Graphs

Juan Mendivelso, Assistant Professor

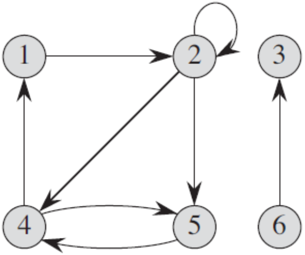
Universidad Nacional de Colombia
School of Science
Department of Mathematics

Contenido

Contenido

Directed Graph (Digraph)

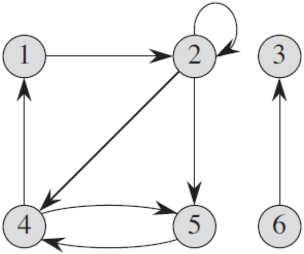
- A directed graph (or digraph) G is a pair (V, E), where V is a finite set and E is a binary relation on V.
- The set V is called the **vertex set** of G, and its elements are called vertices.
- The set E is called the **edge set** of G, and its elements are called edges.



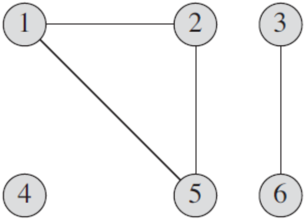
Directed Graph (Digraph)

Example 1.

- $V = \{1, 2, 3, 4, 5, 6\}$
- $E = \{(1,2), (2,2), (2,4), (2,5), (4,1), (4,5), (5,4), (6,3)\}$

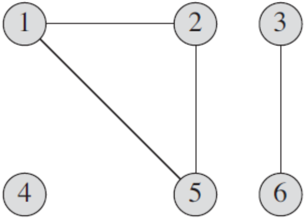


Undirected Graph



- In an undirected graph $G = (V, U)$, the edge set E consists of unordered pairs of vertices, rather than ordered pairs.
- An edge is a set $\{u, v\}$, where $u, v \in V$ and $u \neq v$ (no self-loops).
- By convention, we use the notation (u, v) for an edge, rather than the set notation.

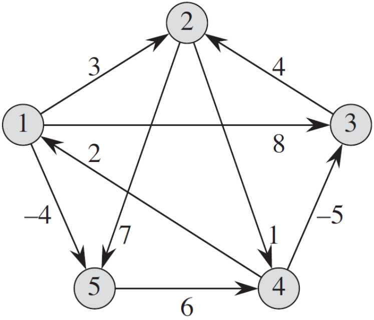
Undirected Graph



Example 2.

- $V = \{1, 2, 3, 4, 5, 6\}$
- $E = \{(1, 2), (1, 5), (2, 5), (3, 6)\}$

Weighted Graph



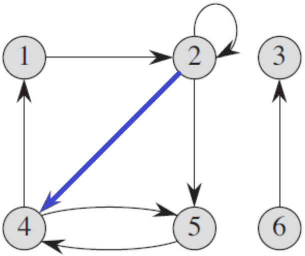
A graph $G = (V, E)$ is **weighted** if there exists a **weight function** $\omega : E \to \mathbb{R}$ that associates a weight $\omega(u, v)$ to each edge $(u, v) \in E$.

Incidence

- Edge (u, v) is **incident from** vertex u and **incident to** vertex v.
- Edge (u, v) **leaves** vertex u and enters vertex v.
- Undirected edge (u, v) is **incident on** both u and v.

Example 3.

Blue edge leaves vertex 2 and enters vertex 4.

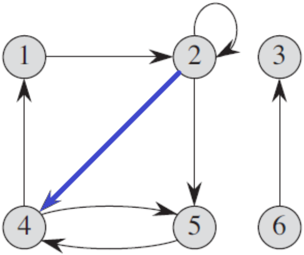


Adjacency

- Given edge $(u, v) \in E$, vertex v is said to be **adjacent** to vertex u. This is denoted by $u \to v$.
- Adjacency is a symmetric relation for undirected graphs.

Example 4.

Vertex 4 is adjacent to vertex 2.

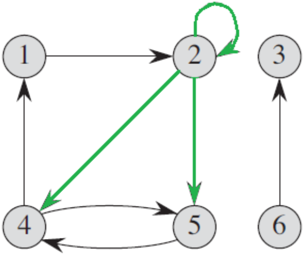


Out-Degree

The **out-degree** of a vertex is the number of edges leaving it.

Example 5.

The out-degree of vertex 2 is 3.

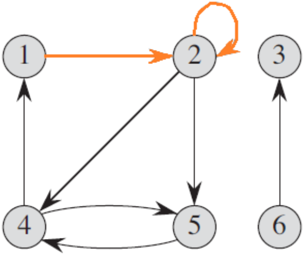


In-Degree

The **in-degree** of a vertex is the number of edges entering it.

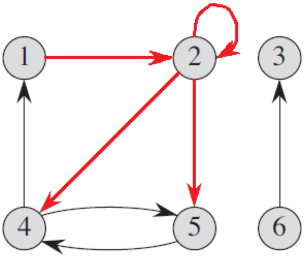
Example 6.

The in-degree of vertex 2 es 2.



Degree

- The **degree** of a vertex in a digraph is in-degree plus its out-degree.
- The **degree** of a vertex in an undirected graph is the number of edges incident on it.
- A vertex whose degree is 0 is called **isolated**.



