The Konigsberg Bridge Problem

This is a classic mathematical problem.

There were seven bridges across the river Pregel at Königsberg.

Is it possible to take a walk in which each bridge is crossed exactly once?

Euler solved this problem in 1736.

- Key insight: represent the problem graphically

Eulerian Paths

Recall that $G(V, E)$ has an Eulerian path if it has a path that goes through every edge exactly once. It has an Eulerian cycle (or Eulerian circuit) if it has an Eulerian path that starts and ends at the same vertex.

How can we tell if a graph has an Eulerian path/circuit?

What's a necessary condition for a graph to have an Eulerian circuit?

Count the edges going into and out of each vertex:

- Each vertex must have even degree!

This condition turns out to be sufficient too.

Theorem: A connected (multi)graph has an Eulerian cycle iff each vertex has even degree.

Proof: The necessity is clear: In the Eulerian cycle, there must be an even number of edges that start or end with any vertex.

To see the condition is sufficient, we provide an algorithm for finding an Eulerian circuit in $G(V, E)$.

First step: Follow your nose to construct a cycle.

Second step: Remove the edges in the cycle from G. Let H be the subgraph that remains.

- every vertex in H has even degree
- H may not be connected; let $H_1, \ldots, H_k$ be its connected components.

Third step: Apply the algorithm recursively to $H_1, \ldots, H_k$, and then splice the pieces together.

Finding cycles

First, find an algorithm for finding a cycle:

Input: $G(V, E)$ [a list of vertices and edges]

procedure Pathgrow(V,E,v)
  [v is first vertex in cycle]
  $P \leftarrow ()$ [P is sequence of edges on cycle]
  $w \leftarrow v$ [w is last vertex in P]
  repeat until $I(w) - P = \emptyset$
    [$I(w)$ is the set of edges incident on w]
    Pick $e \in I(w) - P$
    $w \leftarrow$ other end of e
    $P \leftarrow P \cdot e$ [append e to P]
  endrepeat
  return P
endpro

Claim: If every vertex in V has even degree, then P will be a cycle

- Loop invariant: In the graph $G(V, E - P)$, if the first vertex (v) and last vertex (w) in P are different, they have odd degree; all the other vertices have even degree.

Finding Eulerian Paths

Input: $G(V, E)$ [a list of vertices and edges]

Algorithm ECycle:

procedure Euler(V',E',v')
Pathgrow(V',E',v')
if P is not Eulerian,
delete the edges in P from E;
let $G_1(V_1, E_1), \ldots, G_n(V_n, E_n)$ be
the resulting connected components
let v_i be a vertex in V_i
for $i = 1$ **to** n
Euler(V_i, E_i, v_i)
Attach C to P at v_i
endfor
$C \leftarrow P$
return C
endpro
$v \leftarrow$ any vertex in V
Euler(V,E,v)

Corollary: A connected multigraph has an Eulerian path (but not an Eulerian cycle) if it has exactly two vertices of odd degree.

Which of these graphs have Eulerian paths:

Hamiltonian Paths

Recall that $G(V, E)$ has a Hamiltonian path if it has a path that goes through every vertex exactly once. It has a Hamiltonian cycle (or Hamiltonian circuit) if it has a Hamiltonian path that starts and ends at the same vertex.

There is no known easy characterization or algorithm for checking if a graph has a Hamiltonian cycle/path.

Which of these graphs have a Hamiltonian cycle?

Searching Graphs

Suppose we want to process data associated with the vertices of a graph. This means we need a systematic way of searching the graph, so that we don't miss any vertices.

There are two standard methods.

