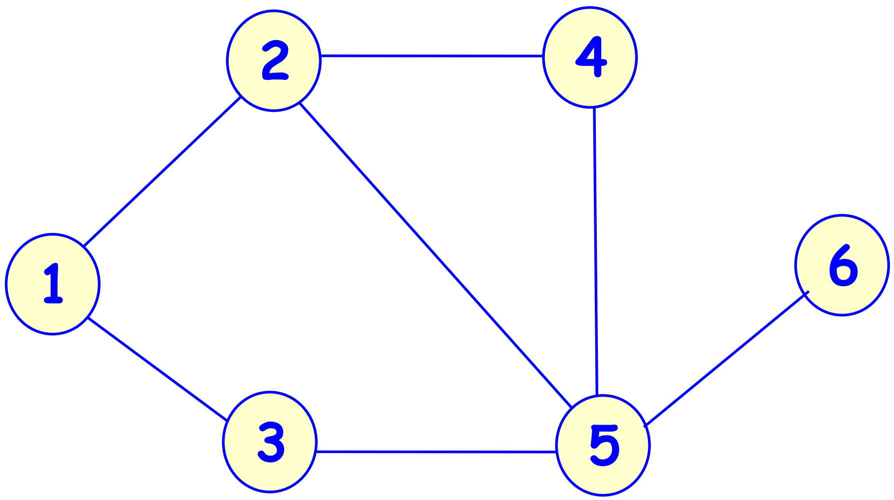


Entrada:
6 7
5 3
2 4
5 6
4 5
2 5
3 1
1 2

```
Leitura():
    ler (n, m)
    ZeraE(n)
    para i ← 1..m incl.:
        ler (u, w)
        E[u,w] ← 1
        E[w,u] ← 1
```

Grafos - Representação Computacional

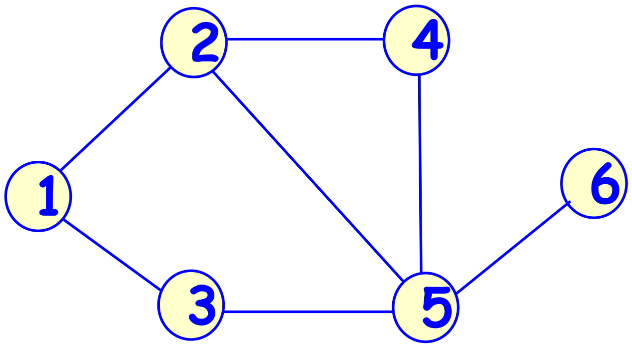
Lista de adjacências-



1 → 2 → 3
2 → 1 → 5 → 4
3 → 5 → 1
4 → 2 → 5
5 → 2 → 3 → 4 → 6
6 → 5

Grafos - Representação Computacional

Lista de adjacências



-Vetor de listas encadeadas, uma para cada vértice, contendo os vizinhos do mesmo.

-Cada aresta é representada 2 vezes.

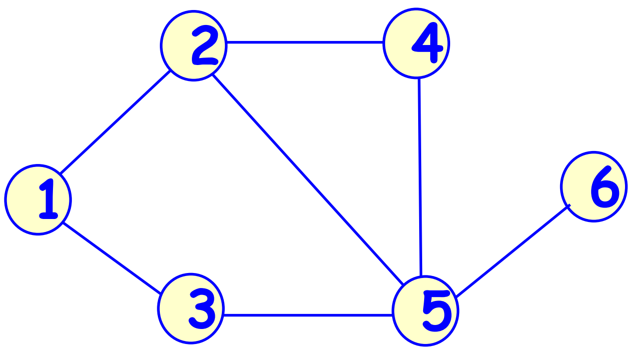
-Boa para grafos esparsos. A complexidade de espaço é O(m).

$1 \to 2 \to 3$
$2 \to 1 \to 5 \to 4$
$3 \to 5 \to 1$
$4 \to 2 \to 5$
$5 \to 2 \to 3 \to 4 \to 6$
$6 \to 5$

-Pode ser estendida para representar parcialmente multigrafos ou grafos simples com pesos nas arestas.

Grafos - Representação Computacional

Leitura de um grafo simples e armazenamento em uma lista de adjacências

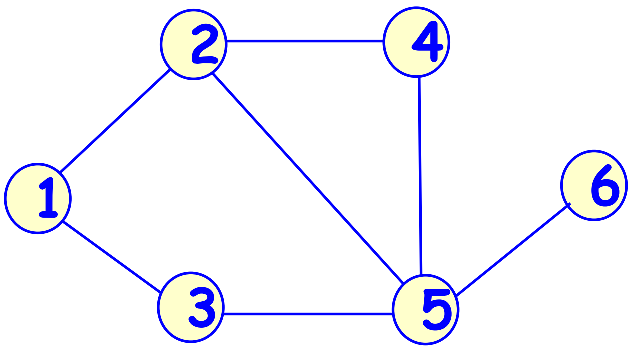


Entrada:
6 7
5 3
2 4
5 6
4 5
2 5
3 1
1 2

```
Leitura():
    ler (n, m)
    AnulaListas()
    para i ← 1..m incl.:
        ler(u, w)
         p ← Nó();          p.v ← w;
         p.prox ← A[u];  A[u] ← p;
         p ← Nó();          p.v ← u;
         p .prox ← A[w];A[w] ← p;
```

Grafos - Representação Computacional

Leitura de um grafo simples e armazenamento em uma lista de adjacências usando vector em C++

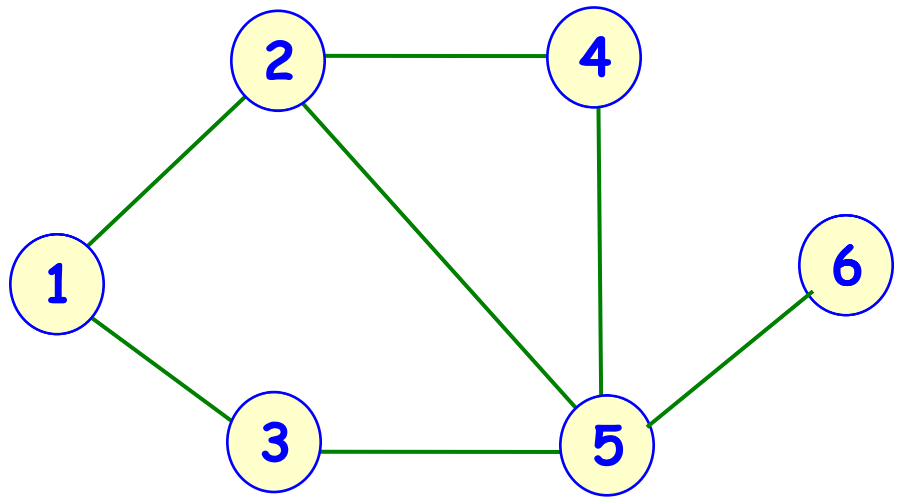


Entrada:
6 7
5 3
2 4
5 6
4 5
2 5
3 1
1 2

```
vector int A[nmax];

Leitura():
    ler (n, m)
    LimpaVectors()
    para i ← 1..m incl.:
        ler(u, w)
        A[u].push_back(w)
        A[w].push_back(u)
```

Grafos - Representação Computacional

Matriz de incidências



	1	2	3	4	5	6	7
1	1	1	0	0	0	0	0
2	1	0	1	1	0	0	0
3	0	1	0	0	1	0	0
4	0	0	1	0	0	1	0
5	0	0	0	1	1	1	1
6	0	0	0	0	0	0	1

Matriz de Incidências

-matriz n x m, onde a célula (u,e) = 1 indica que a aresta e incide no vértice u.

-usada em situações especiais

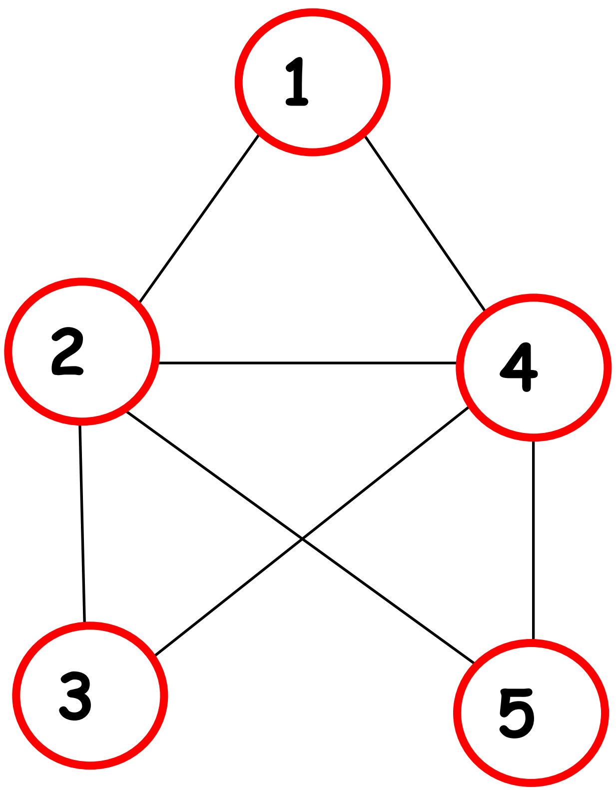
-2m 1's

-2 1's p/ coluna

Grafos – Ciclo e Caminho Euleriano

Ciclo Euleriano- Problema da casinha

Um ciclo Euleriano em um grafo G é um ciclo que contém todas as arestas do grafo.



Um ciclo Euleriano no grafo:

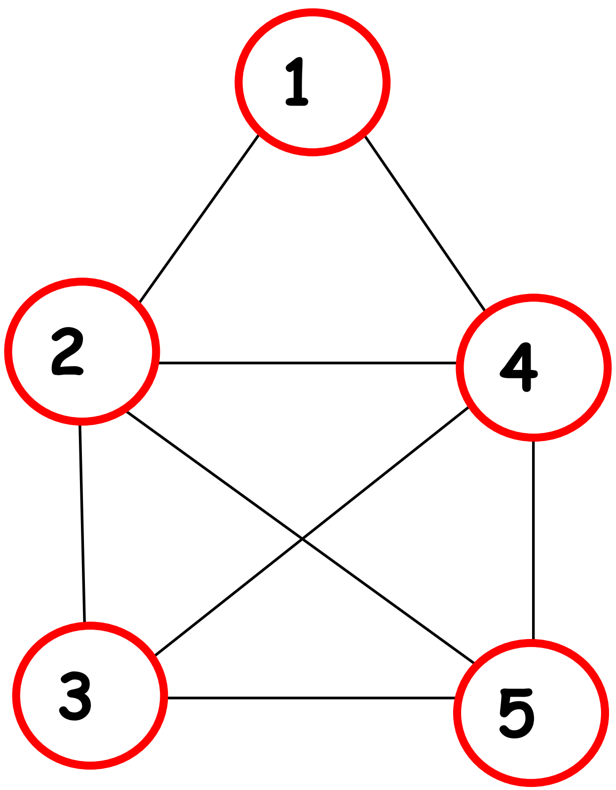
3, 2, 4, 5, 2, 1, 4, 3

Este é um problema importante para empresas de entregas, tais como CocaCola e Correios.

Grafos – Ciclo e Caminho Euleriano

Caminho Euleriano- Problema da casinha

Um caminho Euleriano em um grafo G é um caminho que contém todas as arestas do grafo, podendo o primeiro vértice ser diferente ao último do caminho.



Um caminho Euleriano no grafo:

3, 2, 4, 5, 2, 1, 4, 3, 5

Este grafo possui caminho Euleriano, mas não possui ciclo Euleriano.

Grafos - Determinação de um ciclo Euleriano

Lema 1: Um grafo G simples cujos vértices tenham todos grau maior ou igual a 2 possui um ciclo.

Prova:

A prova é construtiva. Tome um vértice qualquer v e, sucessivamente, vizinhos do último vértice tomado. O novo vértice a entrar no caminho ou já foi tomado antes, o que termina o ciclo, ou possui um vizinho ainda não considerado. Como o número de vértices é finito, o processo de construção sempre acaba.

Grafos - Determinação de um ciclo Euleriano

Teorema 2: Um grafo G possui Ciclo Euleriano sse for conexo e todos os vértices tiverem grau par.

Prova:

⇒: Seja $v_1, \ldots v_k, v_1$ um ciclo Euleriano de G.
Cada ocorrência de um dado vértice v_i no ciclo contribui com 2 unidades para o grau de v_i. Logo, v_i tem grau par.
Para verificar que o grafo é conexo, basta tomar qualquer par de vértices (u, v) e exibir um caminho entre esse par. Isso é feito considerando, no ciclo, uma sequência de vértices que começa com u e termina com v, ou vice-versa. Essa sequência sempre existirá, claro. Eliminam-se todos os ciclos dessa sequência (trechos que começam e terminam no mesmo vértice), deixando apenas o vértice inicial do ciclo. O resultado é claramente um caminho de u para v.

Grafos - Determinação de um ciclo Euleriano

Teorema 9: Um grafo G é Euleriano sse for conexo e todos os vértices tiverem grau par.

Prova:

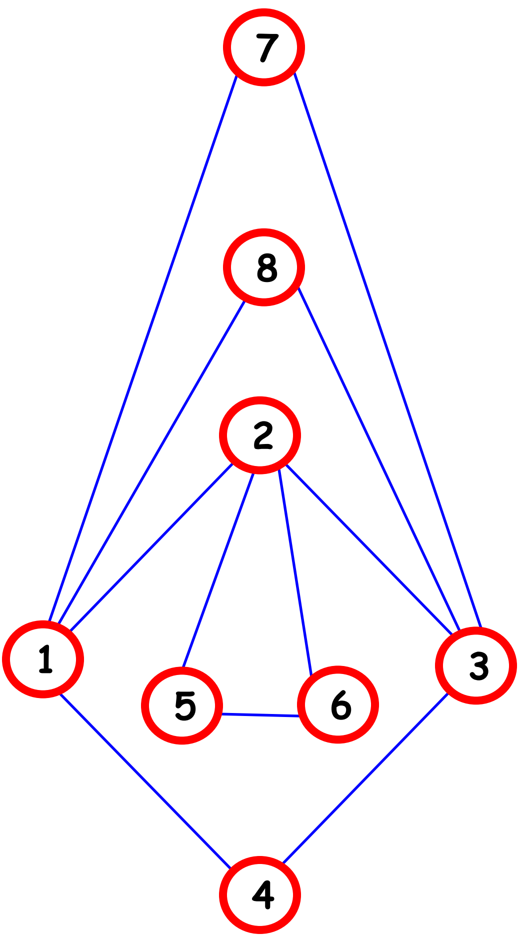
⇐: A prova é por indução em m.