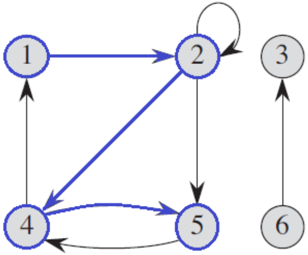
Example 7.

The degree of vertex 2 es 5.

Contenido

Path

A **path** of length k from a vertex u to a vertex u' in a graph $G = (V, E)$ is a sequence $\langle v_0, v_1, v_2, \ldots, v_k \rangle$ of vertices such that $u = v_0$, $u' = v_k$ and $(v_{i-1}, v_i) \in E$ for $i \in \{1, 2, \ldots, k\}$.

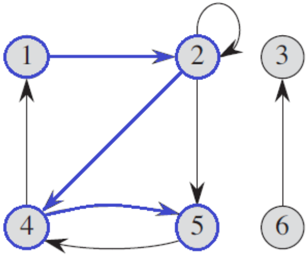


Characteristics of Path

- **Length of a path:** Number of edges in it.
- A path **contains** vertices v_0, v_1, ..., v_k and edges (v_0, v_1), (v_1, v_2), ..., (v_{k-1}, v_k).

Example 8.

The path $\langle 1, 2, 4, 5 \rangle$ is highlighted in blue. It contains edges $(1, 2)$, $(2, 4)$, $(4, 5)$.

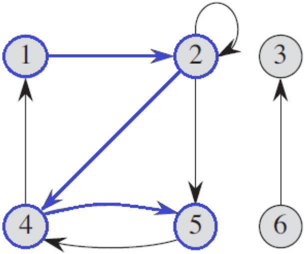


Reachable Vertices

If there is a path p from vertex u to vertex u', we say that u' is **reachable** from u via p. It is denoted by $u \overset{p}{\rightsquigarrow} u'$.

Example 9.

Vertex 5 is reachable from vertex 1.

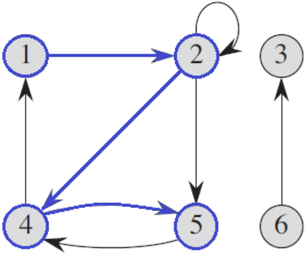


Simple Path

- A path is **simple** if all vertices in the path are distinct.
- Other notation: **walk** (path) and **path** (simple path).

Example 10.

The path in blue is simple.

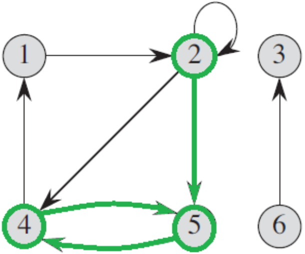


Simple Path

- A path is **simple** if all vertices in the path are distinct.
- Other notation: **walk** (path) and **path** (simple path).

Example 11.

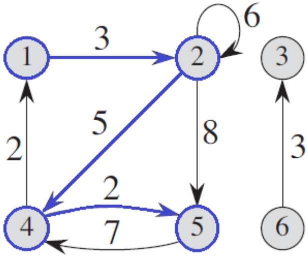
The path in green is not simple.



Weight of a Path

Given a weighted digraph $G = (V, E)$, whose weight function is $\omega : E \to \mathbb{R}$, the **weight of the path** $p = \langle v_0, v_1, \ldots, v_k \rangle$ is defined as:

$$\omega(p) = \sum_{i=1}^{k} \omega(v_{i-1}, v_i) \tag{1}$$



Example 12.

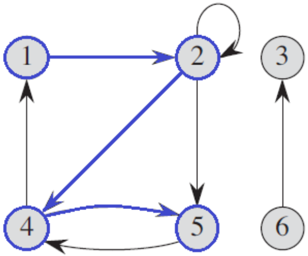
The weight of the blue path is 10.

Subpath

Given a path $p = \langle v_0, v_1, \ldots v_k \rangle$, a length-$k$ **subpath** of p is a contiguous subsequence $\langle v_i, v_{i+1}, \ldots, v_j \rangle$ of its vertices, where $0 \leq i \leq j \leq k$.

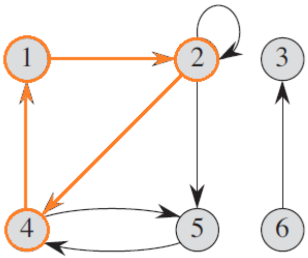
Example 13.

A subpath of the path in blue is $\langle 2, 4, 5 \rangle$.



Cycle

- In a directed graph, a path $p = \langle v_0, v_1, \ldots, v_k \rangle$ forms a **cycle** if $v_0 = v_k$ and the path contains at least one edge.
- The cycle is **simple** if, in addition, $v_1, v_2, \ldots, v_k$ are distinct.
- A self-loop is a cycle of length 1.

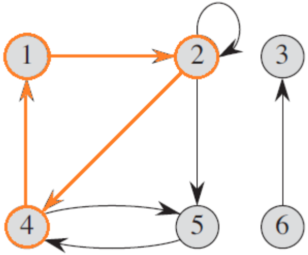


Example 14.

The path in orange is a simple cycle.

Cycle

- Two paths $\langle v_0, v_1, v_{k-1}, v_0 \rangle$ and $\langle v'_0, v'_1, v'_{k-1}, v'_0 \rangle$ form the same cycle if there exists an integer j such that $v'_i = v_{(i+j) \mod k}$ for $i = 0, 1, \ldots, k-1$.

