

# **Data Structures and Algorithms**

**Session 20. April 8, 2009**

**Instructor: Bert Huang**

**<http://www.cs.columbia.edu/~bert/courses/3137>**

# Announcements

- \* Homework 5 is out:
  - \* You can use adjacency lists if you prefer

# Review

- \* Extensions of Dijkstra's Algorithm
  - \* Critical Path Analysis
  - \* All Pairs Shortest Path (Floyd-Warshall)
- \* Maximum Flow
  - \* Floyd-Fulkerson Algorithm

# Today's Plan

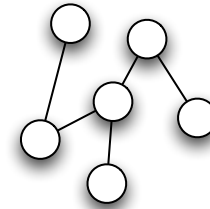
- \* Minimum Spanning Tree
  - \* Prim's Algorithm
  - \* Kruskal's Algorithm
- \* Depth first search
  - \* Euler Paths

# Minimum Spanning Tree Problem definition

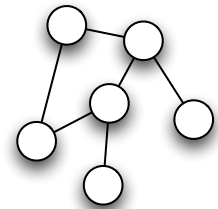
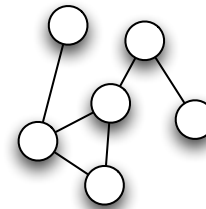
- \* Given connected graph  $\mathbf{G}$ , find the connected, acyclic subgraph  $\mathbf{T}$  with minimum edge weight
- \* A tree that includes every node is called a **spanning tree**
- \* The method to find the MST is another example of a greedy algorithm

# Motivation for Greed

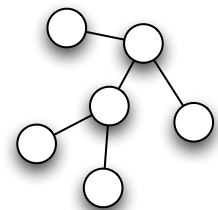
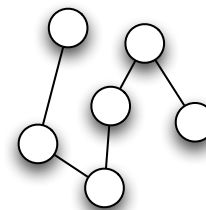
- \* Consider any spanning tree



- \* Adding another edge to the tree creates exactly one cycle



- \* Removing an edge from that cycle restores the tree structure



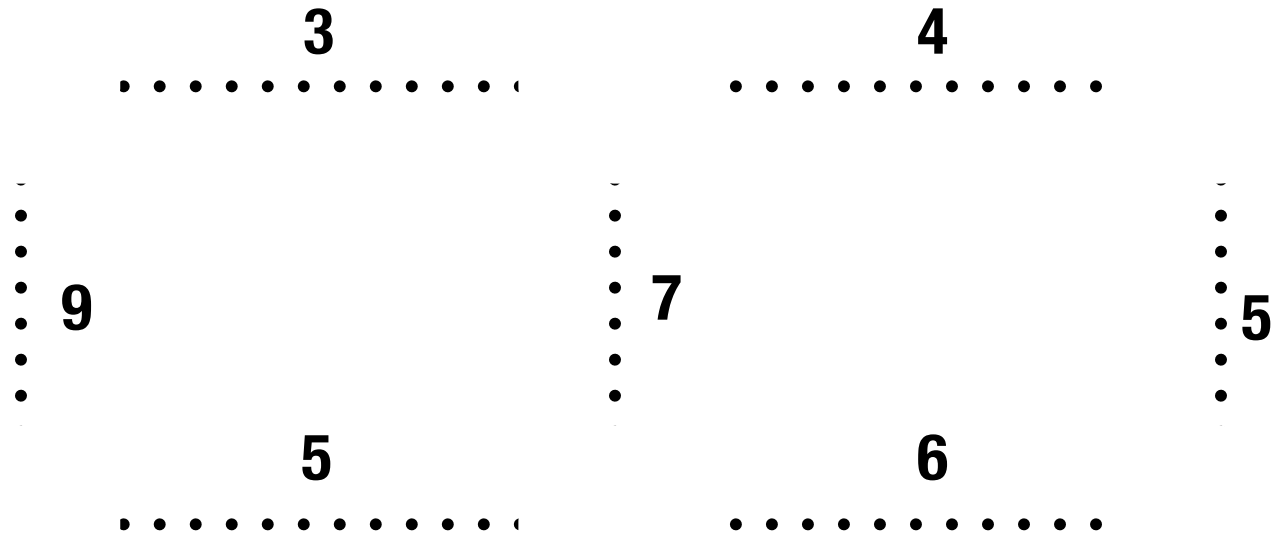
# Prim's Algorithm

- \* Grow the tree like Dijkstra's Algorithm
- \* Dijkstra's: grow the set of vertices to which we know the shortest path
- \* Prim's: grow the set of vertices we have added to the minimum tree
- \* Store shortest edge  $\mathbf{D}[\ ]$  from each node to tree