Para m=3, o menor valor possível, o único grafo com as características do teorema é C_3, que tem ciclo Euleriano.

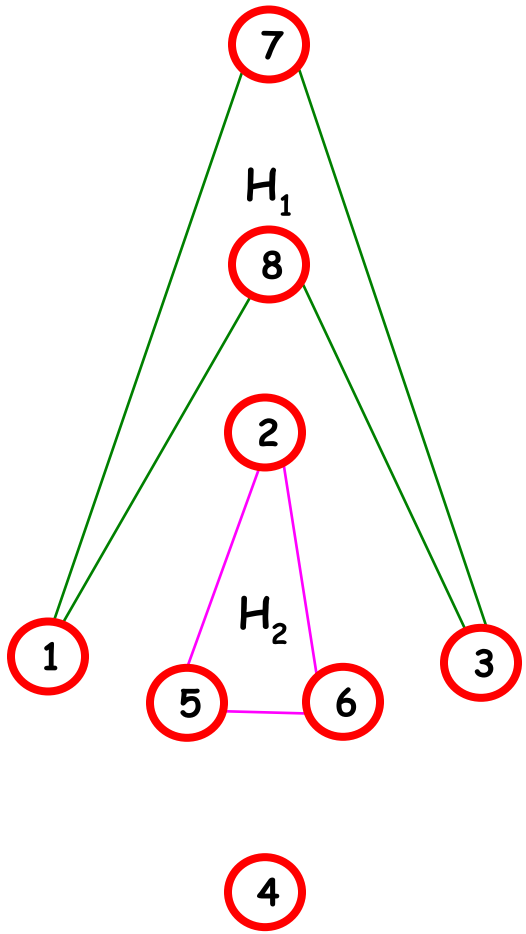
Para m > 3, como todo vértice tem grau par ≥ 2, existe um ciclo C_1 em G (Lema 1). Retirando esse ciclo de G, o grafo restante tem um ou mais componentes onde, cada um deles, é trivial ou tem todos os vértices com grau par. Cada componente tem menos arestas que o grafo original. Além disso, cada componente não trivial tem pelo menos um vértice em comum com C_1. Por indução, cada um desses componentes tem um ciclo Euleriano. Então é possível compor um ciclo Euleriano para G, tomando o ciclo C_1 e inserindo o ciclo Euleriano de cada componente logo após o primeiro vértice de C_1 que é comum ao componente.

Grafos - Determinação de um ciclo Euleriano - Exemplo

Resultado da retirada do ciclo C_1 (1,2,3,4,1)



G



H_3

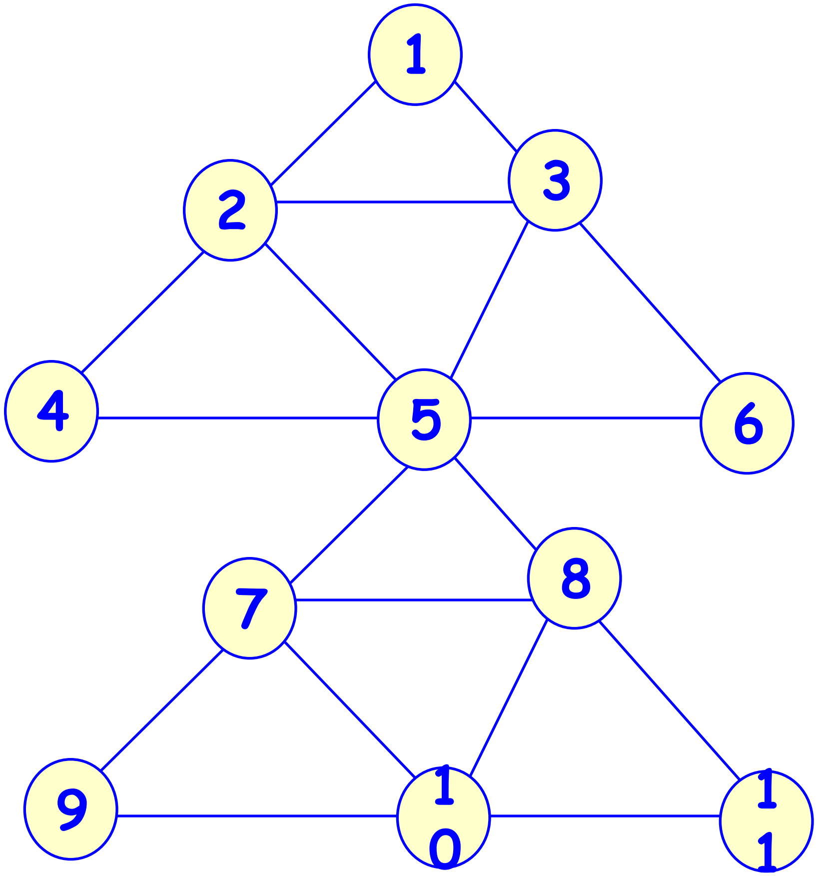
Ciclo Euleriano de H_1:
(1,7,3,8,1)

Ciclo Euleriano de H_2:
(2,6,5,2)

Ciclo Euleriano de G:
(1,7,3,8,1,2,6,5,2,3,4,1)

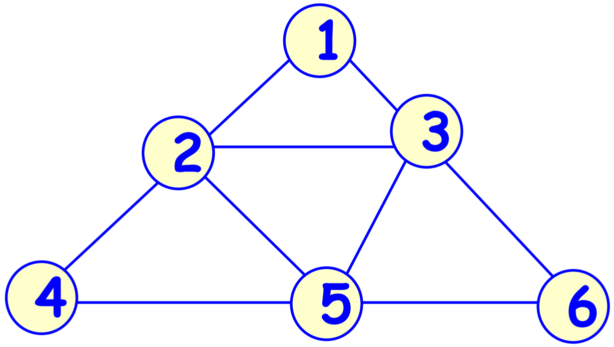
Grafos – Determinação de um ciclo Euleriano

Ex. recomendado: mostrar a construção de um ciclo euleriano usando a idéia do Teorema 2.



Grafos – Determinação de um ciclo Euleriano

Algoritmo iterativo (que destrói o grafo)




```
Ciclo Euleriano(G): #dados grafo G, pilha P, inteiro v
    Esvazia(P)
    Push (P, v)
    enquanto topo ≠ 0:
        v ← P[topo]
        w ← ProxV(v)
        Push (P,w)
        Eliminar (v, w)
        enquanto topo ≠ 0 e d[P[topo]] = 0:
            w ← Pop(P)
            Imprimir w
```

Complexidade: O(m)

Grafos – Determinação de um ciclo Euleriano

Teorema 10: O algoritmo Circuito Euleriano encontra um ciclo euleriano em um grafo euleriano em tempo O(m).

Prova:

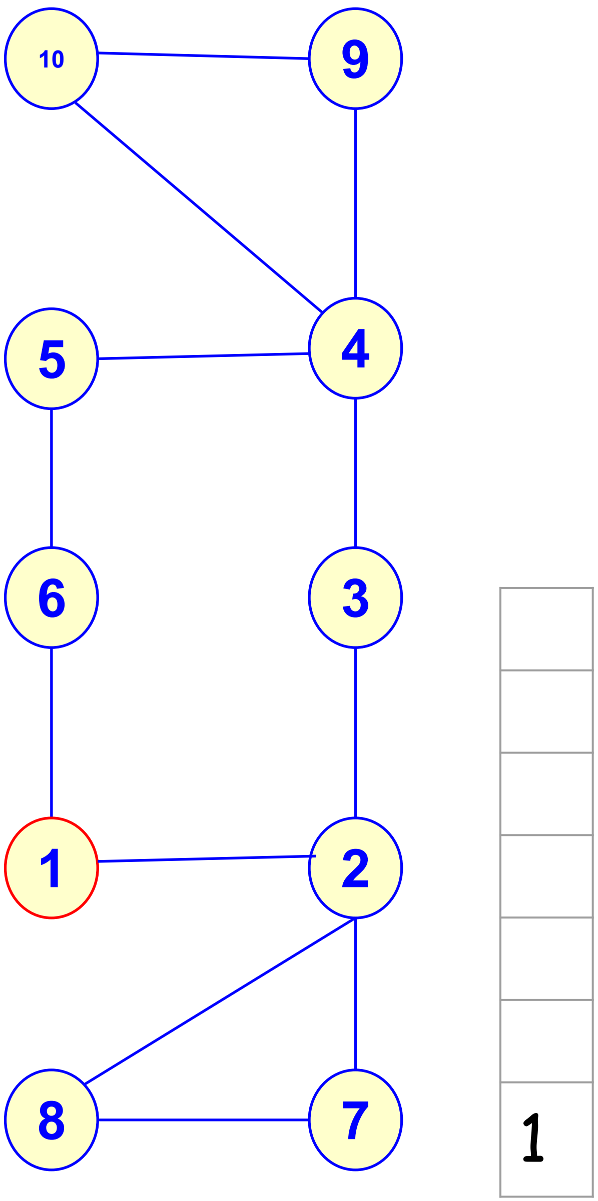
Seja v o vértice inicial escolhido. A sequência da retirada das arestas define um ciclo básico quando é retirada a última aresta incidente ao vértice v, ou seja, seu grau se torna 0. Observe que, nesse momento, a pilha contém exatamente todo esse ciclo básico, não tendo sido retirado nenhum vértice do mesmo.

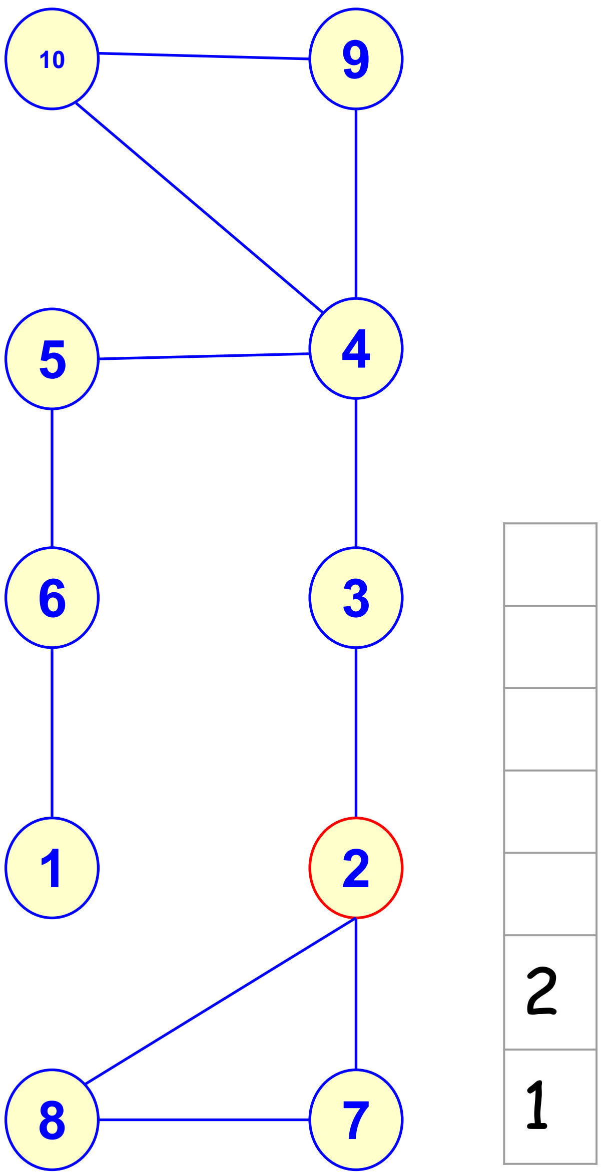
A prova é uma indução no número de vértices com grau diferente de zero ao se determinar o ciclo básico.

Se todos os vértices da pilha têm grau 0, então o ciclo retirado inicialmente é todo o grafo. o algoritmo claramente funciona, pois vai desempilhar o ciclo básico e encerrar.

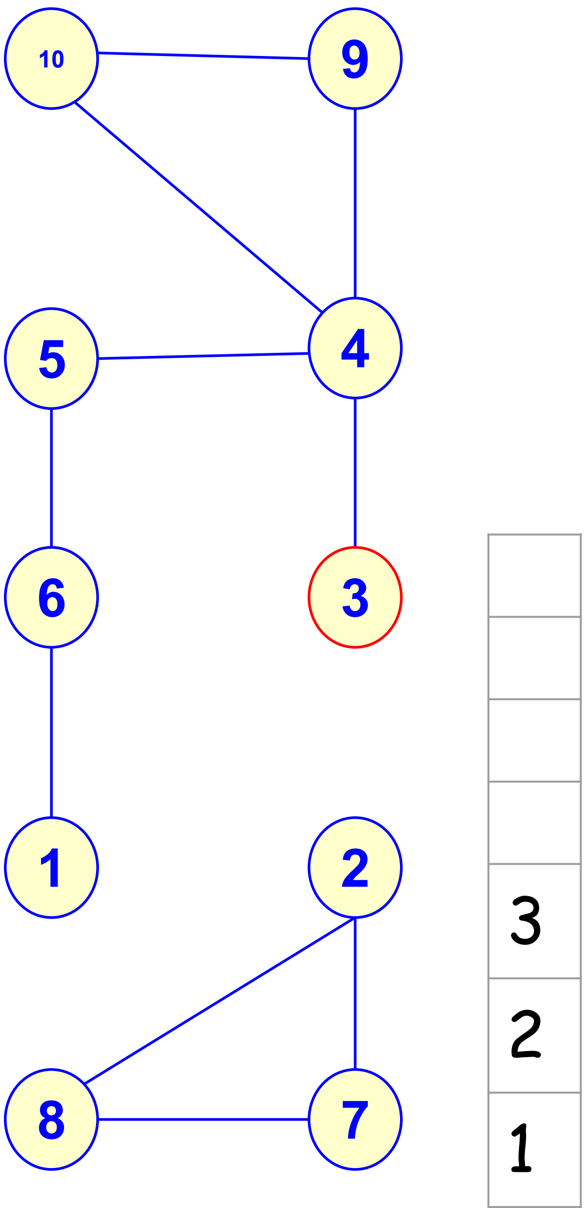
Se o ciclo inicial não é todo o grafo, então sua retirada deixa o grafo com um ou mais componentes, e os vértices do ciclo básico que interceptam os componentes estão todos na pilha e têm grau diferente de zero. Seja w um desses vértices. w só poderá sair da pilha após a saída da pilha de todo(s) o(s) componente(s) que ele intercepta. Isso só ocorre se todos os vértices desse(s) componente(s) forem desempilhados. Logo, a saída de cada componente é um ciclo que se mescla com o ciclo básico, exatamente conforme o teorema de caracterização de um grafo euleriano prova.

