Graph fundamentals

Paolo Boldi
DSI
LAW (Laboratory for Web Algorithmics)
Università degli Studi di Milan

Graphs: an ubiquitous notion

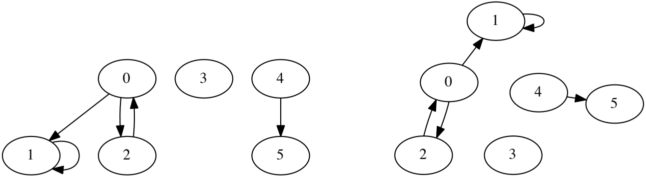
- Graphs appear everywhere: they are an extremely useful formalism!
- Unfortunately: nomenclature and notation is not enough standardized
- The purpose of this lesson is
  - establishing the notation we shall be using in the future
  - singling out some properties that will be of help
  - providing some algorithmic highlight on graphs

Graph

- A *graph* $G = (N_G, A_G)$ is defined by a finite set of *nodes* N_G and by a set of *arcs* $A_G \subseteq N_G \times N_G$
- The subscript G is omitted when clear from the context
- This is sometimes called "network", or "directed graph" or "digraph": I will add the adjective *directed* only when doubts can arise
- Usually $n_G = |N_G|$ and $m_G = |A_G|$. Of course $0 \leq m_G \leq n_G^2$.

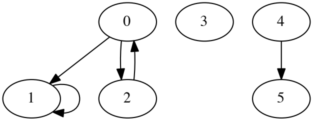
Graph representation

- It is customary to represent graphically a graph, depicting every node with a small circle and every arc (x, y) as a directed arrow from x to y.
- Observe that there is no commitment as to where nodes should be placed (the *embedding* used) or how arcs should be drawn.



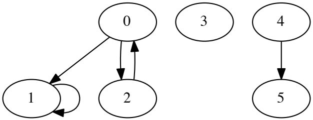
- Drawing graphs “nicely” is by itself an important part of graph theory.

Arcs



- An arc (x, y) is said to *start* (or stem) from x and to *end* on y; x is its *source* and y is its *target*
- We say that (x, y) is incident on x and y.
- y is a *successor* (or *out-neighbor*) of x
- x is a *predecessor* (or *in-neighbor*) of y

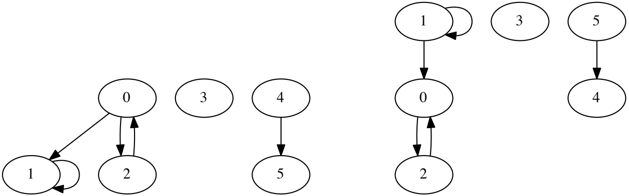
Arcs



- An arc of the form (x, x) is called a *(self-)loop*; graphs without loops are called *loopless* (and may have at most $n(n-1)$ arcs).
- Two arcs of the form (x, y) and (y, x) are called reciprocal of each other
- IMPORTANT: it is impossible to have two parallel arcs with the same source and target (if you need them, you should resort to "multigraphs")

Transpose graphs

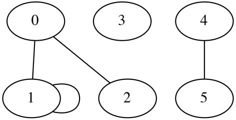
- Given a graph G, define its *transpose* as $G^T = (N_G, A_G^T)$ where $A_G^T = \{(y,x) \mid (x,y) \in A_G\}$.
- Example:



Symmetric graphs

- A graph is G *symmetric* iff $G = G^T$. In symmetric graphs, arcs come in pairs: every arc (x, y) correspond to an arc (y, x).
- The pair of arcs $\{(x, y), (y, x)\}$ is called an *edge* and may be thought of as a *set of (at most) two nodes* $\{x, y\}$.
- This is how most people define *undirected graphs*: we shall actually consider the latter as a synonym of "symmetric (loopless)".

Symmetric graphs



- Edges of an undirected graph can be thought of as elements of $\binom{V}{2}$
- In undirected graphs, nodes are often called *vertices*.
- Notational problem: should m_G denote the number of arcs or the number of edges? I prefer to stick to arcs, and reserve e_G for the number of edges (that is $m_G/2$, in the loopless case)

Neighborhood, degrees

- We let $N_G^+(x)$ be the set of out-neighbors of x; its cardinality, $d^+(x)$, is called the *out-degree* of x
- We let $N_G^-(x)$ be the set of in-neighbors of x; its cardinality, $d^-(x)$, is called the *in-degree* of x
- For undirected graphs, we use $N_G(x)$ and $d_G(x)$

