CS 163 & CS 265: Graph Algorithms

Week 4: More paths

Lecture 12: Euler tours

David Eppstein
University of California, Irvine

Fall Quarter, 2022

Overview

Walks that visit everything in the graph

Tour: start and end at same vertex

Undirected graphs

"Everything": May mean all vertices or all edges

But most of this also works for directed graphs and for walks that start and end at different vertices

Four variants of the problem

Euler tour (today)

Unweighted, walk that uses each edge exactly once

Linear time

Hamiltonian cycle

Unweighted, cycle that uses each vertex exactly once

NP-complete

Chinese postman tour

Weighted, shortest walk that uses each edge at least once

Named after Meigu Guan

Polynomial time

Traveling salesperson tour

(later this week)

Weighted, shortest walk that visits each vertex at least once

NP-complete

The Seven Bridges of Königsberg

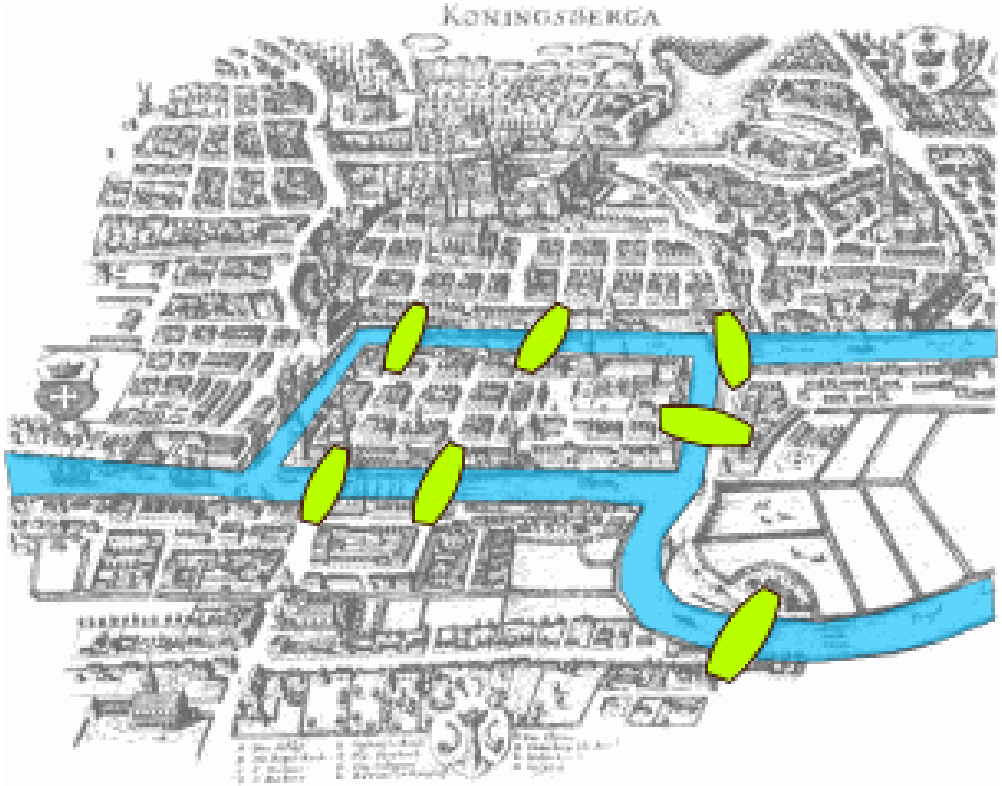
Königsberg

- City on the Baltic Sea coast, at the mouth of the Pregel river
- Name means "king's mountain"
- Formerly in Prussia (pre-WWII Germany)
- Taken over by Russia in 1945, renamed Kaliningrad
- Disconnected from the main part of Russia
- Home of the Baltic Fleet of the Russian Navy

The Seven Bridges of Königsberg

Early-1700s Königsberg had seven bridges, connecting two islands with the river banks

