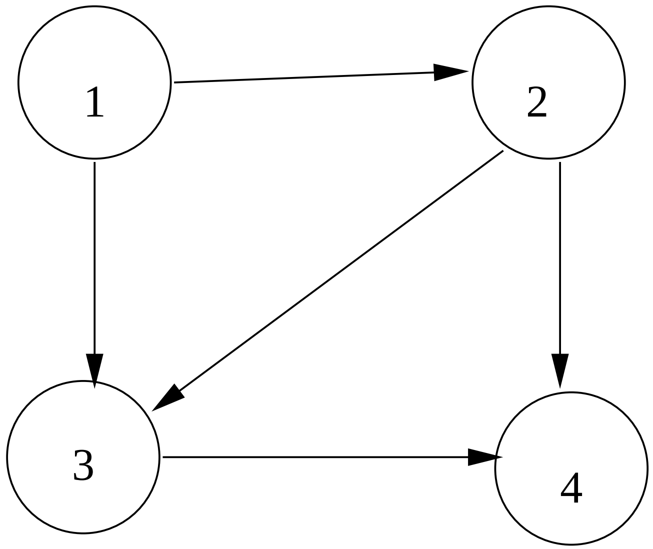
Graphs

(Version of 21 November 2005)

Definition: A *graph* G is a pair (V, E),
where V is a finite set of items, called the *vertices* of G,
and E is a binary relation on V (that is $E \subseteq V \times V$);
the elements of E connect vertices and are called the *edges* of G.

Example: $(\{1,2,3,4\}, \{(1,2),(1,3),(2,3),(2,4),(3,4)\})$



Applications of Graphs

Graphs can be used to represent *relationships* between things.

Example: An undirected graph where each vertex represents an intersection and an edge (i_1, i_2) between two intersections indicates that there is a road from intersection i_1 to intersection i_2.
See any city map or road map.

Example: A directed graph where each vertex represents a website and an edge (w_1, w_2) between two websites indicates that there is a link on website w_1 to website w_2.
See `http://research.lumeta.com/ches/map/gallery/`.

Representations of Graphs

Store E as an $\Theta(|V|^2)$ *adjacency matrix* of 0/1, indexed by V and V:

- Advantage: Constant (that is $\Theta(1)$) search time for edges.
- Advantage: Compact representation of *dense* graphs (where $|E|$ is close to $|V|^2$).
- Disadvantage: Wasteful of memory on *sparse* graphs (where $|E|$ is much smaller than $|V|^2$).

Store E as an $\Theta(|V| + |E|)$ array of *adjacency lists*, indexed by V:

- Advantage: Compact representation of sparse graphs.
- Disadvantage: Wasteful of memory on dense graphs.
- Disadvantage: No constant search time for edges.

The performance of algorithms depends on the graph representation.

Paths

A *path* of length k in a graph G is a sequence of k vertices

$$v_1, v_2, \ldots, v_k$$

such that for every $1 \leq i < k$ there is an edge (v_i, v_{i+1}) in G.

Often, paths are written as

$$v_1 \longrightarrow v_2 \longrightarrow \ldots \longrightarrow v_k$$

Examples: (see the graph on slide 1)

$$1 \longrightarrow 2 \longrightarrow 3$$

$$1 \longrightarrow 3 \longrightarrow 4$$

$$3 \longrightarrow 4$$

Weighted Graphs

Often we do not just want to express relationships,
but also some extra information.

Example: A graph with cities for vertices and roads for edges.
It would then also be useful to represent the lengths of these roads.

A *weighted graph* is a graph with a weight function $w : E \to \mathbb{R}$
from the edges E to the set $\mathbb{R}$ of the possible weights.

Often, we just *label* the edges with the weights.
Adjacency lists and matrices can readily be adapted to do so.

Example: The graph of slide 1 with added weights:

$$(\{1,2,3,4\},\{(1,2,0.5),(1,3,5.5),(2,3,0.5),(2,4,6.0),(3,4,0.1)\})$$

Weights of Paths

On the weighted graph of slide 5,
there are three ways of reaching vertex 4 from vertex 1:

$$1 \longrightarrow 2 \longrightarrow 4$$

$$1 \longrightarrow 3 \longrightarrow 4$$

$$1 \longrightarrow 2 \longrightarrow 3 \longrightarrow 4$$

But each of these paths has a different *weight*:

$$weight(1 \xrightarrow{0.5} 2 \xrightarrow{6.0} 4) = 0.5 + 6.0 = 6.5$$

$$weight(1 \xrightarrow{5.5} 3 \xrightarrow{0.1} 4) = 5.5 + 0.1 = 5.6$$

$$weight(1 \xrightarrow{0.5} 2 \xrightarrow{0.5} 3 \xrightarrow{0.1} 4) = 0.5 + 0.5 + 0.1 = 1.1$$

Common Questions on Graphs

What are the *shortest* (*minimum-weight*) *paths* between two given vertices of a weighted, directed graph?

What are the *connected components* of an undirected graph?
What are the *strongly connected components* of a directed graph?

Is there a path between every pair of vertices of a graph?
In other words: Is the graph (*strongly*) *connected*?

What is the *minimum*(*-weight*) *spanning tree* of a connected, undirected graph?

Finding the Shortest Paths in a Graph

Dijkstra's shortest-paths algorithm (1959) finds shortest (minimal-weight), cycle-free paths (of at most $|V| - 1$ edges) between a given *source* vertex and all the other vertices in a directed graph (V, E) with *non-negative* weights on the edges in E.

Dijkstra's algorithm is another example of a greedy algorithm; it has been shown to indeed compute shortest paths.

It relies on an instance of the *optimal-substructure property*:

> Any shortest path P between two vertices of a graph contains shortest paths between any two vertices of P.

It runs in $O((|V| + |E|) \lg |V|)$ time when using binary heaps, and in $O(|V| \lg |V| + |E|)$ time when using Fibonacci heaps.

Representation Choices for Dijkstra's Algorithm

The algorithm assumes the graph is represented as an array Adj of *adjacency lists*, for $\Theta(1)$ lookup of the neighbours of a vertex.

We are not just interested in the weights of the shortest paths,
but also in actual shortest paths:
the algorithm maintains an array π of *predecessors*, giving
for each vertex v its predecessor $\pi[v]$, which is either a vertex or $\perp$.

It maintains an array d of *distance estimates*, giving
for each vertex v an upper bound $d[v]$
on the weight of a shortest path from the source s to v.

It maintains a set S of vertices whose final shortest-path weights from the source s have already been determined.

It maintains a min-priority queue $Q = V - S$ of vertices, keyed by d.

Dijkstra's Algorithm

$Dijkstra(V, Adj, s)$:
for each vertex $v \in V$ **do**
  $d[v] \leftarrow \infty$
  $\pi[v] \leftarrow \perp$
$d[s] \leftarrow 0$
$S \leftarrow \emptyset$
$Q \leftarrow V$, using the values of d as priorities
while $Q \neq \emptyset$ **do**
$\{$invariant: $d[v]$ is the shortest-path weight from s to v, for all $v \in S\}$
  $u \leftarrow extractMin(Q)$ $\{u$ is estimated closest to s among $Q = V - S\}$
  $S \leftarrow S \cup \{u\}$
  for each vertex $v \in Adj[u]$ **do if** $d[v] > d[u] + w(u, v)$
    then $\{d[v] \leftarrow d[u] + w(u, v)$; update Q; $\pi[v] \leftarrow u\}$

Example for Dijkstra's Algorithm