# Prim's Algorithm

- \* Start with a single node tree, set distance of adjacent nodes to edge weights, infinite elsewhere
- \* Repeat until all nodes are in tree:
  - \* Add the node **v** with shortest known distance
  - \* Update distances of adjacent nodes **w**:  
 $\mathbf{D}[w] = \min(\mathbf{D}[w], \text{weight}(\mathbf{v}, \mathbf{w}))$

# Prim's Example



# Implementation Details

- \* Store “previous node” like Dijkstra’s Algorithm; backtrack to construct tree after completion
- \* Of course, use a priority queue to keep track of edge weights. Either
  - \* keep track of nodes inside heap & decreaseKey
  - \* or just add a new copy of the node when key decreases, and call deleteMin until you see a node not in the tree

# Prim's Algorithm Justification

- \* At any point, we can consider the set of nodes in the tree  $T$  and the set outside the tree  $Q$
- \* Whatever the MST structure of the nodes in  $Q$ , at least one edge must connect the MSTs of  $T$  and  $Q$
- \* The greedy edge is just as good structurally as any other edge, and has minimum weight

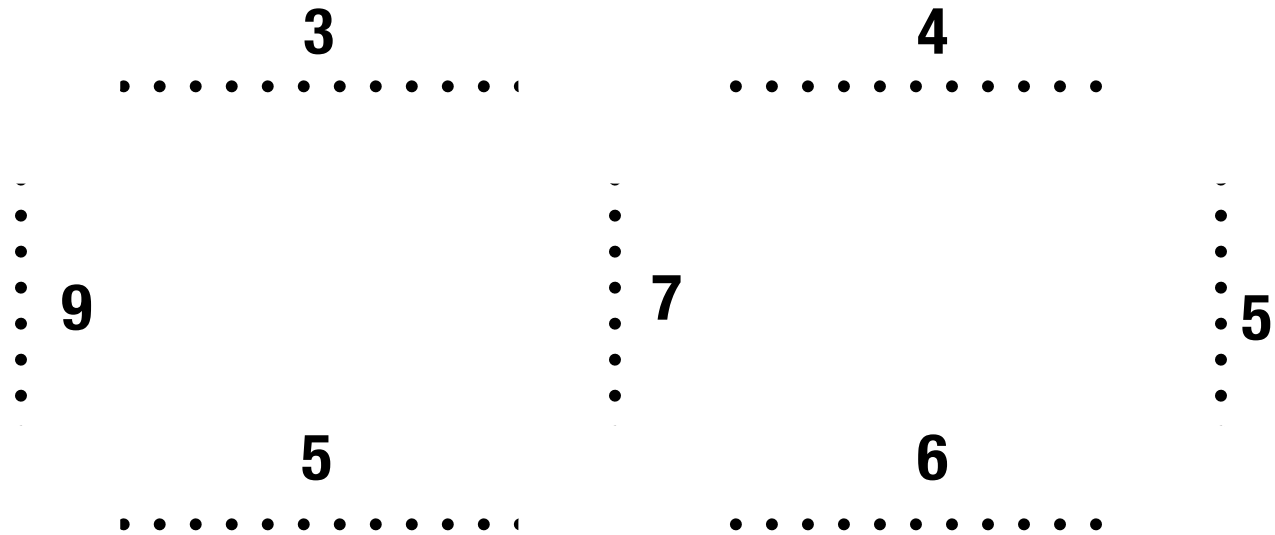
# Prim's Running Time

- \* Each stage requires one deleteMin  $O(\log |V|)$ , and there are exactly  $|V|$  stages
- \* We update keys for each edge, updating the key costs  $O(\log |V|)$  (either an insert or a decreaseKey)
- \* Total time:  $O(|V| \log |V| + |E| \log |V|) = O(|E| \log |V|)$

# Kruskal's Algorithm

- \* Somewhat simpler conceptually, but more challenging to implement
- \* Algorithm: repeatedly add the shortest edge that does not cause a cycle until no such edges exist
- \* Each added edge performs a union on two trees; perform unions until there is only one tree
- \* Need special ADT for unions  
(Disjoint Set... we'll cover it later)

