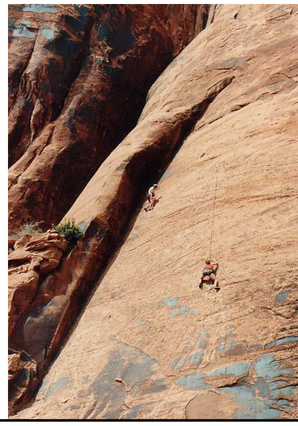
CSE 326: Data Structures
lecture 17

# Really, *Really* Hard Problems

Winter Quarter 2008



---

## Today's Agenda

- Solving two pencil-on-paper puzzles
  - Euler Circuits
  - Hamiltonian circuits
- Hamiltonian circuits and NP complete problems
- The NP = P problem
  - Your chance to win a Turing award!
- Weiss Chapter 9.7

$e^{i\pi} = -1$

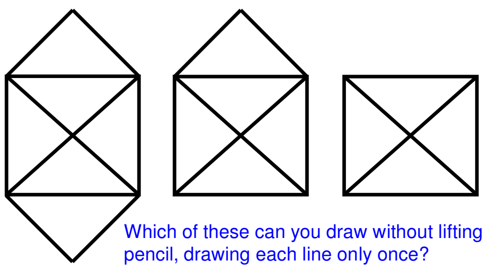
L. Euler (1707-1783)

W. R. Hamilton (1805-1865)



---

## It's Puzzle Time!

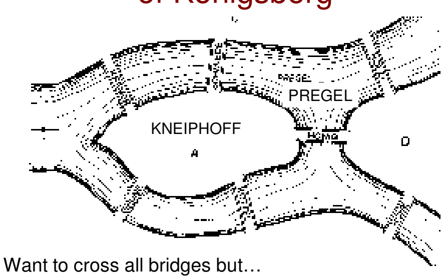


Which of these can you draw without lifting your pencil, drawing each line only once?
Can you start and end at the same point?



---

## Historical Puzzle: Seven Bridges of Königsberg

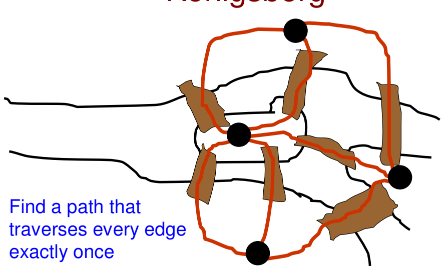



Want to cross all bridges but…
Can cross each bridge only once



---

## A "Multigraph" for the Bridges of Königsberg



Find a path that traverses every edge exactly once



---

## Euler Circuits and Tours

- Euler tour: a path through a graph that *visits each edge exactly once*
- Euler circuit: an Euler tour that *starts and ends at the same vertex*
- Named after Leonhard Euler (1707-1783), who cracked this problem and founded graph theory in 1736
- Some observations for undirected graphs:
  - An Euler circuit exists *iff* the graph is connected and each vertex has even degree (= # of edges on the vertex)
  - An Euler tour exists *iff* the graph is connected and either all vertices have even degree or exactly two have odd degree

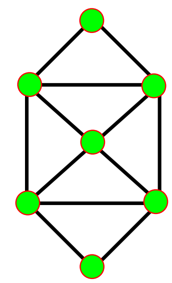
## Euler Circuit Problem

- Problem: Given an undirected graph G, find an Euler circuit
- How can we check if one exists in linear time?
- Given that an Euler circuit exists, how do we *construct* an Euler circuit for G?



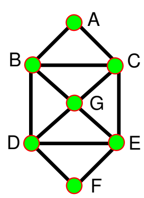
## Finding Euler Circuits

- Given a graph G = (V,E), find an Euler circuit in G
  - Can check if one exists in O(|V|+|E|) time (check degrees)
- Basic Euler Circuit Algorithm:
  1. Do an edge walk from a start vertex until you are back to the start vertex. You never get stuck because of the even degree property.
  2. The walk is removed leaving several components each with the even degree property. Recursively find Euler circuits for these.
  3. Splice all these circuits into an Euler circuit
- Running time = O(|V| + |E|)





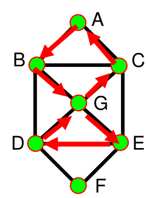
## Euler Circuit Example




Euler(A) :