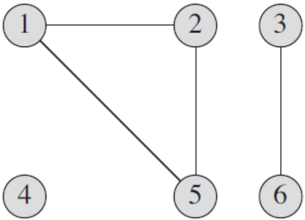


Connected Component

- An undirected graph is **connected** if every vertex is reachable from all other vertices.
- The connected components of a graph are the equivalence classes of vertices under the **reachable from** relation.

Example 15.

This graph has three components: $\{1, 2, 5\}$, $\{3, 6\}$ and $\{4\}$.

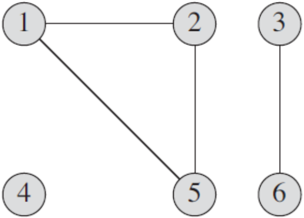


Connected Component

- An undirected graph is connected if it has exactly one connected component.
- The edges of a component are those that are incident only on the components of the component.

Example 16.

The edges of the component $\{1, 2, 5\}$ are $(1, 2)$, $(2, 5)$ and $(1, 5)$.

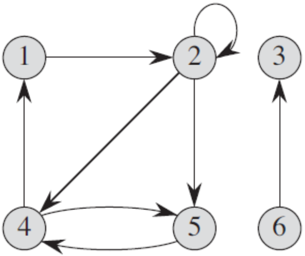


Strongly Connected Component (SCC)

- A digraph is **strongly connected** if every two vertices are reachable from each other.
- The strongly connected components of a digraph are the equivalence classes of vertices under the **mutually reachable** relation.
- A digraph is strongly connected if has only one strongly connected component.

Example 17.

The SCC are $\{1, 2, 4, 5\}$, $\{3\}$ and $\{6\}$.



Contenido

Graph Types regarding Number of Edges

Empty Graph

A graph $G = (V, E)$ is **empty** iff $E = \emptyset$.

Complete Graph

A graph $G = (V, E)$ is **complete** iff G is undirected and every pair of vertices is adjacent.

Sparse Graph

A graph $G = (V, E)$ is **sparse** iff $|E|$ is much less than $|V|^2$.

Dense Graph

A graph $G = (V, E)$ is **dense** iff $|E|$ is close to $|V|^2$.

Graph Types regarding Reachability

Connected Graph

- An undirected graph is connected if every vertex is reachable from all other vertices.
- An undirected graph is connected if it has exactly one connected component.

Strongly Connected Graph

- A digraph is **strongly connected** if every two vertices are reachable from each other.
- A digraph is strongly connected if has only one strongly connected component.

Graph Types regarding Cycles

(Free) Tree

A graph $G = (V, E)$ is a **(free) tree** iff G is a connected acyclic undirected graph.

Forest

A graph $G = (V, E)$ is a **forest** iff G is an acyclic undirected graph.

DAG

A graph $G = (V, E)$ is a **DAG** iff G is a directed acyclic graph.

Other Graph Types

Bipartite Graph

A graph $G = (V, E)$ is **bipartite** iff G is undirected and V can be partitioned into two sets V_1 and V_2 such that $(u, v) \in E$ implies that $(u \in V_1 \land v \notin V_2) \lor (u \notin V_1 \land v \in V_2)$.

Graph Variants

Multigraphs

Similar to undirected graphs but the set of edges can include self-loops and multiple edges between the same pair of vertices.

Hypergraphs

- Similar to undirected graphs, but it contains **hyperedges** instead of edges.
- A **hyperedge** connects an arbitrary number of vertices rather than a pair.

Contenido

Representation of Graphs

Adjacency Matrix

Good for dense graphs or graphs where fast reachability queries are required.

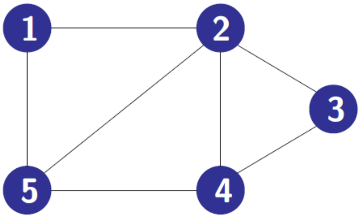
Adjacency List

Ideal for sparse graphs.

Adjacency Matrix

Given the graph $G = (V, E)$, assume vertices in V are numbered: 1, 2, . . ., $|V|$. Then, the adjacency matrix $A = (a_{ij})$ has length $|V| \times |V|$ and it is defined as:

$$a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise.} \end{cases}$$

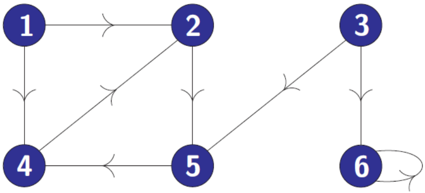


$$\begin{array}{c|ccccc|} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 0 & 0 & 1 \\ 2 & 1 & 0 & 1 & 1 & 1 \\ 3 & 0 & 1 & 0 & 1 & 0 \\ 4 & 0 & 1 & 1 & 0 & 1 \\ 5 & 1 & 1 & 0 & 1 & 0 \\ \hline \end{array}$$

Adjacency Matrix

Given the graph $G = (V, E)$, assume vertices in V are numbered: 1, 2, …, $|V|$. Then, the adjacency matrix $A = (a_{ij})$ has length $|V| \times |V|$ and it is defined as:

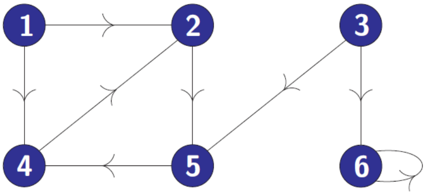
$$a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise.} \end{cases}$$



$$\begin{array}{c|cccccc|} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 1 & 0 \\ 3 & 0 & 0 & 0 & 0 & 1 & 1 \\ 4 & 0 & 1 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 1 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline \end{array}$$

Adjacency Matrix

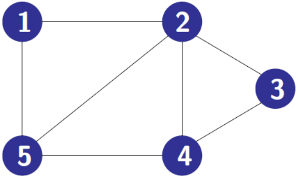
- It requires $\Theta(|V|^2)$ space.
- We can query adjacency in $\Theta(1)$ time.
- It supports weighted graphs by storing $w(u, v)$ at a_{ij} rather than 1.