# Kruskal's Example



# Kruskal's Justification

- \* At each stage, the greedy edge  $e$  connects two nodes  $v$  and  $w$
- \* Eventually those two nodes must be connected;
  - \* we must add an edge to connect trees including  $v$  and  $w$
- \* We can always use  $e$  to connect  $v$  and  $w$ , which must have less weight since it's the greedy choice

# Kruskal's Running Time

- \* First, buildHeap costs  $O(|E|)$
- \* Each edge, need to check if it creates a cycle (costs  $O(\log V)$ )
- \* In the worst case, we have to call  $|E|$  deleteMins
- \* Total running time  $O(|E| \log |E|)$ ; but  $|E| \leq |V|^2$   
 $O(|E| \log |V|^2) = O(2|E| \log |V|) = O(|E| \log |V|)$

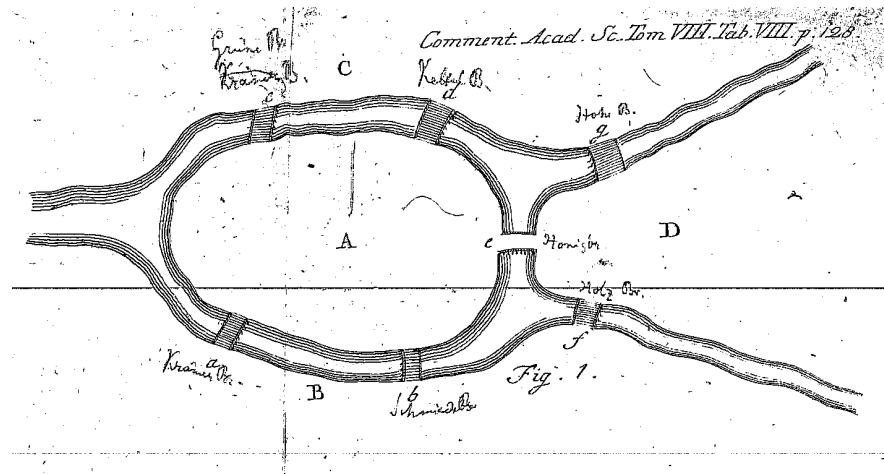
# MST Wrapup

- \* Connect all nodes in graph using minimum weight tree
- \* Two greedy algorithms:
  - \* Prim's: similar to Dijkstra's. Easier to code
  - \* Kruskal's: easy on paper

# Depth First Search

- \* Level-order  $\leftrightarrow$  Breadth-first Search  
Preorder  $\leftrightarrow$  Depth-first Search
- \* Visit vertex  $v$ , then recursively visit  $v$ 's neighbors
- \* To avoid visiting nodes more than once in a cyclic graph, mark visited nodes,
- \* and only recurse on unmarked nodes

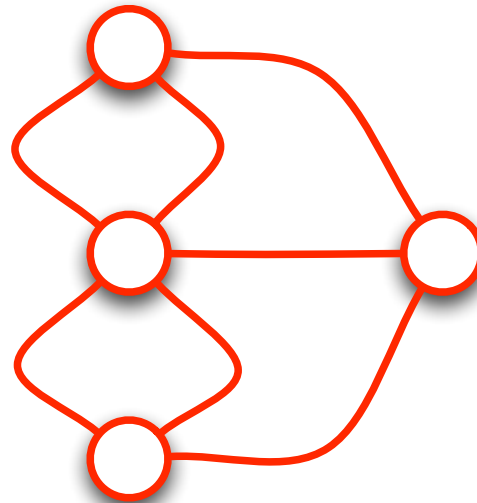
# The Seven Bridges of Königsberg



<http://math.dartmouth.edu/~euler/docs/originals/E053.pdf>

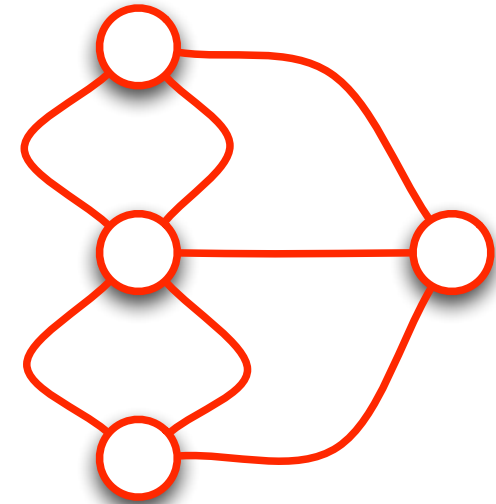
- ✳ Königsburg Bridge Problem: can one walk across the seven bridges and never cross the same bridge twice?

