Massive Graph Management & Analytics

Preliminaries

Nacéra Seghouani

Computer Science Department, CentraleSupélec
Laboratoire Interdisciplinaire des Sciences du Numérique, LISN
nacera.seghouani@centralesupelec.fr

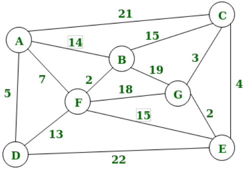
2024-2025

Graph Theory Preliminaries

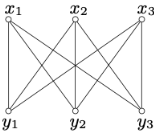
Graph Typology

☞ $\mathcal{G}(V, E)$, V set of vertices, $E = \{(v_i, v_j) | v_i, v_j \in V\}$ set of edges, $|V| = n, |E| = m$

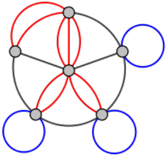
- ✓ **Undirected** - edge: symmetric pair of vertices - **Directed** - edge: asymmetric pair of vertices -
- ✓ **weighted** vertice $w_v : V \rightarrow \mathbb{R}$ or edge $w_e : E \rightarrow \mathbb{R}$
- ✓ **labeled** vertice $w_v : V \rightarrow \mathbb{L}$ or edge $w_e : E \rightarrow \mathbb{L}$
- ✓ **Bipartite** - $V = V_1 \cup V_2$, $E = \{(v_i, v_j) | v_i \in V_1, v_j \in V_2\}$ - Generalization to k**-partite**
- ✓ **Multigraph** or **Multidigraph** - $r : E \rightarrow v_i, v_j \in V$ where r assigns to each $e \in E$ a pair of vertices -
- ✓ **Hypergraph** - edge: relates a subset of vertices -
- ✓ **Complete graph**: $\forall (v_i, v_j) \in VxV, (v_i, v_j) \in E$



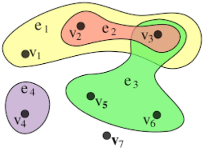
weighted graph



bipartite graph



multigraph



hypergraph

Graph Properties

SNA, pageram - read baout these things

- Let $\mathcal{G}(V, E)$ a directed graph, d_i^+, d_i^- denote resp. the number of edges coming out and coming to v_i. The degree of v_i:
$$d_i = d_i^+ + d_i^-$$
- $\mathcal{N}_i^+, \mathcal{N}_i^-$ denote resp. the set of the successors and predecessors of v_i. The set of the neighbors of v_i:
$$\mathcal{N}_i = \mathcal{N}_i^+ \cup \mathcal{N}_i^-$$
- A (directed) path $(v_i \rightsquigarrow v_j)$ is a sequence of vertices in the graph $(v_i, v_k, .., v_j)$ where each consecutive vertices pair $\in E$
- A (directed) cycle is $(v_i \rightsquigarrow v_j = v_i)$
- The length of a path $(v_i \rightsquigarrow v_j)$ is the number of the edges in $(v_i \rightsquigarrow v_j)$.
- A distance between (v_i, v_j) is the shortest path length between (v_i, v_j)

$$dist(v_i, v_j) = Min_{v_i \rightsquigarrow v_j} length(v_i \rightsquigarrow v_j)$$

Graph Properties

- The eccentricity *ecc* of v: the greatest distance between v and any other vertex;

$$ecc(v) = \max_{s \in V} dist(v, s)$$

- The diameter of $\mathcal{G}$ is

$$\max_{v,s \in V} dist(v, s)$$

  It is also the maximum eccentricity of any v in $\mathcal{G}$

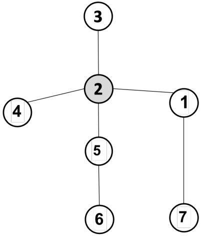
- The radius of $\mathcal{G}$ is the minimum eccentricity of any vertex

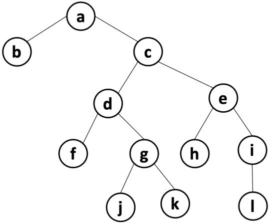
$$\min_{v \in V} ecc(v)$$

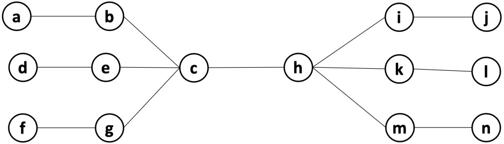
- The center of a graph is the set of all vertices of minimum eccentricity, equal to the graph's radius.

