Lecture: Analysis of Algorithms (CS583 - 004)

Amarda Shehu

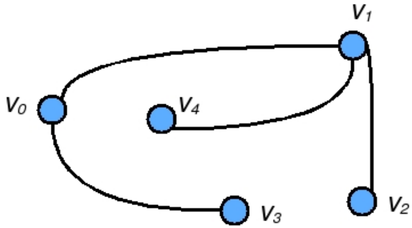
Spring 2019

1. Graphs
   - Definition of a Graph
   - Omnipresence of Graphs
2. Graph Representations
   - Adjacency List Representation
   - Adjacency Matrix Representation
   - Alternative Graph Representations
3. Solving Problems with Graph Algorithms

What is a Graph?

Graph $G = (V, E)$

- V : set of vertices
- E : set of edges consisting of pairs of vertices from V

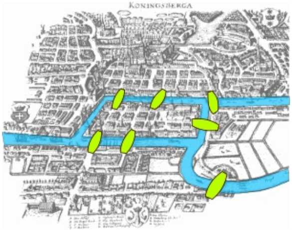


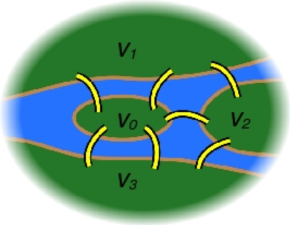
$$V = \{v_0, v_1, v_2, v_3, v_4\}$$
$$E = \{(v_0, v_1), (v_0, v_3), (v_1, v_2), (v_1, v_4)\}$$

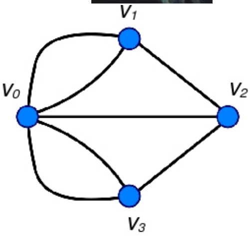
First Graph Problem

Seven Bridges of Koenigsberg [1736]:

Find a route that crosses each bridge exactly once.
Posed by Leonard Euler [1707 - 1783].



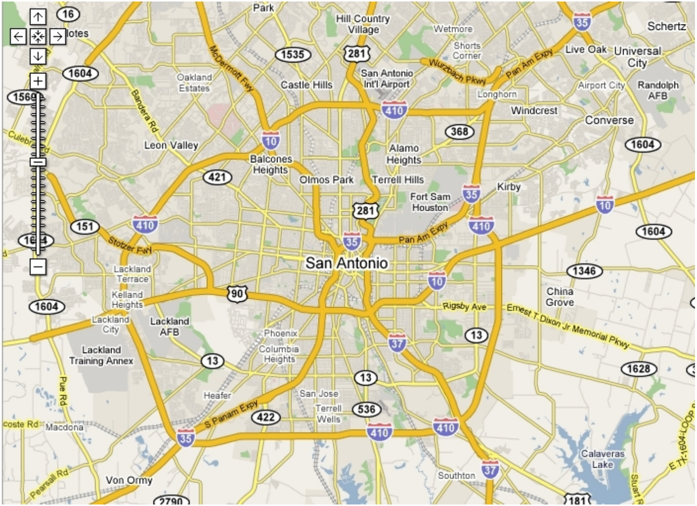




modified from wikipedia

What is the minimum number of bridges that need to be added so that there exists a route that crosses each bridge exactly once?

