Recitation 21: Complexity

0-1 Knapsack Revisited

- 0-1 Knapsack
  - Input: Knapsack with volume S, want to fill with items: item i has size s_i and value v_i.
  - Output: A subset of items (may take 0 or 1 of each) with $\sum s_i \leq S$ maximizing $\sum v_i$
  - Solvable in $O(nS)$ time via dynamic programming
- How does running time compare to input?
  - What is size of input? If numbers written in binary, input has size $O(n \log S)$ bits
  - Then $O(nS)$ runs in exponential time compared to the input
  - If numbers polynomially bounded, $S = n^{O(1)}$, then dynamic program is polynomial
  - This is called a **pseudopolynomial** time algorithm
- Is 0-1 Knapsack solvable in polynomial time when numbers not polynomially bounded?
- No if **P** $\neq$ **NP**. What does this mean? (More Computational Complexity in 6.045 and 6.046)

Decision Problems

- **Decision problem**: assignment of inputs to No (0) or Yes (1)
- Inputs are either **No instances** or **Yes instances** (i.e. satisfying instances)

Problem	Decision
Shortest Path	Does there exist a path with weight at most d?
Negative Cycle	Does there exist a negative weight cycle?
Longest Path	Does there exist a **simple** path with weight at least d?
0-1 Knapsack	Does there exist set of items with total value v or larger?
Tetris	Can you survive a given sequence of pieces?
Chess	Can a player force a win from a given board?
Halting problem	Does a given computer program terminate for a given input?

- **Algorithm/Program**: code to solve a problem, i.e. produces correct output for every input
- Problem is **decidable** if there exists a finite program to solve the problem in finite time

Decidability

- Program is finite string of bits, problem is function $p : \mathbb{N} \to \{0, 1\}$, i.e. infinite string of bits
- (# of programs $|\mathbb{N}|$, countably infinite) $\ll$ (# of problems $|\mathbb{R}|$, uncountably infinite)
- (Proof by Cantor's diagonal argument, probably covered in 6.042)
- Proves that most decision problems not solvable by any program (undecidable)
- e.g. the **halting problem** is undecidable
- Fortunately most problems we think of are algorithmic in structure and are decidable

Decidable Problem Classes

R	problems decidable in finite time	'R' comes from recursive languages
EXP	problems decidable in exponential time $2^{n^{O(1)}}$	most problems we think of are here
P	problems decidable in polynomial time $n^{O(1)}$	efficient algorithms, the focus of this class

- These sets are distinct, i.e. **P** $\subsetneq$ **EXP** $\subsetneq$ **R**

Nondeterministic Polynomial Time (NP)

- Set of decision problems with certificates that can be verified in polynomial time
- Certificate: polynomially verifiable information proving that a Yes instance is satisfying

Problem	Certificate
Shortest Path	A path with weight at most d
Negative Cycle	A negative weight cycle
Longest Path	A **simple** path with weight at least d
Knapsack	A set of items with total value at least v
Tetris	Sequence of moves that allows survival

- If you can check certificate in polynomial time, prove instance is a Yes instance by finding a satisfying certificate non-deterministically (i.e. making lucky guesses)
- **P** $\subset$ **NP** (if you can solve the problem, the solution is a certificate)
- **Open:** Does **P** $=$ **NP**? **NP** $=$ **EXP**?
- Most people think **P** $\subsetneq$ **NP** ($\subsetneq$ **EXP**), i.e. generating solutions harder than checking
- If you prove either way, people will give you lots of money. ($1M Millennium Prize)

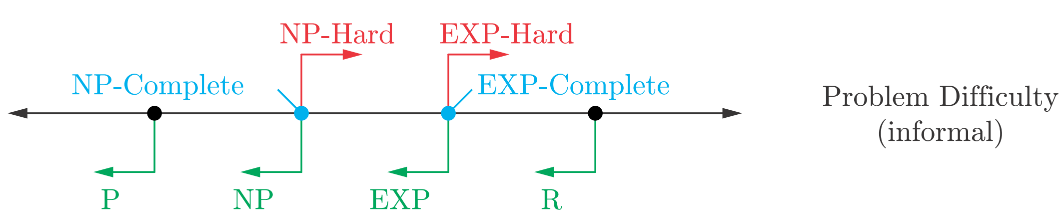
- Why do we care? If can show a problem is hardest problem in **NP**, then problem cannot be solved in polynomial time if **P** $\neq$ **NP**
- How do we relate difficulty of problems? Reductions!

Reductions

- Suppose you want to solve problem A
- One way to solve is to convert A into a problem B you know how to solve
- Solve using an algorithm for B and use it to compute solution to A
- This is called a **reduction** from problem A to problem B ($A \rightarrow B$)
- Because B can be used to solve A, B is at least as hard ($A \leq B$)
- General algorithmic strategy: reduce to a problem you know how to solve

A	Conversion	B
Unweighted Shortest Path	Give equal weights	Weighted Shortest Path
Product Weighted Shortest Path	Logarithms	Sum Weighted Shortest Path
Sum Weighted Shortest Path	Exponents	Product Weighted Shortest Path

- Problem A is **NP-Hard** if every problem in **NP** is polynomially reducible to A
- i.e. A is at least as hard as (can be used to solve) every problem in **NP** ($X \leq A$ for $X \in$ **NP**)
- **NP-Complete = NP** $\cap$ **NP-Hard**
- All **NP-Complete** problems are equivalent, i.e. reducible to each other
- First **NP-Complete**? Every decision problem reducible to satisfying a logical circuit.
- Longest Path, Tetris are **NP-Complete**, Chess is **EXP-Complete**



0-1 Knapsack is NP-Hard

- Reduce known NP-Hard Problem to 0-1 Knapsack: **Partition**
  - Input: List of n numbers a_i
  - Output: Does there exist a partition into two sets with equal sum?
- Reduction: $s_i = v_i = a_i$, $S = \frac{1}{2}\sum a_i$
- 0-1 Knapsack at least as hard as Partition, so since Partition is **NP-Hard**, so is 0-1 Knapsack
- 0-1 Knapsack in **NP**, so also **NP-Complete**