# Euler Paths and Circuits



- \* Euler path – a (possibly cyclic) path that crosses each edge exactly once
- \* Euler circuit - an Euler path that starts and ends on the same node

# Euler's Proof

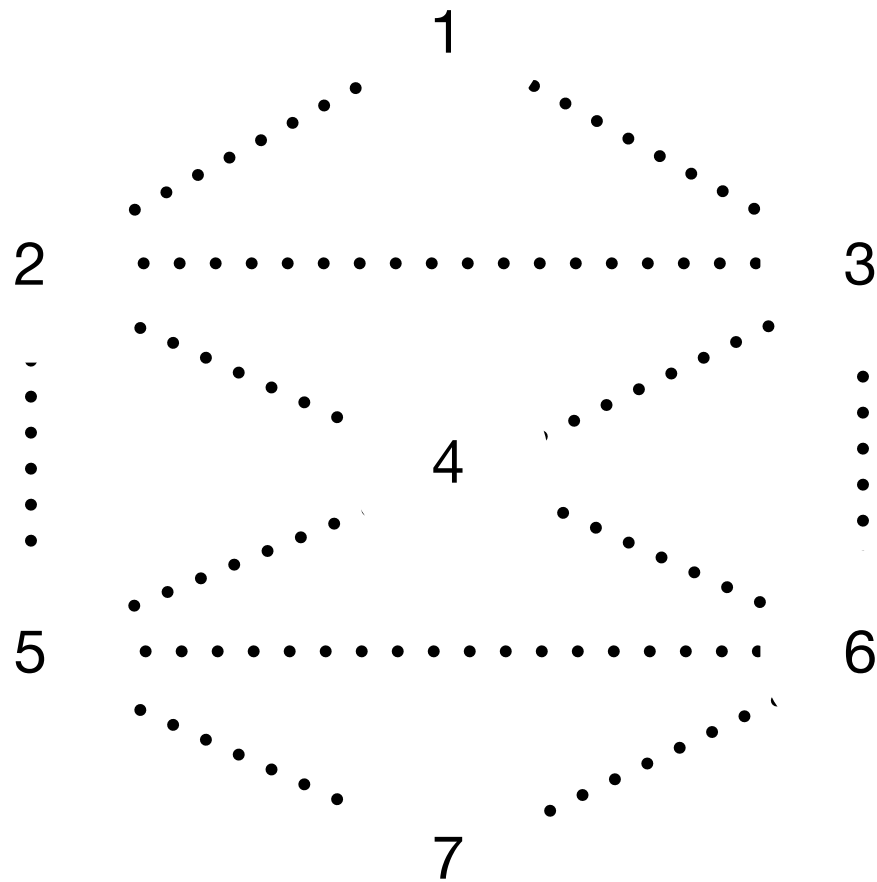


- \* Does an Euler path exist? **No**
- \* Nodes with an odd degree must either be the start or end of the path
- \* Only one node in the Königsberg graph has odd degree; the path cannot exist
- \* What about an Euler circuit?

# Finding an Euler Circuit

- \* Run a partial DFS; search down a path until you need to backtrack (mark edges instead of nodes)
- \* At this point, you will have found a circuit
- \* Find first node along the circuit that has unvisited edges; run a DFS starting with that edge
- \* Splice the new circuit into the main circuit, repeat until all edges are visited

# Euler Circuit Example



# Euler Circuit Running Time

- \* All our DFS's will visit each edge once, so at least  $O(|E|)$
- \* Must use a linked list for efficient splicing of path, so searching for a vertex with unused edge can be expensive
- \* but cleverly saving the last scanned edge in each adjacency list can prevent having to check edges more than once, so also  $O(|E|)$

# Hamiltonian Cycle

- \* Now that we know how to find Euler circuits efficiently, can we find Hamiltonian Cycles?
- \* Hamiltonian cycle - path that visits each *node* once, starts and ends on same node

# Reading

- \* Weiss 9.5 (MST - today's material)
- \* Weiss 9.6 (DFS - today's material)
- \* Weiss 9.7 (P vs. NP - Monday)