Data Structure and Algorithms [CO2003]

Graph

Lecturer: Nguyen Huynh-Tuong, PhD. & Duc-Dung Nguyen, PhD.
Contact: {htnguyen;nddung}@hcmut.edu.vn

Faculty of Computer Science and Engineering
Hochiminh city University of Technology

Contents

Outcomes

- **L.O.7.1** – Depict the following concepts: connected graph, disconnected graph, direct/undirect graph, etc.
- **L.O.7.2** – Depict storage structures for graph and describe graph using pseudocode in the cases of using adjacency matric and adjacency list.
- **L.O.7.3** – List necessary methods supplied for graph, and describe them using pseudocode.
- **L.O.7.4** – Depict basic traversal methods step-by-step (depth first and bread-first).
- **L.O.7.5** – Implement storage structures for graphs using C/C++.
- **L.O.7.6** – Implement basic traversal methods using C/C++.
- **L.O.7.7** – Depict the working steps of Dijkstra and Prim step-by-step.

Graph definition

Definition
A graph G is defined by

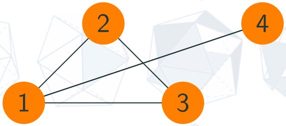
- a set of vertices $V(G)$
- a set of edges (or arcs) $E(G) \subseteq [V(G)]^2$

Graph captures/represents the abtract relations between objets (vertices).

Definition
A graph G is defined by

- a set of vertices $V(G)$
- a set of edges (or arcs) $E(G) \subseteq [V(G)]^2$

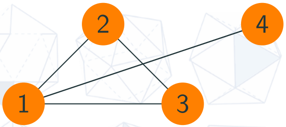
Graph captures/represents the abtract relations between objets (vertices).

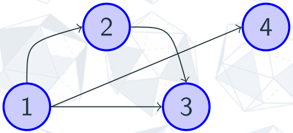


Definition
A graph G is defined by

- a set of vertices $V(G)$
- a set of edges (or arcs) $E(G) \subseteq [V(G)]^2$

Graph captures/represents the abtract relations between objets (vertices).





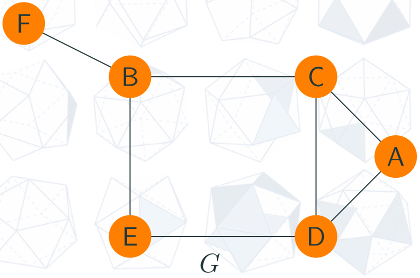
Adjacency matrices

Give the graph defined by the following adjacency matrices.

$$
\begin{array}{c c}
 & \begin{array}{ccccc} A & B & C & D & E \end{array} \\
\begin{array}{c} A \\ B \\ C \\ D \\ E \end{array} &
\begin{bmatrix}
0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 0
\end{bmatrix}
\end{array}
\qquad
\begin{array}{c c}
 & \begin{array}{ccccc} A & B & C & D & E \end{array} \\
\begin{array}{c} A \\ B \\ C \\ D \\ E \end{array} &
\begin{bmatrix}
1 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 & 0
\end{bmatrix}
\end{array}
$$

Incidence matrice

Define incidence matrice of the following graph.



Graph $G = (V, E)$ with adjacency lists Adj

- class Vertex
-     Integer id, name, color, ..
- end
- class AdjList
-     vertex [] l, ..
-     Integer nVertice (number of adjacent vertice)
- end
- class Graph
-     VerticeList [] g;
-
- end

- class Vertex
-     Integer id, name, color, ..
- end
- class AdjList
-     vertex [] l, ..
-     Integer nVertice (number of adjacent vertice)
- end
- class Graph
-     VerticeList [] g;
-
- end

Find all adjacency neighbors of a vertex u

1. Input: Graph g,
2. for

Depth-First Search

DFS(G)

1. for each vertice u of V do
2.   color[u] $\leftarrow$ **W** ("White")
3.   p[u] $\leftarrow$ null
4. time $\leftarrow$ 0
5. for each vertex u of V do
6.   if color[u] $=$ **W** then
7.   DFSVisit(Adj, u)

