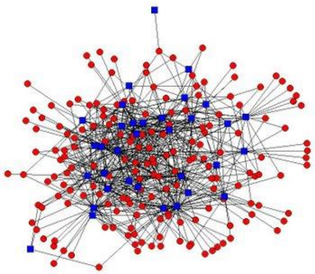
CSE 421: Algorithms

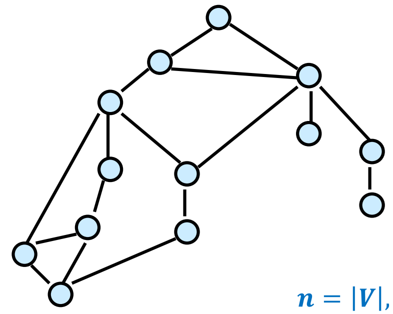
Winter 2014
Lecture 5: Graphs and graph traversal II

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 $u, v \in V$.

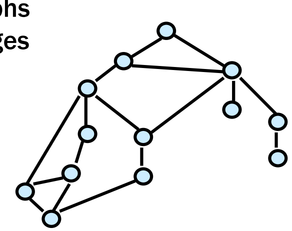


$n = |V|, m = |E|$

undirected graphs

Two representations:
- adjacency list
- adjacency matrix

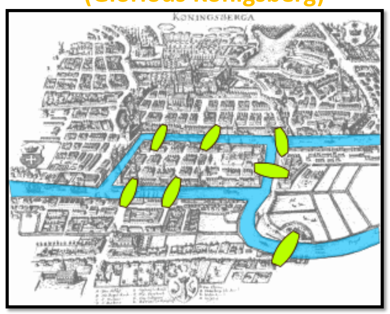
Dense graphs vs. sparse graphs
$\Theta(n^2)$ edges $O(n)$ edges



$n = |V|, m = |E|$

euler tours

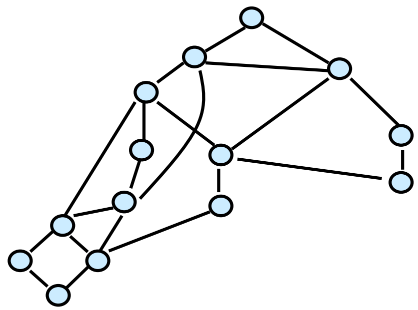
(Glorious Konigsberg)



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

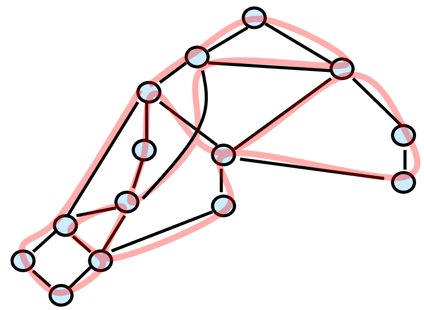
eulerian graphs

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



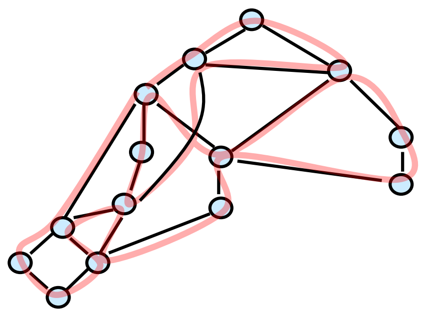
eulerian graphs

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 it is connected and every vertex has even degree!

Proof.

Easier direction:

Eulerian → connected and all degrees even.

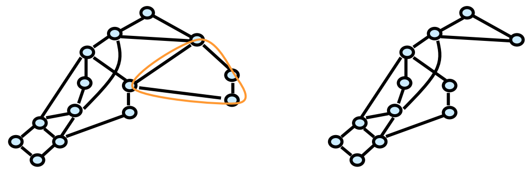
eulerian graphs

Theorem: An undirected graph is Eulerian if and only if it is connected and 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 it is connected and 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 it is connected and 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?

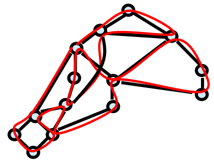
one last step…

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.

