Introduction to Graphs

Summary

- Definition: Graph
- Related Definitions
- Applications
- Graph representation
- Graph visits
Definition: Graph

Introduction to Graphs
Definition: **Graph**

- A **graph** is a collection of **points** and **lines** connecting some (possibly empty) subset of them.
- The points of a graph are most commonly known as **graph vertices**, but may also be called “nodes” or simply “points.”
- The lines connecting the vertices of a graph are most commonly known as **graph edges**, but may also be called “arcs” or “lines.”

http://mathworld.wolfram.com/
What's in a name?

"MATHEMATICS IS THE ART OF GIVING THE SAME NAME TO DIFFERENT THINGS."
JULES HENRI POINCARE (1854–1912)

http://spikedmath.com/382.html
Big warning: Graph ≠ Graph ≠ Graph

Graph (plot)
(italiano: grafico)

Graph (maths)
(italiano: grafo)

Graph (chart)
(italiano: grafico)
The study of graphs is known as **graph theory**, and was first systematically investigated by D. König in the 1930s.

Euler’s proof about the *walk across all seven bridges of Königsberg* (1736), now known as the *Königsberg bridge problem*, is a famous precursor to graph theory.

In fact, the study of various sorts of paths in graphs has many applications in real-world problems.
Königsberg Bridge Problem

- Can the 7 bridges of the city of Königsberg over the river Preger all be traversed in a single trip without doubling back, with the additional requirement that the trip ends in the same place it began?

Today: Kaliningrad, Russia
Königsberg Bridge Problem

- Can the 7 bridges of the city of Königsberg over the river Preger all be traversed in a single trip without doubling back, with the additional requirement that the trip ends in the same place it began?
Unless…

http://spikedmath.com/541.html
Types of graphs: edge cardinality

- **Simple graph:**
  - At most one edge (i.e., either one edge or no edges) may connect any two vertices

- **Multigraph:**
  - Multiple edges are allowed between vertices

- **Loops:**
  - Edge between a vertex and itself

- **Pseudograph:**
  - Multigraph with loops
Types of graphs: edge direction

- Undirected
- Oriented
  - Edges have **one** direction (indicated by arrow)
- Directed
  - Edges may have **one or two** directions
- Network
  - Oriented graph with weighted edges
Types of graphs: labeling

- **Labels**
  - None
  - On Vertices
  - On Edges

- **Groups (=colors)**
  - Of Vertices
    - no edge connects two identically colored vertices
  - Of Edges
    - adjacent edges must receive different colors
  - Of both
Directed and Oriented graphs

A Directed Graph (*di-graph*) $G$ is a pair $(V,E)$, where

- $V$ is a (finite) set of vertices
- $E$ is a (finite) set of edges, that identify a binary relationship over $V$
  - $E \subseteq V \times V$
Example
Example

Loop
Example

V = \{1, 2, 3, 4, 5, 6\}

E = \{(1, 2), (2, 2), (2, 5), (5, 4), (4, 5), (4, 1), (2, 4), (6, 3)\}
Undirected graph

- Ad **Undirected** Graph is still represented as a couple $G=(V,E)$, but the set $E$ is made of **non-ordered pairs** of vertices
Example

\[ V = \{1, 2, 3, 4, 5, 6\} \]
\[ E = \{(1, 2), (2, 5), (5, 1), (6, 3)\} \]
Example

\[ V = \{1,2,3,4,5,6\} \]
\[ E = \{(1,2), (2,5), (5,1), (6,3)\} \]

Edge (1,5) adjacent (or incident) to vertices 1 and 5

Vertex 5 is adjacent to vertices 1 and 2

Vertex 4 is isolated
Related Definitions

Introduction to Graphs
Degree

- In an *undirected* graph,
  - the *degree* of a vertex is the number of incident edges
- In a *directed* graph
  - The *in-degree* is the number of incoming edges
  - The *out-degree* is the number of departing edges
  - The *degree* is the sum of in-degree and out-degree
- A vertex with degree 0 is *isolated*
Degree

1

2

3

4

5

6

0

2

1

2

1

2

0

2

1
Degree

1
In: 1
Out: 1

2
In: 1 or 2
Out: 2 or 3

3
In: 1
Out: 0

4
In: 2
Out: 2

5
In: 2
Out: 1

6
A path on a graph $G=(V,E)$ also called a trail, is a sequence $\{v_1, v_2, \ldots, v_n\}$ such that:
- $v_1, \ldots, v_n$ are vertices: $v_i \in V$
- $(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n)$ are graph edges: $(v_{i-1}, v_i) \in E$
- $v_i$ are distinct (for “simple” paths).

