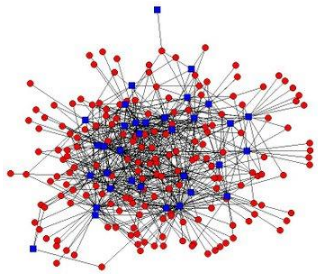
CSE 421: Algorithms

Winter 2014

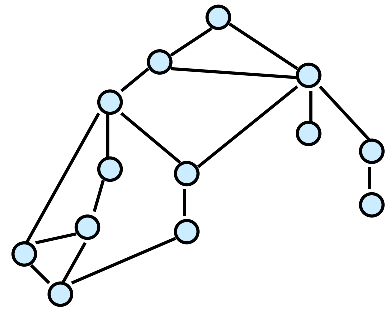
Lecture 4: Graphs and graph traversal

Reading: Sections 3.1-3.2



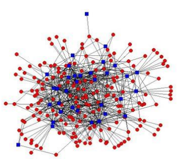
undirected graphs

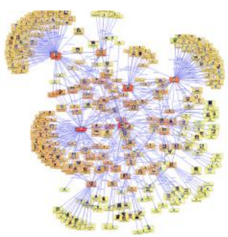
Mathematically, a graph is a pair $G = (V, E)$ of *vertices* (V) and *edges* (E). The edges are simply *unordered* pairs of vertices, i.e. $\{u, v\}$ for $u, v \in V$.

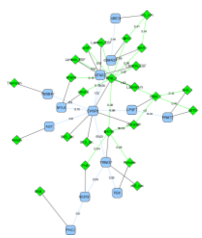


undirected graphs

Graphs can be used to model all sorts of things:
Networks (computer, social, transportation), similarity (e.g. proteins, genes, Amazon users, web pages, …).
Anything with pairwise relationships.

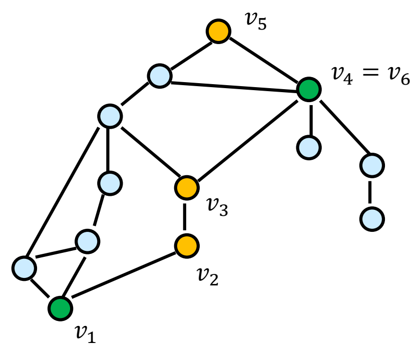






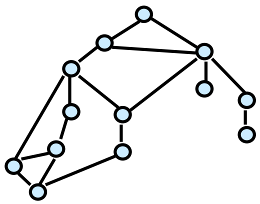
paths

A *path* in a graph is a sequence of nodes $v_1, v_2, \ldots, v_k$ such that consecutive pairs $\{v_i, v_{\{i+1\}}\}$ are connected by an edge.

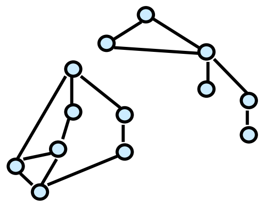


connectivity

An undirected graph is *connected* if every pair of vertices is connected by some path.



connected



not connected

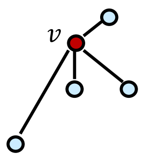
exercise!

Prove that every connected graph with n vertices has at least $n - 1$ edges.

handshaking lemma

Let's look at some simple facts about graphs.

Definition: The *degree* of a vertex v is the number of edges touching v.



$\deg(v) = 4$

Theorem: For any undirected graph $G = (V, E)$,

$$\sum_{v \in V} \deg(v) = 2|E|$$

handshaking lemma

Theorem: For any undirected graph $G = (V, E)$,

$$\sum_{v \in V} \deg(v) = 2|E|$$

Proof:

The LHS counts every edge twice (once from each endpoint).

handshaking lemma

Theorem: For any undirected graph $G = (V, E)$,

$$\sum_{v \in V} \deg(v) = 2|E|$$

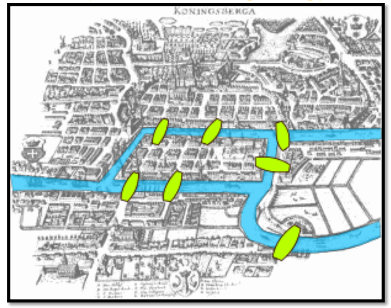
Proof:

The LHS counts every edge twice (once from each endpoint).