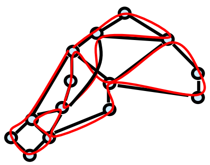
the profundity of algorithms



find a tour that visits every **edge** exactly once

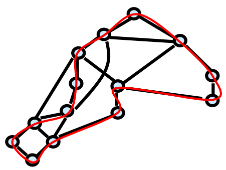
$\ll 1$ second on $100{,}000$ node graph (fastest algorithm, 1 op/nanosec)

the profundity of algorithms



find a tour that visits every **edge** exactly once

$\ll 1$ second on $100{,}000$ node graph (fastest algorithm, 1 op/nanosec)

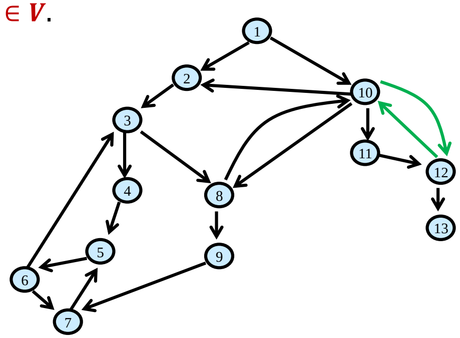


find a tour that visits every **vertex** exactly once

fastest known algorithm on 100 node graph takes $\gg 1{,}000{,}000$ years

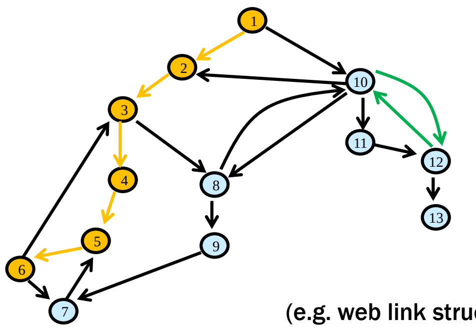
directed graphs

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

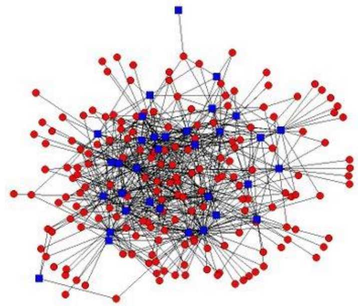
A *path* in a directed graph is a sequence of nodes $v_1, v_2, \ldots, v_k$ such that (v_i, v_{i+1}) are connected by an edge for $i = 1,2,\ldots,k-1$.



(e.g. web link structure)

graph traversal

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



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.

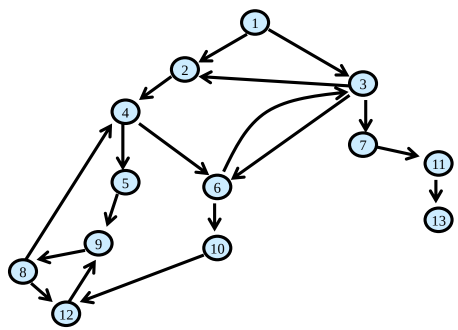
Different orders lead to algorithms with different properties.

Two main examples:

BFS (breadth-first search) and DFS (depth-first search)

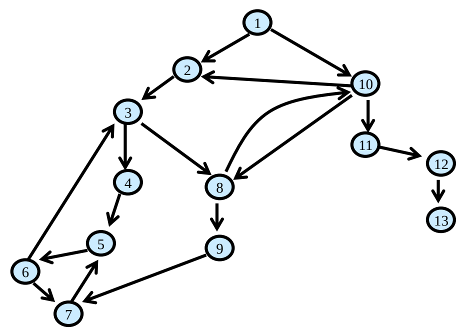
breadth-first search

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



depth-first search

- Completely explore the vertices in DFS order (duh)
- Naturally implemented using recursion



BFS

Global initialization: mark all vertices "unvisited"
BFS(s)
  mark s "visited"; R←{s}; layer L_0←{s}
  while L_i not empty
    $L_{i+1} \leftarrow \varnothing$
    for each **u**∈**L_i**
      for each edge (**u**,**v**)
        if (**v** is "unvisited")
          mark **v** "visited"
          Add **v** to set **R** and to layer **L_{i+1}**
      Mark **u** "fully-explored"
    i ← i+1

BFS

Global initialization: mark all vertices "unvisited"
BFS(s)
  mark s "visited"; R←{s}; layer L_0←{s}
  while L_i not empty
    $L_{i+1} \leftarrow \varnothing$
    for each **u**∈**L_i**
      for each edge (**u**,**v**)
        if (**v** is "unvisited")
          mark **v** "visited"
          Add **v** to set **R** and to layer **L_{i+1}**
      mark **u** "fully-explored"
    i ← i+1

