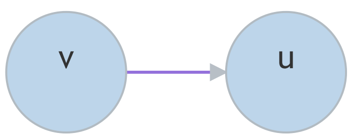
Graph theory

Glossary

- **Graph G (V, E)**:
  - **V**: vertices set
  - **E**: edges set
- **Degree**: number of vertices
- **Walk**: series of connected vertices
- **Path**: walk without repeated vertices
- **Closed walk**: walk where $v_0 = v_n$
- **Cycle**: closed walk without repeating vertices
- **Euler path**: visit each edge exactly once
- **Hamilton path**: visit each vertex exactly once
- **Directed graph**: edges are ordered pairs
- **Ancestor**: v, **Successor**: u in



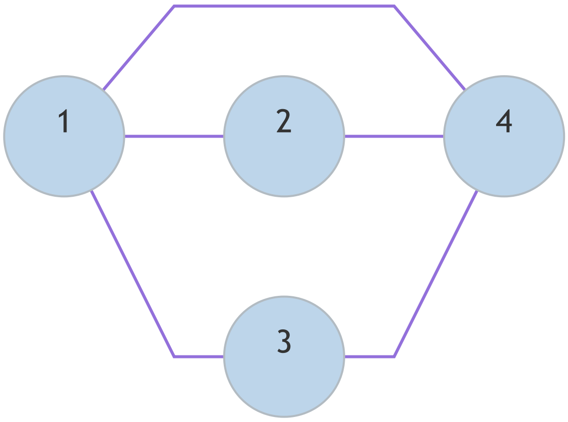
- **$deg_{in}(v)$**: number of incoming edges into v
- **$deg_{out}(v)$**: number of outgoing edges into v

Graph Representation

Adjacency matrix:

matrix where $A_{uv} = \begin{cases} 1 & \text{if}(u,v) \in E \\ 0 & \text{otherwise} \end{cases}$

Graph:



Matrix:

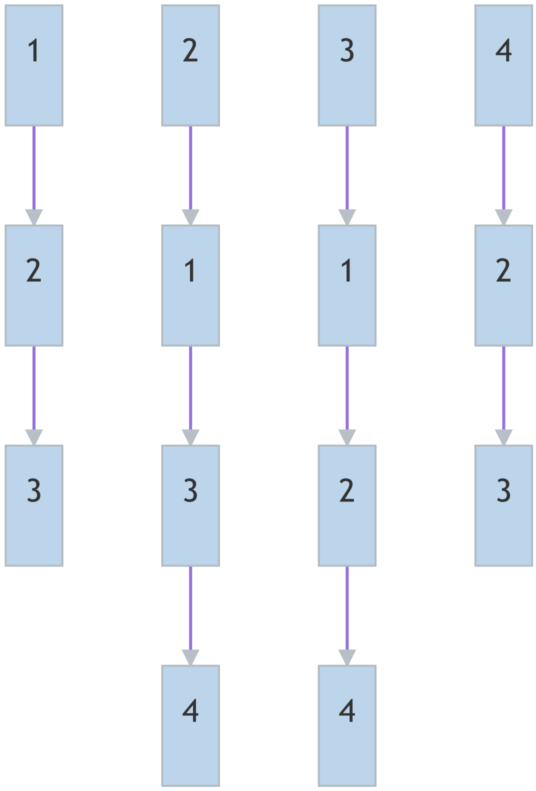
$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Adjacency list

Array of linked lists, where Adj[u] contains a list containing all the neighbors of u.

Graph: Same as above

List:



Runtimes

	Matrix	List
Find all neighbors of v	$\mathcal{O}(n)$	$\mathcal{O}(deg_{out}(v))$
Find $v \in V$ without neighbors	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$
Check if $(v, u) \in E$	$\mathcal{O}(1)$	$\mathcal{O}(1 + min(deg_{out}(v), deg_{out}(u)))$
Insert edge	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Remove edge v	$\mathcal{O}(1)$	$\mathcal{O}(deg_{out}(v))$
Check whether an Eulerian path exists or not	$\mathcal{O}(\|V\| * \|E\|)$	$\mathcal{O}(\|V\| + \|E\|)$

Algorithms

Depth-First Search (DFS)

Used mainly to check whether a Graph can be topological sorted or not ($\Leftrightarrow$ has a cycle). A **topological sorting** of a graph it's a sequence of all its nodes with the property that a node u comes after a node v **if and only if** either a walk from v to u exists or u cannot be reached starting from v.