Graph Properties

Examples:







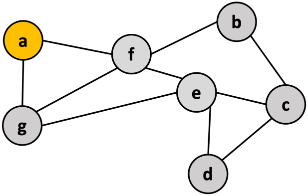
Graph Properties

- ☞ $\mathcal{G}\prime(V, E\prime \subset E)$ is a partial graph of $\mathcal{G}(V, E)$
- ☞ $\mathcal{G}\prime(V\prime \subset V, E\prime \subset E)$ is a subgraph of $\mathcal{G}(V, E)$
- ☞ $\mathcal{G}(V, E)$ is a connected graph $\Longleftrightarrow \forall(v_i, v_j) \in V \exists(v_i \rightsquigarrow v_j)$
- ☞ (strongly) connected component of $\mathcal{G}(V, E)$ is a subgraph - $\mathcal{G}_{cc}(V_{cc}, E_{cc})$ where $\exists(v_i \rightsquigarrow v_j)$, a (directed) path between each v_i and $v_j \in V_{cc}$,
- ☞ $\mathcal{G}(V, E)$ is a tree $\Leftrightarrow$ $\mathcal{G}$ is a connected graph without cycle $\Rightarrow$ graph with $m = n - 1$ edges
- ☞ $\mathcal{G}(V, E)$ is a forest $\Longleftrightarrow$ each connected component is a tree

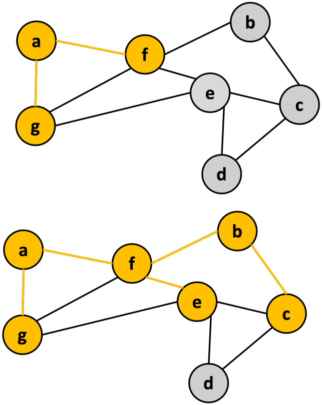
Breadth First Search (BFS)

- ☞ Queue data structure: an element first added in the list first removed out the list FIFO (First In First Out)

procedure BFS($\mathcal{G}(V, E), r$)
    $Q \leftarrow \emptyset$, *enqueue*(Q, r),
    $r.label$ = *true*
    while $Q \neq \emptyset$ **do**
        $v \leftarrow$ *dequeue*(Q)
        for $w \in \mathcal{N}_v$ **do**
            if $\neg w.label$ **then**
                enqueue(Q, w)
                $w.label$ = *true*
            end if
        end for
    end while
end procedure



Breadth First Search (BFS)

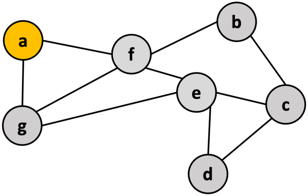


CentraleSupélec

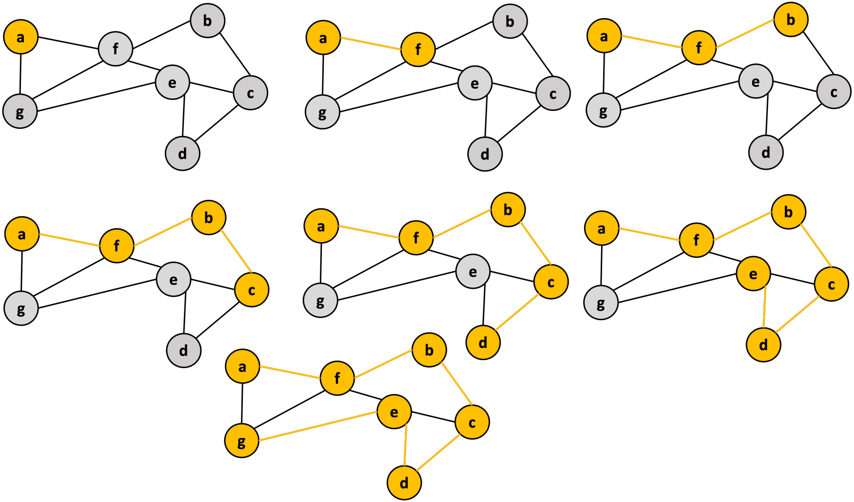
Depth First Search (DFS)

