# Graphs

Graphs are built from

- Vertices (aka nodes)
- Edges (perhaps with extra data)

Two kinds of graphs:

- directed: edge is from source $u$ to target $v$ ($u = v$ possible)
- undirected: between $u$ and $v$ ($u = v$ not possible)

We do not model

- hypergraphs (edges with $> 2$ end points)
- multigraphs

We often study paths in graphs; a cycle is a path that returns to itself without following the same edge twice.

## Size Measures

The size of a graph depends on

- $n$, the number of nodes
- $a$, the number of edges

$n$ and $a$ may not be related, except

$$a \in O(n^2)$$

Many graph algorithms run in time $\Theta(n + a)$ which

- is in $\Theta(n)$ if $a \in O(n)$
- is in $\Theta(n^2)$ if $a \in \Theta(n^2)$

# Connectivity

For an undirected graph, two nodes are connected if there is a path between them.

For a directed graph, two nodes $u, v$ are

- strongly connected if path from $u$ to $v$ and path from $v$ to $u$
- unilateraly connected if path from $u$ to $v$ or path from $v$ to $u$
- weakly connected if path between $u$ and $v$ in corresponding undirected graph

# Trees

A tree is an undirected graph that is

- connected, and
- acyclic

In a tree, any node may be picked as the root. Then one can define various notions:

- parent (exactly one for each non-root node)
- child (each node may have arbitrarily many)
- leaf (node with no children)
- depth (distance from root)
- height (maximum depth).

Special case: binary trees.

# Graph Operations

- Get to check if a given edge exists (and if so perhaps get extra data)
- Put to add an edge to the graph (we do not require it checks for duplicates)
- Delete
- AllFrom returns a list of all edges from given node

# Representations

An Adjacency Matrix has an entry in row $i$, column $j$ iff the graph has an edge from $i$ to $j$

- get, put, delete all run in constant time
- allFrom runs in linear time (even if resulting list very short)

With Adjacency Lists, each node has a pointer to a list of the nodes to which there is an edge from that node

- put runs in constant time (as no check for duplicates)
- allFrom runs in constant time (though the returned list may be long)
- get (and delete) do not run in constant time.

# Assessment

An Adjacency Matrix

+ allows for quick get (and delete) even in dense graph
- but for a sparse graph,
  - allFrom is unnecessarily slow
  - wastes space