total running time:
$O(m+n)$

properties of BFS

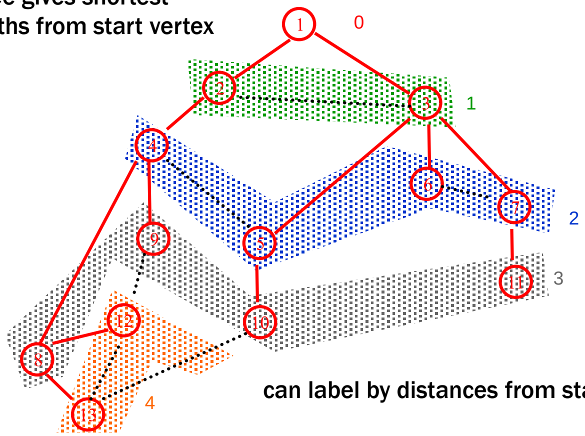
- BFS(s) visits x if and only if there is a path in G from s to x.
- Edges followed to undiscovered vertices define a "breadth first spanning tree" of G
- Layer i in this tree, L_i
  - those vertices u such that the shortest path in G from the root s is of length i.
- On undirected graphs
  - All non-tree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

BFS application: shortest paths

Tree gives shortest paths from start vertex



connected components

Want to answer questions of the form:

- **Given**: vertices u and v in G
- Is there a path from u to v?

connected components

Want to answer questions of the form:

- **Given**: vertices u and v in G
- Is there a path from u to v?

Idea: create array A such that

$A[u]$ = smallest numbered vertex that is connected to u

- question reduces to whether $A[u] = A[v]$?

connected components

Want to answer questions of the form:

- **Given**: vertices u and v in G
- Is there a path from u to v?

Q: Why not create an array **Path**[u,v]?

Idea: create array A such that

$A[u]$ = smallest numbered vertex that is connected to u

- question reduces to whether $A[u] = A[v]$?

connected components

- initial state: all **v** unvisited
  for **s** ← **1** to **n** do
    if state(**s**) ≠ "fully-explored" then
      BFS(**s**): setting **A**[**u**] ←**s** for each **u** found
        (and marking **u** visited/fully-explored)
    endif
  endfor

- Total cost: $O(n+m)$
  - each vertex is touched once in this outer procedure and the edges examined in the different BFS runs are disjoint
  - works also with depth first search

DFS(u) – recursive version

Global Initialization: mark all vertices "unvisited"
DFS(**u**)
  mark **u** "visited" and add **u** to **R**
  for each edge (**u**,**v**)
    if (**v** is "unvisited")
      DFS(**v**)
  mark **u** "fully-explored"

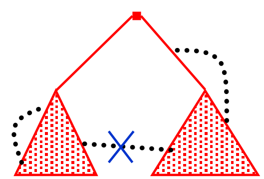
properties of DFS(s)

- Like BFS(**s**):
  - DFS(s) visits x if and only if there is a path in G from s to x
  - Edges into undiscovered vertices define a "depth first spanning tree" of G
- Unlike the BFS tree:
  - the DFS spanning tree isn't minimum depth
  - its levels don't reflect min distance from the root
  - non-tree edges never join vertices on the same or adjacent levels
- but…

non-tree edges

- All non-tree edges join a vertex and one of its descendents/ancestors in the DFS tree

- No cross edges.

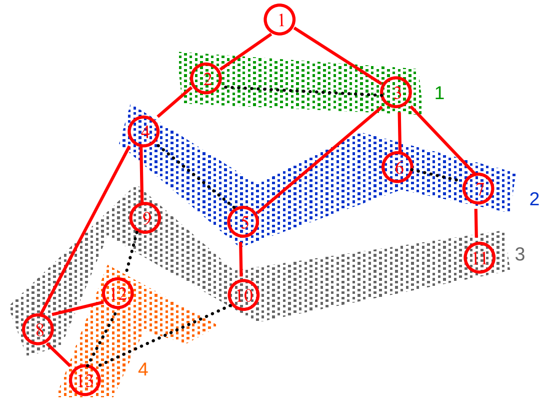


bipartite graph

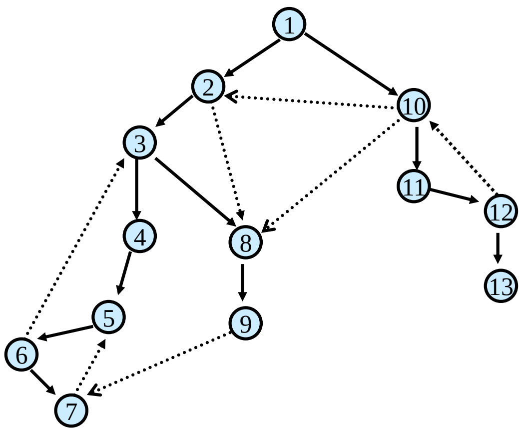
Definition:

Theorem:
Graph is bipartite iff does not contain an odd cycle.

BFS for bipartite testing



DFS(v) for a directed graph



DFS(v)