☞ Recursive DFS

```
procedure DFS*(𝒢(V, E), r)
    r.label = true
    for v ∈ 𝒩 do
        if ¬v.label then
            DFS*(𝒢(V, E), v)
        end if
    end for
end procedure
```



Breadth First Search (BFS)



Graph Representation using Matrices

☞ $\mathcal{G}(E, V)$ with n vertices and m edges can be encoded using:

✓ Adjacency Matrix $\mathbf{A}(n \times n)$, $n = |V|$

$$\mathbf{A}_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E, \\ 0 & \text{otherwise} \end{cases}$$

Symmetric matrix if $\mathcal{G}$ is an undirected (without loops)

✓ Adjacency list $\mathbf{L}$ each vertex holds a list of its neighbours

$$\forall v_i \in V,\ \mathbf{L}_i = \{v_j | (v_i, v_j) \in E\}$$

If $\mathcal{G}$ is directed the choice of the direction depends on analytic needs

✓ Incidence matrix $\mathbf{B}$, $n \times m$

$$\mathbf{B}_{ij} = \begin{cases} 1 & \text{if } e_j = (v_i, v_k) \in E, \\ 0 & \text{otherwise} \end{cases}$$

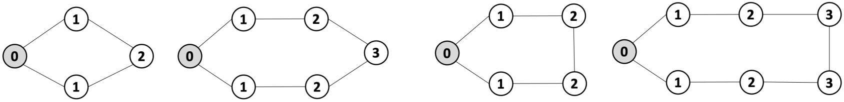
Graph Theory Preliminaries

Some Exercises

Exercise: Breadth-First Search and Bipartite graphs

1) Using graph traversal algorithms, propose an algorithm that computes the number of edges between a given vertex and all other vertices.

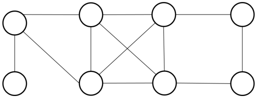
2) Given the following cycles with even and odd length (with the distances or depths from the grey vertex), what do you think about the case of graphs with an **odd** cycle (in number of edges)? Is this a characteristic property? State the general case.



3) Propose an algorithm that determines if a graph contains an odd cycle.

4) In a bipartite graph, can there be a cycle with an odd number of edges? Is this a characteristic property? Justify your answer.

5) Propose an algorithm that allows to determine if a graph is bipartite. Test your algorithm on the following graph. Is it bipartite? Justify your answer



Exercise: Depth-First Search and 2-colorable graphs

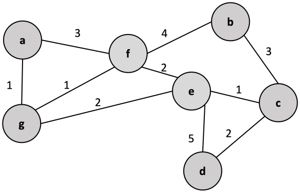
Graph coloring is a way of coloring the vertices of a graph such that no two adjacent vertices share the same color. A 2-colorable graph is a graph that can be colored with only 2 colors.

1) What is the link with the previous exercise? Justify your answer.
2) We want to write an algorithm, inspired by DFS search which takes as input a graph $\mathcal{G}(V, E)$ and which returns a pair `(result, color)` where *result* is *true* if the graph is colorable, *false* otherwise and *color* is a dictionary associating a color 0 or 1 to each vertex. This algorithm should *stop as soon as possible* when the graph is not 2-colorable. Propose an **iterative** version or a **recursive** version.

Exercise: Shortest path

Compute the shortest path using Dijkstra algorithm

procedure $\textsc{Dijkstra}(\mathcal{G}(V, E, W), s)$
    $dist \leftarrow \{s : 0\}$
    $P \leftarrow \emptyset$
    for $v \in V \wedge v \neq s$ **do**
        $dist[v] \leftarrow +\infty$
    end for
    $w \leftarrow select(v \in V - P \wedge dist[v] = min_v\, dist[v]$
    $P \leftarrow P \cup \{w\}$
    for $v \in \mathcal{N}_w \wedge v \notin P$ **do**
        if $dist[w] + w_{(v,w)} < dist[v]$ **then**
            $predecessor(v) \leftarrow w$
            $dist[v] \leftarrow w_{(v,w)} + dist[w]$
        end if
    end for
end procedure



Exercise: Matrix Multiplication & Power

1) Give the different representations of these graphs.
2) Compute $\mathbf{A}^2$, $\mathbf{A}^3$. What $\mathbf{A}^r_{ij}$ represents?
3) What is the complexity of $\mathbf{A}^r$, Is it possible to reduce it?