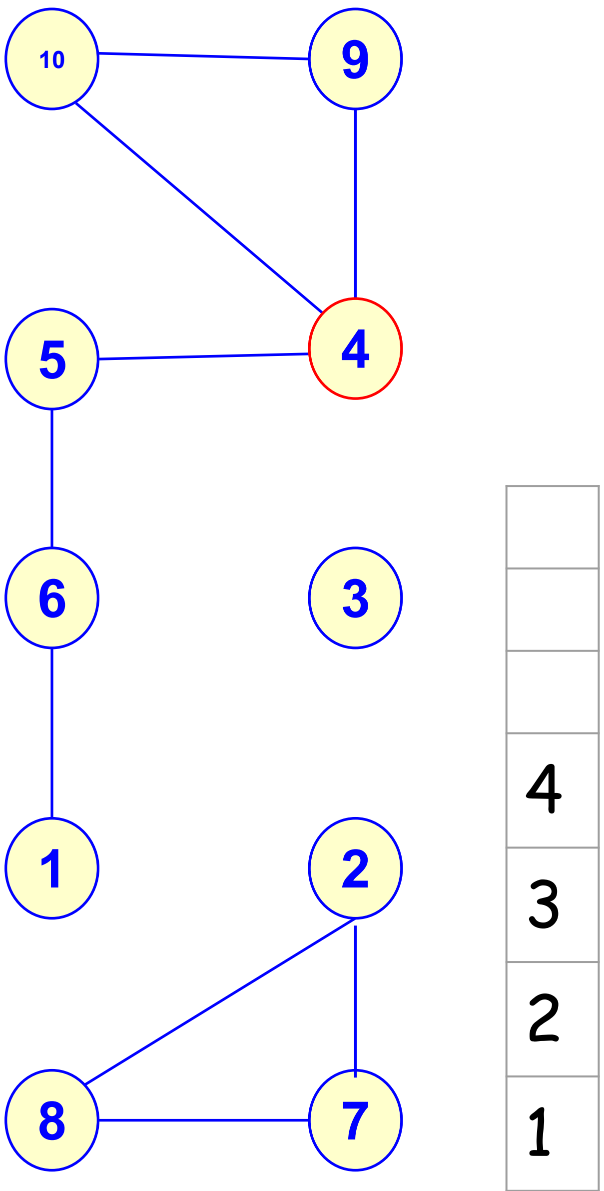
Grafos - Determinação de um ciclo Euleriano



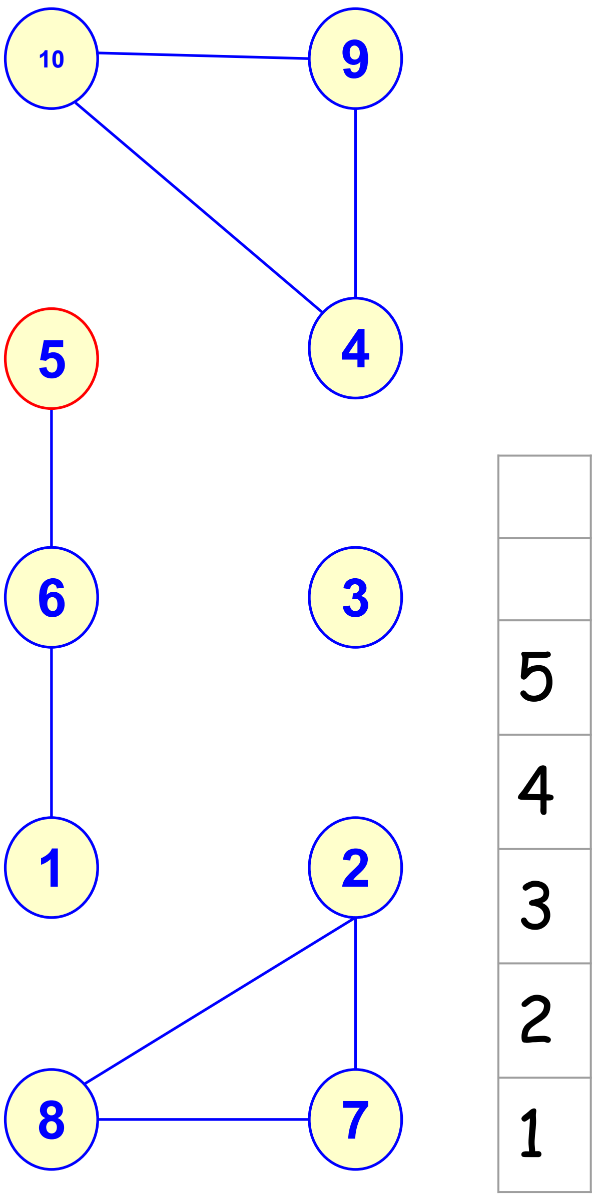


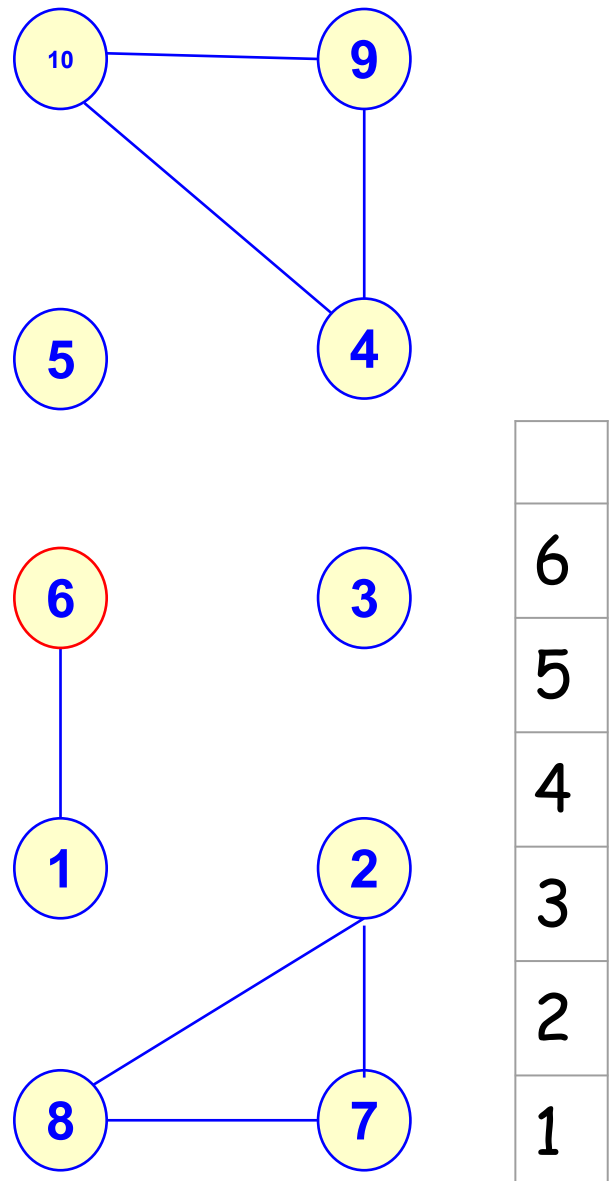
Grafos - Determinação de um ciclo Euleriano



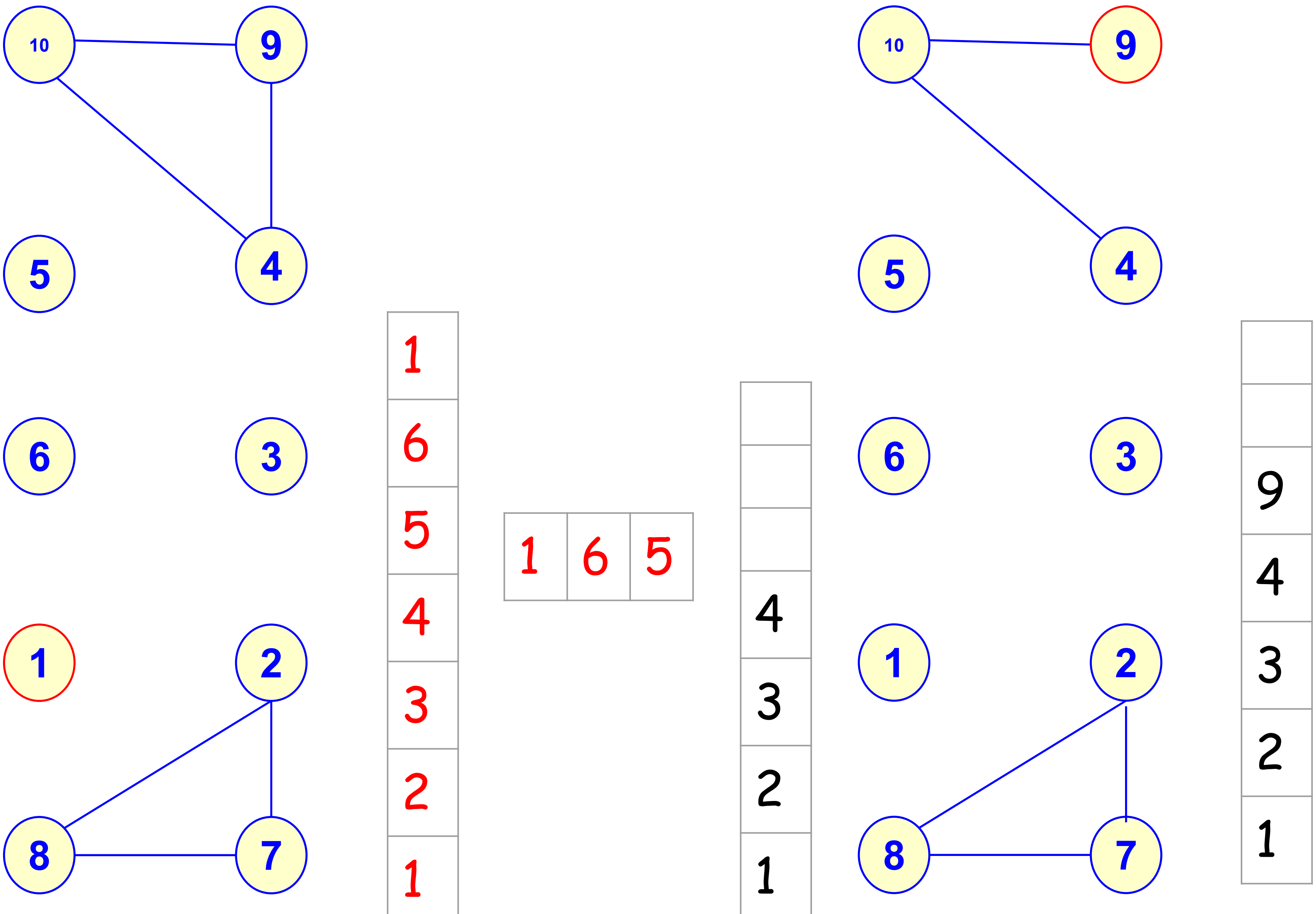


Grafos – Determinação de um ciclo Euleriano

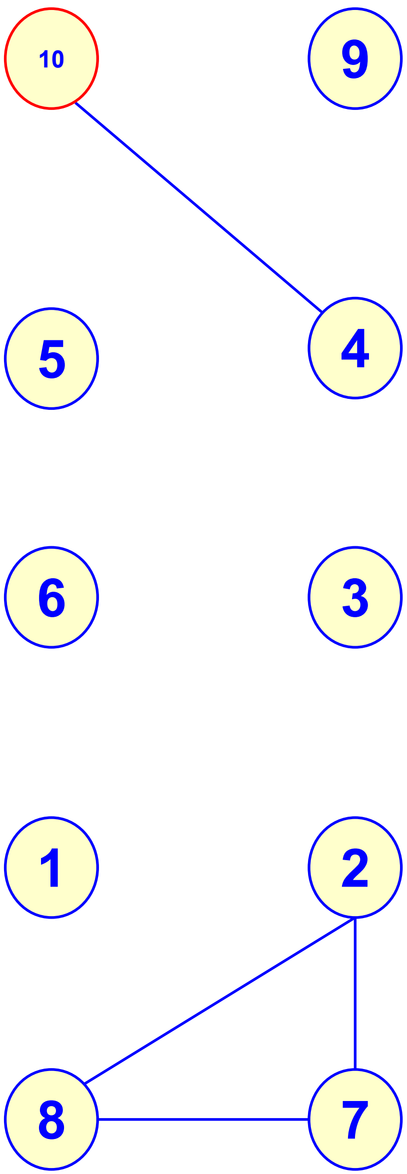




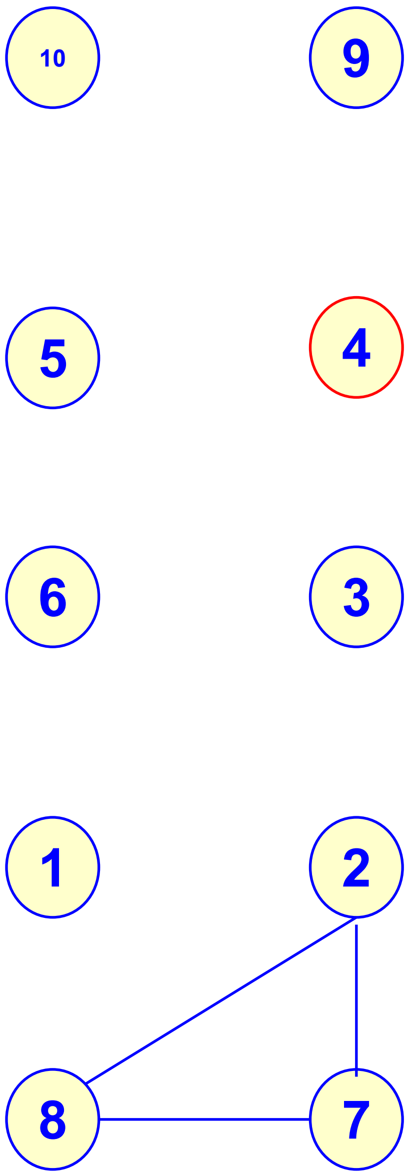
Grafos - Determinação de um ciclo Euleriano