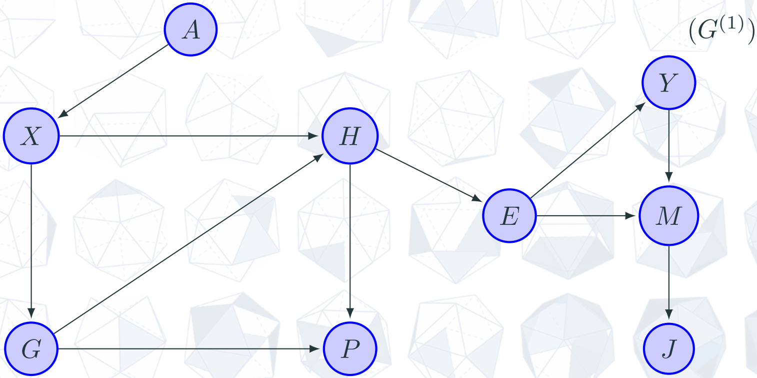
DFS(G)

1. for each vertice u of V do
2. color[u] $\leftarrow$ **W** ("White")
3. p[u] $\leftarrow$ null
4. time $\leftarrow 0$
5. for each vertex u of V do
6. if color[u] $=$ **W** then
7. DFSVisit(Adj, u)

DFSVisit(Adj, u)

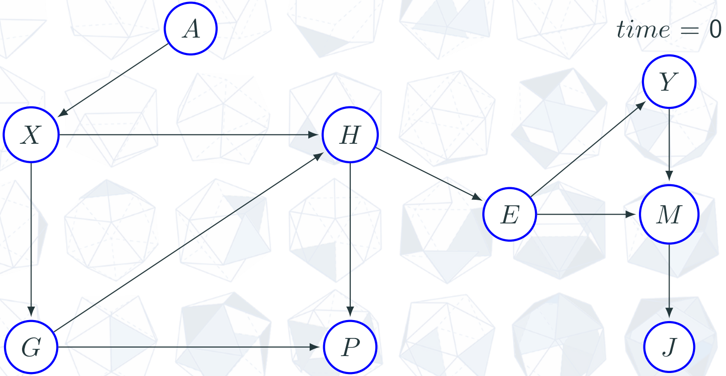
1. color[u] $\leftarrow$ **G** ("Gray")
2. d[u] $\leftarrow$ time $\leftarrow$ time $+ 1$
3. for each v of Adj[u] do
4. if color[v] $=$ **W** then
5. p[v] $\leftarrow u$
6. DFSVisit(Adj, v)
7. color[u] $\leftarrow$ **B** ("Black")
8. f[u] $\leftarrow$ time $\leftarrow$ time $+ 1$

Example



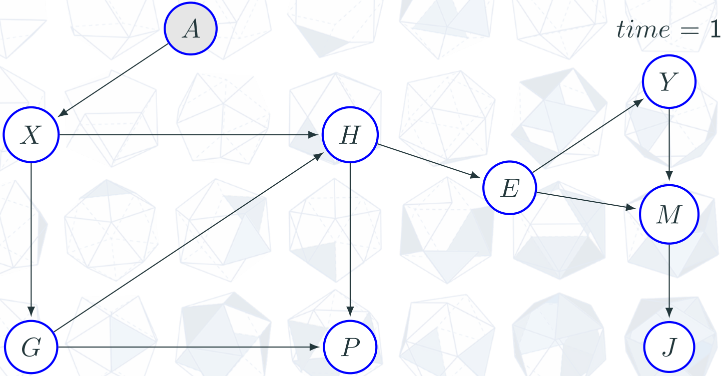
Example

Node	p[v]	d[v]	f[v]
A	null		
X	null		
H	null		
Y	null		
E	null		
G	null		
P	null		
M	null		
J	null		