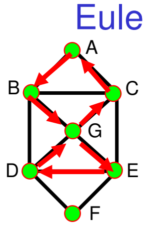
## Euler Circuit Example



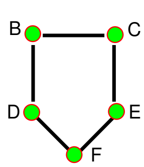

Euler(A) :
A B G E D G C A



## Euler Circuit Example



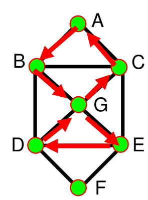

Euler(A) :
A B G E D G C A



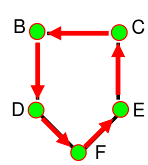

Euler(B)



## Euler Circuit Example

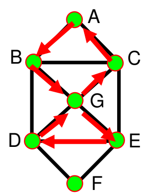


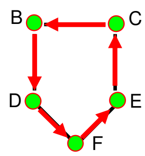

Euler(A) :
A B G E D G C A




Euler(B):
B D F E C B

## Euler Circuit Example




Euler(A) :
A B G E D G C A

Euler(B):
B D F E C B

Splice

A B D F E C B G E D G C A

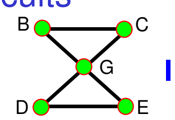


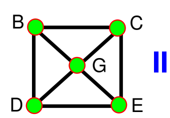
## Data Structure?



## Euler with a Twist: Hamiltonian Circuits

- Euler circuit: A cycle that goes through each *edge* exactly once
- Hamiltonian circuit: A cycle that goes through each *vertex* exactly once
- Does graph **I** have:
  - An Euler circuit?
  - A Hamiltonian circuit?
- Does graph **II** have:
  - An Euler circuit?
  - A Hamiltonian circuit?





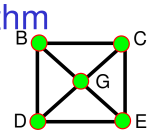

## Finding Hamiltonian Circuits in Graphs

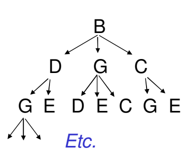
- Problem: Find a Hamiltonian circuit in a graph G
  - Sub-problem: Does G contain a Hamiltonian circuit?
  - No known easy algorithm for checking this…
- One solution: Search through *all paths* to find one that visits each vertex exactly once
  - Can use your favorite graph search algorithm (DFS!) to find various paths
- This is an *exhaustive search* ("brute force") algorithm
- Worst case need to search all paths
  - How many paths??



## Analysis of our Exhaustive Search Algorithm

- Worst case need to search all paths
  - How many paths?
- Can depict these paths as a *search tree*
- Let the average branching factor of each node in this tree be B
- |V| vertices, each with ≈ B branches
- Total number of paths ≈ B·B·B … ·B = $O(B^{|V|})$
- Worst case Exponential time!

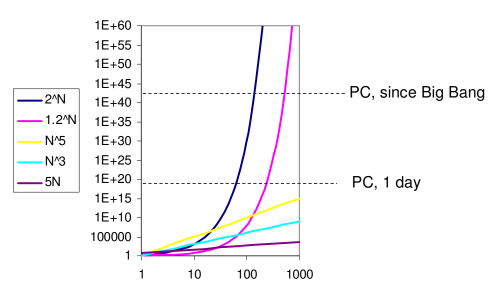



*Search tree* of paths from B



## Exponential Time



## Review: Polynomial versus Exponential Time

- Most of our algorithms so far have been O(log N), O(N), O(N log N) or $O(N^2)$ running time for inputs of size N
  - These are all *polynomial time* algorithms
  - Their running time is $O(N^k)$ for some k > 0
- Exponential time $B^N$ is asymptotically worse than any polynomial function $N^k$ for any k



## The Complexity Class P

- The set P is defined as the set of all problems that can be solved in *polynomial worse case time*
  - Also known as the *polynomial time* complexity class
  - All *problems* that have some *algorithm* whose running time is $O(N^k)$ for some *k*
- Examples of problems in P: sorting, shortest path, Euler circuit, *etc.*



## The Complexity Class NP

- *Definition*: NP is the set of all problems for which a given *candidate solution* can be *tested* in polynomial time
- Example of a problem in NP:
  - Hamiltonian circuit problem: *Why is it in NP?*