Grafos - Determinação de um ciclo Euleriano



10
9
4
3
2
1

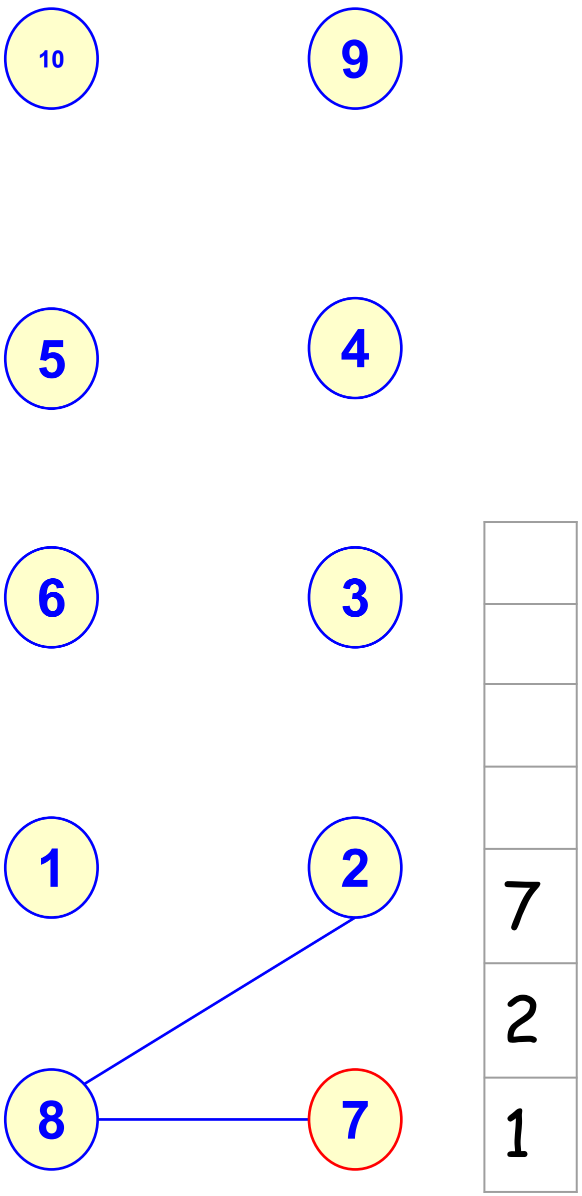


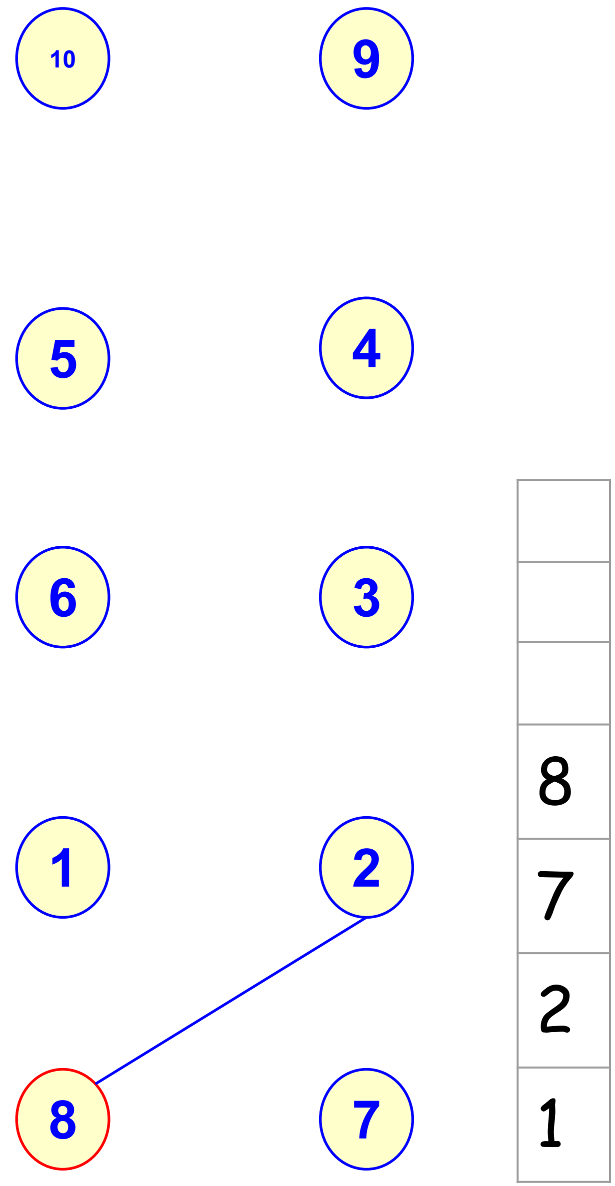
4
10
9
4
3
2
1

4	10	9	4	3

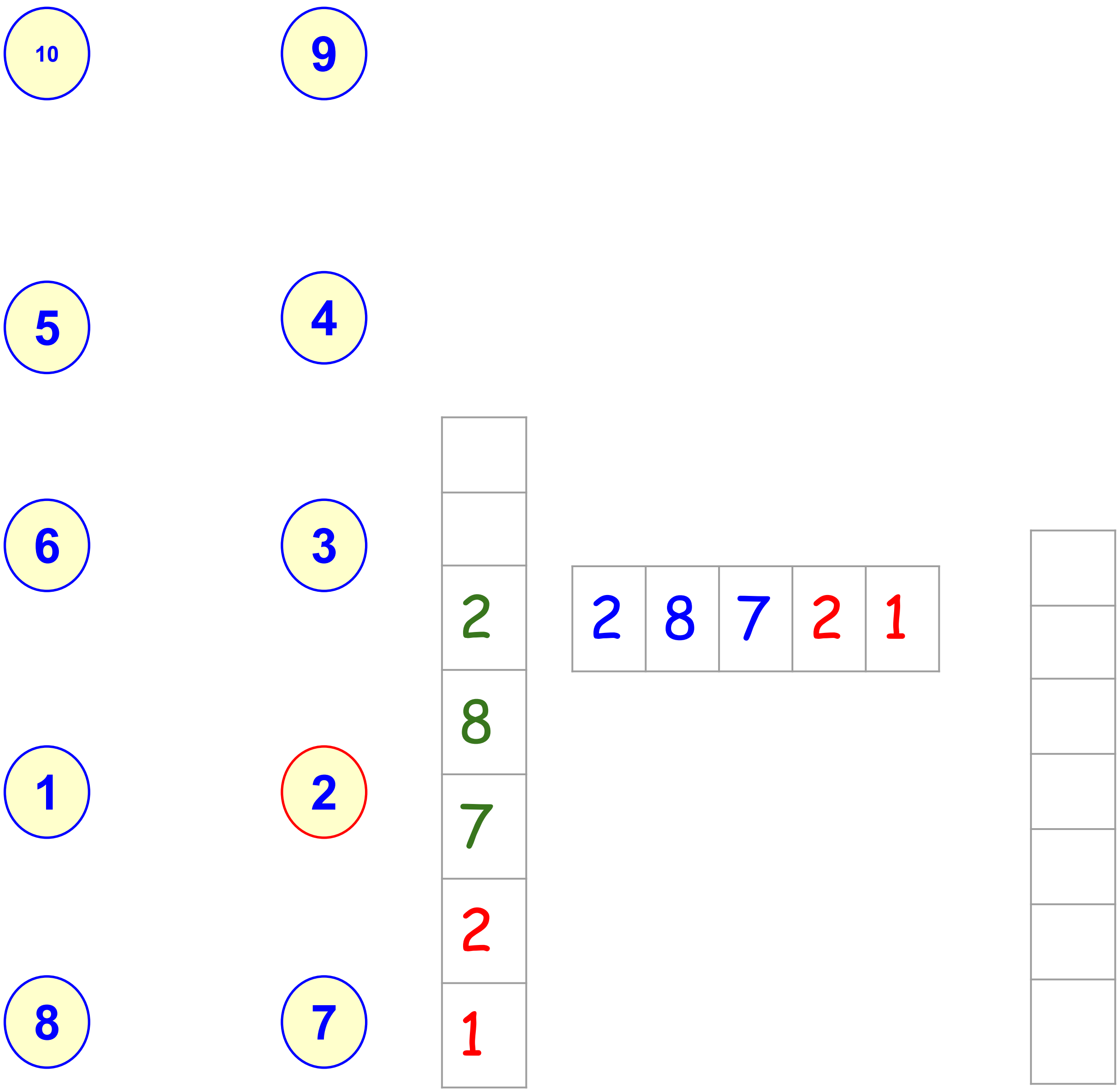
2
1

Grafos – Determinação de um ciclo Euleriano



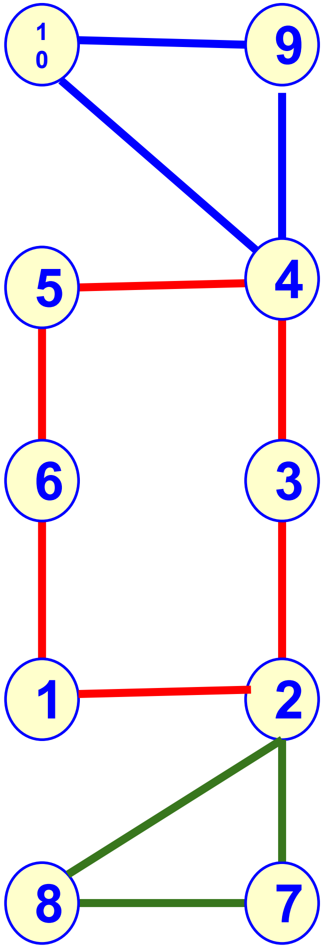


Grafos – Determinação de um ciclo Euleriano



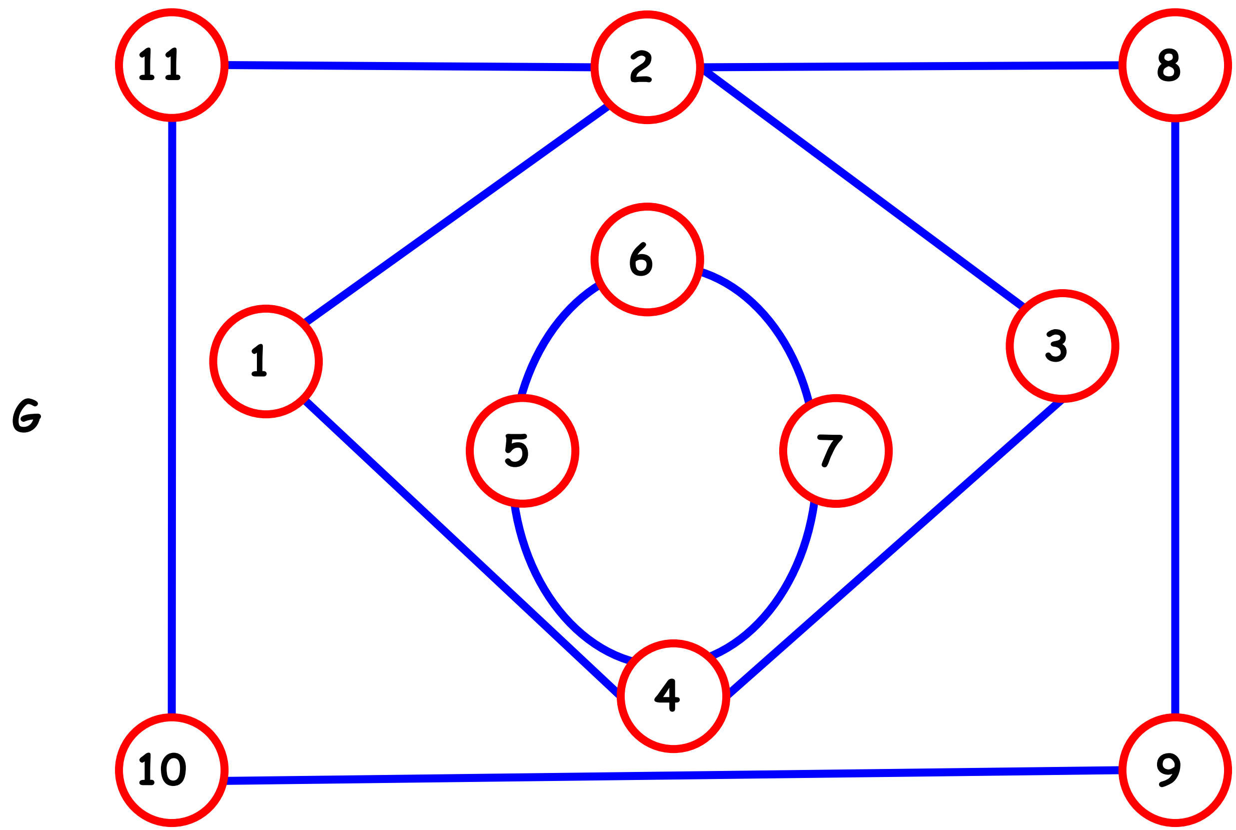
Grafos – Determinação de um ciclo Euleriano

1	6	5	4	10	9	4	3	2	8	7	2	1



Grafos - Determinação de um ciclo Euleriano

ES1: Mostrar a aplicação do algoritmo Ciclo Euleriano ao grafo abaixo, começando no vértice v = NLN mod 11+1, representado com matriz de adjacências.