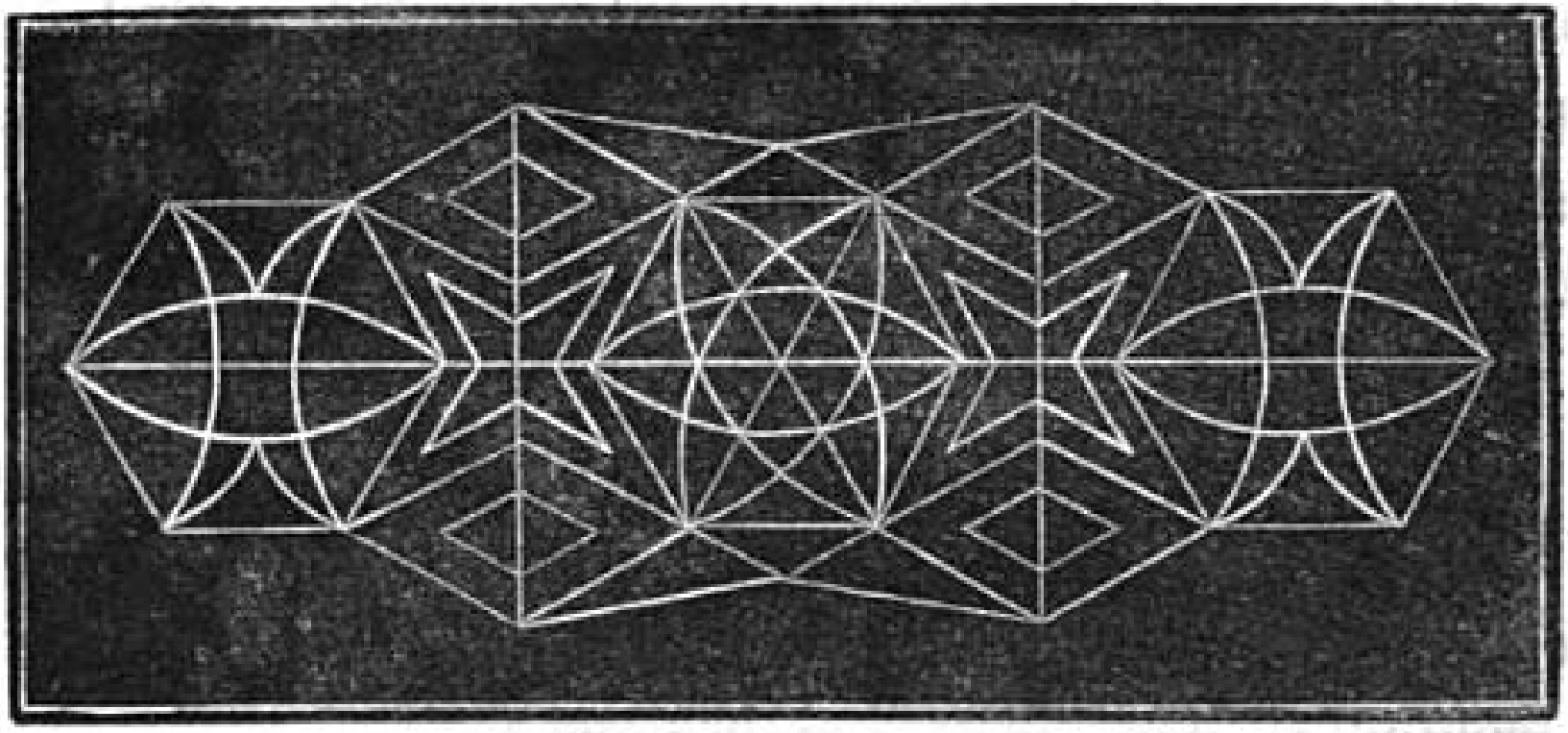
A puzzle from the time asked whether one could walk through the city crossing each bridge exactly once



Mathematician Leonhard Euler published a solution in 1736 that became the first published work in graph theory [Euler 1736]

Another more complicated example

Puzzle: Can you draw this pattern using a pen or pencil, as a single continuous line, without lifting your pen or pencil from the paper?