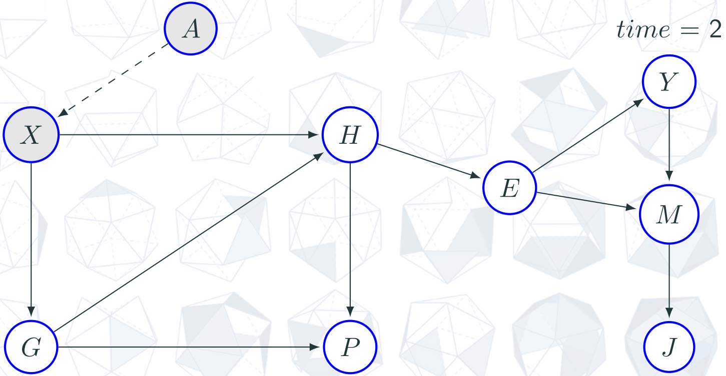


Example

Node	p[v]	d[v]	f[v]
A	null	1	
X	null		
H	null		
Y	null		
E	null		
G	null		
P	null		
M	null		
J	null		

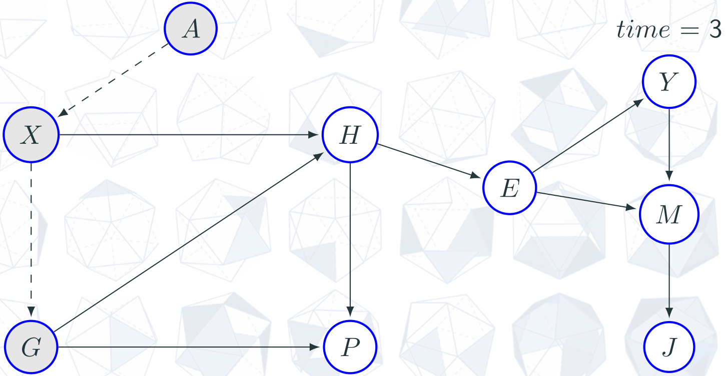


Node	p[v]	d[v]	f[v]
A	null	1	
X	A	2	
H	null		
Y	null		
E	null		
G	null		
P	null		
M	null		
J	null		



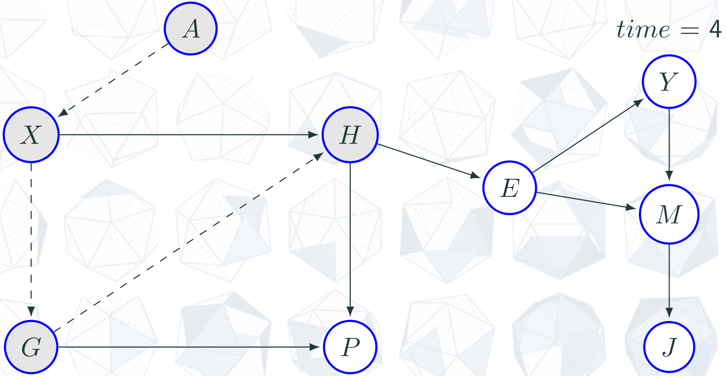
Example

Node	p[v]	d[v]	f[v]
A	null	1	
X	A	2	
H	null		
Y	null		
E	null		
G	X	3	
P	null		
M	null		
J	null		



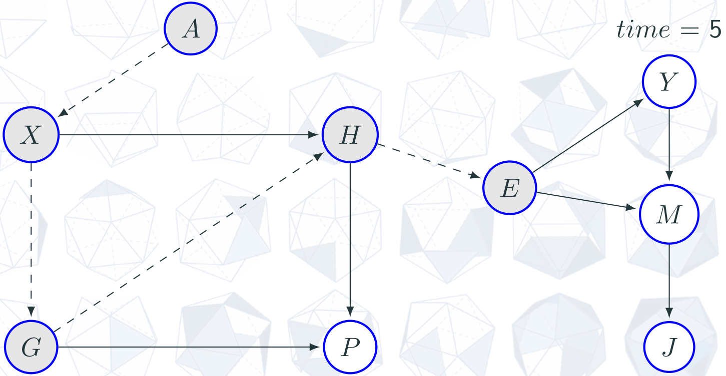
Example

Node	p[v]	d[v]	f[v]
A	null	1	
X	A	2	
H	G	4	
Y	null		
E	null		
G	X	3	
P	null		
M	null		
J	null		



