Decomposition of Graphs: Depth First Search

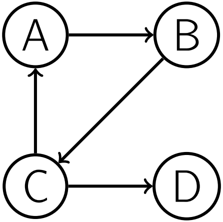
Daniel Kane

Department of Computer Science and Engineering
University of California, San Diego

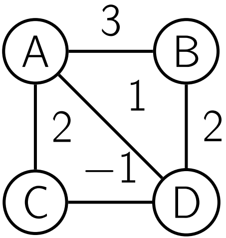
Graph Algorithms
Data Structures and Algorithms

Outline

Graphs

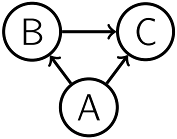


$$V = \{A, B, C, D\}$$
$$E = \{(A, B), (C, D), (C, A), (B, C)\}$$

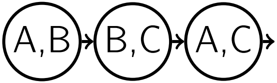


$$V = \{A, B, C, D\}$$
$$E = \{(\{A, B\}, 3), (\{A, D\}, 1),$$
$$(\{B, D\}, 2), (\{C, D\}, -1), (\{A, C\}, 2)\}$$

Ways to Represent



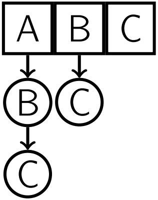
edge list



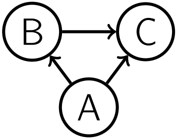
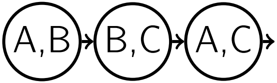
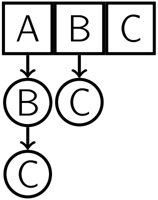
adjacency matrix

	A	B	C
A	0	1	1
B	0	0	1
C	0	0	0

adjacency list

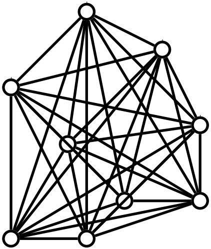


Ways to Represent

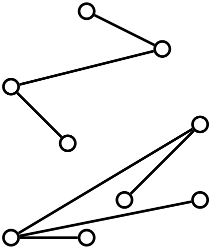
	edge list	adjacency matrix	adjacency list
		$\begin{array}{cccc} & A & B & C \\ A & 0 & 1 & 1 \\ B & 0 & 0 & 1 \\ C & 0 & 0 & 0 \end{array}$	
space	$\Theta(\lvert E \rvert)$	$\Theta(\lvert V \rvert^2)$	$\Theta(\lvert V \rvert + \lvert E \rvert)$
$(u, v) \in E$?	$\Theta(\lvert E \rvert)$	$\Theta(1)$	$\deg(u)$
neighbors of u	$\Theta(\lvert E \rvert)$	$\Theta(\lvert V \rvert)$	$\deg(u)$

Sparse and Dense Graphs

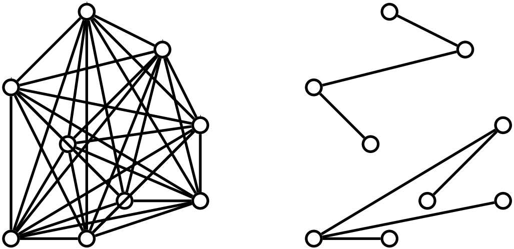
dense graph



sparse graph



Sparse and Dense Graphs



	dense graph	sparse graph
# edges	$\Theta(\lvert V\rvert^2)$	$\Theta(\lvert V\rvert)$
average degree	$\Theta(\lvert V\rvert)$	$\Theta(1)$
adj. matrix	$\Theta(\lvert V\rvert^2)$	$\Theta(\lvert V\rvert^2)$
adj. list	$\Theta(\lvert V\rvert^2)$	$\Theta(\lvert V\rvert)$

Outline

Explore(v)

```
{Input: a node v of a graph
G = (V, E).}
{Output: visited[u] = true for all
  nodes u reachable from v.}
visited[v] ← true
Previsit(v)
for each edge (v, u) ∈ E:
  if visited[u] = false:
    Explore(u)
Postvisit(v)
```

Formal Proof

- Clearly only vertices reachable from v are visited.

Formal Proof

- Clearly only vertices reachable from v are visited.
- To show that all of them are visited assume, for the sake of contradiction, that a vertex u is reachable from v but was not visited.