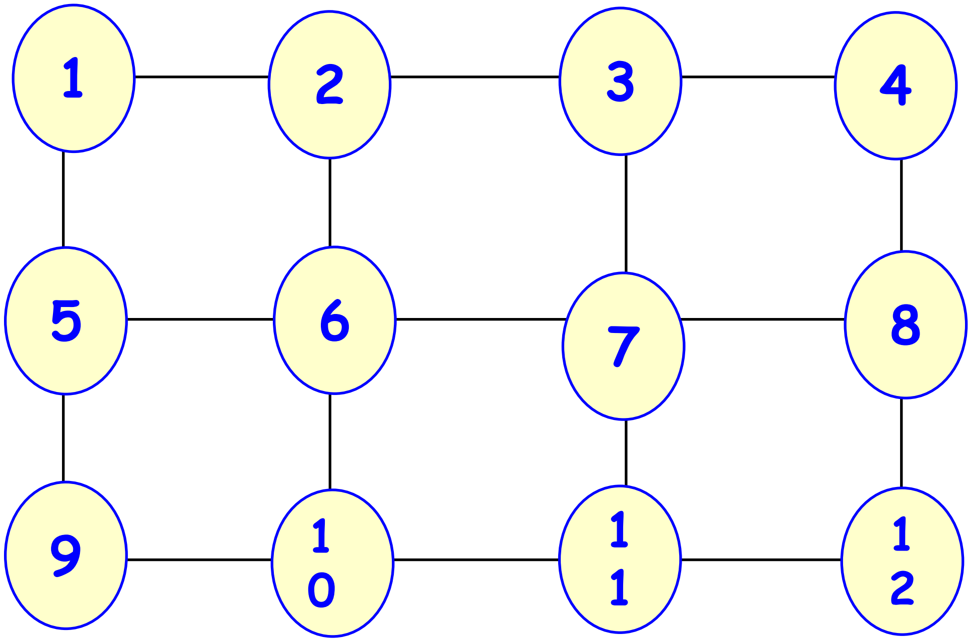


Grafos – Determinação de um ciclo Euleriano

ES2: Fazer as alterações pedidas no programa C++ CircEuleM.cpp

Grafos - Representação Computacional

Outras representações: Aplicam-se a situações especiais.



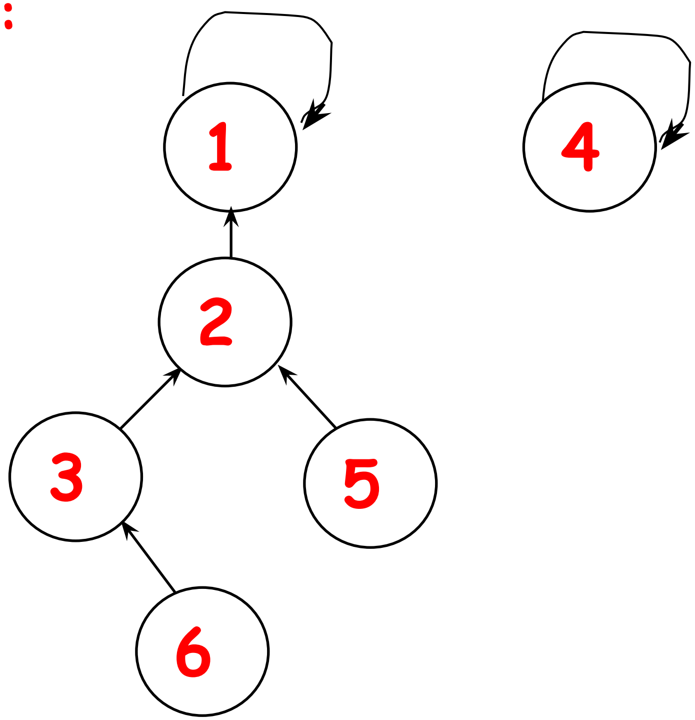
Grid pxq: a lista de adjacências pode ser uma matriz pq x 4, já que cada vértice tem, no máximo 4 vizinhos.

Exercício recomendado: Escrever um algoritmo para gerar um grafo grid p x q

Grafos - Determinação de componentes usando up-trees

Conhecendo-se as arestas do grafo, pode-se determinar seus componentes, usando-se os algoritmos de tratamento de conjuntos (Union-Find), representados em up-trees.

Exemplo:



Uma up-tree é uma árvore onde, para cada nó representa-se apenas um link para seu pai. A raiz da up-tree aponta para si mesma.

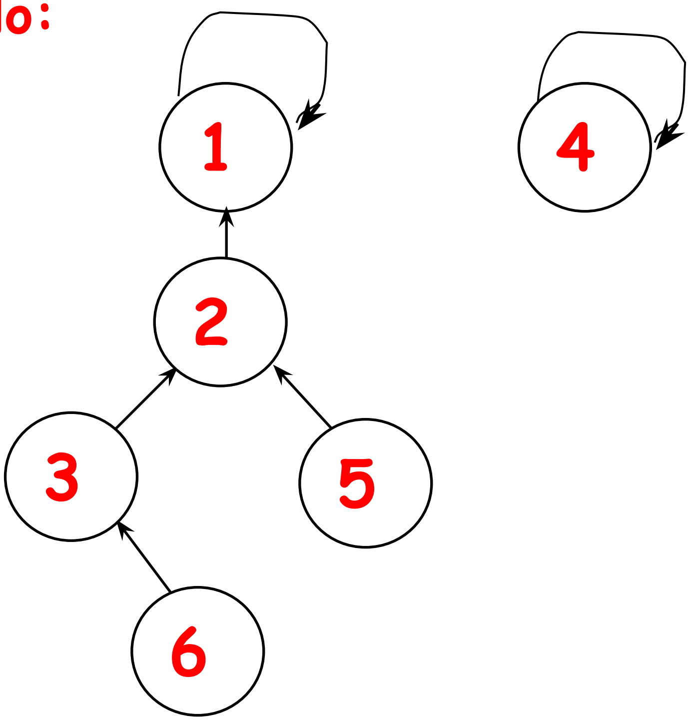
Esta árvore é guardada como um vetor de inteiros para os elementos 1 a n e o link é o conteúdo do vetor.

1	2	3	4	5	6
1	1	2	4	2	3

Grafos - Determinação de componentes usando up-trees

Conhecendo-se as arestas do grafo, pode-se determinar seus componentes, usando-se os algoritmos de tratamento de conjuntos (Union-Find), representados em up-trees.

Exemplo:



1	2	3	4	5	6
1	**1**	**2**	**4**	**2**	**3**

Up-trees são usadas para guardar conjuntos disjuntos. Cada conjunto é definido pela raiz da up-tree.
No exemplo, são representados dois conjuntos: A = {1,2,3,5,6} e B = {4}.

Nesse tipo de representação usam-se duas operações que devem ser bastante eficientes:

Find: determina a raiz da up-tree onde está certo elemento.

Union: une duas up-trees.

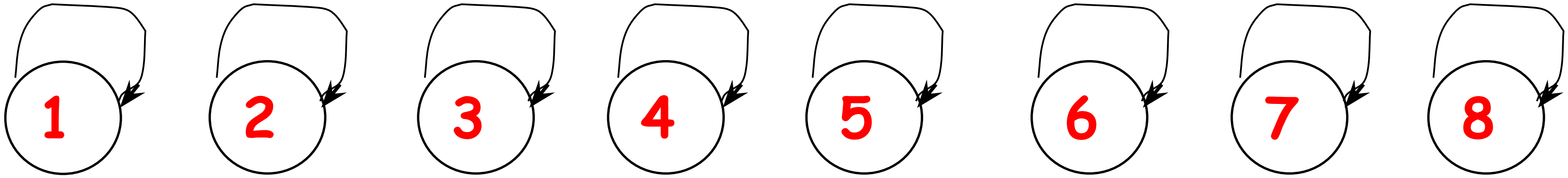
Grafos - Determinação de componentes usando up-trees

Conhecendo-se as arestas do grafo, pode-se determinar seus componentes, usando-se os algoritmos de tratamento de conjuntos (Union-Find), representados em up-trees.

a) Criação de conjuntos com up-trees (representadas em vetor):

Criação(): #dados: inteiro n;

para i ← 1 até n incl.:
pai[i] ← i



1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8

Grafos - Determinação de componentes usando up-trees

b) Busca (Find) da raiz de um elemento com compactação de caminho:

Esta é uma operação para determinar, para dado elemento, qual é a raiz da up-tree à qual ele pertence. É implementada recursivamente, tal que, na volta da recursão, faz-se uma compactação do caminho percorrido até a raiz. Essa compactação consiste em fazer todos os nós visitados apontarem para a raiz.

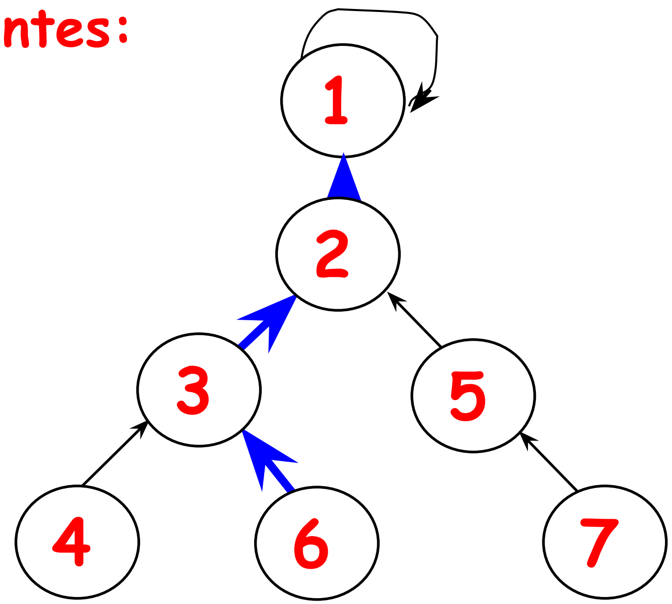
```
Busca(p):  #dados: inteiro p

    se pai[p] ≠ p:
        pai[p] ← Busca(pai[p])
    retornar pai[p]
```

Grafos - Determinação de componentes usando up-trees

b) Busca (Find) da raiz de um elemento com compactação de caminho:

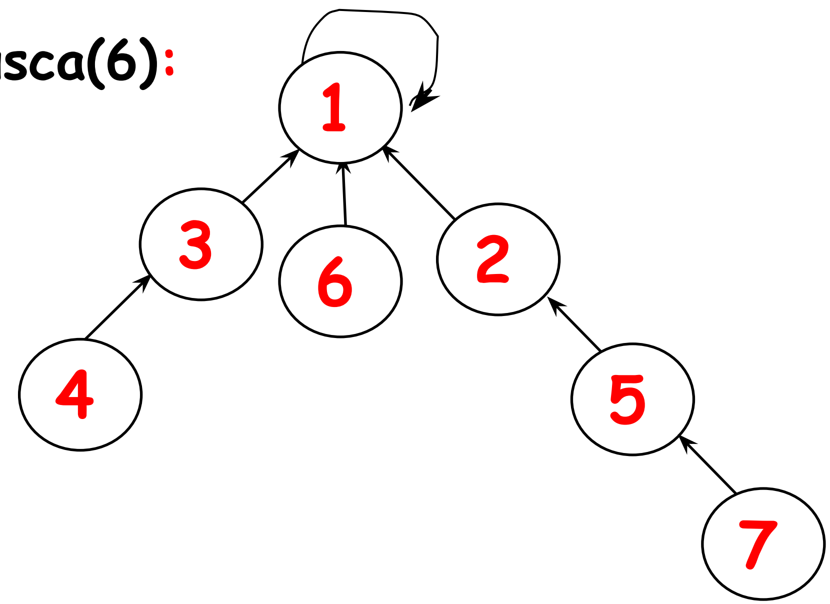
```
Busca(p):  #dados: inteiro p
    se pai[p] ≠ p:
        pai[p] ← Busca(pai[p])
    retornar pai[p]
```

Antes:



1	2	3	4	5	6	7
1	1	2	3	2	3	5

Após Busca(6):



1	2	3	4	5	6	7
1	1	1	3	2	1	5

Grafos – Determinação de componentes

c) União de conjuntos por valor da raiz;

Dados dois elementos, esta operação determina em qual up-tree está cada elemento e, quando são up-trees diferentes, junta-as em uma única. A raiz passa a ser aquela menor dentre as duas.
Observe que esta operação também pode ser implementada tomando-se como raiz aquela que tem mais elementos.