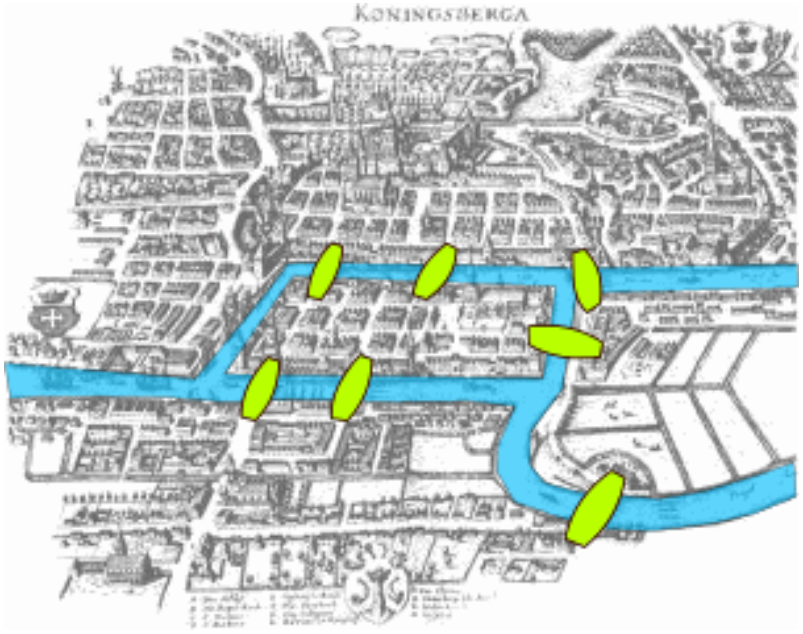


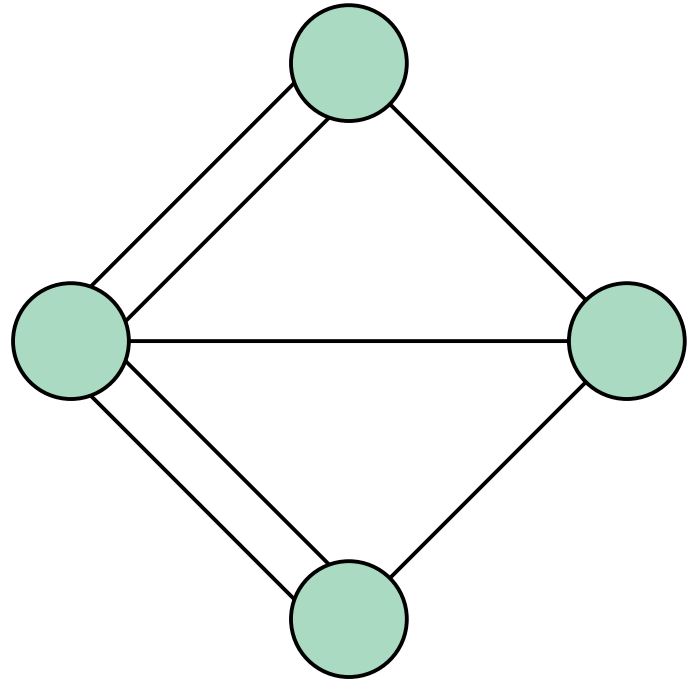
[Listing 1848]

Translation into a graph problem

Make a vertex for each piece of land and an edge for each bridge

(It's actually a multigraph because some pieces of land are connected by two bridges)



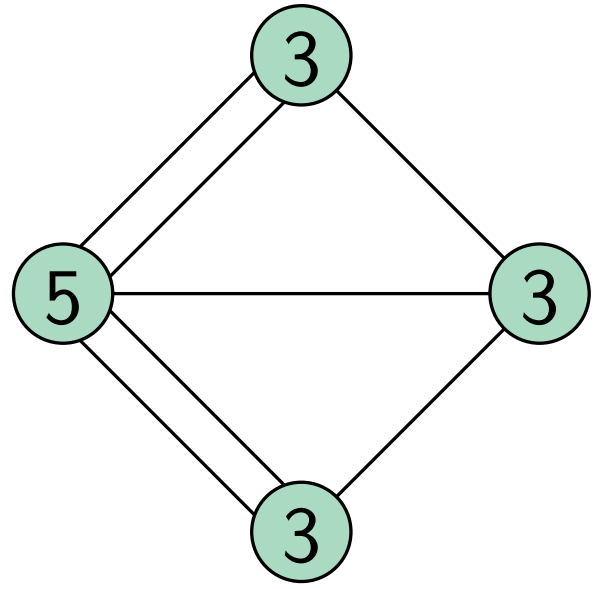


Problem becomes: Does some walk use each edge exactly once?

Degree and the handshaking lemma

Degree of a vertex:
how many edges touch it?

An odd vertex is a vertex whose degree is an odd number



Euler's handshaking lemma:
Every finite graph has an even number of odd vertices

(The Königsberg graph has four)

Proof idea: Each edge has 2 endpoints $\Rightarrow$ the whole graph has $2m$ vertex–edge contacts, an even number

Counting by vertices instead of edges, $\#\text{contacts} = \sum_v \deg(v)$.