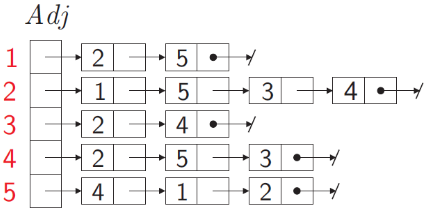


	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

Adjacency Lists

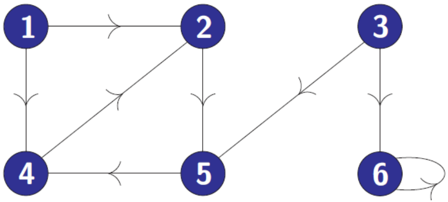
- The **adjacency-list** representation of a graph $G = (V, E)$ consists of an array Adj of $|V|$ lists, one for each vertex in V.
- For each $u \in V$, the adjacency list $Adj[u]$ contains all the vertices v such that there is an edge $(u, v) \in E$.

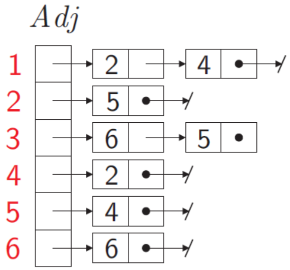




Adjacency Lists

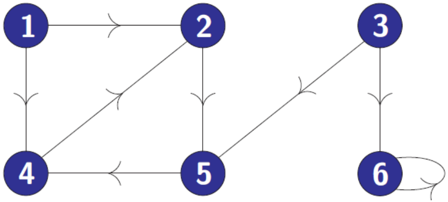
- The **adjacency-list** representation of a graph $G = (V, E)$ consists of an array *Adj* of $|V|$ lists, one for each vertex in V.
- For each $u \in V$, the adjacency list $Adj[u]$ contains all the vertices v such that there is an edge $(u, v) \in E$.

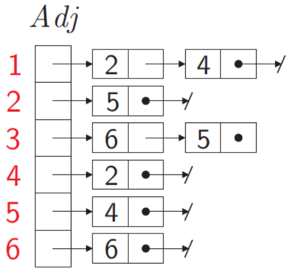




Adjacency Lists

- They require $\Theta(|V| + |E|)$ space.
- To determine whether vertex u' is adjacent to vertex u, we need to traverse $Adj[u]$.
- We can consider weights by adding an extra field at each node.





Contenido

Breadth First Tree

Predecessor Subgraph

Given the graph $G = (V, E)$, the **predecessor subgraph** of G is defined as $G_\pi = (V_\pi, E_\pi)$, where

- $V_\pi = \{v \in V : v.\pi \neq NIL\} \cup \{s\}$.
- $E_\pi = \{(v.\pi, v) : v \in (V_\pi - \{s\})\}$.

It is a **tree** that contains a shortest path from s to each reachable vertex $v \in V$.

Lemma 6

When applied to a graph $G = (V, E)$, BFS constructs π so that the predecessor subgraph is a breadth-first tree.

Printing the Shortest Paths

```
PRINT-PATH(G, s, v)
if v == s
    print s
elseif v.π == NIL
    print "no path from" s "to" v "exists"
else PRINT-PATH(G, s, v.π)
    print v
```

Time Complexity

Linear on the number of vertices on the path.

Contenido

Depth First Search (DFS)

Main Ideas

- It searches deeper in the graph whenever possible.
- In particular, it searches the edges that lead to undiscovered vertices from the most recently discovered vertex v.
- When v has no edges to undiscovered vertices, the search **backtracks** to explore edges leaving from the vertex from which v was discovered.

Depth First Search (DFS)

About the sources

- In BFS, we start from a single source s to find all the reachable vertices from s. We obtain a Breadth First Tree.
- In DFS, we start from different (unexplored) sources until all vertices in the graph are discovered. We obtain a Depth First Forest with possibly many Depth First Trees.

Colors

Colors

Same meaning as in BFS:

- **White:** Undiscovered.
- **Gray:** Discovered but not all its neighbors have been explored.
- **Black:** Finished. All its neighbors have already been explored.

Predecessor Subgraph

Predecessor Subgraph

- The predecessor subgraph of $G = (V, E)$ is defined as $G_\pi = (V, E_\pi)$, where
  - $E_\pi = \{(v.\pi, v) : v \in V \wedge v.\pi \neq NIL\}$
  - v is a descendant of u was discovered when u was gray.
- It forms a Depth First Forest comprised of possibly many Depth First Trees.
- Each v ends up in exactly one DFS tree.

Timestamps

Timestamps

- Integers from 1 to $2|V|$ that represent moments in time.
- Each vertex $v \in V$ has two timestamps:
  - $v.d$: time when v was discovered and grayed.
  - $v.f$: time when v was blackened, i.e. $Adj[v]$ was fully explored.
- They provide important information about the structure of the graph.
- They are helpful for analizing the behavior of DFS.
- $v.d < v.f$.
- v is
  - white before $v.d$.
  - gray at $v.d$ and before $v.f$.
  - black at $v.f$ thereafter.