- Breadth-first search
- Depth-first search

It's best to think of these on a tree:

Breadth-first search would visit the nodes in the following order:

$$1, 2, 3, \ldots, 10$$

Depth-first search would visit the nodes in the following order:

$$1, 2, 4, 5, 7, 8, 11, 3, 6, 9, 10$$

Breadth-First Search

Input $G(V, E)$ [a connected graph]
v [start vertex]

Algorithm Breadth-First Search
visit v
$V' \leftarrow \{v\}$ [V' is the vertices already visited]
Put v on Q [Q is a queue]
repeat while $Q \neq \emptyset$
$u \leftarrow head(Q)$ [$head(Q)$ is the first item on Q]
for $w \in A(u)$ [$A(u) = \{w | \{u, w\} \in E\}$]
if $w \notin V'$
then visit w
Put w on Q
$V' \leftarrow V' \cup \{w\}$
endif
endfor
Delete u from Q

The BFS algorithm basically finds a tree embedded in the graph.

- This is called the *BFS search tree*

BFS and Shortest Length Paths

If all edges have equal length, we can extend this algorithm to find the shortest path length from v to any other vertex:

- Store the path length with each node when you add it.
- Length(v) = 0.
- Length(w) = Length(u) + 1

With a little more work, can actually output the shortest path from u to v.

- This is an example of how BFS and DFS arise unexpectedly in a number of applications.
  - We'll see a few more

Depth-First Search

Input $G(V, E)$ [a connected graph]
v [start vertex]

Algorithm Depth-First Search
visit v
$V' \leftarrow \{v\}$ [V' is the vertices already visited]
Put v on S [S is a stack]
$u \leftarrow v$
repeat while $S \neq \emptyset$
if $A(u) - V' \neq \emptyset$
then Choose $w \in A(u) - V'$
visit w
$V' = V' \cup \{w\}$
Put w on stack
$u \leftarrow w$
else $u \leftarrow top(S)$ [Pop the stack]
endif
endrepeat

DFS uses *backtracking*

- Go as far as you can until you get stuck
- Then go back to the first point you had an untried choice

Spanning Trees

A *spanning tree* of a connected graph $G(V, E)$ is a connected acyclic subgraph of G, which includes all the vertices in V and only (some) edges from E.

Think of a spanning tree as a "backbone"; a minimal set of edges that will let you get everywhere in a graph.

- Technically, a spanning tree isn't a tree, because it isn't directed.

The BFS search tree and the DFS search tree are both spanning trees.

- In the text, they give algorithms to produce minimum weight spanning trees
- That's done in CS 482, so we won't do it here.

Topological Sorting

[NOT IN TEXT]

If $G(V, E)$ is a *dag*: directed acyclic graph, then a *topological sort* of G is a total ordering $\prec$ of the vertices in V such that if $(v, v') \in E$, then $v \prec v'$.

- Application: suppose we want to schedule jobs, but some jobs have to be done before others
  - vertices on dag represent jobs
  - edges describe precedence
  - topological sort gives an acceptable schedule

Theorem: Every dag has at least one topological sort.

- If $V \neq \emptyset$, some vertices in V have indegree 0.
  - If all vertices in V have indegree > 0, then G has a cycle: start at some $v \in V$, go to a parent v' of v, a parent v'' of v', etc.
    - Eventually a node is repeated; this gives a cycle
- Add a "virtual node" v^* to the graph, and an edge from v^* to all nodes with indegree 0
- Do a BFS starting at v^*.
- Order the vertices in the order they're discovered by BFS

Ignoring v^*, this algorithm gives a topological sort of G

- It's running time is polynomial
- actually, linear in the number of vertices $O(|V|+|E|)$.

Graph Coloring

How many colors do you need to color the vertices of a graph so that no two adjacent vertices have the same color?

- Application: scheduling
  - Vertices of the graph are courses
  - Two courses taught by same prof are joined by edge
  - Colors are possible times class can be taught.

Lots of similar applications:

- E.g. assigning wavelengths to cell phone conversations to avoid interference.
  - Vertices are conversations
  - Edges between "nearby" conversations
  - Colors are wavelengths.
- Scheduling final exams
  - Vertices are courses
  - Edges between courses with overlapping enrollment
  - Colors are exam times.