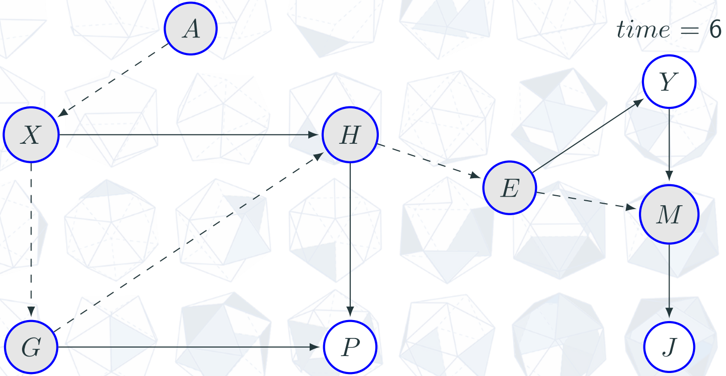
Example

Node	p[v]	d[v]	f[v]
A	null	1	
X	A	2	
H	G	4	
Y	null		
E	H	5	
G	X	3	
P	null		
M	null		
J	null		



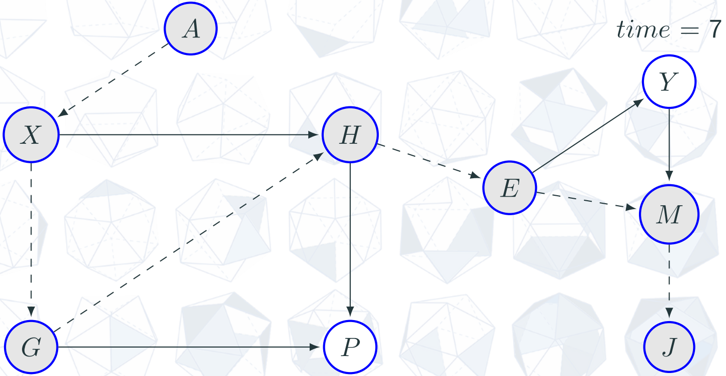
Example

Node	p[v]	d[v]	f[v]
A	null	1	
X	A	2	
H	G	4	
Y	null		
E	H	5	
G	X	3	
P	null		
M	E	6	
J	null		



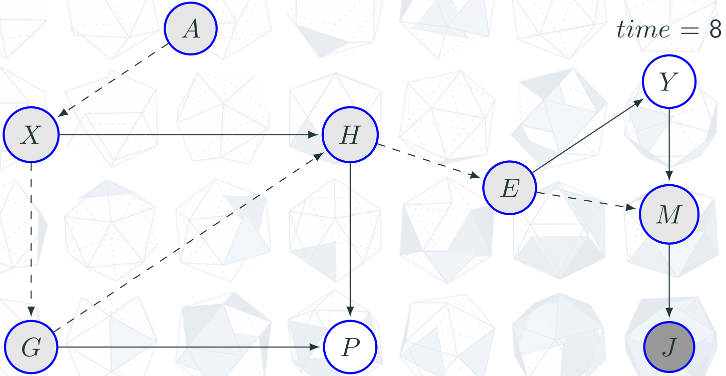
Example

Node	p[v]	d[v]	f[v]
A	null	1	
X	A	2	
H	G	4	
Y	null		
E	H	5	
G	X	3	
P	null		
M	E	6	
J	M	7	

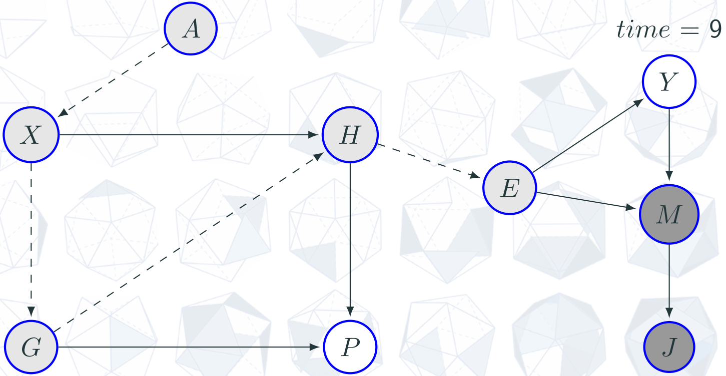


Example

Node	p[v]	d[v]	f[v]
A	null	1	
X	A	2	
H	G	4	
Y	null		
E	H	5	
G	X	3	
P	null		
M	E	6	
J	M	7	8