The length of a path is the number of edges $(n-1)$

If there exist a path between $v_A$ and $v_B$ we say that $v_B$ is reachable from $v_A$
Example

Path = \{ 1, 2, 5 \}
Length = 2
Cycles

- A cycle is a path where $v_1 = v_n$
- A graph with no cycles is said acyclic
Example

Path = \{ 1, 2, 5, 1 \}
Length = 3
Reachability (Undirected)

- An undirected graph is **connected** if, for every couple of vertices, there is a path connecting them.
- The connected sub-graph of maximum size are called **connected components**.
- A connected graph has exactly one connected component.
Connected components

The graph is not connected.
Connected components =
\{ 4 \}, \{ 1, 2, 5 \}, \{ 3, 6 \}
Reachability (Directed)

- A directed graph is **strongly connected** if, for every ordered pair of vertices \((v, v')\), there exists at least one path connecting \(v\) to \(v'\)
Example

The graph is **strongly connected**
Example

The graph is not strongly connected
A graph is complete if, for every pair of vertices, there is an edge connecting them (they are adjacent).

Symbol: $K_n$
Complete graph: edges

- In a **complete** graph with $n$ vertices, the number of **edges** is
  - $n(n-1)$, if the graph is directed
  - $n(n-1)/2$, if the graph is undirected
- If self-loops are allowed, then
  - $n^2$ for directed graphs
  - $n(n-1)$ for undirected graphs
Density

- The density of a graph $G=(V,E)$ is the ratio of the number of edges to the total number of edges.

$$d = \frac{|E(G)|}{|E(K_{|V(G)|})|}$$
Esempio

Density = 0.5
Existing: 3 edges
Total: 6 possible edges

1 — 2

4 — 3
Trees and Forests

- An undirected acyclic graph is called **forest**
- An undirected acyclic connected graph is called **tree**
Example

Tree
Example

Forest
Example

This is not a tree nor a forest
(it contains a cycle)
Rooted trees

- In a tree, a special node may be singled out
- This node is called the “root” of the tree
- Any node of a tree can be the root
Tree (implicit) ordering

- The root node of a tree **induces an ordering** of the nodes
- The root is the “ancestor” of all other nodes/vertices
  - “children” are “away from the root”
  - “parents” are “towards the root”
- The root is the only node without parents
- All other nodes have exactly one parent
- The furthermost (children-of-children-of-children…) nodes are “leaves”
Example

Rooted Tree
Example

Rooted Tree
Weighted graphs

- A weighted graph is a graph in which each branch (edge) is given a numerical weight.
- A weighted graph is therefore a special type of labeled graph in which the labels are numbers (which are usually taken to be positive).
Applications

Introduction to Graphs
Graph applications

- Graphs are everywhere
  - Facebook friends (and posts, and ‘likes’)
  - Football tournaments (complete subgraphs + binary tree)
  - Google search index (V=page, E=link, w=pagerank)
  - Web analytics (site structure, visitor paths)
  - Car navigation (GPS)
  - Market Matching
Market matching

- $H = \text{Houses (1, 2, 3, 4)}$
- $B = \text{Buyers (a, b, c, d)}$
- $V = H \cup B$
- Edges: $(h, b) \in E$ if $b$ would like to buy $h$
- Problem: can all houses be sold and all buyers be satisfied?
- Variant: if the graph is weighted with a purchase offer, what is the most convenient solution?
- Variant: consider a ‘penalty’ for unsold items

This graph is called “bipartite”: $H \cap B = \emptyset$
Connecting cities

- We have a water reservoir
- We need to serve many cities
  - Directly or indirectly
- What is the most efficient set of inter-city water connections?

- Also for telephony, gas, electricity, …

We are searching for the “minimum spanning tree”
Google Analytics (Visitors Flow)
Customer behavior

User actions encoded as frequencies
Street navigation

TSP: The traveling salesman problem

We must find a “Hamiltonian cycle”
Train maps
Chemistry (Protein folding)
Facebook friends
Flow chart
Graph representation

Representing and visiting graphs
Representing graphs

**List structures**

- **Adjacency list**
  - Each vertex has a list of which vertices it is adjacent to.
  - For undirected graphs, information is duplicated.