Consequence: Every graph has an even number of odd degree vertices. (Why?)

euler tours

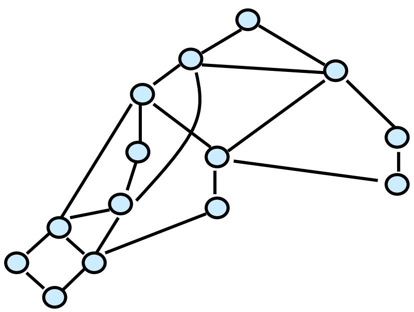
(Glorious Konigsberg)



Euler: Is it possible to walk over each bridge exactly once and then return to the starting point?

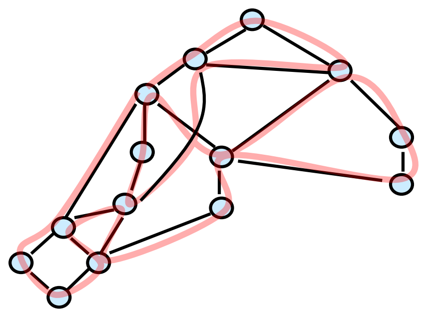
eulerian graphs

A graph is *Eulerian* if there exists a tour that crosses every edge exactly once.



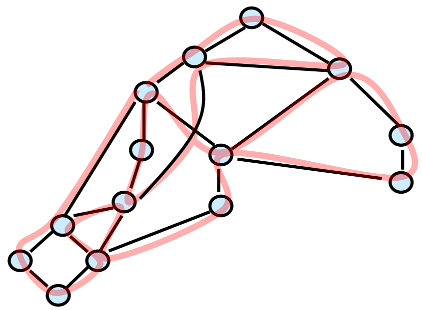
eulerian graphs

A graph is *Eulerian* if there exists a tour that crosses every edge exactly once.



eulerian graphs

Theorem: An undirected graph is Eulerian if and only if it is connected and every vertex has even degree!



eulerian graphs

Theorem: An undirected graph is Eulerian if and only if every vertex has even degree!

Proof.

Easier direction: Eulerian → connected and all degrees even. Why?

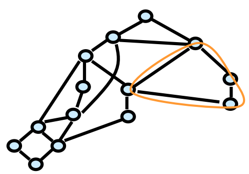
eulerian graphs

Theorem: An undirected graph is Eulerian if and only if every vertex has even degree!

Proof.

Harder direction: connected and all degrees → Eulerian

Strategy: Find a simple cycle (no vertex repeated) in the graph. Then remove it and induct!



eulerian graphs

Theorem: An undirected graph is Eulerian if and only if every vertex has even degree!

Proof.

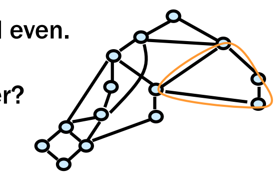
Harder direction: connected and all degrees → Eulerian

Strategy: Find a simple cycle (no vertex repeated) in the graph. Then remove it and induct!

After cycle removed, degrees still even.

Why can we patch cycles together?

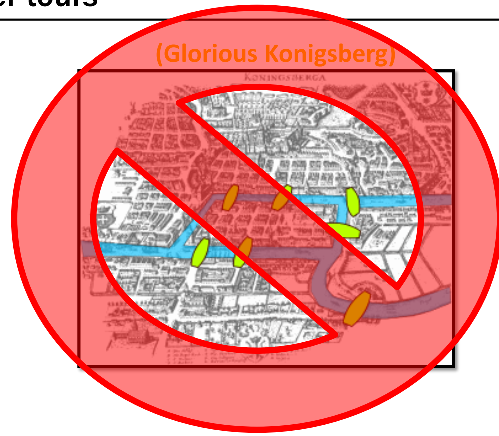
Base case?



exercise!

Prove that every graph with all degrees even contains a simple cycle (i.e. with no vertices repeated).

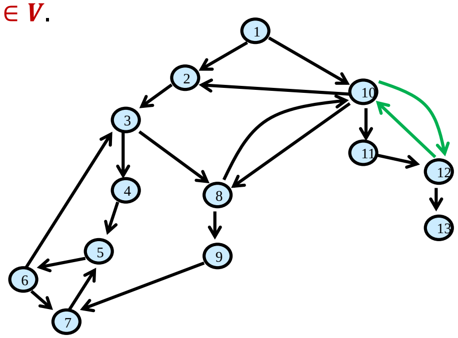
euler tours



NO: There are odd degree vertices.

