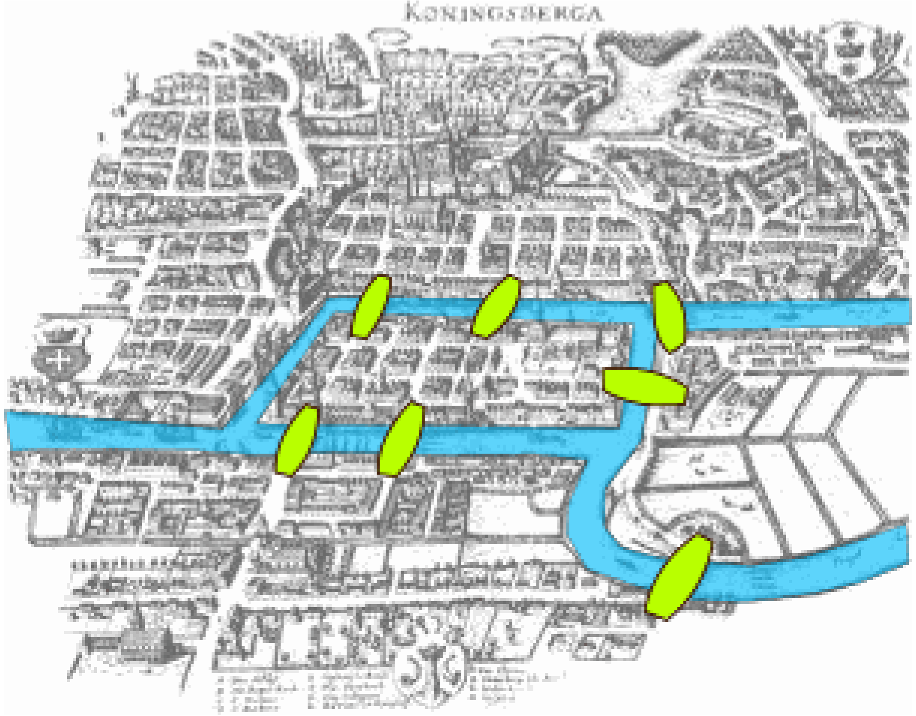
Theory of Computing

Lecture 8

MAS 714

Hartmut Klauck

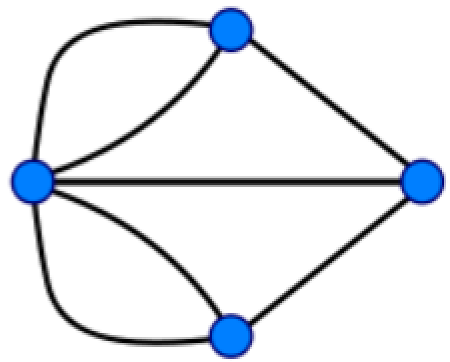
Seven Bridges of Königsberg



Can one take a walk that crosses each bridge exactly once?

Seven Bridges of Königsberg

- Model as a graph



- Is there a path that traverses each edge exactly once?
  - Original problem allows different start and end vertex
  - Answer is no.

Euler Tours

- For an undirected graph G=(V,E) an *Euler circuit* is a path that traverses every edge exactly once, and ends at the same vertex as it starts
- Same definition for directed graphs
- Graph G is Eulerian if it has an Euler circuit

Euler Circuits

- **Theorem:** an undirected graph is Eulerian, iff all vertices have even degree and all vertices of nonzero degree are in the same connected component.
- *Proof:* Clearly the condition is necessary.
  To see that it is sufficient we will give an algorithm that will find an Euler tour in linear time.
- Note: vertices of degree 0 do not need to be visited

Finding an Euler circuit

- Start at some vertex v_1, follow any edge $\{v_i,v_{i+1}\}$ until v_1 is reached again (initial tour)
  - On the way mark edges as *used* and vertices as *visited*
- At this time some edges may be unused
- Find any vertex on the tour with unused edges and start a path from it until a cycle is formed
- Join the new cycle with the tour
- Continue until no vertex on the tour has any unused edges
- By assumption there are no unvisited vertices with degree>0
- Why don’t we get stuck?

Why it works

- There are two ways to get stuck:
  - Not possible to return to the starting vertex
  - Cycle constructed so far as no outgoing edges to look for a new cycle that can be joined with the current one
- But all vertices have even degree
  - We can continue on the current path until it eventually come back to starting vertex
- And graph is connected
  - If there are no edges left from current cycle we are already done
- Furthermore:
  - Removing a cycle leaves the Euler condition intact

Implementation

- Store the circuit as a (doubly) linked list T
  - initially empty, linked forward and backward
- Augment the adjacency list A to store also pointers from the vertices to an occurrence in T and their degree (if nonzero)
  - For visited vertices
  - Delete a vertex if degree is 0
- Create the initial tour while traversing the graph from some vertex
  - Delete used edges from the adjacency list and update degrees
- Continue with a vertex v with nonzero degree and find a cycle, and insert the cycle from v to v into T

Time

- An Euler circuit can be found in time O(n+m)
- Note that we can decide the Euler condition very easily from the degree sequence if we know the graph is connected

Euler Paths

- A graph has an Euler path if there is a path that traverses all edges exactly once
  - Start and end of the path need not be the same
- ***Theorem:*** A graph has an Euler path if all vertices except two have even degree and the two vertices have odd degree
  - And all vertices with degree>0 connected

Proof

- Again, the condition is necessary
- Sufficient too:
  - Add an extra edge connecting the two vertices
  - Might have a multigraph now
    - Everything here works for multigraphs
  - New graph has an Euler circuit
  - Remove extra edge from circuit to get path