Pseudocode

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

```
DFS-VISIT(G, u)
time = time + 1              // white vertex u has just been discovered
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]        // explore edge (u, v)
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK              // blacken u; it is finished
time = time + 1
u.f = time
```

Analysis

DFS(G)

```
for each vertex u ∈ G.V                Θ(V)
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V                Θ(V)
    if u.color == WHITE
        DFS-VISIT(G, u)
```

DFS-VISIT(G, u)

```
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]                  Σ_{v∈V} |Adj[v]| = Θ(E)
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

- DFS may vary depending on the order of the adjacency lists.
- DFS-VISIT(u) is called once for each $u \in V$.
- The total complexity is $\Theta(|V| + |E|)$.

Analysis

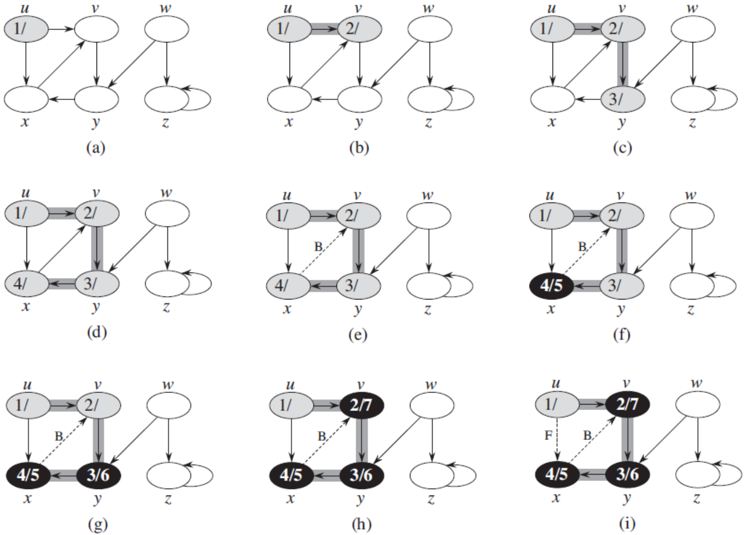
```
DFS(G)
for each vertex u ∈ G.V              Θ(V)
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V              Θ(V)
    if u.color == WHITE
        DFS-VISIT(G, u)
```

```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

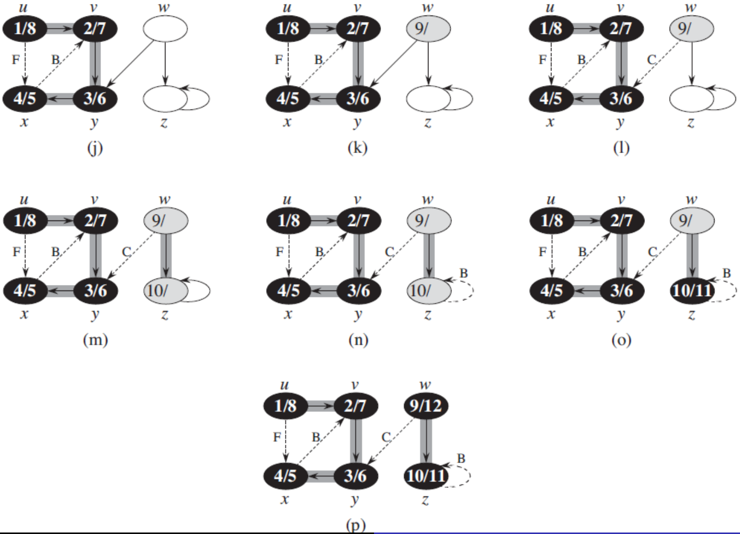
$$\boxed{\sum_{v \in V} |Adj[v]| = \Theta(E)}$$

- DFS may vary depending on the order of the adjacency lists.
- $\text{DFS-VISIT}(u)$ is called once for each $u \in V$.
- The total complexity is $O(|V| + |E|)$.

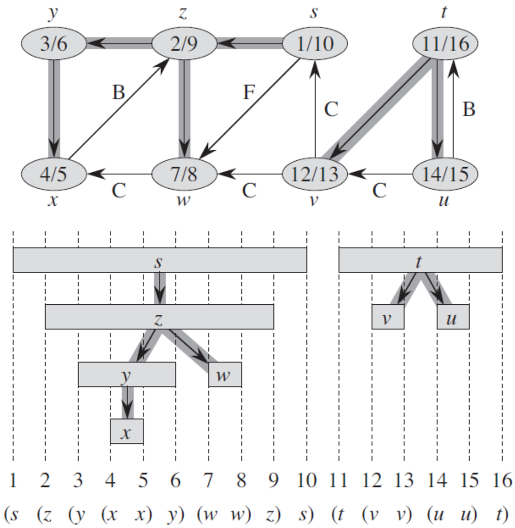
Example



Example



Parenthesis Structure



Parenthesis Structure

Theorem 7 (Parenthesis Theorem)

In any DFS of a graph $G = (V, E)$, for any $u, v \in V$, exactly one of the following conditions hold:

- $[u.d, u.f]$ and $[v.d, v.f]$ are entirely disjoint and u and v are not descendant of each other in the forest.
- $[u.d, u.f]$ is entirely contained in $[v.d, v.f]$ and u is a descendant of v in a Depth First tree.
- $[v.d, v.f]$ is entirely contained in $[u.d, u.f]$ and v is a descendant of u in a Depth First tree.