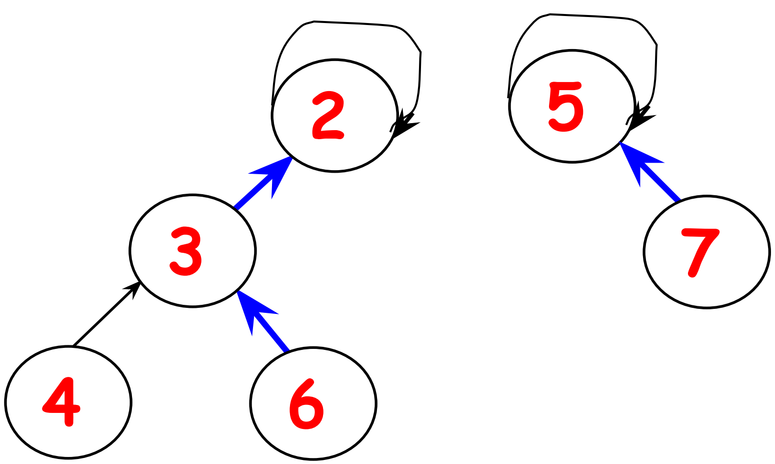
```
Uniao(p,q):  #dados: inteiro p, q;

    Pp ← Busca(p);   Pq ← Busca(q);
    se (Pp < Pq):
        pai[Pq] ← Pp
    senão:
        pai[Pp] ← Pq
```

Grafos - Determinação de componentes

c) União de conjuntos por valor da raiz;

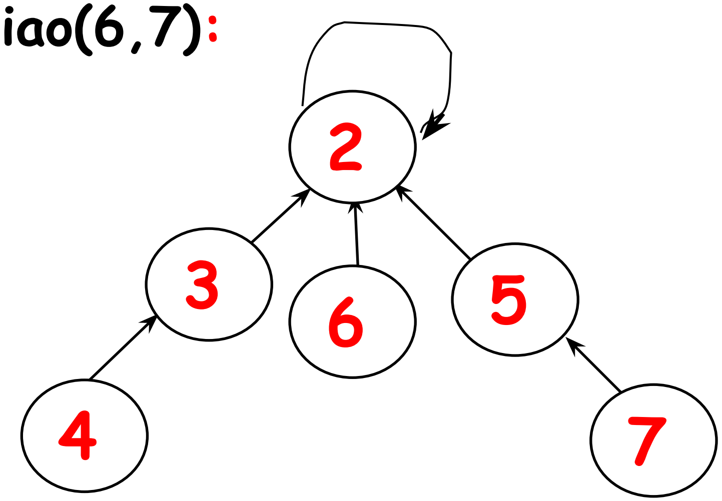
Uniao(p,q): #dados: inteiro p, q;
    Pp ← Busca(p); Pq ← Busca(q);
    se (Pp < Pq):
        pai[Pq] ← Pp
    senão:
        pai[Pp] ← Pq

Antes:



1	2	3	4	5	6	7
1	2	2	3	5	3	5

Após Uniao(6,7):



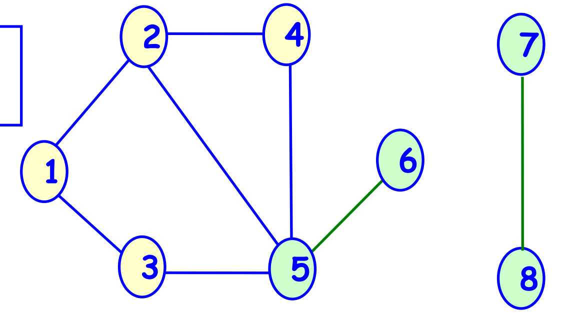
1	2	3	4	5	6	7
1	2	2	3	2	2	5

Grafos – Determinação de conectividade

Leitura do grafo e determinação de conectividade;

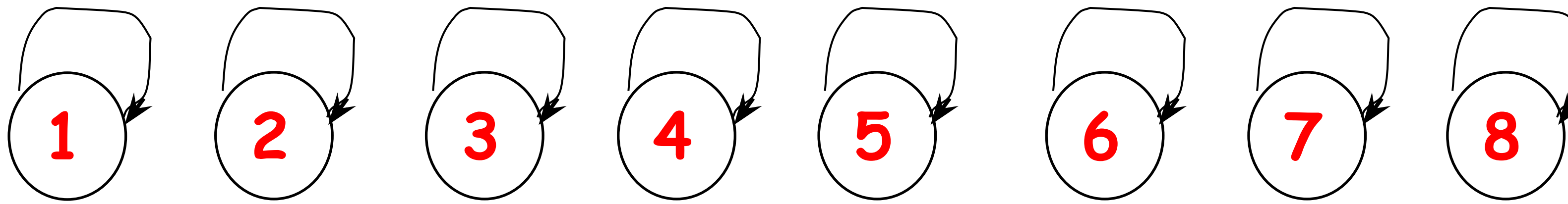
```
Componentes():

    ler(n,m)
    Criação(n)
    para i ← 1.. m incl.:
        ler(u,v)
        Uniao(u,v)
    c ← 0
    para i ← 1.. n incl.:
        se pai[i] = i:
            c ← c+1
    se c=1:
        escrever ("Conexo")
    senão:
        escrever ("Desconexo")
```

Grafos - Determinação de componentes

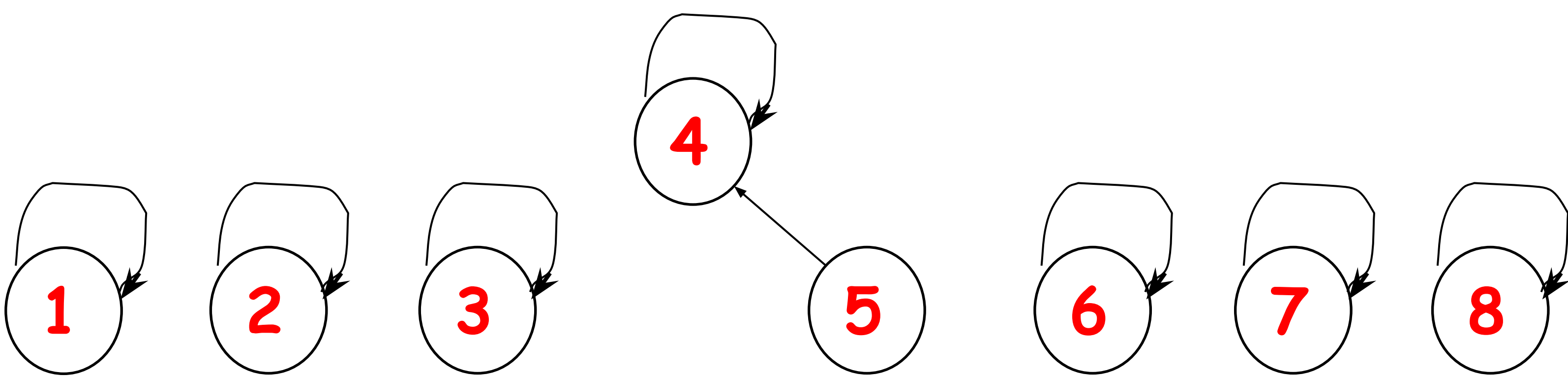


Exemplo (n=8):

u	v
5	4

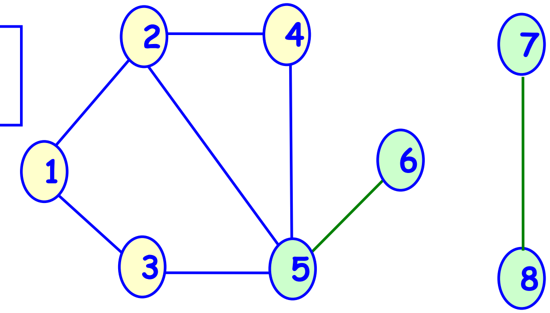


1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8



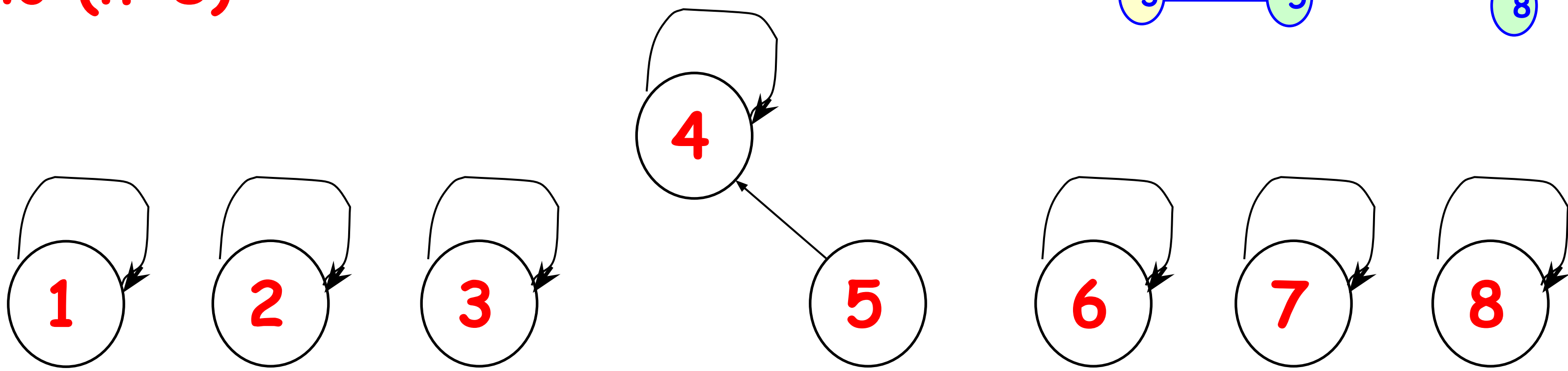
1	2	3	4	5	6	7	8
1	2	3	4	4	6	7	8

Grafos - Determinação de componentes

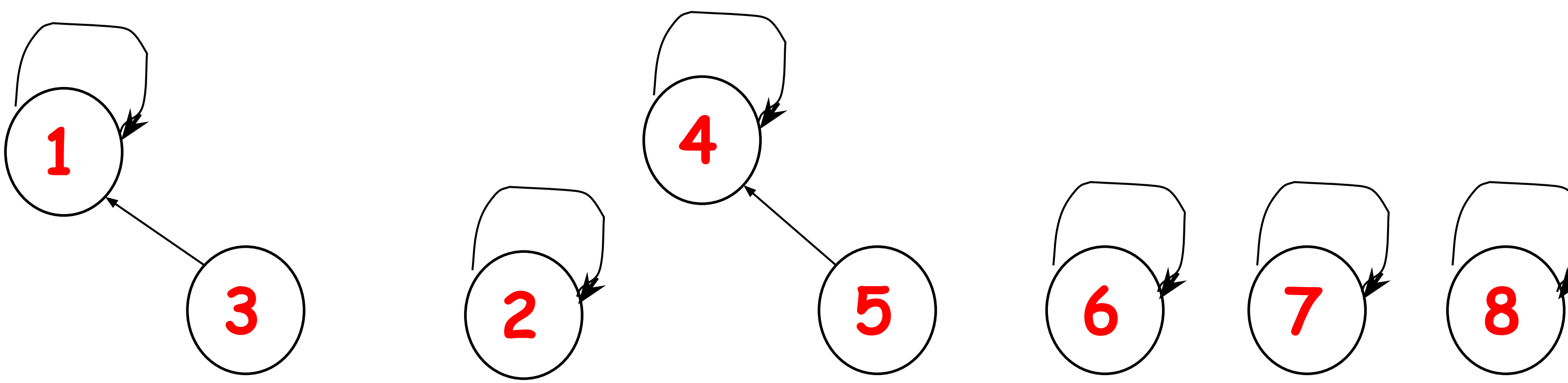


Exemplo (n=8):

u	v
5	4
1	3

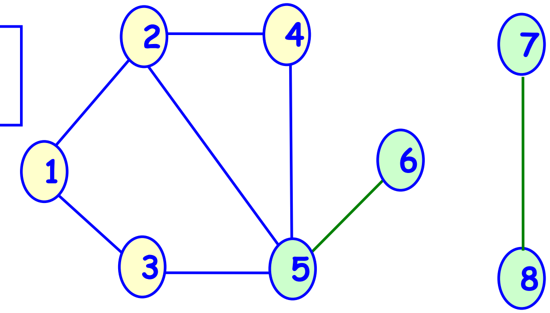


1	2	3	4	5	6	7	8
1	2	3	4	4	6	7	8

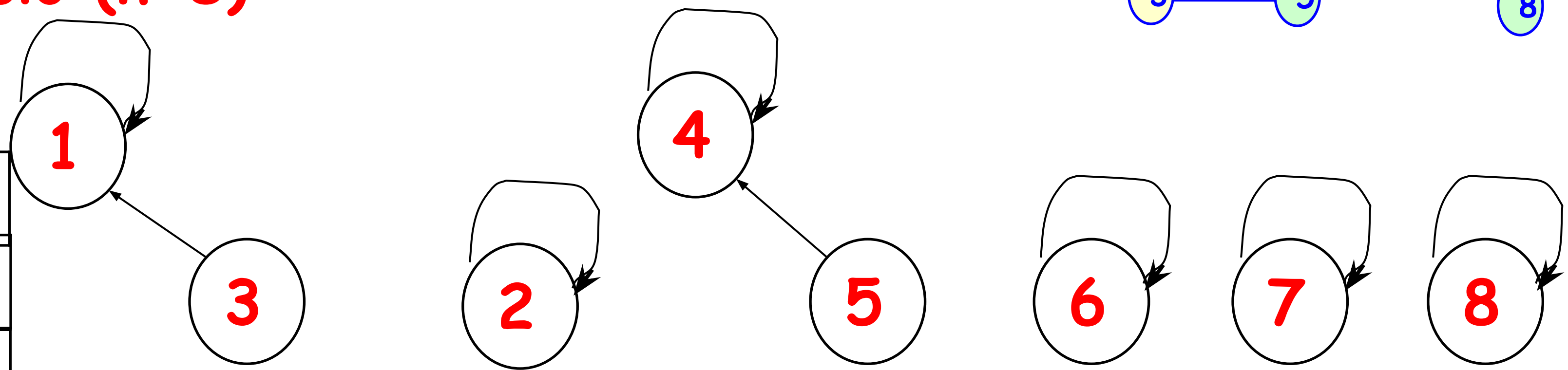


1	2	3	4	5	6	7	8
1	2	1	4	4	6	7	8

Grafos - Determinação de componentes

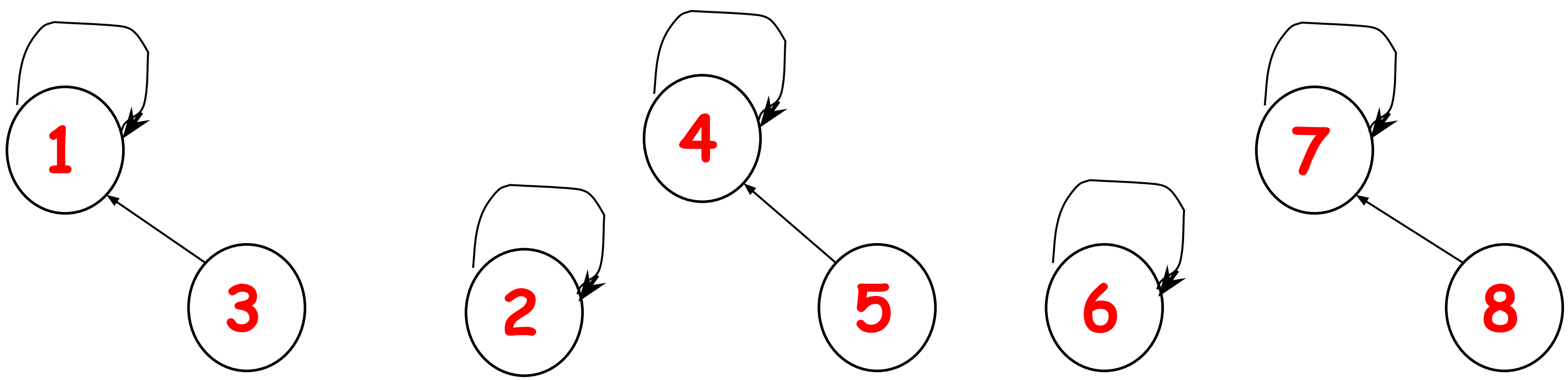


Exemplo (n=8):