Pseudocode

```
DFS(G):
    for (v in V not marked):
        DFS-Visit(v)
```

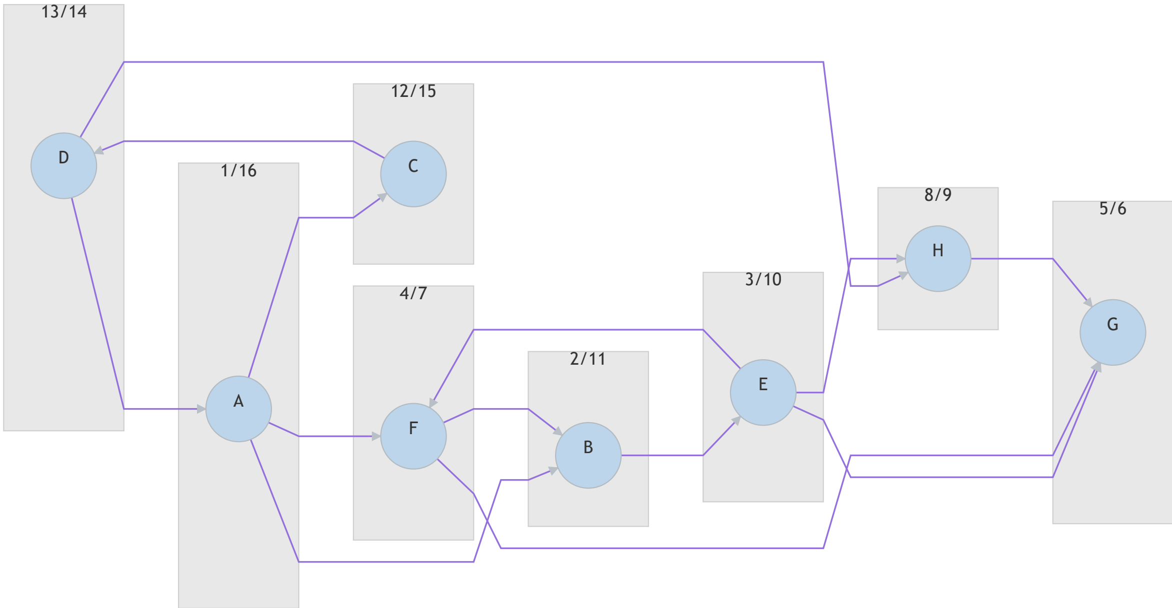
```
DFS-Visit(v):
    t = 0
    pre[v] = t++
    marked[v] = true
    for ((u, v) in E not marked)
        DFS_Visit(u)
    post[u] = t++
```

Runtime

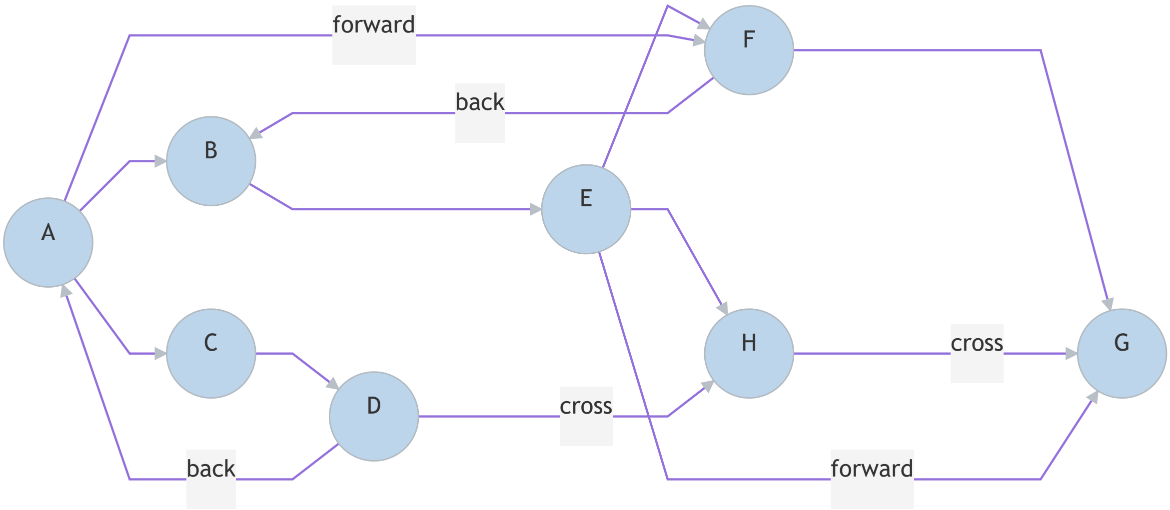
Operations	$T(n) \in \Theta(\|E\| + \|V\|)$
Memory	$T(n) \in \Theta(\|V\|)$

Edge classification (post and pre numbers)

Example: `DFS(A)` got called



This graph generate the following tree (rotated of 90 degree to save space):



Pre and post number	Name of the edge $(v, u) \in E$
$pre(u) < pre(v)$ and $post(u) < post(v)$	Not possible
$pre(u) < pre(v)$ and $post(u) > post(v)$	**forward** or simply no name, since it is part of the tree
$pre(u) < pre(w)$ and $post(u) < post(v)$ but $(u, v) \notin E$	**forward edge**
$pre(u) > pre(v)$ and $post(u) > post(v)$	**back edge**
$pre(u) > pre(v)$ and $post(u) > post(v)$	**cross edge**
$pre(u) < pre(v)$ and $post(u) < post(v)$	Not possible

Remark: $\nexists$ back edge $\Leftrightarrow \nexists$ closed walk (cycle)

Breadth-First Search (BFS)

Instead of searching through the depth of a graph, one can also go first through all the successor of the root with the BFS algorithm.

Pseudocode

```
BFS(G):
    for (v in V not marked):
        DFS-Visit(v)
```

```
DFS-VIsit(v):
    Q = new Queue()
    active[v] = true //used to check whether a vertex is in the queue or not
    enqueue(v, Q)
    while (!isEmpty(Q)):
        w = dequeue(Q)
        visited[w] = true
        for ((w, x) in E):
            if(!active[x] && !visited[x]):
                active[x] = true
                enqueue(x, Q)
```

Runtime

Operations	$T(n) \in \Theta(\|E\| + \|V\|)$
Memory	$T(n) \in \Theta(\|V\|)$

Find shortest path in DAG (Directed Acyclic Graph)

We can compute a recurrence following the topological sorting of the graph.

Psudocode

```
ShortestPath(V):
    d[s] = 0, d[v] = inf
    for (v in V \ {s}, following topological sorting):
        for (u, v, s.t. (u, v) in E):
            d[v] = min(d[u] + c(u,v))
```

Runtime

$T(n) \in \mathcal{O}((|E| * |V|) * log(|V|))$

Djikstra

Used to find the shortest (cheapest) path between two nodes in a graph.
Remark: The graph must **not** have negative weights