Road Networks as Graphs



Airline Routes as Graphs

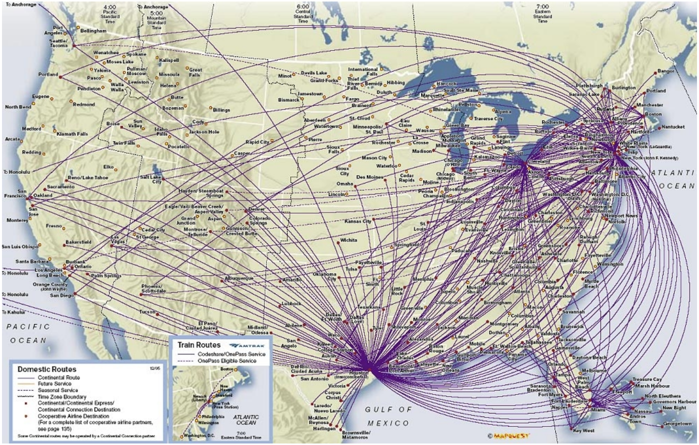


Figure: http://www.airlineroutemaps.com/

Social Networks as Graphs

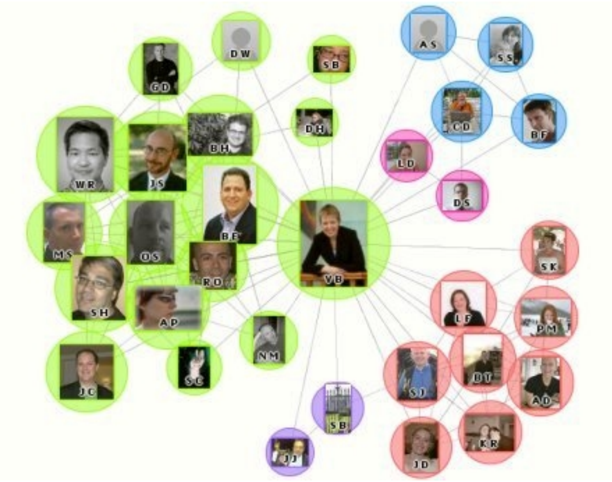


Figure: http://hbr.idnet.net/images/

The Internet as a Graph

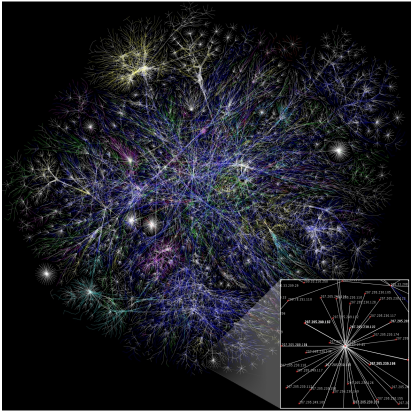


Figure: Credit: Matt Britt

Visualization of the various routes through a portion of the Internet.

