Lecture 27: The Grand Finale

- Agenda for the final class:
  - P, NP, and NP-completeness
  - The NP =? P problem
    - Major extra-credit problem
  - Final Review
    - Summary of what you've learned in this course
  - End-of-quarter M.o.M. party
    - Munch on munchies as you leave…

From Last Time: 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 – contains problems whose time complexity is $O(N^k)$ for some k
- Examples of problems in P: searching, sorting, topological sort, single-source 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
  - Suppose someone gives you a solution – can it be tested in polynomial time? (testing is easier than solving it)
- Example of a problem in NP:
  - Our new friend, the 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 (except start/finish vertex)

Why NP?

- NP stands for Nondeterministic Polynomial time
  - Why “nondeterministic”? Corresponds to algorithms that can search all possible solutions in parallel and pick the correct one → each solution can be checked 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
  - Sorting: Can test in linear time if a candidate ordering is sorted
  - Sorting is also in P. Are any other problems in P also in NP?

More revelations about NP

- Sorting is in P. Are any other problems in P also in NP?
  - YES! All problems in P are also in NP → P ⊆ NP
    - If you can solve a problem in polynomial time, can definitely verify a solution in polynomial time
- Question: Are all problems in NP also in P?
  - Is NP ⊆ P?

Your chance to win a Turing award: P = NP?

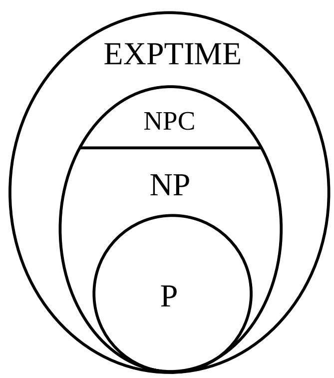
- Nobody knows whether NP ⊆ P
  - Proving or disproving this will bring you instant fame!
- It is generally believed that P ≠ NP i.e. there are problems in NP that are not in P
  - But no one has been able to show even one such problem
- A very large number of problems are in NP (such as the Hamiltonian circuit problem)
  - No one has found fast (polynomial time) algorithms for these problems
  - On the other hand, no one has been able to prove such algorithms don't exist (i.e. that these problems are not in P)!

NP-complete problems

- The "hardest" problems in NP are called NP-complete (NPC) problems
- Why "hardest"? A problem X is NP-complete if:
  1. X is in NP and
  2. *any* problem Y in NP can be *converted to* X in polynomial time, such that *solving X also provides a solution for Y* → Can use algorithm for X as a *subroutine* to solve Y
- Thus, if you find a poly time algorithm for just one NPC problem, all problems in NP can be solved in poly time
- Example: The Hamiltonian circuit problem can be shown to be NP-complete (not so easy to prove!)

P, NP, and Exponential Time Problems

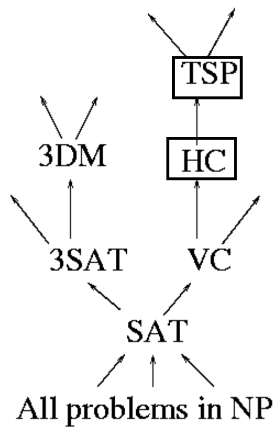
- All algorithms for NP-complete problems so far have tended to run in nearly exponential worst case time
  - But this doesn't mean fast sub-exponential time algorithms don't exist! Not proven yet…
- Diagram depicts relationship between P, NP, and EXPTIME (class of problems that require exponential time to solve)



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

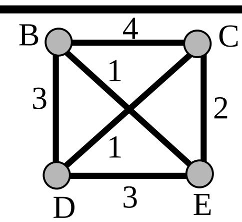
The “graph” of NP-completeness

- Cook first showed (in 1971) that satisfiability of Boolean formulas (SAT) is NP-complete
- Hundreds of other problems (from scheduling and databases to optimization theory) have since been shown to be NPC
- How? By showing an algorithm that converts a known NPC problem to your pet problem in poly time → then, your problem is also NPC!

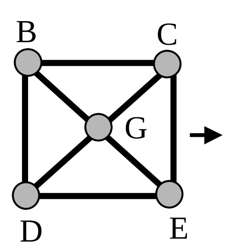


Showing NP-completeness: An example

- Consider the **Traveling Salesperson (TSP) Problem**: Given a <u>fully connected</u>, <u>weighted</u> graph G = (V,E), is there a cycle that visits all vertices exactly once and <u>has total cost ≤ K</u>?
- TSP is in NP (why?)
- Can we show TSP is NP-complete?
  - Hamiltonian Circuit (HC) is NPC
  - Can show TSP is also NPC if we can convert any input for HC to an input for TSP in poly time



Cycle with cost ≤ 8 → BDCEB

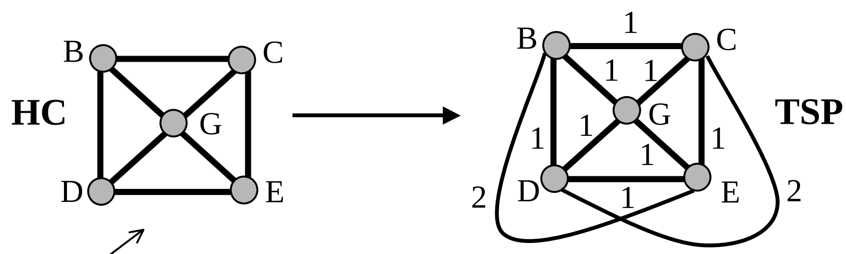


→ Convert to input for TSP

Input for HC

TSP is NP-complete!