Every even vertex contributes an even number of contacts, and every odd vertex contributes an odd number, so for the sum to be even the number of odd vertices must be even

Necessary conditions for an Euler tour

Whenever a tour enters and then leaves a vertex, the two edges it crosses contribute an even number to the degree of the vertex

When a tour starts at a vertex or ends at a vertex, the edge at the start or end of the tour contributes an odd number to the degree

So if an Euler tour (through each edge exactly once) starts and ends at the same vertex, all vertices must have even degree

If the tour starts and ends at different vertices, those two vertices have odd degree and all the rest must be even

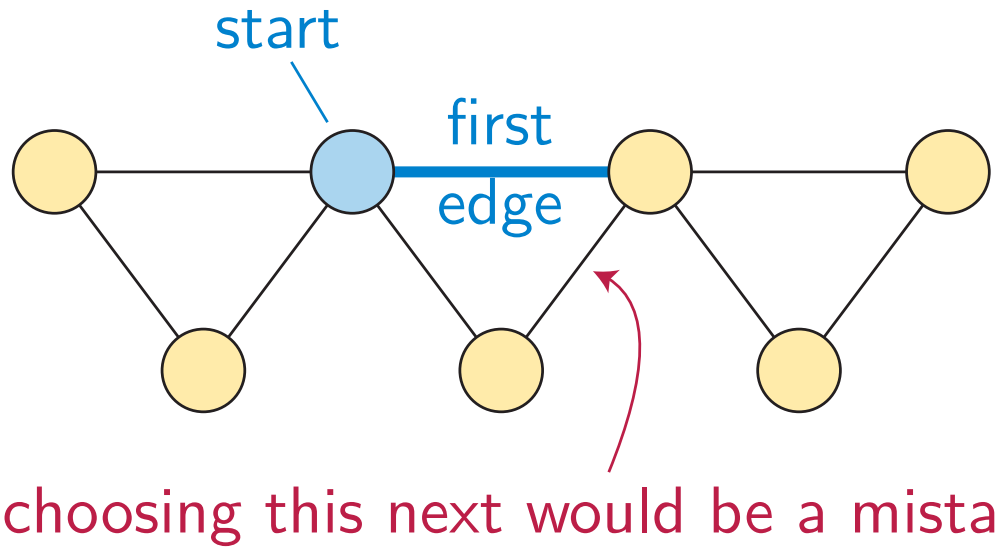
Königsberg has four vertices of odd degree, too many $\Rightarrow$ no tour

Euler claimed that every connected finite graph with zero or two odd vertices has a tour, but did not provide a proof

Algorithms

Fleury's algorithm [Fleury 1883]

- A good method for pencil and paper but slow & complicated for computers
- Start anywhere when all degrees are even, or at an odd vertex when there are two odd vertices
- Greedy algorithm: keep adding edges to the end of the curve, but don't make mistakes
- Mistake: taking the last edge connecting two uncovered parts of the graph



Why does Fleury's algorithm work correctly?

Whenever the path so far does not end at the eventual goal vertex, both have an odd # of uncovered edges (odd $\Rightarrow$ nonzero)

$\Rightarrow$ algorithm can only get stuck when it is at the eventual goal (but maybe has not covered all edges)

Only possible mistake: edge vw that separates a part of the graph X containing current vertex v from the the part of the graph containing goal (any other possibility would leave an odd number of odd vertices in separated pieces, violating handshaking lemma)

$\Rightarrow$ it is the only edge connecting X to the rest of the graph, because otherwise it would not be a mistake

$\Rightarrow$ every other edge we could take from v is not a mistake

$\Rightarrow$ until all edges covered, a non-mistaken next edge always exists

How fast is Fleury's algorithm?

Naïve implementation

Use reachability to test whether each choice is a mistake

If it is, choose any other edge (no need to test again)

$O(m)$ each time we add an edge to the tour, $O(m^2)$ total

Complicated data structures

Use a data structure that can keep track of the uncovered edges and tell which ones disconnect the rest, in small time per test

Randomized $O(m \log n (\log \log n)^3)$ [Thorup 2000]

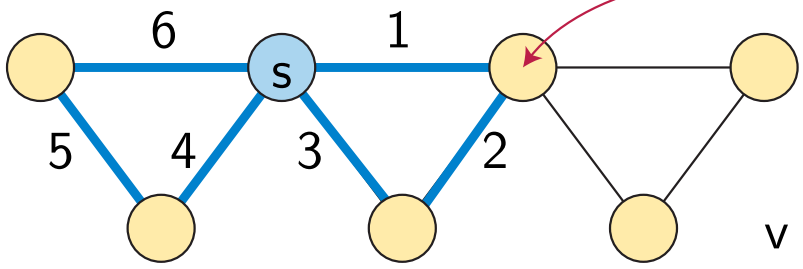
Non-random $O(m(\log n)^2 / \log \log n)$ [Wulff-Nilsen 2013]

Hierholzer's algorithm [Hierholzer 1873]

- Choose a starting vertex and then form tour one edge at a time, like Fleury
- Keep following edges until getting stuck, but without checking for making mistakes
- Then, while there exists vertex v on the current tour at which some edges are still uncovered:
  - Start a new tour that follows uncovered edges in the same way from v until getting stuck at v again
  - Splice the new tour into the position of v of the main tour