Parenthesis Structure

Corollary 8

Vertex v is a proper descendant of vertex u in the Depth First forest for a graph G if and only if $u.d < v.d < v.f < u.f$.

White-path Theorem

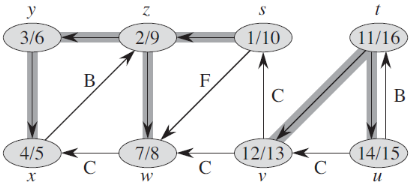
White-path Theorem

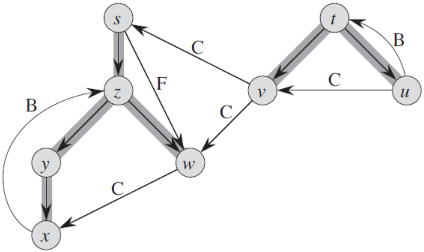
In a Depth First forest of a graph $G = (V, E)$, vertex v is a descendant of vertex u if and only if at the time $u.d$ that the search discovers u, there is a path from u to v consisting entirely of white vertices.

Classification of Edges

- **Tree Edges:** Edges in the Depth First forest. Edge (u, v) is a tree edge if v was discovered by exploring edge (u, v). In particular, $u.d < v.d < v.f < u.f$.
- **Back Edges:** Edges (u, v) connecting vertex u to an ancestor v in the Depth First tree. It holds that $v.d < u.d < u.f < v.f$.
- **Forward Edges:** Non-tree edges (u, v) connecting a vertex u to a descendant v in the Depth First tree. Like in tree edges, $u.d < v.d < v.f < u.f$.
- **Cross Edges:** It holds that $v.d < v.f < u.d < u.f$.
  - They can go between vertices in the same Depth First tree as long as one of them is not an ancestor of the other.
  - They can go between vertices in different Depth First trees.

Classification of Edges

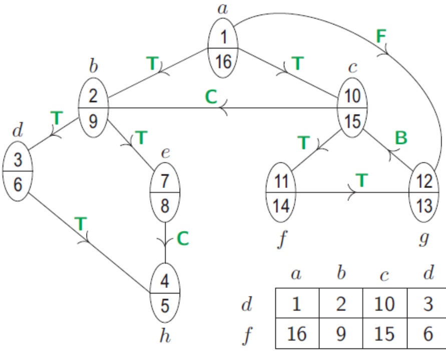




DFS can classify edges (u, v) according to the color of v:

- **White:** Tree edge.
- **Gray:** Back edge.
- **Black:** Forward Edge $(u.d < v.d)$ or Cross Edge $(u.d > v.d)$.

Classification of Edges



	a	b	c	d	e	f	g	h
d	1	2	10	3	7	11	12	4
f	16	9	15	6	8	14	13	5

Tree edges	(a,b), (b,d), (b,e), (d,h), (a,c), (c,f), (f,g)
Back edges	(g,c)
Forward edges	(a,g)
Cross edges	(e,h), (c,b)

Classification of Edges in Undirected Graphs

- Ambiguous since (u, v) and (v, u) are the same.
- The first type that applies in the list is taken.
- Forward and Cross edges never occur.

Theorem 10

In a DFS of an undirected graph $G = (V, E)$, every edge is either a tree edge or a back edge.

Contenido

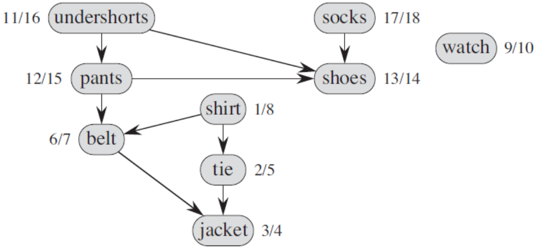
Topological Sort

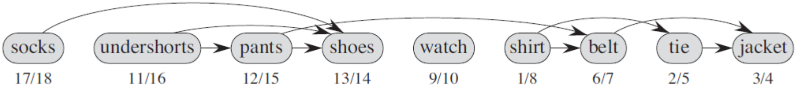
- A **topological sort** of a dag $G = (V, E)$ is a linear ordering of all its vertices such that if G contains (u, v), the u appears before v in the ordering.
- Put vertices on a line so that edges go from left to right.
- If a graph contains a cycle, such ordering is not possible.
- **Time Complexity:** $\Theta(|V| + |E|)$.

TOPOLOGICAL-SORT(G)

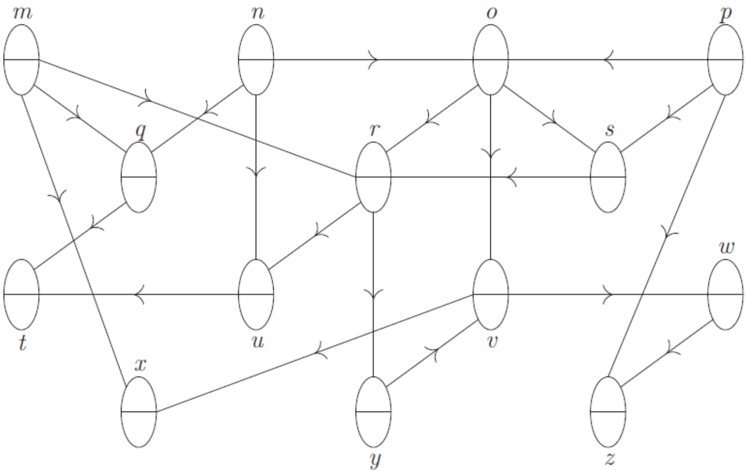
```
call DFS(G) to compute finishing times v.f for each vertex v
as each vertex is finished, insert it onto the front of a linked list
return the linked list of vertices
```