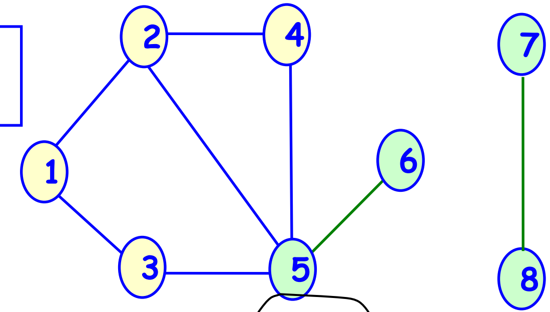
u	v
5	4
1	3
8	7

1	2	3	4	5	6	7	8
1	2	1	4	4	6	7	8



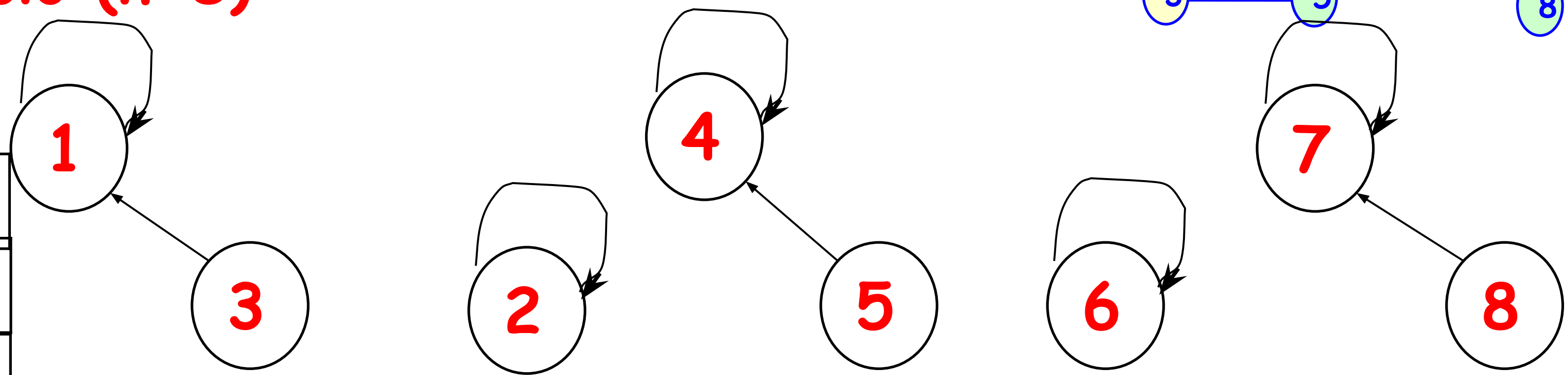
1	2	3	4	5	6	7	8
1	2	1	4	4	6	7	7

Grafos – Determinação de componentes

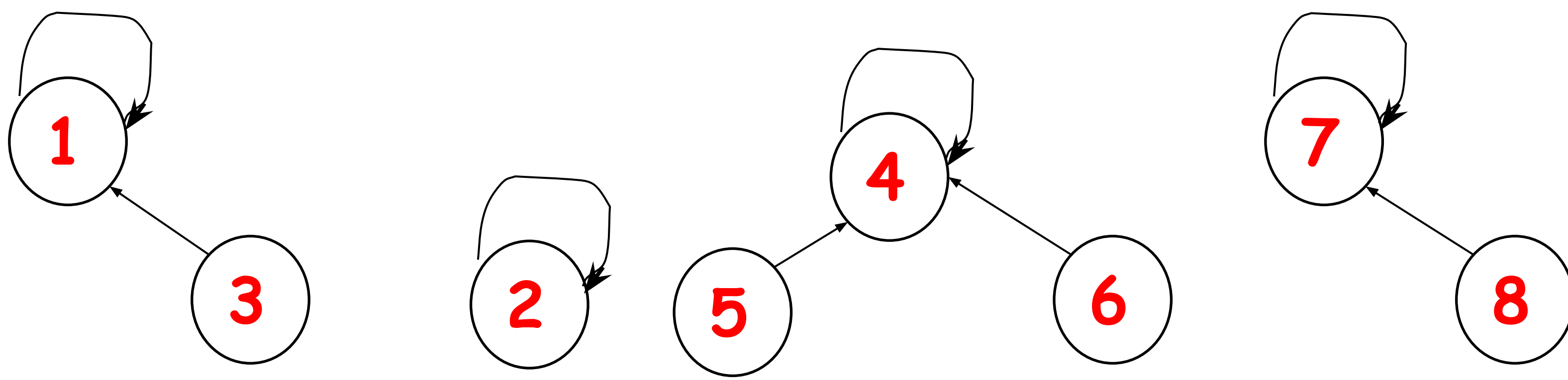


Exemplo (n=8):

u	v
5	4
1	3
8	7
5	6

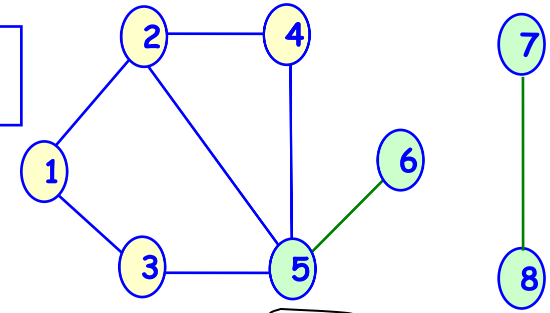


1	2	3	4	5	6	7	8
1	2	1	4	4	6	7	7



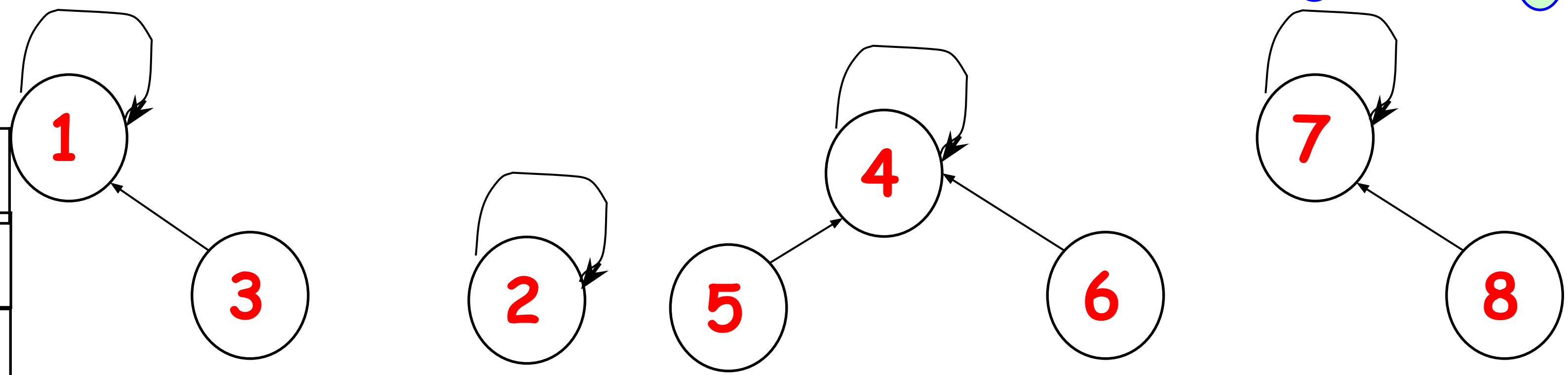
1	2	3	4	5	6	7	8
1	2	1	4	4	4	7	7

Grafos - Determinação de componentes

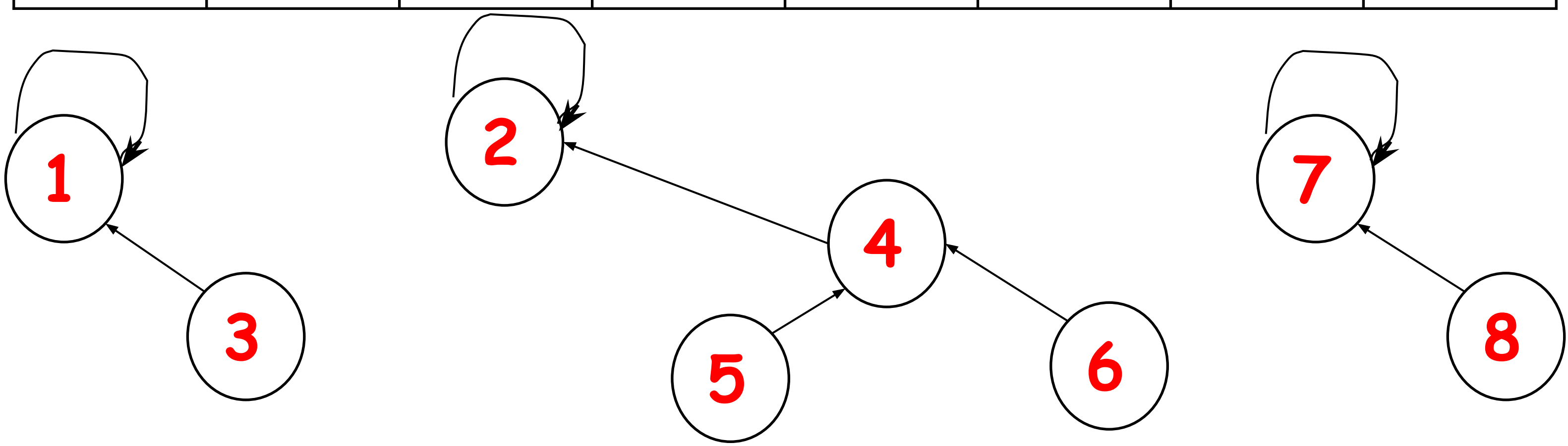


Exemplo (n=8):

u	v
5	4
1	3
8	7
5	6
2	4

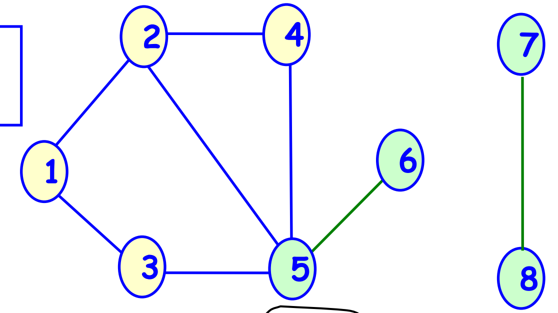


1	2	3	4	5	6	7	8
1	2	1	4	4	4	7	7



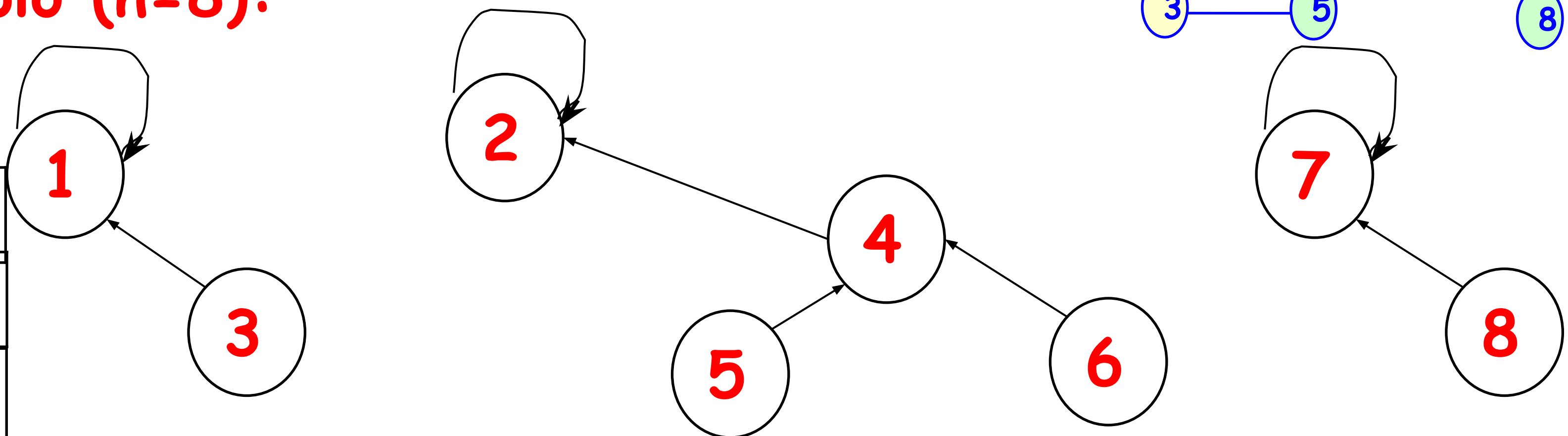
1	2	3	4	5	6	7	8
1	2	1	2	4	4	7	7

Grafos - Determinação de componentes

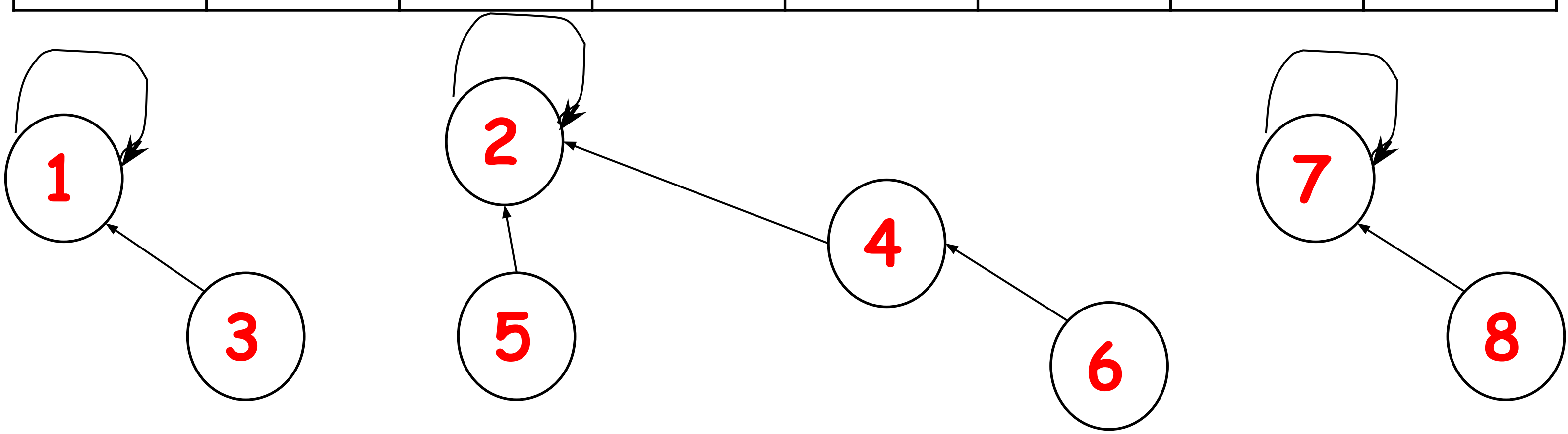


Exemplo (n=8):