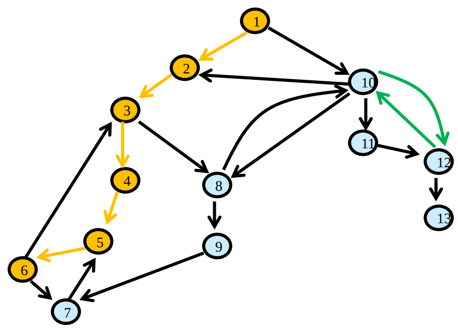
directed graphs

A *directed graph* (**digraph** for short) is a pair $G = (V, E)$ of *vertices* (V) and *edges* (E). The edges are now *ordered* pairs of vertices, i.e. (u, v) for $u, v \in V$.



directed path

A *path* in a directed graph is a sequence of nodes $v_1, v_2, \dots, v_k$ such that $(v_i, v_{\{i+1\}})$ are connected by an edge for $i = 1,2, \dots, k-1$.



graph traversal

We are interested in *algorithmic* questions like:
How can we determine if a graph is connected (and do it fast)?

Goal of traversal:
- Learn the basic structure of a graph
- Walk from a fixed starting vertex s to find all vertices reachable from s

Three states of vertices
- **unvisited**
- **visited/discovered**
- **fully-explored**

generic graph traversal algorithm

Find: Set R of vertices reachable from $s \in V$

Reachable(s):
  $R \leftarrow \{s\}$
  While there is an edge $(u, v) \in E$ with $u \in R$ and $v \notin R$
    Add v to R

generic traversal always works

Claim:
At termination R is the set of nodes reachable from s

Proof:

$\subseteq$: For every node $v \in R$ there is a path from s to v

$\supseteq$: Suppose there is a node $w \notin R$ reachable from s via a path P

Take first node v on P such that $v \notin R$

Predecessor u of v in P satisfies

$u \in R$

$(u, v) \in E$

But this contradicts the fact that the algorithm exited the while loop.

generic graph traversal algorithm

Find: Set R of vertices reachable from $s \in V$

Reachable(s):
  $R \leftarrow \{s\}$
  While there is an edge $(u, v) \in E$ with $u \in R$ and $v \notin R$
    Add v to R

We didn't specify the order in which to check the edges.
Different orders lead to algorithms with different properties.
Two main examples:
BFS (breadth-first search) and DFS (depth-first search)

breadth-first search

- Completely explore the vertices in order of their distance from s
- Naturally implemented using a queue

BFS

Global initialization: mark all vertices "unvisited"

BFS(**s**)

    mark s "visited"; R←{s}; layer L0←{s}
    while Li not empty
        Li+1 ← ∅
        For each u∈Li
            for each edge {u,v}
                if (v is "unvisited")
                    mark v "visited"
                    Add v to set R and to layer Li+1
            mark u "fully-explored"
        i ← i+1

properties of BFS

- BFS(**s**) visits **x** if and only if there is a path in **G** from **s** to **x**.
- Edges followed to undiscovered vertices define a "breadth first spanning tree" of **G**
- Layer **i** in this tree, L_i
  - those vertices **u** such that the shortest path in **G** from the root **s** is of length **i**.
- On undirected graphs
  - All non-tree edges join vertices on the same or adjacent layers

properties of BFS

On undirected graphs

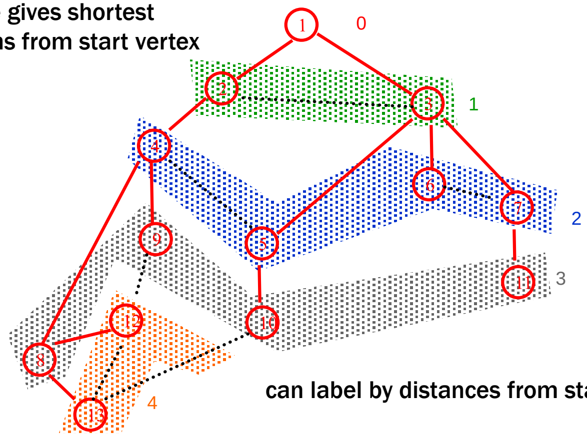
- All non-tree edges join vertices on the same or adjacent layers
- Suppose not

  Then there would be vertices **(x,y)** such that $x \in L_i$ and $y \in L_j$ and $j > i+1$

  Then, when vertices incident to **x** are considered in BFS **y** would be added to L_{i+1} and not to L_j

BFS application: shortest paths

Tree gives shortest paths from start vertex



can label by distances from start