Topological Sort

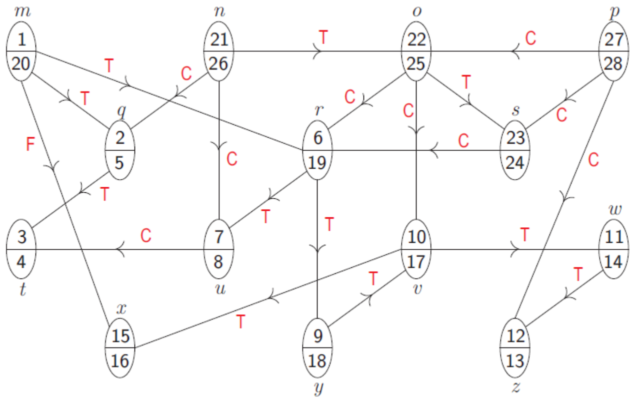




Topological Sort



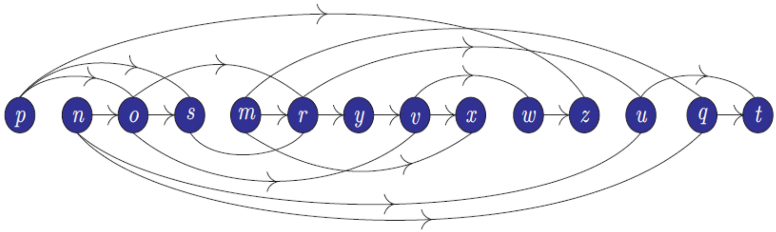
Topological Sort



	m	n	o	p	q	r	s	t	u	v	w	x	y	z
d	1	21	22	27	2	6	23	3	7	10	11	15	9	12
f	20	26	25	28	5	19	24	4	8	17	14	16	18	13

Topological Sort

	m	n	o	p	q	r	s	t	u	v	w	x	y	z
d	1	21	22	27	2	6	23	3	7	10	11	15	9	12
f	20	26	25	28	5	19	24	4	8	17	14	16	18	13



Correctness of Topological Sort

Lemma 11

A directed graph $G = (V, E)$ is acyclic if and only if a DFS of G yields no back edges.

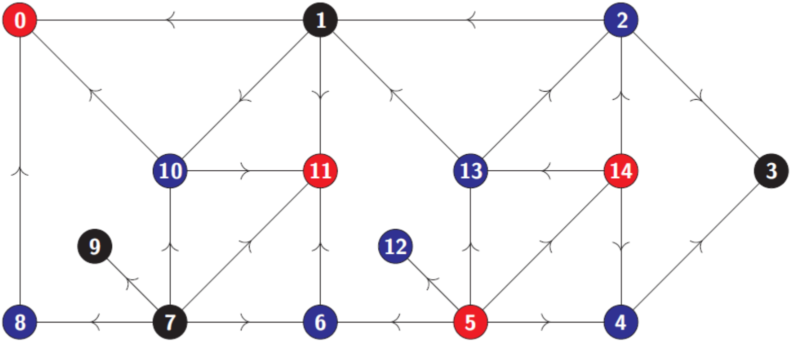
Theorem 12

$\textsc{TopologicalSort}(G)$ produces a topological sort of graph $G = (V, E)$.

Exercise ofTopological Sort

Exercise

Calculate a topological sort for the following graph.



Contenido

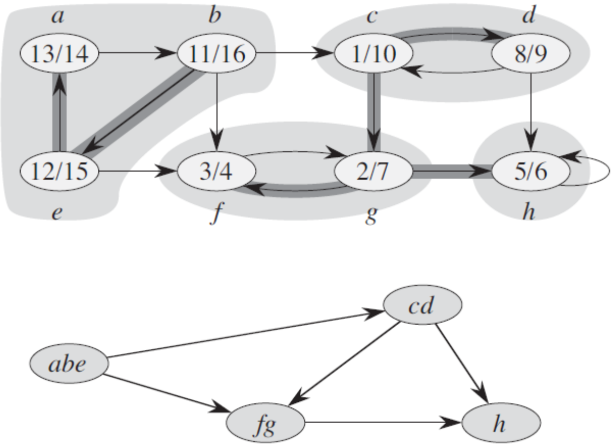
Strongly Connected Components

- Classical application of DFS.
- Many algorithms on graphs start by decomposing the digraph in strongly connected components.
- They work separately in each component.
- Then, they combine the solutions according to the connections of the components.

Strongly Connected Component

A strongly connected component of a digraph $G = (V, E)$ is a maximal set of vertices $C \subseteq V$ such that for every pair of vertices $u, v \in C$, we have both $u \overset{p}{\rightsquigarrow} v$ and $v \overset{p}{\rightsquigarrow} u$.