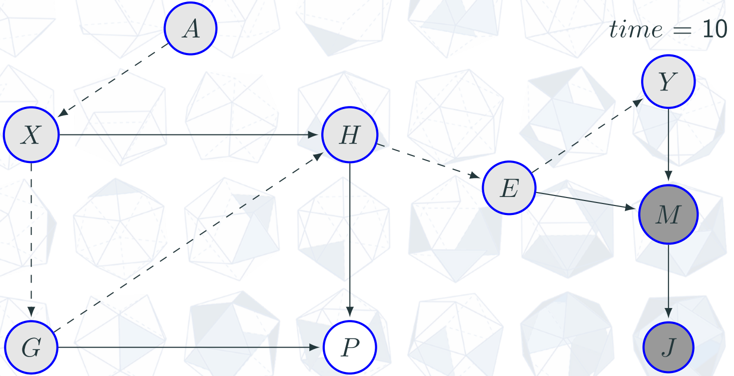


Node	p[v]	d[v]	f[v]
A	null	1	
X	A	2	
H	G	4	
Y	null		
E	H	5	
G	X	3	
P	null		
M	E	6	9
J	M	7	8

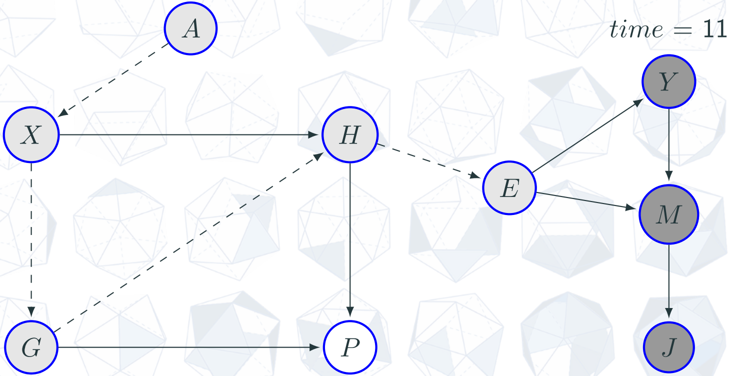


Example

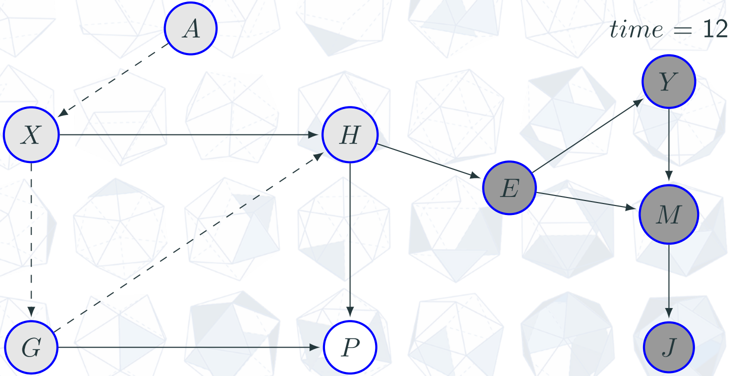
Node	p[v]	d[v]	f[v]
A	null	1	
X	A	2	
H	G	4	
Y	E	10	
E	H	5	
G	X	3	
P	null		
M	E	6	9
J	M	7	8



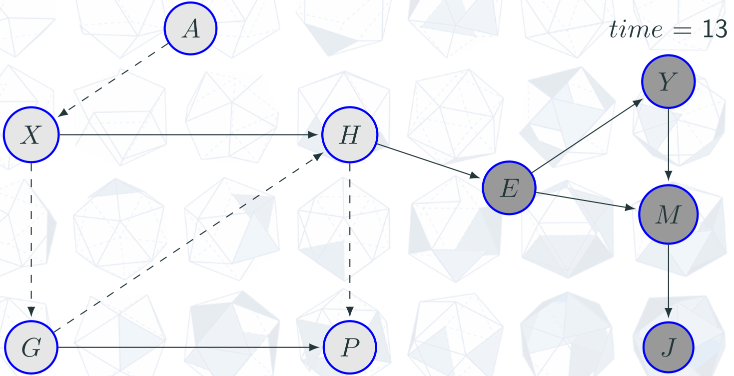
Node	p[v]	d[v]	f[v]
A	null	1	
X	A	2	
H	G	4	
Y	E	10	11
E	H	5	
G	X	3	
P	null		
M	E	6	9
J	M	7	8