Hamilton Paths

- A Hamilton path in an undirected graph is a path that visits every vertex exactly once
- A Hamilton circuit is a circuit/cycle that visits every vertex exactly once

- Note: No efficient algorithms are known that decide if there is a Hamiltonian path in a graph
  - And likely none exist

Minimum spanning trees

- **Definition**
  - A spanning tree of an undirected connected graph is a set of edges E‘$\subseteq$ E:
    - E‘ forms a tree
    - every vertex is in at least one edge in E‘
  - When the edges of G have weights, then a *minimum* spanning tree is a spanning tree with the smallest sum of edge weights

MST

- Motivation: measure costs to establish connections between vertices
- Basic procedure in many graph algorithms
- Problem first studied by Boruvka in 1926

- Other algorithms: Kruskal, Prim
- Inputs: adjacency list with weights

Application Example

- Traveling Salesman Problem [TSP]
- Input: matrix of edge weights and number K
- Decision: is there a path through the graph that visits each vertex once and has cost at most K?

- Problem is believed to be hard
  – i.e., not solvable in polynomial time

Application Example

- Metric TSP (Traveling Salesman Problem)
  - weights form a metric (symmetric, triangle inequality)
  - Still believed to be hard
- Approximation algorithm:
  - Find an MST T
  - Replace each edge of T by two edges
  - Traverse T in an Euler tour of these 2(n-1) edges
  - Make shortcuts to generate a cycle that goes through all vertices once
- Euler tour cost is twice the MST cost, hence at most twice the TSP cost, shortcuts cannot increase cost

MST

- Generic algorithm:
  - Start with an empty set of edges
  - Add edges, such that current edge set is always subset of a minimum spanning tree
  - Edges that can be added are called *safe*

Generic Algorithm

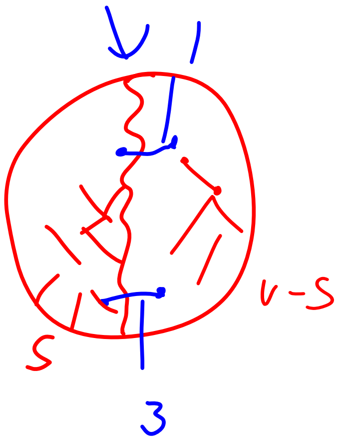
- Set $A=\emptyset$
- As long as A is not (yet) a spanning tree add a safe edge e
- Output A

Safe Edges

- How can we find safe edges?

- **Definition:**
  - A *cut* C=(S, V-S) is a partition of V
  - A set of edges *repects* the cut, if no edge crosses
  - An edge is *light* for a cut, if it is the edge with smallest weight that crosses the cut

- Example: red edges respect the cut



Safe Edges

- **Theorem:**
  Let G be an undirected, connected, weighted graph.
  A a subset of a minimum spanning tree. C=(S, V-S) a cut that A respects.
  Then the lightest edge of C is safe.
- Proof:
  - T is an MST containing A
  - Suppose e is not in T (otherwise we are done)
  - Construct another MST that contains e

Safe Edges

- Inserting e={u,v} into T creates a cycle p in T∪{e}
- u and v are on different sides of the cut
- Another edge e‘ in T crosses the cut
- e‘ is not in A (A respects the cut)
- Remove e‘ from T (T is now disconnected into 2 trees)
- Add e to T (the two trees reconnect into one)
- W(e)=W(e‘), so T‘ is also minimal
- Hence A∪{e} subset of T ‘
- e is safe for A.

Which edges are not in a min. ST?

- **Theorem:**
  - G a graph with weights W.
    All edge weights distinct.
    C a cycle in G and e={u,v} the largest edge in C.
  - Then e is in no minimum spanning tree.
- Proof:
  - Assume e is in a min. ST T
  - Remove e from T
  - Result is two trees (containing all vertices)
  - The vertices of the two trees form a cut
  - Follow C-{e} from u to v
  - Some edge e‘ crosses the cut
  - T-{e}∪{e‘} is a spanning tree with smaller weight T

Algorithms

- We complete the algorithm „skeleton“ in two ways
  - Prim: A is always a tree
  - Kruskal: A starts as a forest that joins into a single tree
    - initially every vertex its own tree
    - join trees until all are joined up

Data structures: Union-Find

- We need to store a set of disjoint sets with the following operations:
  - Make-Set(v):
    generate a set {v}. Name of the set is v
  - Find-Set(v):
    Find the name of the set that contains v
  - Union(u,v):
    Join the sets named u and v. Name of the new set is either u or v
- As with Dijkstra/priority queues the running time will depend on the implementation of the data structure

Kruskal

1. Input: graph G, weights $W: E \mapsto R$
2. $A=\emptyset$
3. For each vertex v:
   a) Make-Set(v)
4. Sort edges in E (increasing) by weight
5. For all edges {u,v} (order increasing by weight):
   a) a=FindSet(u), b=FindSet(v)
   b) If $a \neq b$ then
      $A:=A\cup \{\{u,v\}\}$
      Union(a,b)
6. Output A

Running time Kruskal

- We will have:
  - until 3: O(n) times Time for Make-Set
  - 4: O(m log n)
  - 5: O(m) time Time for Find/Union
  - Total will be O(n+m log n)

Correctness

- We only have to show that all edges inserted are safe
  - Choose a cut respected by A, which is crossed by the new edge e
  - e has minimum weight under all edges forming no cycle, hence e has minimum weight among all edges crossing the cut
  - Hence e must be safe for A