Strongly Connected Components



Transpose of a Graph

Transpose of a Graph

- Given a digraph $G = (V, E)$, the **transpose** of G is defined as $G^T = (V, E^T)$ where
  - $E^T = \{(u, v) : (v, u) \in E\}$.
- G^T can be computed in $O(|V| + |E|)$.
- G and G^T have the same strongly connected components.

Component Graph

Component of a Graph

- Given a digraph $G = (V, E)$ whose strongly connected components are C_1, C_2, ..., C_k, the **component graph** of G is defined as $G^{SCC} = (V^{SCC}, E^{SCC})$ where
  - $V^{SCC} = \{v_1, v_2, \ldots v_k\}$ contains a vertex v_i for each strongly connected component C_i of G, $i \in \{1, 2, \ldots, k\}$.
  - There is an edge $(v_i, v_j) \in E^{SCC}$ if G contains a directed edge (x, y) for some $x \in C_i$ and $y \in C_j$.
- G^{SCC} is the result of contracting each component in a vertex.
- G^{SCC} is a dag.

Properties

Lemma 13

,

- Let C and C' be distinct strongly connected components in digraph $G = (V, E)$.
- Let $u, v \in C$.
- Let $u', v' \in C'$.
- Suppose that G contains a path $u \overset{p}{\rightsquigarrow} u'$.

Then, G cannot also contain a path $v' \overset{p}{\rightsquigarrow} v$.

Pseudocode

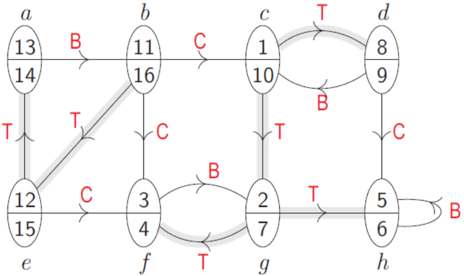
STRONGLY-CONNECTED-COMPONENTS(G)

1 call DFS(G) to compute finishing times $u.f$ for each vertex u
2 compute G^{T}
3 call DFS(G^{T}), but in the main loop of DFS, consider the vertices
    in order of decreasing $u.f$ (as computed in line 1)
4 output the vertices of each tree in the depth-first forest formed in line 3 as a
    separate strongly connected component

- **Complexity:** $\Theta(|V| + |E|)$.
- Let $u.d$ and $u.f$ respectively denote the discovery and finishing times of the first call to DFS (line 1).
- Extend the notion of discovery and finishing times to sets of vertices. If $U \subseteq V$, we define
  - $d(U) = \min_{u \in U}\{u.d\}$ (earliest discovery).
  - $f(U) = \max_{u \in U}\{u.f\}$ (latest finishing).

Strongly Connected Components

G

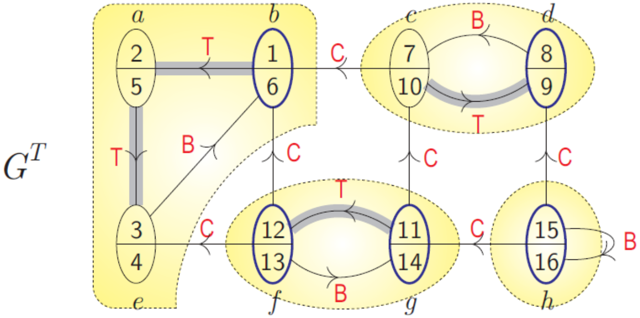


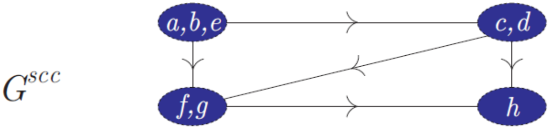
	a	b	c	d	e	f	g	h
d	13	11	1	3	12	3	2	5
f	14	16	10	9	15	4	7	6

Vertices in order of decreasing f_u : b, e, a, c, d, g, h, f

Strongly Connected Components

Vertices in order of decreasing f_u : b, e, a, c, d, g, h, f





Correctness

Lemma 14

- Let C and C' be distinct strongly connected components in digraph $G = (V, E)$.
- Suppose there is an edge $(u, v) \in E$, where $u \in C$ and $v \in C'$.

Then, $f(C) > f(C')$.

Corollary 15

- Let C and C' be distinct strongly connected components in digraph $G = (V, E)$.
- Suppose there is an edge $(u, v) \in E^T$, where $u \in C$ and $v \in C'$.

Then, $f(C) < f(C')$.

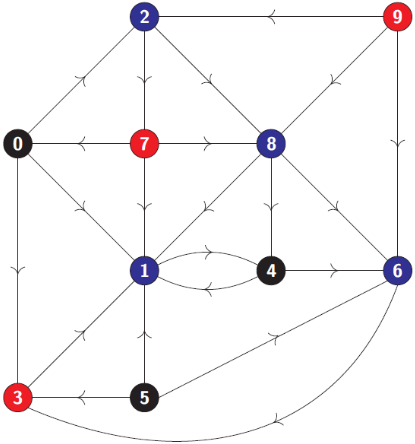
Correctness

Theorem 16

$\textsc{StronglyConnectedComponents}(G)$ correctly computes the strongly connected components of digraph $G = (V, E)$.

Exercise of Strongly Connected Components

Find the SCC of the following graph.



Bibliography

- Cormen TH, Leiserson CH, Rivest RL, Stein C. **Introduction to Algorithms, 3rd Edition.** The MIT Press. 2009.
- Curso de Algoritmia Avanzada de Yoan Pinzón. 2012.