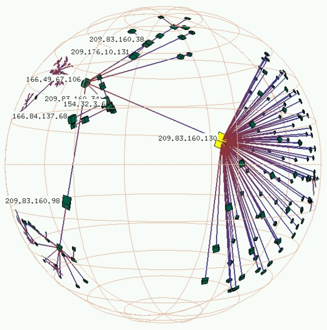


Figure: Credit: Young Hyun, CAIDA

Websites as Graphs

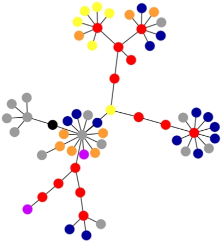


Figure: http://www.google.com

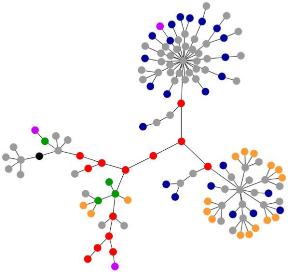


Figure: http://www.cs.gmu.edu

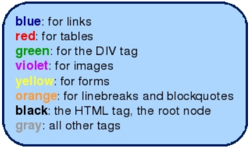


Figure: Credit: Marcel Salathe http://www.aharef.info

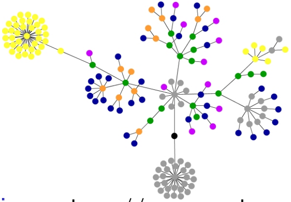


Figure: http://www.apple.com

Biological Networks as Graphs

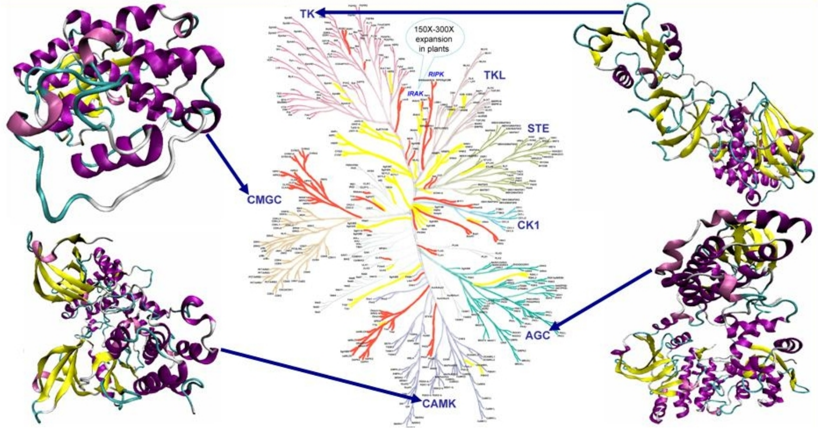


Figure: Adapted from A. Barabasi, University of Notre Dame

Applications of Graphs

- Compilers
- Databases
- Neural Networks
- Machine Learning
- Artificial Intelligence
- Robotics
- Computational Biology
- ...

Formal Definition of a Graph

- A graph $G = (V, E)$ is a pair consisting of:
- a set V of vertices (or nodes)
- a set $E \subseteq V \times V$ of edges (or arcs)
  - edge $e_i \in E$ is a pair (u, v) connecting vertices u and v

A graph $G = (V, E)$ is:

- directed (referred to as a digraph) if E is a set of ordered pairs of vertices. The edges here are often referred to as directed edges or arrows.
- undirected if E is a set of unordered pairs of vertices.
- weighted if there are weights associated with the edges.

Illustrations of Types of Graphs

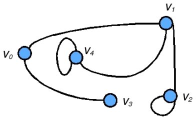


Figure: undirected graph

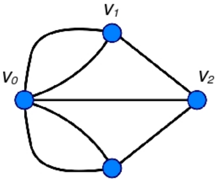


Figure: multigraph

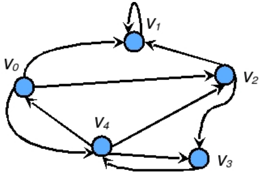


Figure: directed graph

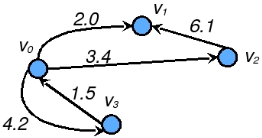


Figure: weighted graph

General Definition of a Graph

In a graph $G = (V, E)$:

- E may be a set of unorderered pairs of vertices not necessarily distinct. More than one edge can connect two vertices.
- An edge in E may connect more than two vertices.
- These graphs are referred to as multigraphs or pseudo-graphs.

Focusing on Simple Graphs

Simple Graphs

- A simple graph, or a strict graph, is an unweighted, undirected graph containing no loops or multiple edges
- Given that $E \subseteq V \times V$, $|E| \in O(|V|^2)$.
- If a graph is connected, $|E| \geq |V| - 1$
- Combining the two, show that $lg(|E|) \in \theta(lg(|V|))$

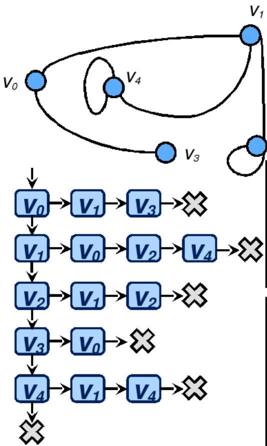
More Definitions, Conventions, Nomenclature

- A **subgraph** H of $G = (V, E)$ is $H = (V_1, E_1)$ where $V_1 \subseteq V$ and $E_1 \subseteq E$, where $\forall e = (k, j) \in E_1$, $k, j \in V_1$.
- A **path** is a sequence of vertices, where each pair of successive vertices is connected by an edge.
- The **length of the path** is the number of edges in the path.
- A **simple path** contains unique vertices.
- A **cycle** is a simple path with the same first and last vertex.
- Two vertices are **adjacent** if they are connected by an edge.
- The **neighbors** of a vertex are all the vertices adjacent to it.
- The **degree** of a vertex is the number of its neighbors.
- A graph is **connected** if $\exists$ a path between every pair of vertices.
- A **tree** is a connected graph with no cycles.