- We can show TSP is also NPC if we can convert any input for HC to an input for TSP in poly time. Here's one way:



This graph has a Hamiltonian circuit iff this fully-connected graph has a TSP cycle of total cost ≤ K, where K = |*V*| (here, K = 5)

Coping with NP-completeness

- Given that it is difficult to find fast algorithms for NPC problems, what do we do?
- Alternatives:
  1. Dynamic programming: Avoid repeatedly solving the same subproblem – use table to store results (see Chap. 10)
  2. Settle for algorithms that are fast on average: Worst case still takes exponential time, but doesn't occur very often
  3. Settle for fast algorithms that give near-optimal solutions: In TSP, may not give the cheapest tour, but maybe good enough
  4. Try to get a "wimpy exponential" time algorithm: It's okay if running time is $O(1.00001^N)$ – bad only for N > 1,000,000

Yawn…What does all this have to do with data structures and programming?

✦ <u>Top 5 reasons to know and understand NP-completeness:</u>

5. What if there's an NP-completeness question in the final?
4. When you are having a tough time programming a fast algorithm for a problem, you could show it is NP-complete
3. When you are having a tough time programming a fast algorithm for a problem, you could just say it is NPC (and many will believe you (yes, it's a sad state of affairs))
2. When you are at a cocktail party, you can impress your friends with your profound knowledge of NP-completeness
1. Make money with new T-shirt slogan: "And God said: P=NP"

Final Review

("We've covered way too much in this course…

What do I <u>really</u> need to know?")

Final Review: What you need to know

- Basic Math
  - Logs, exponents, summation of series
  - Proof by induction

$$\sum_{i=1}^{N} i = \frac{N(N+1)}{2}$$

$$\sum_{i=0}^{N} A^i = \frac{A^{N+1}-1}{A-1}$$

- Asymptotic Analysis
  - Big-oh, little-oh, Theta and Omega
  - Know the definitions and how to show f(N) is big-oh/little-oh/Theta/Omega of (g(N))
  - How to estimate Running Time of code fragments
    - E.g. nested "for" loops
- Recurrence Relations
  - Deriving recurrence relation for run time of a recursive function
  - Solving recurrence relations by expansion to get run time

Final Review: What you need to know

- Lists, Stacks, Queues
  - Brush up on ADT operations – Insert/Delete, Push/Pop etc.
  - Array versus pointer implementations of each data structure
  - Header nodes, circular, doubly linked lists
- Trees
  - Definitions/Terminology: root, parent, child, height, depth etc.
  - Relationship between depth and size of tree
    - Depth can be between O(log N) and O(N) for N nodes

Final Review: What you need to know

- Binary Search Trees
  - How to do Find, Insert, Delete
    - Bad worst case performance – could take up to O(N) time
  - AVL trees
    - Balance factor is +1, 0, -1
    - Know single and double rotations to keep tree balanced
    - All operations are O(log N) worst case time
  - Splay trees – good amortized performance
    - A single operation may take O(N) time but in a sequence of operations, average time per operation is O(log N)
    - Every Find, Insert, Delete causes accessed node to be moved to the root
    - Know how to zig-zig, zig-zag, etc. to "bubble" node to top
  - B-trees: Know basic idea behind Insert/Delete

Final Review: What you need to know

- Priority Queues
  - Binary Heaps: Insert/DeleteMin, Percolate up/down
    - Array implementation
    - BuildHeap takes only O(N) time (used in heapsort)
  - Binomial Queues: Forest of binomial trees with heap order
    - Merge is fast – O(log N) time
    - Insert and DeleteMin based on Merge
- Hashing
  - Hash functions based on the mod function
  - Collision resolution strategies
    - Chaining, Linear and Quadratic probing, Double Hashing
  - Load factor of a hash table

Final Review: What you need to know

- <u>Sorting Algorithms</u>: Know run times and how they work
  - Elementary sorting algorithms and their run time
    - Bubble sort, Selection sort, Insertion sort
  - Shellsort – based on several passes of Insertion sort
    - Increment Sequence
  - Heapsort – based on binary heaps (max-heaps)
    - BuildHeap and repeated DeleteMax's
  - Mergesort – recursive divide-and-conquer, uses extra array
  - Quicksort – recursive divide-and-conquer, Partition in-place
    - fastest in practice, but $O(N^2)$ worst case time
    - Pivot selection – median-of-three works best
  - Know which of these are stable and in-place
  - Lower bound on sorting, bucket sort, and radix sort

Final Review: What you need to know

- <u>Disjoint Sets and Union-Find</u>
  - Up-trees and their array-based implementation
  - Know how Union-by-size and Path compression work
  - No need to know run time analysis – just know the result:
    - Sequence of M operations with Union-by-size and P.C. is $\Theta(M\ \alpha(M,N))$ – basically $\Theta(1)$ amortized time per op
- <u>Graph Algorithms</u>
  - Adjacency matrix versus adjacency list representation of graphs
  - Know how to Topological sort in $O(|V| + |E|)$ time using a queue
  - Breadth First Search (BFS) for unweighted shortest path

Final Review: What you need to know

- Graph Algorithms (cont.)
  - Dijkstra's shortest path algorithm – greed works!
    - Know how a priority queue can speed up the algorithm
  - Depth First Search (DFS)
  - Minimum Spanning trees: Know the 2 greedy algorithms
    - Prim's algorithm – similar to Dijkstra's algorithm
    - Kruskal's algorithm
      - Know how it uses a priority queue and Union/Find
    - Euler versus Hamiltonian circuits – difference in run times
    - Know what P, NP, and NP-completeness mean
      - How one problem can be "reduced" to another (e.g. input to HC can be transformed into input for TSP)

Final Exam:

Where: This room

When: Wednesday, June 6, 2:30-4:20pm

To Do:

Go over sample final exam on web site

Prepare, prepare, prepare (for the final)

Have a great summer!