Paths

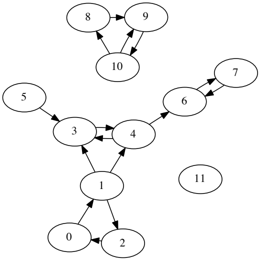
- In a graph G a *path* is a sequence of nodes $\pi = \langle x_0, x_1, \ldots, x_\ell \rangle$ such that $(x_{i-1}, x_i) \in A_G$ for all $i = 1, \ldots, \ell$
- $|\pi| = \ell$ is called the *length* of π
- π is called *simple* iff nodes are all distinct
- We say that π starts from x_0 and ends in x_ℓ, and write $\pi : x_0 \rightsquigarrow x_\ell$
- We say that x_ℓ is *reachable* from x_0 iff there is a path from x_0 to x_ℓ

Cycles

- In a graph G, a *cycle* is a nonempty path of length that starts and ends in the same node.
- It is *simple* if no node (except for the starting/ending one) is repeated.
- A graph is *acyclic* iff it contains no simple cycle.
- For undirected graph, the request is that a cycle be of length ≥ 3.

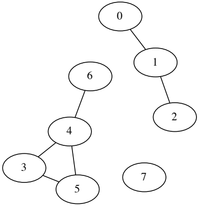
Connected components

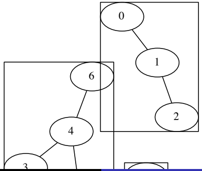
- For a given graph G, the relation $\leadsto$ (reachability) is a pre-order: it is reflexive and transitive.
- Its associated equivalence relation $x \sim y$ is defined by $x \leadsto y$ and $y \leadsto x$.
- The equivalence classes of $\sim$ are called the *(strongly) connected components* of the graph.
- $\leadsto$ is a partial order on the components (hence: it is acyclic).



Connected components

- In an undirected graph, there are *no edges* between different components!





Graph traversal (visit)

- A general technique that is used to do "something" with the nodes of a graph
- Traversals consider all the nodes of the graph exactly once, in some order
- The order depends on the kind of visit
- At any moment, all nodes are classified into:
  - unknown (white)
  - frontier: known but unvisited (grey)
  - visited (black)

Visit: 1) initialization

- all nodes are initially white
- the frontier is empty

$\mathrm{init}()$**: Initialization**
$St[-] \leftarrow \mathrm{white}$
$F \leftarrow \emptyset$

Visit: 2) cycle

- provided that some nodes are used as seed...
- ...i.e., initially set as grey put in the frontier

$\mathrm{visit}()$: **Perform a visit cycle**

while $F \neq \emptyset$ **do**
    $x \leftarrow F.\mathrm{pick}()$
    $\mathrm{visit}(x)$
    $St[x] \leftarrow \mathrm{black}$
    for $y \in N^+(x)$ **do**
        if $S[y] = \mathrm{white}$ **then**
            $St[y] \leftarrow \mathrm{grey}$
            $F.\mathrm{add}(y)$
        end if
    end for
end while

Visit: 3) full visit

- performs a visit starting from the first white nodes
- at the end, all nodes are black

init()
for $x \in N$ **do**
  if $St[x] = \text{white}$ **then**
    for $x \in S$ **do**
      $St[x] \leftarrow \text{grey}$
      $F.\text{add}(x)$
    end for
    visit()
  end if
end for

Types of traversals

The type of traversal changes depending on the data structure used for the frontier (i.e., implementation of the *pick* method).

- If F is a *stack* (LIFO), the visit is a *depth-first search* (DFS)
- If F is a *queue* (FIFO), the visit is a *breadth-first search* (BFS)
- Note: DFS can also be implemented with an implicit stack, using recursion.

Application example: shortest paths

Finding shortest paths (in unweighted graphs) from a given source s.

- Perform a BFS from s.
- When a node enters the frontier, the node under visit is marked as his *parent*: it is the next-to-last node in a shortest path from s.
- Works perfectly even for directed graphs.

Application example: components (undirected)

An example of application is finding the connected components of an *undirected graph*

- Each single visit touches all (and only) the nodes of a single component
- In this case, the visit order is irrelevant: any frontier datastructure will do the job!

Graph representations

The most trivial representation of a graph is its *adjacency matrix*:

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

- Requires n^2 bits: very expensive for sparse graphs (i.e., when $m \ll n^2$)
- Obtaining the successors of a node requires time $O(n)$
- Knowing if (x, y) is an arc requires time $O(1)$

Graph representations

Alternatively, one can use *adjacency lists*:

$$
\begin{aligned}
L[0] &= \langle 2, 3 \rangle \\
L[1] &= \langle 0, 3, 4 \rangle \\
L[2] &= \langle 1, 2 \rangle \\
L[3] &= \langle \rangle \\
L[4] &= \langle 0, 1, 3 \rangle
\end{aligned}
$$

- Requires $m \log n$ bits, plus the space for the offsets of every list
- Obtaining the successors of node x requires time $O(|N^+(x)|)$ (amortized constant)
- Knowing if (x, y) is an arc requires time $O(|N^+(x)|)$ (maximum degree, in the worst case, or n)