## The Complexity Class NP

- *Definition*: NP is the set of all problems for which a given *candidate solution* can be *tested* in polynomial time
- Example of a problem in NP:
  - Hamiltonian circuit problem: *Why is it in NP?*
    - Given a candidate path, can test in linear time if it is a Hamiltonian circuit – just check if all vertices are visited exactly once in the candidate path

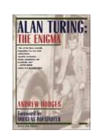


## Why NP?

- NP stands for *Nondeterministic Polynomial time*
  - Why "nondeterministic"? Corresponds to algorithms that can guess a solution (if it exists) the solution is then verified to be correct in polynomial time
  - Nondeterministic algorithms don't exist – purely theoretical idea invented to understand how hard a problem could be
- Examples of problems in NP:
  - *Hamiltonian circuit:* Given a candidate path, can test in linear time if it is a Hamiltonian circuit
  - *Satisfiability:* Given a circuit made out of AND, OR, NOT gates: is there an input that makes it output "1"?
  - *All problems that are in P (why?)*



## Your Chance to Win a Turing Award




Alan Turing (1912-1954)

- It is generally believed that $P \neq NP$, *i.e.* there are problems in NP that are not in P
  - But no one has been able to show even one such problem!
  - This is the fundamental open problem in theoretical computer science
  - Nearly everyone has given up trying to prove it. Instead, theoreticians prove theorems about what follows once we assume $P \neq NP$ !

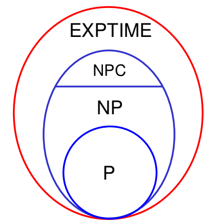
# NP-Complete Problems

- The "hardest" problems in NP are called **NP-complete**
  - › If any NP-complete problem is in P, then all of NP is in P
- Examples:
  - › Hamiltonian circuit
  - › Traveling salesman: find the shortest path that visits all nodes in a weighted graph (okay to repeat edges & nodes)
  - › Graph coloring: can the vertices of a graph be colored using K colors, such that no two adjacent vertices have the same color?
  - › Crossword puzzle construction: can a given set of 2N words, each of length N, be arranged in an NxN crossword puzzle?



# P, NP, and Exponential Time Problems

- All **currently known** algorithms for NP-complete problems run in **exponential** worst case time
  - › Finding a polynomial time algorithm for any NPC problem would mean:
- Diagram depicts relationship between P, NP, and EXPTIME (class of problems that **provably require** exponential time to solve)




It is believed that
$P \neq NP \neq EXPTIME$



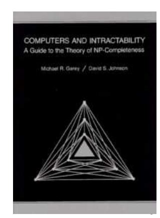
# Coping with NP-Completeness

1. Settle for algorithms that are fast on average: Worst case still takes exponential time, but doesn't occur very often.
   But some NP-Complete problems are also average-time NP-Complete!
2. Settle for fast algorithms that give near-optimal solutions: In traveling salesman, may not give the cheapest tour, but maybe good enough.
   But finding even approximate solutions to some NP-Complete problems is NP-Complete!
3. Just get the exponent as low as possible! Much work on exponential algorithms for satisfiability: in practice can often solve circuits with 1,000+ inputs
   But even $2^{n/100}$ will eventual hit the exponential curve!



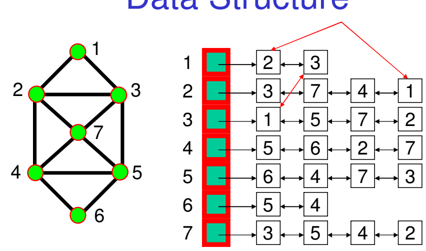
# Great Quick Reference

- *Computers and Intractability: A Guide to the Theory of NP-Completeness*, by Michael S. Garey and David S. Johnson





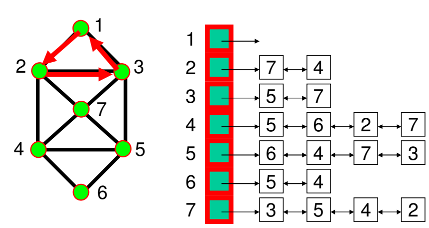

# Data Structure



Path 1



# Data Structure



Path 1 → 2 → 3 → 1

Time = O(n+e)