Lecture 24: How to become Famous with P and NP

- Agenda for today's class:
  - The complexity class P
  - The complexity class NP
  - NP-completeness
  - The P =? NP problem
    - Major extra-credit problem (due: whenever)
  - Fun with Golf Pencils (fill out Evals)

From Last Time: Polynomial vs. Exponential Running Time

N	log N	N log N	N^2	2^N
1	0	0	1	2
2	1	2	4	4
4	2	8	16	16
10	3	30	100	1024
100	7	700	10,000	1,000,000,000,000,00, 000,000,000,000,000
1000	10	10,000	1,000,000	Fo'gettaboutit!
1,000,000	20	20,000,000	1,000,000,000,000	ditto
1,000,000,000	30	30,000,000,000	1,000,000,000,000,000,000	mega ditto plus

Polynomial versus Exponential Time

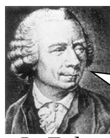
- Most of our algorithms so far have been O(log N), O(N), O(N log N) or $O(N^2)$ running time for inputs of size N
  - These are all *polynomial time* algorithms
  - Their running time is $O(N^k)$ for some $k > 0$
- Exponential time B^N is asymptotically *worse than any* polynomial function N^k for any k
  - For any k, N^k is $o(B^N)$ for any constant $B > 1$
- Polynomial time algorithms are generally regarded as "fast" algorithms – these are the kind we want!
- Exponential time algorithms are generally inefficient – avoid these!

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*
- P contains all problems for which algorithms exist whose worst case running time is **$O(N^k)$** for some **k**
- Thus, P = class of "easy" or "tractable" problems for which fast (i.e. polynomial time) algorithms exist

What's in P?

- Examples of problems in P:
  - Searching
  - Sorting
  - Topological sort
  - Single-source shortest path
  - Euler circuit, etc.



Finding my circuits is easy and in P!

L. Euler
(1707-1783)

Well, what about mine?

W. R. Hamilton
(1805-1865)

Introducing…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 (e.g., by guessing). <u>You should be able to test or verify it in polynomial time</u>
- Note: Testing a given "solution" is typically easier than solving or finding the correct solution!
  - Finding the correct solution may take exponential time but checking is usually much easier and faster

What's the deal with the name 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 should be "checkable" in polynomial time
- Nondeterministic algorithms don't exist – purely theoretical idea invented to understand how hard a problem could be

A Nondeterministic Algorithm for Searching

- Problem: Given a list of integers $A_1, \ldots, A_N$, is integer X in the list?
- Nondeterministic Algorithm:
  1. Guess an index i between 1 and N
  2. If $A_i = X$, then Output "Yes"
- Alternate description:
  - Nondeterministic algorithm produces N "parallel processes"
  - Each process checks if its $A_i = X$
  - Algorithm outputs "Yes" if at least one process outputs "Yes"

Nondeterministic Algorithm for Searching

Guess Index

i=1 i=2 ……… i=N

Check if $A_1 = X$

Check if $A_N = X$?

Output "Yes" if **any one** of these processes outputs "Yes"

Is dis an NP algorithm?

Other problems in NP

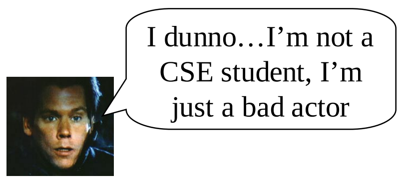
- Recall our friend from last time, the Hamiltonian circuit problem: Find a cycle that goes through each *vertex* exactly once
- Given a candidate path, can test in linear time if it is a Hamiltonian circuit
- NP algorithm for HC:
  - Guess a candidate path
  - Check if all vertices are visited exactly once in this candidate path (except start/finish vertex)
  - Can check in time polynomial in |V|

Pray tell me, why is my problem in NP?

W. R. Hamilton (1805-1865)

Other problems in NP

- Sorting: Can test in linear time if a candidate ordering is sorted
- But sorting is also in P.
  - Are any other problems in P also in NP?



The Intimate Relationship between P and NP

- Sorting is in P. Are any other problems in P also in NP?
  - YES!
  - All problems in P are also in NP i.e. $P \subseteq NP$
  - If you can solve a problem in polynomial time, can definitely verify a solution in polynomial time
- So, some problems in NP like searching, sorting, etc. are also in P.
- Question: Are all problems in NP also in P?
  - Is $NP \subseteq P$?

