What is this class about?

- What is computation
- What is (or is not) computable?
- What is (or is not) computable given limited resources?

What is an example of something that is not computable?

> The halting problem

A Halting Problem solver (HPSolver) is a program that, on input (program $P$, input $x$) outputs "halts" if $P$ halts on input $x" loops" otherwise

Theorem: HPSolver does not exist

Proof: Suppose it existed. Consider the program Can'tSolveMe:

Can'tSolveMe (program $P$):

Run HPSolver (program $P$, input $P$)

- Let status := result of HPSolver

If status = "halts" enter infinite loop
Else Halt

What if we run Can'tSolveMe (program Can'tSolveMe)?

If it halts HPSolver (Can'tSolveMe, Can'tSolveMe) returns "halts" → then Can'tSolveMe(Can'tSolveMe) loops.
If it loops, HPSolver(Can'tSolveMe, Can'tSolveMe) returns "loops" → then Can'tSolveMe(Can'tSolveMe) halts.
Lecture 1  9/6/18

What else is not computable?
- telling if programs are equivalent
- telling running time as function of input length

These things are deemed to be heuristics 😇
But useful to reason about them, and know what is/is not possible 😊

Also important for security/cryptography 😇
- In cryptography, want bad guys to be forced to solve intractable/"impossible" problems
  ex: RSA encryption based on difficulty of factoring primes

Crypto:

"good guys" run efficient algorithms
"bad guys" would have to solve an intractable problem in order to break system

Ex: RSA
- Good guys just need to multiply integers
- Bad guys need to factor them

The Most Famous Open Problem in CompSci ~

P vs NP ↓ definitions
Lecture 1  
1/6/18

P vs. NP

P = {Boolean functions f | f(x) is computable in polynomial time ?}

NP = {Boolean functions f | the statement “f(x) = 1” can be verified in polynomial time using some extra information ?}

\[ P \subseteq NP \]

Ex: Path (directed graph G, vertex u, vertex v):

\[ \begin{cases} 1 & \text{if } \exists \text{ a path from } u \text{ to } v \text{ in } G \\ 0 & \text{otherwise} \end{cases} \]

Path \in P \rightarrow \text{You've probably done it in a programming class!}

Question: What if graph infinite? \rightarrow \text{would need diff. model of computation.}

Ex: HamPath (directed graph G, u, v):

\[ \begin{cases} 1 & \text{if } \exists \text{ a path from } u \text{ to } v \text{ that also visits every other vertex of } G \text{ exactly once} \\ 0 & \text{otherwise} \end{cases} \]

HamPath \in NP \rightarrow \text{Could verify path if I told you one}

HamPath is NP-complete \rightarrow \text{Would solve P vs NP if you knew whether HamPath were in the class P}

More examples ↓
Lecture 1  9/6/18

Ex: **Course Scheduling**
- On input $k$ non-overlapping timeslots
- List of courses
- List of students, what courses they want
- Output 1 if a schedule satisfying everyone exists

Intuition: In NP? (Seems easy to verify, also memes??)

Course Scheduling is **NP**: can verify given schedule in polytime

Ex: **Colorability**
- On input $k$ an integer
- G an undirected graph
- Output 1 if every vertex in $G$ such that no edge has endpoints of the same color

Intuition: In NP? (Seems like it!)

Course Scheduling is **NP**: can verify given coloring works

How do these problems relate?

*Can express instance of Course Scheduling as instance of Colorability (timeslots as colors, courses as vertices, students as cliques of edges)*

**Colorability** is **NP-complete** (as defined here)

Course Scheduling is **NP-complete**