Graph Representations

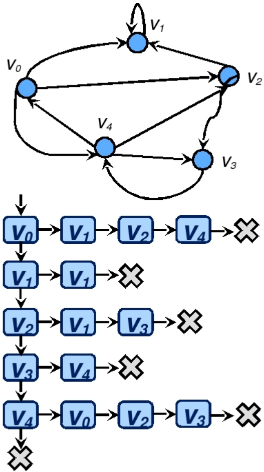
- A graph can be represented as an **adjacency list**.
- A graph can be represented as an **adjacency matrix**.

Adjacency List Representation




```
struct elist
{
 int vto;
 struct elist *next;
};
```

```
struct vlist
{
 int v;
 elist *edges;
 struct vlist *next;
};
```



Basic Graph Functionality

Function	Adjacency List
find(v)	$O(\lvert V\rvert)$
hasVertex(v)	$O(\mathrm{find(v)})$
hasEdge(v_i, v_j)	$O(\mathrm{find(v_i)} + \mathrm{deg(v_i)})$
insertVertex(v)	$O(1)$
insertEdge(v_i, v_j)	$O(\mathrm{find(v_i)})$
removeVertex(v)	$O(\lvert V\rvert + \lvert E\rvert)$
removeEdge(v_i, v_j)	$O(\mathrm{find(v_i)} + \mathrm{deg(v_i)})$
outEdges(v)	$O(\mathrm{find(v)} + \mathrm{deg(v)})$
inEdges(v)	$O(\lvert V\rvert + \lvert E\rvert)$
overall memory	$O(\lvert V\rvert + \lvert E\rvert)$

In undirected graphs:
$|\mathrm{elist[v]}| =$ degree(v).

In digraphs:
$|\mathrm{elist[v]}| =$ out-degree(v).

Handshaking Lemma: $\sum_{v \in V} |\mathrm{elist(v)}| = 2|\mathrm{E}|$ for undirected graphs. $O(|V| + |E|)$ storage $\Rightarrow$ **sparse** representation.

More on Implementation of Adjacency-list Representation

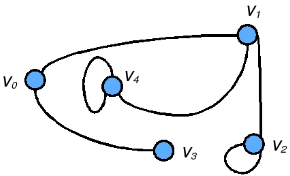
The adjacency list of a vertex can be implemented as a linked list
The list of vertices themselves can be implemented using:

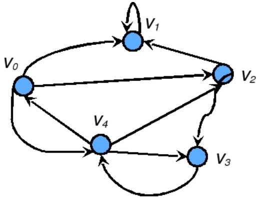
- A linked list
- A binary search tree
- A hash table

In a standard implementation, each edge list has two fields, a data field and a pointer:

- The data field contains adjacent vertex name and edge information
- The pointer points to next adjacent vertex

Adjacency Matrix Representation





$$M[i][j] = 1 \quad \text{iff} \quad (v_i, v_j) \in E$$

M	v_0	v_1	v_2	v_3	v_4
v_0	0	1	0	1	0
v_1		0	1	0	1
v_2			1	0	0
v_3				0	0
v_4					1

```
bool M[n][n];
```

```
bool **M;
```

```
using namespace std;

vector < vector<bool> > M;
```

M	v_0	v_1	v_2	v_3	v_4
v_0	0	1	1	0	1
v_1	0	1	0	0	0
v_2	0	1	0	1	0
v_3	0	0	0	0	1
v_4	1	0	1	1	0

Basic Graph Functionality

Function	Adjacency Matrix
find(v)	$O(1)$
hasVertex(v)	$O(1)$
hasEdge(v_i, v_j)	$O(1)$
insertVertex(v)	$O(\|V\|^2)$
insertEdge(v_i, v_j)	$O(1)$
removeVertex(v)	$O(\|V\|^2)$
removeEdge(v_i, v_j)	$O(1)$
outEdges(v)	$O(\|V\|)$
inEdges(v)	$O(\|V\|)$
overall memory	$O(\|V\|^2)$

$O(|V|^2)$ storage $\Rightarrow$ **dense** representation.

