

## Recitation 20: 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  $\mathbf{P} \neq \mathbf{NP}$ . 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
$s$ - $t$ Shortest Path	Does a given $G$ contain a path from $s$ to $t$ with weight at most $d$ ?
Negative Cycle	Does a given $G$ contain a negative weight cycle?
Longest Path	Does a given $G$ contain a <b>simple</b> path with weight at least $d$ ?
Subset Sum	Does a given set of integers $A$ contain a subset with sum $S$ ?
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**: constant length code (working on a word-RAM with  $\Omega(\log n)$ -bit words) to solve a problem, i.e., it produces correct output for every input and the length of the code is independent of the instance size
- Problem is **decidable** if there exists a program to solve the problem in finite time

## Decidability

- Program is finite string of bits, problem is function  $p : \mathbb{N} \rightarrow \{0, 1\}$ , 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 (many awesome proofs in 6.045)
- Fortunately most problems we think of are algorithmic in structure and are decidable

## Decidable Problem Classes

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

- These sets are distinct, i.e.  $\mathbf{P} \subsetneq \mathbf{EXP} \subsetneq \mathbf{R}$  (via time hierarchy theorems, see 6.045)

## Nondeterministic Polynomial Time (NP)

- **P** is the set of decision problems for which there is an algorithm  $A$  such that for every instance  $I$  of size  $n$ ,  $A$  on  $I$  runs in  $\text{poly}(n)$  time and solves  $I$  correctly
- **NP** is the set of decision problems for which there is an algorithm  $V$ , a “verifier”, that takes as input an instance  $I$  of the problem, and a “certificate” bit string of length polynomial in the size of  $I$ , so that:
  - $V$  always runs in time polynomial in the size of  $I$ ,
  - if  $I$  is a YES-instance, then there is some certificate  $c$  so that  $V$  on input  $(I, c)$  returns YES, and
  - if  $I$  is a NO-instance, then no matter what  $c$  is given to  $V$  together with  $I$ ,  $V$  will always output NO on  $(I, c)$ .
- You can think of the certificate as a proof that  $I$  is a YES-instance. If  $I$  is actually a NO-instance then no proof should work.

Problem	Certificate	Verifier
$s$ - $t$ Shortest Path	A path $P$ from $s$ to $t$	Adds the weights on $P$ and checks if $\leq d$
Negative Cycle	A cycle $C$	Adds the weights on $C$ and checks if $< 0$
Longest Path	A path $P$	Checks if $P$ is a <b>simple</b> path with weight at least $d$
Subset Sum	A set of items $A'$	Checks if $A' \in A$ has sum $S$
Tetris	Sequence of moves	Checks that the moves allow survival

- $\mathbf{P} \subset \mathbf{NP}$  (if you can solve the problem, the solution is a certificate)
- **Open:** Does  $\mathbf{P} = \mathbf{NP}$ ?  $\mathbf{NP} = \mathbf{EXP}$ ?
- Most people think  $\mathbf{P} \subsetneq \mathbf{NP}$  ( $\subsetneq \mathbf{EXP}$ ), 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  $\mathbf{NP}$ , then problem cannot be solved in polynomial time if  $\mathbf{P} \neq \mathbf{NP}$
- How do we relate difficulty of problems? Reductions!

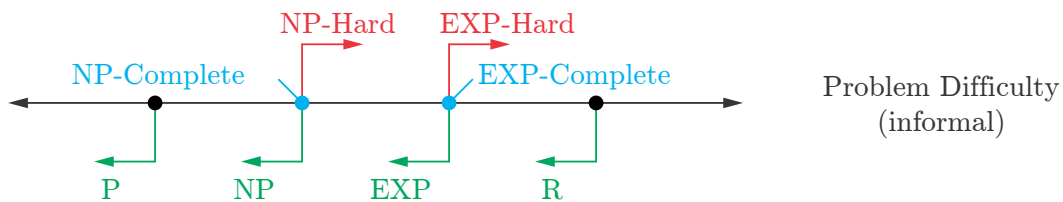
## Reductions

- 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  $\mathbf{NP}$  is polynomially reducible to  $A$
- i.e.  $A$  is at least as hard as (can be used to solve) every problem in  $\mathbf{NP}$  ( $X \leq A$  for  $X \in \mathbf{NP}$ )
- **NP-Complete** =  $\mathbf{NP} \cap \mathbf{NP-Hard}$

- 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**