Chromatic Number

The *chromatic number* of a graph G, written $\chi(G)$, is the smallest number of colors needed to color it so that no two adjacent vertices have the same color.

Examples:

A graph G is *k-colorable* if $k \geq \chi(G)$.

Determining $\chi(G)$

Some observations:

- If G is a complete graph with n vertices, $\chi(G) = n$
- If G has a clique of size k, then $\chi(G) \leq k$.
  - Let $c(G)$ be the *clique number* of G: the size of the largest clique in G. Then
$$\chi(G) \geq c(G)$$
- If $\Delta(G)$ is the maximum degree of any vertex, then
$$\chi(G) \leq \Delta(G) + 1 :$$
  - Color G one vertex at a time; color each vertex with the "smallest" color not used for a colored vertex adjacent to it.

How hard is it to determine if $\chi(G) = k$?

- It's NP complete, just like
  - determining if $c(G) = k$
  - determining if G has a Hamiltonian path
  - determining if a propositional formula is satisfiable

Can guess and verify.

Bipartite Graphs

A graph $G(V, E)$ is *bipartite* if we can partition V into disjoint sets V_1 and V_2 such that all the edges in E joins a vertex in V_1 to one in V_2.

- A graph is bipartite iff it is 2-colorable
- Everything in V_1 gets one color, everything in V_2 gets the other color.

Example: Suppose we want to represent the "is or has been married to" relation on people. Can partition the set V of people into males (V_1) and females (V_2). Edges join two people who are or have been married.

Characterizing Bipartite Graphs

Theorem: G is bipartite iff G has no odd-length cycles.

Proof: Suppose that G is bipartite, and it has edges only between V_1 and V_2. Suppose, to get a contradiction, that $(x_0, x_1, \ldots, x_{2k}, x_0)$ is an odd-length cycle. If $x_0 \in V_1$, then x_2 is in V_1. An easy induction argument shows that $x_{2i} \in V_1$ and $x_{2i+1} \in V_2$ for $0 = 1, \ldots, k$. But then the edge between x_{2k} and x_0 means that there is an edge between two nodes in V_1; this is a contradiction.

- Get a similar contradiction if $x_0 \in V_2$.

Conversely, suppose $G(V, E)$ has no odd-length cycles.

- Partition the vertices in V into two sets as follows:
  - Start at an arbitrary vertex x_0; put it in V_0.
  - Put all the vertices one step from x_0 into V_1
  - Put all the vertices two steps from x_0 into V_0;
  - ...

This construction works if G is connected and has no odd-length cycles.

- What if G isn't connected?

This construction also gives a polynomial-time algorithm for checking if a graph is bipartite.

The Four-Color Theorem

Can a map be colored with four colors, so that no countries with common borders have the same color?

- This is an instance of graph coloring
  - The vertices are countries
  - Two vertices are joined by an edge if the countries they represent have a common border

A *planar graph* is one where all the edges can be drawn on a plane (piece of paper) without any edges crossing.

- The graph of a map is planar

Graphs that are planar and ones that aren't:

Four-Color Theorem: Every map can be colored using at most four colors so that no two countries with a common boundary have the same color.

- Equivalently: every planar graph is four-colorable

Four-Color Theorem: History

- First conjectured by Galton (Darwin's cousin) in 1852
- False proofs given in 1879, 1880; disproved in 1891
- Computer proof given by Appel and Haken in 1976
  - They reduced it to 1936 cases, which they checked by computer
- Proof simplified in 1996 by Robertson, Sanders, Seymour, and Thomas
  - But even their proof requires computer checking
  - They also gave an $O(n^2)$ algorithm for four coloring a planar graph
- Proof checked by Coq theorem prover (Werner and Gonthier) in 2004
  - So you don't have to trust the proof, just the theorem prover

Note that the theorem doesn't apply to countries with non-contiguous regions (like U.S. and Alaska).