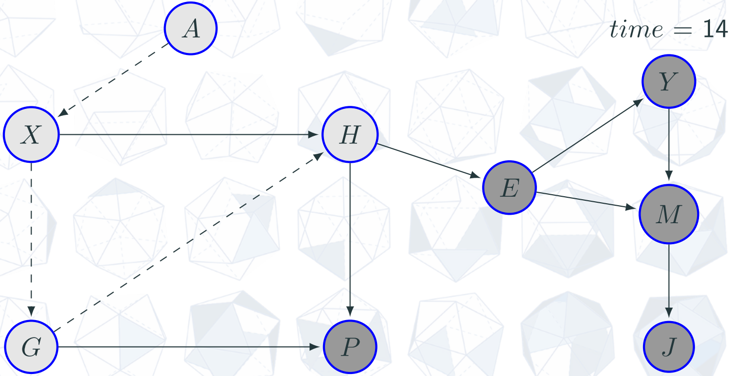
Node	p[v]	d[v]	f[v]
A	null	1	
X	A	2	
H	G	4	
Y	E	10	11
E	H	5	12
G	X	3	
P	null		
M	E	6	9
J	M	7	8



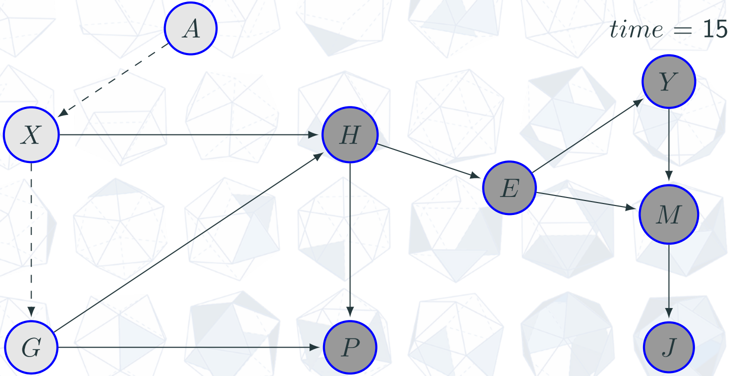
Node	p[v]	d[v]	f[v]
A	null	1	
X	A	2	
H	G	4	
Y	E	10	11
E	H	5	12
G	X	3	
P	H	13	
M	E	6	9
J	M	7	8



Node	p[v]	d[v]	f[v]
A	null	1	
X	A	2	
H	G	4	
Y	E	10	11
E	H	5	12
G	X	3	
P	H	13	14
M	E	6	9
J	M	7	8

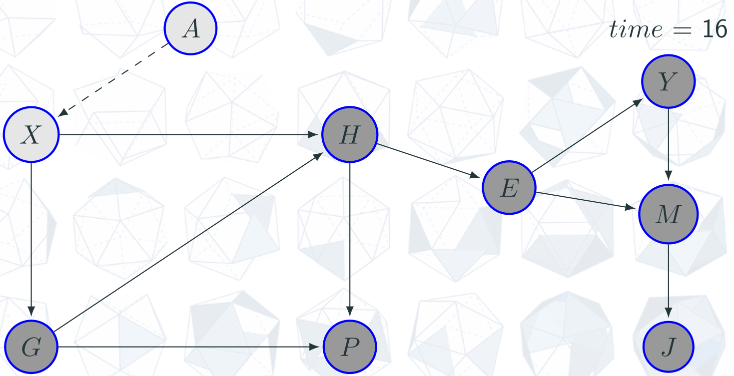


Node	p[v]	d[v]	f[v]
A	null	1	
X	A	2	
H	G	4	15
Y	E	10	11
E	H	5	12
G	X	3	
P	H	13	14
M	E	6	9
J	M	7	8