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

- Nobody knows whether NP $\subseteq$ P
  - Proving or disproving this will bring you instant fame!
- 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
- A very large number of problems are in NP (such as the Hamiltonian circuit problem) but not known to be in P
  - No one has found fast (polynomial time) algorithms for these problems
  - 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

  (If only 2 holds, X is said to be **NP-hard**)

Input to Y ⟶ "Converter" Algorithm ⟶ Input to X
(runs in poly time)

We say that problem Y can be reduced to X

Note: X is NP-hard if all problems in NP can be reduced to X

More on NP-complete problems

- Note that if X is NP-complete, solving X also provides a solution for all problems Y in NP
  - Just use the converter to convert input for Y to input for X and run the algorithm for X
  - Using algorithm for X as a *subroutine* to solve Y

$$\text{Input to Y} \xrightarrow{\text{Converter}} \text{Input to X} \xrightarrow{\text{Algorithm for X}} \text{Solution}$$

All problems Y in NP $\xrightarrow{\text{Convert input}}$ Algorithm for X (any NPC problem) $\longrightarrow$ Solution

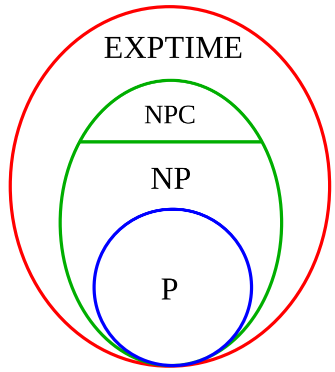
The Power of NP-completeness

All problems Y in NP $\xrightarrow{\text{Convert input}}$ Algorithm for X (any NPC problem) $\longrightarrow$ Solution

- Thus, **if you find a poly time algorithm for just one NPC problem X, 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 from scratch!)

P, NP, and Exponential Time Problems

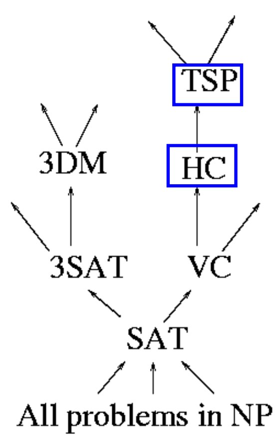
- All algorithms for NP-complete problems so far have tended to run in nearly exponential worst case time
  - But this doesn't mean fast sub-exponential time algorithms don't exist! Not proven yet…
- Diagram depicts relationship between P, NP, and EXPTIME (class of problems that can be solved within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



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

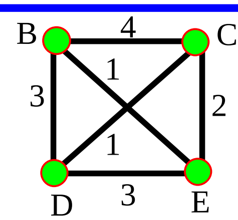
The "graph" of NP-completeness

- 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 giving an algorithm for **converting a known NPC problem to your pet problem in poly time**. Then, **<u>your problem is also NPC</u>**!



Showing NP-completeness: An Example

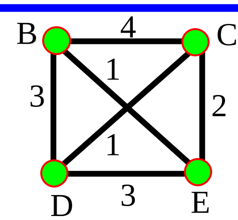
- Consider the **Traveling Salesperson (TSP) Problem**: Given a fully connected, weighted graph G = (V,E), is there a cycle that visits all vertices exactly once and has total cost ≤ K?
- TSP is in NP (why?)
- Can we show TSP is NP-complete? How?



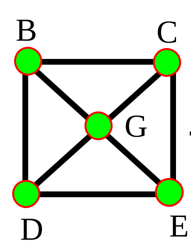
Cycle with cost ≤ 8 = BDCEB

Showing NP-completeness: An Example

- Can we show TSP is NP-complete?
  - We know 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 (Why?)
  - Because all NP problems can be reduced to HC (definition of NPC) which can now be reduced to TSP



Cycle with cost ≤ 8 = BDCEB



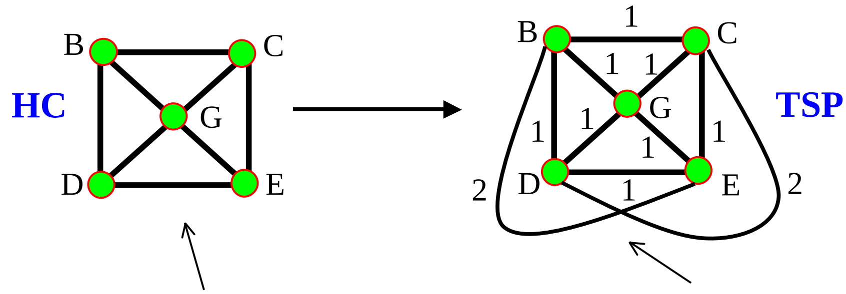
→ Convert to input for TSP

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:

Just assign weight of 1 for all existing edges and 2 to new edges



Can prove: This graph has a Hamiltonian circuit iff this fully connected graph has a TSP cycle of total cost ≤ 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?

- [Top 5 reasons to know and understand NP-completeness:]

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"