u	v
5	4
1	3
8	7
5	6
2	4
2	5

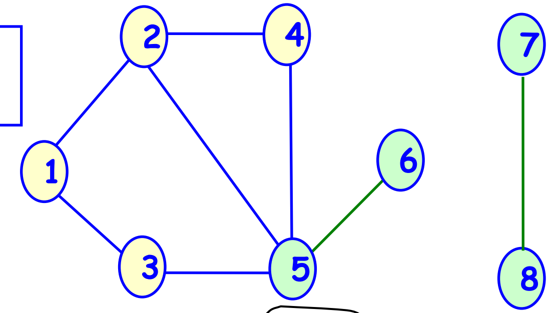


1	2	3	4	5	6	7	8
1	2	1	2	4	4	7	7



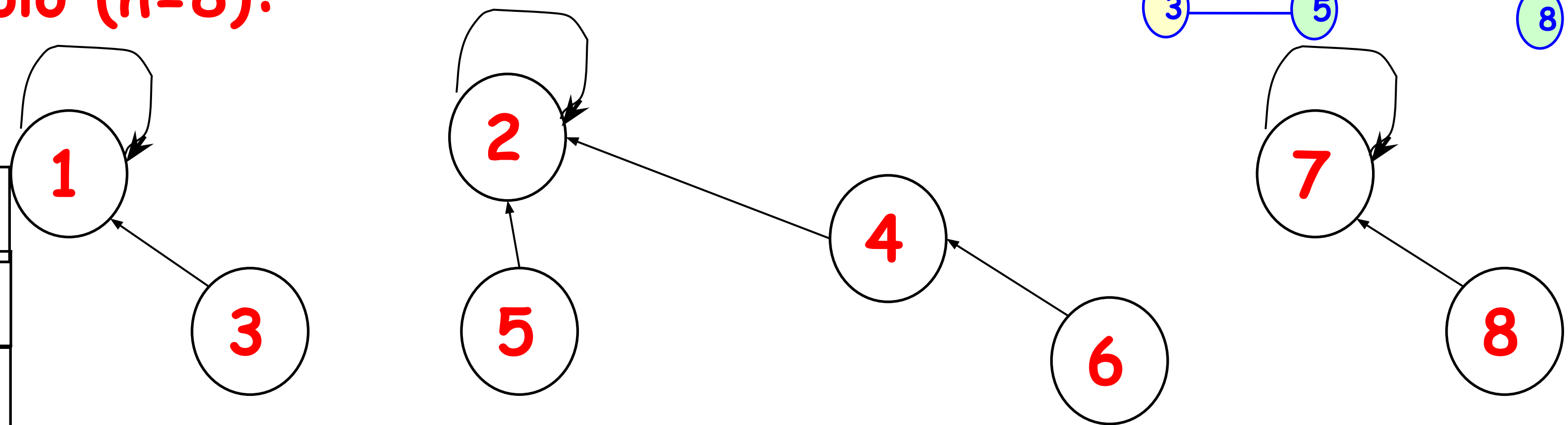
1	2	3	4	5	6	7	8
1	2	1	2	2	4	7	7

Grafos – Determinação de componentes

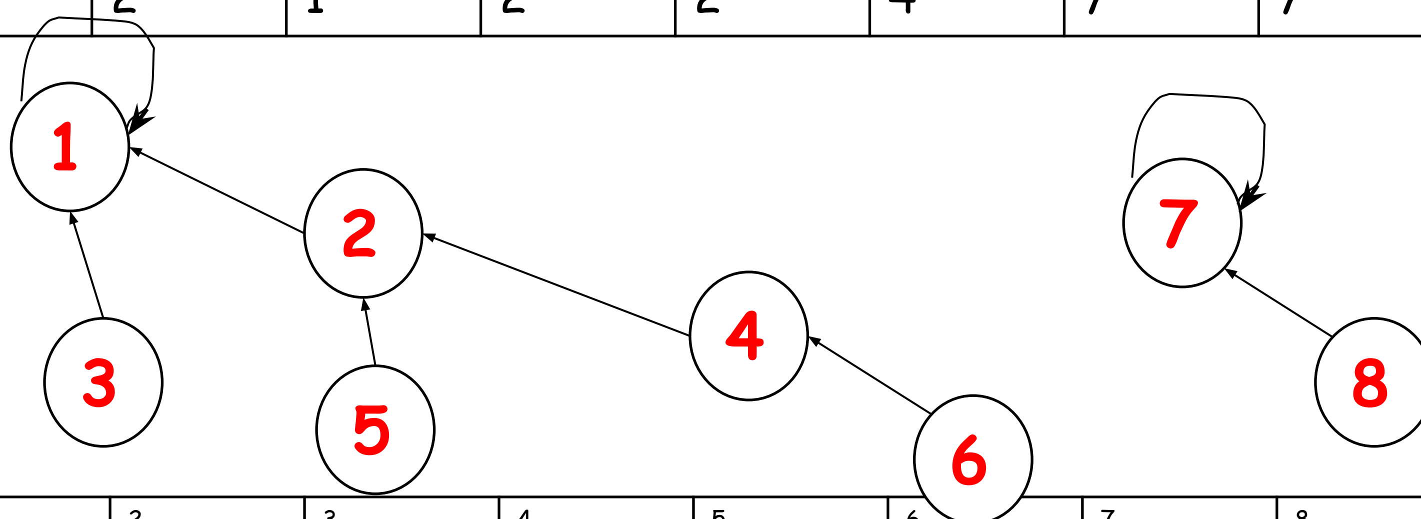


Exemplo (n=8):

u	v
5	4
1	3
8	7
5	6
2	4
2	5
5	3

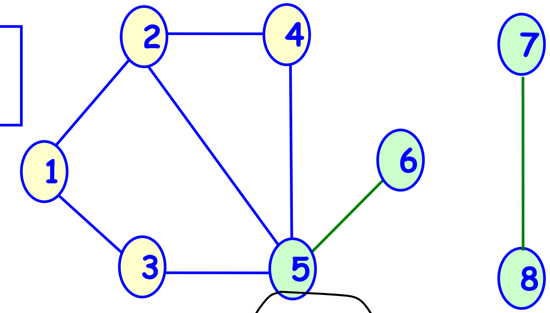


1	2	3	4	5	6	7	8
1	2	1	2	2	4	7	7



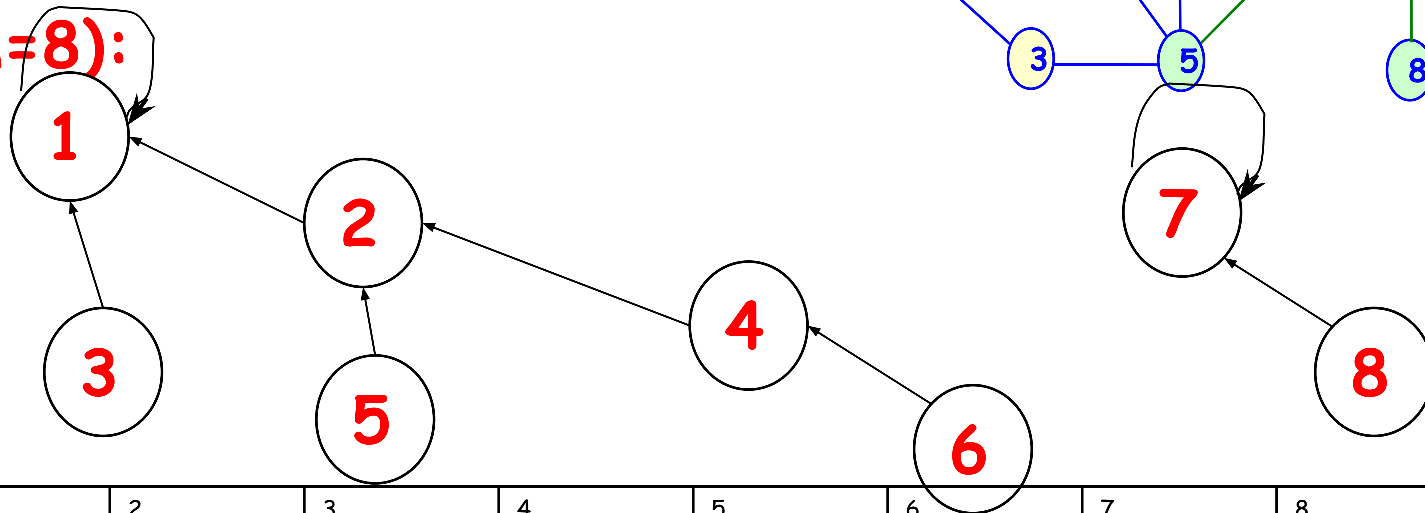
1	2	3	4	5	6	7	8
1	1	1	2	2	4	7	7

Grafos - Determinação de componentes

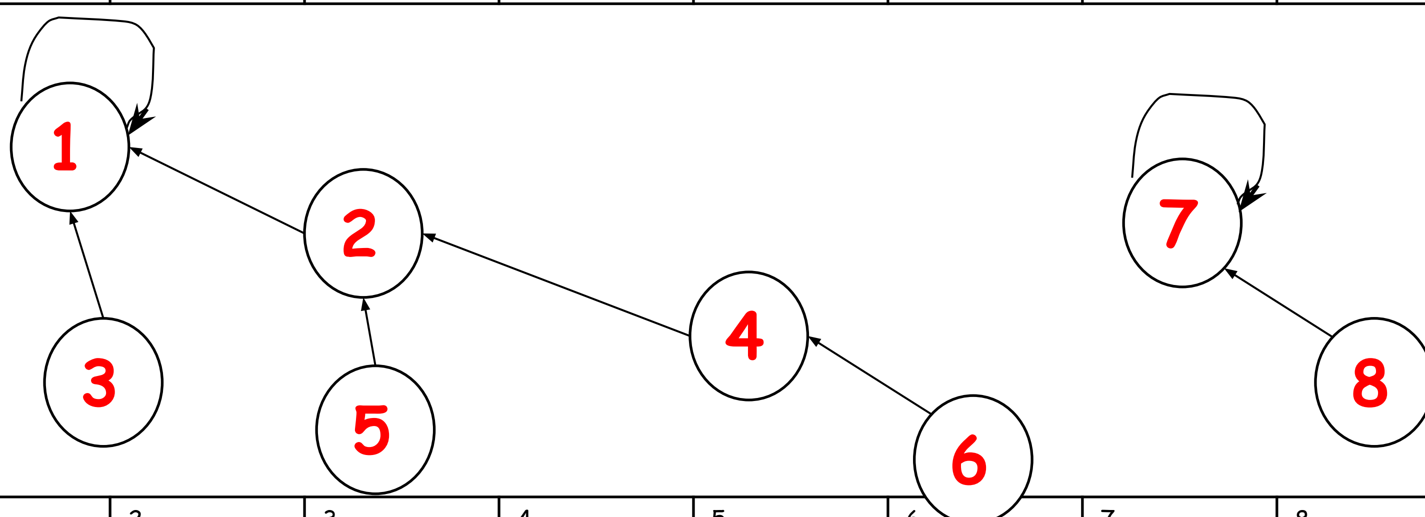


Exemplo (n=8):

u	v
5	4
1	3
8	7
5	6
2	4
2	5
5	3
1	2

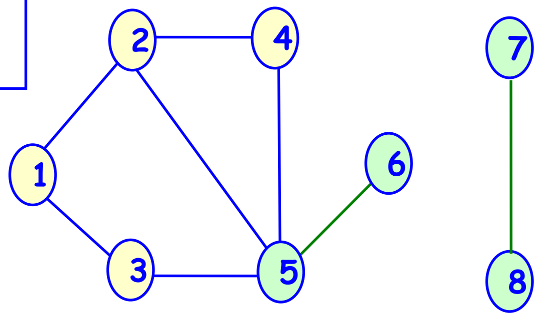


1	2	3	4	5	6	7	8
1	1	1	2	2	4	7	7



1	2	3	4	5	6	7	8
1	1	1	2	2	4	7	7

Grafos - Determinação de componentes



Exemplo (n=8):

u	v
5	4
1	3
8	7
5	6
2	4
2	5
5	3
1	2

Vetor Pai

1	2	3	4	5	6	7	8
1	2	3	4	4	6	7	8
1	2	1	4	4	6	7	8
1	2	1	4	4	6	7	7
1	2	1	4	4	4	7	7
1	2	1	2	4	4	7	7
1	2	1	2	2	4	7	7
1	1	1	2	2	4	7	7
1	1	1	2	2	4	7	7

1	1	1	1	1	1	7	7

Grafos – Determinação de componentes

Leitura do grafo e determinação de componentes;

Componentes():

```
    ler(n,m)
    Criação(n)
    para i ← 1 até m incl.:
        ler(u,v)
        Uniao(u,v)
    c ← 0
    para i ← 1 até n inclusive:
        se pai[i] = i:
            c ← c+1
            escrever ("Componente ",c,":")
            para j ← i até n incl.:
                se Busca(j)=i:
                    escrever (j)
```

Grafos – Determinação de componentes

Observações sobre os algoritmos UNION-FIND

1. É comum que as operações de fusão sejam feitas por tamanho de cada subconjunto. A implementação por valor da raiz é mais simples.

2. A complexidade das operações de n operações de fundir e m operações de buscar é $O(n+m\alpha(m+n,n))$, onde α é o inverso da função de Ackerman. Na prática $\alpha(n,m) \leq 4$.

FIM