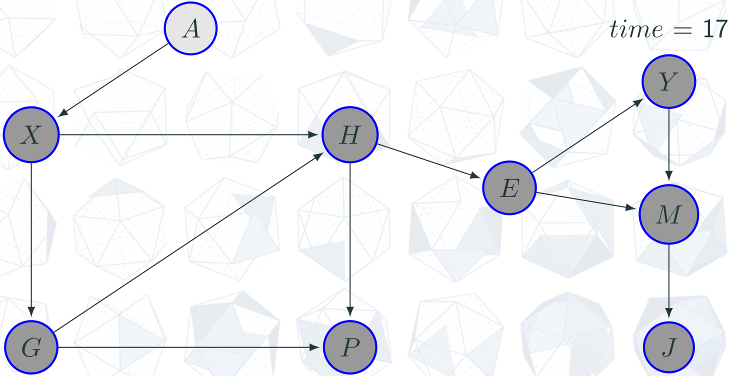
Example

Node	p[v]	d[v]	f[v]
A	null	1	
X	A	2	
H	G	4	15
Y	E	10	11
E	H	5	12
G	X	3	16
P	H	13	14
M	E	6	9
J	M	7	8



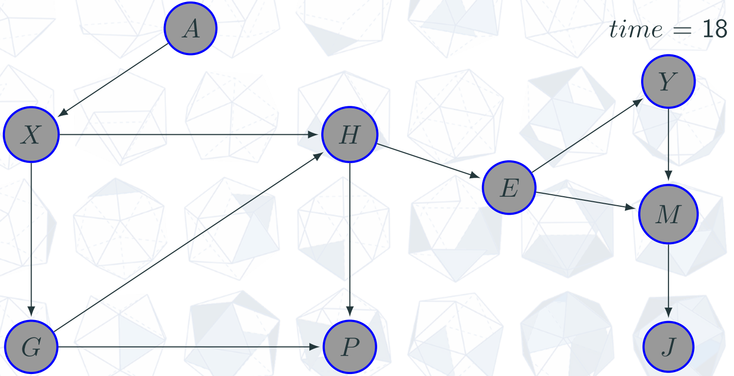
Example

Node	p[v]	d[v]	f[v]
A	null	1	
X	A	2	17
H	G	4	15
Y	E	10	11
E	H	5	12
G	X	3	16
P	H	13	14
M	E	6	9
J	M	7	8



Example

Node	p[v]	d[v]	f[v]
A	null	1	18
X	A	2	17
H	G	4	15
Y	E	10	11
E	H	5	12
G	X	3	16
P	H	13	14
M	E	6	9
J	M	7	8



DFS(G)

for each vertice u of V do
    color[u] $\leftarrow$ **W** ("<u>W</u>hite")
    p[u] $\leftarrow$ null
time $\leftarrow$ 0
for each vertex u of V do
    if color[u] $=$ **W** then
        DFSVisit(Adj, u)

Graph $G = (V, E)$ with adjacency lists/matrix Adj

DFS(G)

1. for each vertice u of V do
2.   color[u] $\leftarrow$ **W** ("White")
3.   p[u] $\leftarrow$ null
4. time $\leftarrow$ 0
5. for each vertex u of V do
6.   if color[u] $=$ **W** then
7.   DFSVisit(Adj, u)

DFSVisit(Adj, u)

1. color[u] $\leftarrow$ **G** ("Gray")
2. d[u] $\leftarrow$ time $\leftarrow$ time $+ 1$
3. for each v of Adj[u] do
4.   if color[v] $=$ **W** then
5.   p[v] $\leftarrow u$
6.   DFSVisit(Adj, v)
7. color[u] $\leftarrow$ **B** ("Black")
8. f[u] $\leftarrow$ time $\leftarrow$ time $+ 1$

Exerxise

1. Implement DFS algorithm in C/C++ by using adjacency matrice/lists and incident matrice.

Exerxise

1. Implement DFS algorithm in C/C++ by using adjacency matrice/lists and incident matrice.
2. Calculate complexity of each implementation and determine the best implementation.