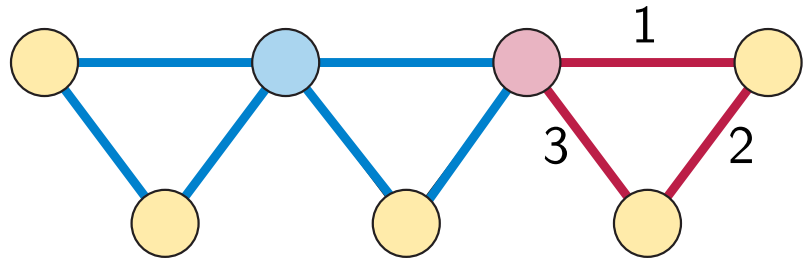
Hierholzer example

Tour 1: walk from s until stuck

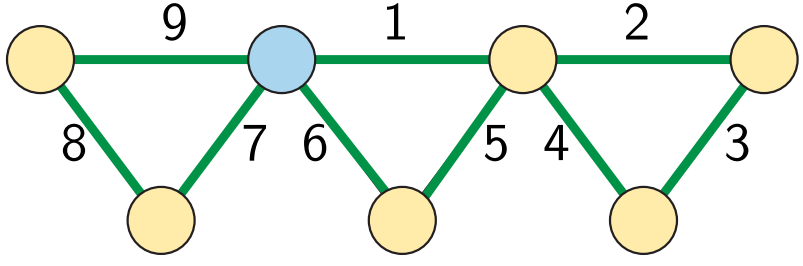


v appears in tour 1, has uncovered edges

Tour 2: walk from v until stuck



Insert tour 2 at the position of v in tour 1



Linear-time implementation details

- Decorate each edge with a Boolean, is it used yet?
- Decorate each vertex with an iterator of its edges
- Represent the tour in a way that allows efficient insertions into the middle (e.g. a linked list, not a Python list); initialize it to just contain the starting vertex
- Keep a pointer to the first vertex v on the main tour that might still have uncovered edges, initially s
- Repeat:
  - Use the iterator for v to find its next edge vw
  - If the iterator runs out of edges, advance v to the next vertex on the main tour, or exit if there is no next vertex
  - Else if vw is already covered, continue
  - Else start a new tour with edge vw, using the iterators for each vertex to add more edges to the end of the tour until reaching a vertex whose iterators have run out, and then attach the new tour after v in the main tour

A parallel algorithm

More complicated, but allows many things to be done simultaneously:

- At each vertex, group edges into pairs
- Following pairs (if you come into a vertex by one edge, go out on its pair) covers the graph by many cycles, but we want only one big cycle
- Use reachability to find the cycles
- Make an auxiliary graph whose vertices describe cycles and whose edges describe certain ways of gluing pairs of cycles together
- Find a spanning tree in the auxiliary graph
- Use the spanning tree to guide which pairs of cycles to glue

[Awerbuch et al. 1984]

Morals of the story

Sometimes recreational puzzles lead to important mathematics

There are multiple ways of asking for a tour that visits everything, with different complexities

The simplest one, a walk that visits each edge once, has an easy test for whether it exists, and can be found in linear time

References I

Baruch Awerbuch, Amos Israeli, and Yossi Shiloach. Finding Euler circuits in logarithmic parallel time. In *Proc. 16th ACM Symposium on Theory of Computing (STOC 1984)*, pages 249–257, 1984. doi:10.1145/800057.808688.

Leonhard Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii Academiae Scientiarum Imperialis Petropolitanae*, 8:128–140, 1736. URL `https://math.dartmouth.edu/~euler/docs/originals/E053.pdf`.

Pierre-Henry Fleury. Deux problemes de geometrie de situation. *Journal de mathematiques elementaires*, pages 257–261, 1883.

Carl Hierholzer. Ueber die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. *Mathematische Annalen*, 6(1):30–32, 1873. doi:10.1007/BF01442866.

References II

Johann Benedikt Listing. *Vorstudien zur Topologie*. Vandenhoeck und Ruprecht, 1848.

Mikkel Thorup. Near-optimal fully-dynamic graph connectivity. In *Proc. 32nd ACM Symposium on Theory of Computing (STOC 2000)*, pages 343–350, 2000. doi:10.1145/335305.335345.

Christian Wulff-Nilsen. Faster Deterministic Fully-Dynamic Graph Connectivity. In *Proc. 24th ACM-SIAM Symposium on Discrete Algorithms (SODA 2013)*, pages 1757–1769, 2013. doi:10.1137/1.9781611973105.126.