- **Incidence list**
  - Each vertex has a list of ‘edge’ objects.
  - Edges are represented by a pair (a tuple if directed) of vertices (that the edge connects) and possibly weight and other data.

**Matrix structures**

- **Adjacency matrix**
  - \( A = |V| \times |V| \) matrix of Booleans or integers.
  - If there is an edge from a vertex \( v \) to a vertex \( v' \), then the element \( A[v,v'] \) is 1, otherwise 0.

- **Incidence matrix**
  - \( IM = |V| \times |E| \) matrix of integers.
  - \( IM[v,e] = 1 \) (incident), 0 (not incident).
  - For directed graphs, may be -1 (out) and +1 (in).
Example

Adjacency matrix

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>d</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>e</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>f</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>g</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>h</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>i</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Adjacency list

- a → b → c → d → e
- b → a → c → e
- c → b → d → e
- d → b → c → e → f
- e → b → c → d → f
- f → d → e
- g → h → i
- h → g → i
- i → g → h
Adjacency list (undirected graph)
Adjacency list (undirected graph)

Undirected ==> All edges are represented twice
Adjacency list (un-connected graph)

Un-connected graph, same rules

L_1 = [2, 8]  
L_2 = [1, 6]  
L_3 = [4]  
L_4 = [3]  
L_5 = [6, 8]  
L_6 = [2, 5, 8]  
L_7 = []  
L_8 = [1, 5, 6]
Adjacency list (directed graph)
Adjacency list

![Graph Diagram]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>6</td>
<td>4</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>3</td>
<td>7</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>9</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>8</td>
<td>6</td>
<td>7</td>
<td>11</td>
<td>10</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Adjacency matrix (undirected graph)

Undirected => symmetric matrix

No self-loops: zero diagonal
Adjacency matrix (undirected graph)

Undirected => symmetric matrix

50% of memory can be saved
Adjacency matrix (directed graph)

From-vertex (out-edges)

To-vertex (in-edges)

Self-loops in the diagonal
Adjacency matrix (weighted graph)

Values = edge weight

1 2 3 4 5
1 0 2 0 0 0
2 0 0 3 5 0
3 0 0 1 0 0
4 0 0 0 0 2
5 1 3 0 0 0
Adjacency matrix

\[
\begin{array}{cccccccccccc}
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
2 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
4 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
5 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
6 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\
7 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
8 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
9 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\
10 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\
11 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
\end{array}
\]
Incidence matrix (undirected graph)

Vertices v1…v5

Edges e1…e7

# of “ones” in a row = vertex degree

Exactly 2 “ones” in every column

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
**Incidence matrix (directed graph)**

### Vertices
- $v_1...v_5$

### Edges
- $e_1...e_7$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-1</td>
<td>+1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>+1</td>
<td>0</td>
<td>+1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>+1</td>
<td>-1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>+1</td>
<td>-1</td>
<td>0</td>
<td>+1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>+1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **indegree** = count(+1)
- **outdegree** = count(-1)

- Self loops can't be represented

- -1: exit
- +1: enter
### Complexity & trade-offs

<table>
<thead>
<tr>
<th></th>
<th>Adjacency List</th>
<th>Adjacency Matrix</th>
<th>Incidence Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Space</strong></td>
<td>$O(</td>
<td>V</td>
<td>+</td>
</tr>
<tr>
<td><strong>Space</strong></td>
<td>For sparse graphs $</td>
<td>E</td>
<td>\ll</td>
</tr>
<tr>
<td><strong>Check edge</strong></td>
<td>$O(1 + \text{deg}(v))$</td>
<td>$O(1)$</td>
<td>$O(</td>
</tr>
<tr>
<td><strong>Find all adjacent</strong></td>
<td>$O(1 + \text{deg}(v))$</td>
<td>$O(</td>
<td>V</td>
</tr>
</tbody>
</table>
Graph visits

Representing and visiting graphs
Visit Algorithms

- **Visit =**
  - Systematic exploration of a graph
  - Starting from a ‘source’ vertex
  - Reaching all reachable vertices

- **Main strategies**
  - Breadth-first visit (“in ampiezza”)
  - Depth-first visit (“in profondità”)
Breadth-First Visit

- Also called Breadth-first search (BFV or BFS)
- All reachable vertices are visited “by levels”
  
  - $L$ – level of the visit
  - $S_L$ – set of vertices in level $L$
  - $L=0$, $S_0=\{ \text{v}_{\text{source}} \}$
  - Repeat while $S_L$ is not empty:
    
