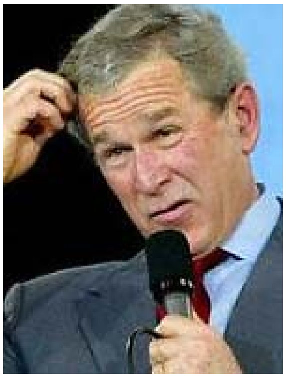


Design Patterns & Some Unresolved Problems

Lecture 26
CS211 – Fall 2006

Announcements

- Final Exam
  - Wednesday, Dec 13
  - 7:00 - 9:30 pm
  - Uris Hall, Auditorium
- Review Session
  - To be determined
- Check your final exam schedule!
- For exam conflicts:
  - Notify Kelly Patwell (patwell@cs.cornell.edu)
  - You must provide
    - Your entire exam schedule
    - Include the course numbers
- Definition of exam conflict:
  - Two exams at the same time or
  - *Three* or more exams within 24 hours

Late-Breaking Announcements

- Thinking about a Masters of Engineering degree?
  - Come to the next ACSU general meeting
  - Wednesday, November 29th at 4:45pm in Phillips 203
  - Professor Bailey, director of the Computer Science M. Eng. program, will discuss CS M. Eng. opportunities
  - As always, pizza will be served!
- Jealous of the glamorous life of CS consultants?
  - We're recruiting next-semester consultants for CS100 and CS211
  - Interested students should fill out an application, available in 303 Upson Hall

Design Patterns

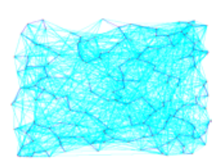
- Design Patterns: A catalog of common interactions between objects that programmers have frequently found useful
  - Influential book:
    Design Patterns: Elements of Reusable Software,
    Gamma, Helm, Johnson and Vlissides (1995)
- Design patterns are often divided into 3 groups:
  - Creational patterns
    - Used to create objects
  - Structural patterns
    - Composing groups of objects to build more complex structures
  - Behavioral patterns
    - Communication & flow control

A Few Design Patterns in Java

- Design patterns can be used with any Object-Oriented language
- Chain of Responsibility Pattern
  - A request is passed along a "chain" of classes until one of the classes can handle it
  - Java example: The inheritance structure itself; a method-call is passed up the inheritance chain until the first parent class containing that method is found
- Adapter pattern
  - Provides an alternate interface to a class
  - Java example: MouseAdapter is a class used in Java to simplify the MouseListener interface
- Iterator Pattern
  - A standard interface for moving through a collection
  - Java example: Iterator interface in the JCF

Complexity of Bounded-Degree Euclidean MST?

- The Euclidean MST (Minimum Spanning Tree) problem:
  - Given n points in the plane, determine the MST
  - Can be solved in O(n log n) time by first building the Delaunay Triangulation



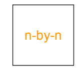
- Bounded-degree version:
  - Given n points in the plane determine the MST where each vertex has degree ≤ k
    - Known to be NP-hard for k=3 [Papadimitriou & Vazirani 84]
    - O(n log n) algorithm for k=5 (or greater)
      - Can show Euclidean MST has degree ≤ 5
    - Unknown for k=4

Runtime for Euclidean MST in R^d?

- Given n points in dimension d, determine the MST
  - Is there an algorithm with runtime close to the $\Omega(n \log n)$ lower bound?
- Can solve in time $O(n \log n)$ for d=2
- For large d, it appears that runtime approaches $O(n^2)$
- Best algorithms for general graphs run in time linear in m = number of edges
  - But for Euclidean distances on points, the number of edges is n(n-1)/2

$O(n^2)$ Time for X+Y Sorting?

How long does it take to a sort an n-by-n table of numbers?



+	1	3	5	8
2	3	5	7	10
10	11	13	15	18
12	13	15	17	20
14	15	17	19	22

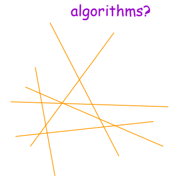
- $O(n^2 \log n)$ because there are n^2 numbers in the table
- What if it's an *addition table*?
  - Shouldn't it be easier to sort than an arbitrary set of n^2 numbers?
- There is a technique [Fredman 76] that uses just $O(n^2)$ *comparisons*
  - But it uses $O(n^2 \log n)$ time [Lambert 92] to decide *which* comparisons to use
- This problem is closely related to the problem of sorting the vertices of a line arrangement

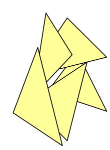
O(n log n) Time for ShellSort?

- Is there a sequence of ShellSort step-sizes for which ShellSort runs in time O(n log n)?
- There *is* a sequence for which ShellSort runs in time $O(n \log^2 n)$
  - Pratt sequence: numbers of the form $2^p 3^q$ arranged in order

3SUM in Subquadratic Time?

- Given a set of n integers, are there three that sum to zero?
  - $O(n^2)$ algorithms are easy (e.g., use a hashtable)
  - Are there better algorithms?

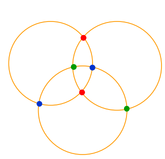




- This problem is closely related to many other problems [Gajentaan & Overmars 95]
  - Given n lines in the plane, are there 3 lines that intersect in a point?
  - Given n triangles in the plane, does their union have a hole?

Great-Circle Graph 3-Colorable?

- Build a graph by drawing great-circles on a sphere
  - Create a vertex for each intersection
  - Assume no three great circles intersect in a point
- Is the resulting graph 3-colorable?
- All arrangements for up to 11 great circles have been verified as 3-colorable
- For *general* circles on the sphere (or for circles on the plane) the graph can require 4 colors



The Big Question: Is P=NP?

- P represents problems that can be *solved* in polynomial time
  - These problems are said to be *tractable*
  - Problems that are not in P are said to be *intractable*
- NP represents problems that, for a *given solution*, the solution can be *checked* in polynomial time
- For ease of comparison, problems are usually stated as yes-or-no questions
- Examples
  - Given a weighted graph G and a bound k, does G have a spanning tree of size ≤ k?
    - This is in P because we have an algorithm for the MST with runtime O(m + n log n)
  - Given graph G, does G have a cycle that visits all vertices?
    - This is in NP because, given a possible solution, we can check in polynomial time that it's a cycle and that it visits all vertices

Current Status: P vs. NP

- It's easy to show that $P \subseteq NP$
- Most researchers believe that $P \neq NP$
  - But at present, there is no proof
  - We do have a large collection of *NP-complete problems*
    - If *any* NP-complete problem has a polynomial time algorithm then they *all* do
- Definition: A problem B is *NP-complete* if, by making use of an *imaginary* fast subroutine for B, any problem in NP could be solved in polynomial time
  - [Cook 1971] showed a particular problem to be NP-complete
  - [Karp 1972] showed that many useful problems are NP-complete

NP-Complete Problems

- Graph coloring: Given graph G and bound k, is G k-colorable?
- Planar 3-coloring: Given planar graph G, is G 3-colorable?
- Traveling Salesman: Given weighted graph G and bound k, is there a cycle of cost ≤ k that visits each vertex exactly once
- Hamiltonian Cycle: Give graph G, is there a cycle that visits each vertex exactly once?
- What if you really *need* an algorithm for an NP-complete problem?
  - Some special cases can be solved in polynomial time
    - If you're lucky, you have such a special case
  - Otherwise, once a problem is shown to be NP-complete, the best strategy is to start looking for an approximation
- For a while, a new proof showing a problem NP-complete was enough for a a paper
  - Nowadays, no one is interested unless the result is somehow unexpected