Comparing The Two Representations

Function	Adjacency List	Adjacency Matrix
find(v)	$O(\|V\|)$	$O(1)$
hasVertex(v)	$O(\mathrm{find(v)})$	$O(1)$
hasEdge(v_i, v_j)	$O(\mathrm{find(v_i) + deg(v_i)})$	$O(1)$
insertVertex(v)	$O(1)$	$O(\|V\|^2)$
insertEdge(v_i, v_j)	$O(\mathrm{find(v_i)})$	$O(1)$
removeVertex(v)	$O(\|V\| + \|E\|)$	$O(\|V\|^2)$
removeEdge(v_i, v_j)	$O(\mathrm{find(v_i) + deg(v_i)})$	$O(1)$
outEdges(v)	$O(\mathrm{find(v) + deg(v)})$	$O(\|V\|)$
inEdges(v)	$O(\|V\| + \|E\|)$	$O(\|V\|)$
overall memory	$O(\|V\| + \|E\|)$	$O(\|V\|^2)$

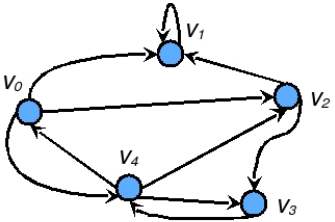
Alternative Graph Representations

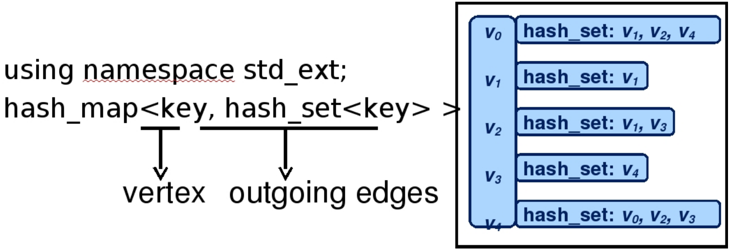
HashMap

Fast to query	[hasVertex, hasEdge]	$O(1)$
Fast to scan	[outEdges]	$O(\|V\|)$
Fast to insert	[insertVertex, insertEdge]	$O(1)$
Fast to remove	[removeEdge]	$O(1)$

Graph Representation: Hash Map

- Vertex set as a hash map
  - key: vertex
  - data: outgoing edges
- Outgoing edges of each vertex as a hash set





Comparing The Three Representations

Function	Adj. List	Adj. Matrix	Hash Map
find(v)	$O(\|V\|)$	$O(1)$	$O(1)$
hasVertex(v)	$O(\|V\|)$	$O(1)$	$O(1)$
hasEdge(v_i, v_j)	$O(\|V\| + \mathrm{deg(v_i)})$	$O(1)$	$O(1)$
insertVertex(v)	$O(1)$	$O(\|V\|^2)$	$O(1)$
insertEdge(v_i, v_j)	$O(\|V\|)$	$O(1)$	$O(1)$
removeVertex(v)	$O(\|V\| + \|E\|)$	$O(\|V\|^2)$	$O(\|V\|)$
removeEdge(v_i, v_j)	$O(\|V\| + \mathrm{deg(v_i)})$	$O(1)$	$O(1)$
outEdges(v)	$O(\|V\| + \mathrm{deg(v)})$	$O(\|V\|)$	$O(\mathrm{deg(v)})$
inEdges(v)	$O(\|V\| + \|E\|)$	$O(\|V\|)$	$O(\|V\|)$
overall memory	$O(\|V\| + \|E\|)$	$O(\|V\|^2)$	linear-quadratic

Graph modeling: Problem Solving with Graph Algorithms

- Identify the vertices and the edges in your problem formulation
- Identify the objective of the problem
- State this objective in graph terms
- Implementation:
  - Construct the graph from the input instance
  - Run the suitable graph algorithm on the graph
  - Convert the output into a suitable/required format