Formal Proof

- Clearly only vertices reachable from v are visited.
- To show that all of them are visited assume, for the sake of contradiction, that a vertex u is reachable from v but was not visited.
- Take any path from v to u and denote by z the last vertex on this path that was visited and by w its subsequent vertex.

Formal Proof

- Clearly only vertices reachable from v are visited.
- To show that all of them are visited assume, for the sake of contradiction, that a vertex u is reachable from v but was not visited.
- Take any path from v to u and denote by z the last vertex on this path that was visited and by w its subsequent vertex.

- Hence `Explore` was not called for w while iterating over the neighbors of z.

Depth-First Search

$\mathtt{DFS}(G)$

```
for all v ∈ V:
  visited[v] ← false
for all v ∈ V:
  if visited[v] = false:
    Explore(v)
```

Depth-First Search

$\texttt{DFS}(G)$

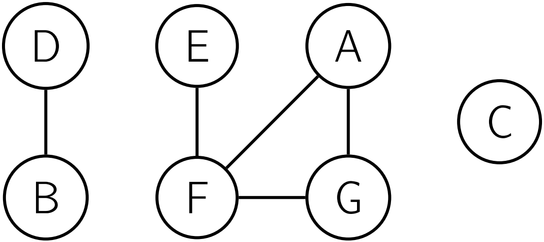
$\texttt{for all } v \in V:$
  $\texttt{visited}[v] \leftarrow \texttt{false}$

$\texttt{for all } v \in V:$
  $\texttt{if visited}[v] = \texttt{false:}$
    $\texttt{Explore}(v)$

Running time: $O(|V| + |E|)$ since $\texttt{Explore}$ is called exactly once for each vertex $v \in V$ and each edge is examined either once (for

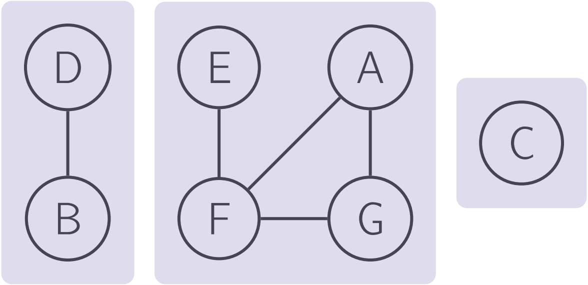
Connected Components

A connected component of un undirected graph is an inclusion-wise maximal subset of vertices such that there is a path between any two of them.



Connected Components

A connected component of un undirected graph is an inclusion-wise maximal subset of vertices such that there is a path between any two of them.



Finding Connected Components

Previsit(v)

```
ccnum[v] ← cc
```

DFS(G)

```
cc ← 0
for all v ∈ V:
  visited[v] ← false
  ccnum[v] ← −1

for all v ∈ V:
  if visited[v] = false:
    cc ← cc + 1
    Explore(v)
```

Outline

Previsit and Postvisit Orderings

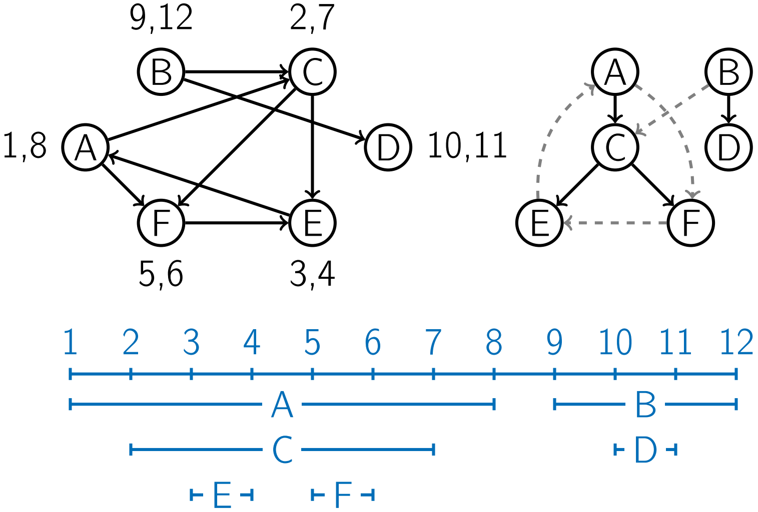
Previsit(v)