1. Find vertice set accessible from a vertex s

Application

1. Find vertice set accessible from a vertex s
2. Determine topological order in an acyclic graph

Application

1. Find vertice set accessible from a vertex s
2. Determine topological order in an acyclic graph
3. Calculate number of connected components in a directed graph

Application

1. Find vertice set accessible from a vertex s
2. Determine topological order in an acyclic graph
3. Calculate number of connected components in a directed graph
4. Determine whether there exists a cycle in graph

1. Find vertice set accessible from a vertex s
2. Determine topological order in an acyclic graph
3. Calculate number of connected components in a directed graph
4. Determine whether there exists a cycle in graph
5. Verify whether the graph is bipartite

1. Find vertice set accessible from a vertex s
2. Determine topological order in an acyclic graph
3. Calculate number of connected components in a directed graph
4. Determine whether there exists a cycle in graph
5. Verify whether the graph is bipartite
6. Identify articulation points

DFS(G)

```
for each vertex u of V do
    color[u] ← W ("White")
    parent[u] ← null
time ← 0
for each vertex u of V do
    if color[u] = W then
        DFSVisit(Adj, u)
```

DFS(G)

1. for each vertex u of V do
2. color[u] $\leftarrow$ **W** ("White")
3. parent[u] $\leftarrow$ null
4. time $\leftarrow$ 0
5. for each vertex u of V do
6. if color[u] $=$ **W** then
7. DFSVisit(Adj, u)

DFSVisit(Adj, u)

1. color[u] $\leftarrow$ **G** ("Gray")
2. d[u] $\leftarrow$ temps $\leftarrow$ temps $+ 1$
3. for each vertex v of Adj[u] do
4. if couleur[v] $=$ **W** then
5. p[v] $\leftarrow u$
6. DFSVisit(Adj, v)
7. if (color[v] $=$ **G**) & (p[u] $\neq v$) then
8. There exists a cycle here!
9. color[u] $\leftarrow$ **B** ("Black")
10. f[u] $\leftarrow$ time $\leftarrow$ time $+ 1$

Breath-First Search

BFS(G)

```
create a queue Q
for each vertex u of V do
    color[u] ← W ("White")
    p[u] ← null
time ← 0
for each vertex u of V do
    if color[u] = W then
        Q.EnQueue(u)
        color[u] ← G ("Gray")
        d[u] ← time ← time + 1
        BFSVisit(Adj, Q)
```

BFS(G)

```
create a queue Q
for each vertex u of V do
    color[u] ← W ("White")
    p[u] ← null
time ← 0
for each vertex u of V do
    if color[u] = W then
        Q.EnQueue(u)
        color[u] ← G ("Gray")
        d[u] ← time ← time + 1
        BFSVisit(Adj, Q)
```

BFSVisit(Adj, Q)

```
while (Q is not null) do
    u ← Q.front()
    for each v of Adj[u] do
        if color[v] = W then
            color[v] ← G;
            p[v] ← u
            d[v] ← time ← time + 1
            Q.EnQueue(v)
    Q.DeQueue()
    color[u] ← B ("Black")
    f[u] ← time ← time + 1
```

BFS(G)

```
create a queue Q
for each vertex u of V do
    color[u] ← W ("White")
    p[u] ← null
time ← 0
for each vertex u of V do
    if color[u] = W then
        Q.EnQueue(u)
        color[u] ← G ("Gray")
        X[u] ← 0
        d[u] ← time ← time + 1
        if (BFSVisit(Adj, Q) = 'N' then
            return 'N'
```

BFSVisit(Adj, Q)

```
while (Q is not null) do
    u ← Q.front()
    for each v of Adj[u] do
        if color[v] = W then
            color[v] ← G;
            p[v] ← u
            if (X[u]=0) then X[v]=1 else X[v]=0
            d[v] ← time ← time + 1
            Q.DeQueue()
        else if (X[u]=X[v]) return 'N'
            Q.EnQueue(v)
    color[u] ← B ("Black")
    f[u] ← time ← time + 1
```