    - $S_{L+1}$ = set of all vertices:
      - not visited yet, and
      - adjacent to at least one vertex in $S_L$
    
    - $L=L+1$
Example

Source = s
L = 0
S₀ = {s}
Example

L = 1
S_0 = \{s\}
S_1 = \{r, w\}
Example

$L = 2$

$S_1 = \{r, w\}$

$S_2 = \{v, t, x\}$
Example

$L = 3$
$S_2 = \{v, t, x\}$
$S_3 = \{u, y\}$
BFS Tree

- The result of a BFV identifies a “visit tree” in the graph:
  - The tree root is the source vertex
  - Tree nodes are all graph vertices
    - (in the same connected component of the source)
  - Tree are a subset of graph edges
    - Those edges that have been used to “discover” new vertices.
BFS Tree
Minimum (shortest) paths

- Shortest path: the minimum number of edges on any path between two vertices
- The BFS procedure computes all minimum paths for all vertices, starting from the source vertex
- NB: unweighted graph: path length = number of edges
Depth First Visit

- Also called Depth-first search (DFV or DFS)
- Opposite approach to BFS
- At every step, visit one (yet unvisited) vertex, adjacent to the last visited one
- If no such vertex exist, go back one step to the previously visited vertex
- Lends itself to recursive implementation
  - Similar to tree visit procedures
DFS Algorithm

- **DFS(Vertex v)**
  - For all ( w : adjacent_to(v) )
    - If( not visited (w) )
      - Visit (w)
      - DFS(w)

- Start with: DFS(source)
Example

Source = s
Example

Source = s
Visit r
Example

Source = s
Visit r
Visit v
Example

Source = s
Back to r
Back to s
Visit w
Example

Source = s
Visit w
Visit t

Diagram with nodes labeled r, s, t, u, v, w, x, y.
Example

Source = s
Visit w
Visit t
Visit u
Example

Source = s
Visit w
Visit t
Visit u
Visit y
Example

Source = s
Visit w
Visit t
Visit u
Visit y
Visit x
Example

Source = s
Back to y
Back to u
Back to t
Back to w

DFS tree
Edge classification

- In an directed graph, after a DFS visit, all edges fall in one of these 4 categories:
  - **T**: *Tree* edges (belonging to the DFS tree)
  - **B**: *Back* edges (not in T, and connect a vertex to one of its ancestors)
  - **F**: *Forward* edges (not in T and B, and connect a vertex to one of its descendants)
  - **C**: *Cross* edges (all remaining edges)
Example

Directed graph

\[ \text{Directed graph} \]
Example

DFS visit
(sources: s, t)
Example

Edge classification

Graph with nodes labeled as Y, Z, S, T, X, W, V, U and edges labeled with T, B, F, C.
Theorem:

A directed graph is acyclic if and only if a depth-first visit does not produce any B edge
Visits have linear complexity in the graph size
- BFS: $O(V+E)$
- DFS: $\Theta(V+E)$

N.B. for dense graphs, $E = O(V^2)$
Resources

- Basic Graph Theory with Applications to Economics [http://www.isid.ac.in/~dmishra/mpdoc/lecgraph.pdf](http://www.isid.ac.in/~dmishra/mpdoc/lecgraph.pdf)
Resources

- Open Data Structures (in Java), Pat Morin, http://opendatastructures.org/
Licenza d’uso

- Queste diapositive sono distribuite con licenza Creative Commons “Attribuzione - Non commerciale - Condividi allo stesso modo (CC BY-NC-SA)”

- Sei libero:
  - di riprodurre, distribuire, comunicare al pubblico, esporre in pubblico, rappresentare, eseguire e recitare quest’opera
  - di modificare quest'opera

- Alle seguenti condizioni:
  - Attribuzione — Devi attribuire la paternità dell'opera agli autori originali e in modo tale da non suggerire che essi avallino te o il modo in cui tu usi l'opera.
  - Non commerciale — Non puoi usare quest'opera per fini commerciali.
  - Condividi allo stesso modo — Se alteri o trasformi quest'opera, o se la usi per crearne un'altra, puoi distribuire l'opera risultante solo con una licenza identica o equivalente a questa.

- [http://creativecommons.org/licenses/by-nc-sa/3.0/](http://creativecommons.org/licenses/by-nc-sa/3.0/)