$\texttt{pre}[v] \leftarrow \texttt{clock}$
$\texttt{clock} \leftarrow \texttt{clock} + 1$

Postvisit(v)

$\texttt{post}[v] \leftarrow \texttt{clock}$
$\texttt{clock} \leftarrow \texttt{clock} + 1$

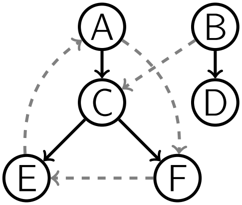
Types of Edges

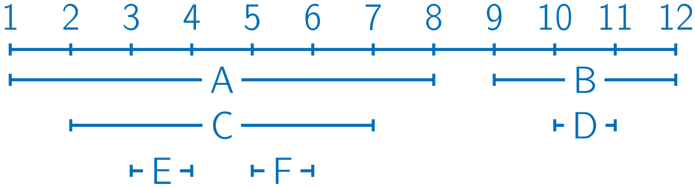


Types of Edges

Types of edges:

- tree edge: (A, C), (C, E), (C, F), (B, D)
- forward edge: (A, F)
- cross edge: (B, C), (F, E)
- back edge: (E, A)





Types of edges

tree/forward edge (u, v):

back edge (u, v):

cross (u, v):

Directed acyclic graphs

Lemma

A directed graph has a cycle if and only if its depth-first search reveals a back edge.

Proof

$\Rightarrow$ If (u, v) is a back edge, then there is a path from v to u in DFS tree.

Directed acyclic graphs

Lemma

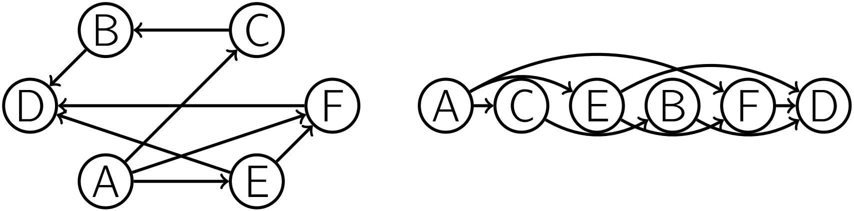
A directed graph has a cycle if and only if its depth-first search reveals a back edge.

Proof

- $\Rightarrow$ If (u, v) is a back edge, then there is a path from v to u in DFS tree.
- $\Leftarrow$ Let $u_1 \to u_2 \to \ldots \to u_k \to u_1$ be a cycle and assume w.l.o.g. that u_1 is the first vertex `Explore` was

Topological ordering

A topological ordering of a directed graph is a linear ordering of its vertices such that for any edge (u, v), u comes before v.



Lemma

A directed graph can be linearized iff it is a DAG.

Proof

$\Rightarrow$ If there is a cycle the graph cannot be linearized.

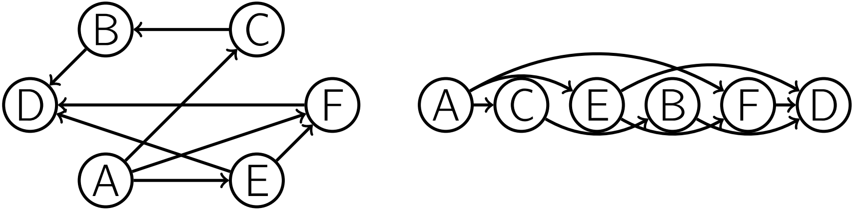
Lemma

A directed graph can be linearized iff it is a DAG.

Proof

- $\Rightarrow$ If there is a cycle the graph cannot be linearized.
- $\Leftarrow$ Each DAG contains at least one source (a vertex with no incoming edges) and at least one sink (no outgoing edges). This suggests the

Example



Visualization:

http://www.cs.usfca.edu/~galles/visualization/TopoSortIndegree.html

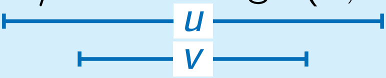
Lemma

In a DAG every edge leads to a vertex with a lower $\texttt{post}$ number.

Proof

If $\texttt{post}[v] > \texttt{post}[u]$ for an edge (u, v) then (u, v) is a back edge.

tree/forward edge (u, v):



back edge (u, v):

Example