# Algorithm and Data Structure Analysis (ADSA)

Complexity Classes

# Computational Complexity

- In the previous lectures, we used Turing machine to show that there are problems that are not decidable (such as the Halting problem).
- In the following, we consider problems that are decidable but would like to understand which problems have an algorithm that solves it in polynomial time.

- We use again Turing machines to define complexity classes.
- For a given language L, the Turing machine should accept x if x is in L.

# Example: Hamiltonian Cycle (HC) Problem

- Given: Undirected graph G=(V,E).
- Decide whether G contains a Hamiltonian cycle. A Hamiltonian cycle is cycle that visits each node exactly once and returns to the start vertex.

- Input x (the given graph) is in L if it contains a Hamiltonian cycle, it’s not in L if it doesn’t contain a Hamiltonian cycle.

# Computational Complexity

## A deterministic Turing machine M is a 6-tuple

$$M = (Q, \Sigma, q_{start}, q_{accept}, q_{reject}, \delta)$$

Q: finite set of states

$q_{start}, q_{accept}, q_{reject} \in Q$

$\Sigma$ : nonempty finite alphabet that includes symbol # called blank

Transition function $\delta : Q \times \Sigma \to Q \times \Sigma \times \{L, R\}$

For given $q \in Q$ and $\sigma \in \Sigma$ if $\delta(q, \sigma) = (q', \sigma', D)$, we mean that if $M$ is in state q and encounters symbol $\sigma$ then it changes $\sigma$ to $\sigma'$ and moves one step in the direction of $D \in \{L, R\}$ and enters state $q'$.

# Deterministic Turing Machine

We can also write the transition function as

$$\delta' : Q \times \Sigma \times Q \times \Sigma \times \{L, R\} \to \{0, 1\}$$

- For our deterministic transition functions, we have

$$(\forall q \in Q)(\forall \sigma \in \Sigma) \sum_{q' \in Q, \sigma' \in \Sigma, D \in \{L,R\}} \delta'(q, \sigma, q', \sigma', D) = 1$$

Classical Church-Turing Thesis: Every problem that is intuitively computable can be computed by a deterministic Turing machine.

Complexity class P: P is the set of problems that can be solved by a deterministic Turing machine in a polynomial number of steps.

Cook-Karp Thesis: Problems that are "tractably computable" can be computed by a deterministic Turing machine in polynomial time, i.e. are in P.

# Nondeterministic Turing machine

- A nondeterministic Turing machine M is a 6-tuple

$$M = (Q, \Sigma, q_{start}, q_{accept}, q_{reject}, \delta)$$

Q: finite set of states

$q_{start}, q_{accept}, q_{reject} \in Q$

$\Sigma$ : nonempty finite alphabet that includes symbol # called blank

Transition function:

$$\delta : Q \times \Sigma \to \mathcal{P}(Q \times \Sigma \times \{L, R\})$$

$\mathcal{P}$ is the powerset function

$$\hat{\delta} : Q \times \Sigma \to \{0, 1\}^{Q \times \Sigma \times \{L, R\}}$$

# Nondeterministic Turing machine

- We can rewrite

$$\delta' : Q \times \Sigma \times Q \times \Sigma \times \{L, R\} \to \{0, 1\}$$

- But don't require:

$$\cancel{(\forall q \in Q)(\forall \sigma \in \Sigma) \sum_{q' \in Q, \sigma' \in \Sigma, D \in \{L,R\}} \delta'(q, \sigma, q', \sigma', D) = 1}$$

Nondeterministic Turing machine M for language L accepts x if there is a path from $q_{start}$ to the accepting state $q_{accept}$.

Complexity class NP: NP is the set of problems that can be solved by a nondeterministic Turing machine in a polynomial number of steps.

We have P $\sqsubseteq$ NP.

# Clique Problem

- Given: Undirected graph G=(V,E) and an integer k.
- Decide whether the graph contains a complete subgraph (clique) on k nodes.

- NTM M counts the number of nodes of input G and then guesses (nondeterministic part) a word $w \in \{0,1\}^n$. 1 means node $v_i$ is selected, 0 means node $v_i$ is not selected.
- M tests whether w contains exactly k nodes and whether it is a clique. If both tests are positive, then it accepts.
- Verification of whether w represents a clique on k nodes can be done in polynomial time.

# Variants

- Decision variant: Decide whether the graph contains a complete subgraph (clique) on k nodes.
- Optimisation variant (Maximum Clique): Compute a clique C such that there is no clique in G with a larger number of nodes.

- Decision variant for CLIQUE is in NP.
- Optimisation variant for CLIQUE is not in NP (unless P=NP). Why?
- You can’t verify in polynomial time that a given clique C is a clique with a maximal number of nodes.

# Complement

- If a problem has a ”yes” answer, then the complement of the problem has a “no” answer and vice versa.

Complexity class coP: coP is the set of problems whose complements can be solved by a deterministic Turing machine in a polynomial number of steps.

Complexity class coNP: coNP is the set of problems whose complements can be solved by a nondeterministic Turing machine in a polynomial number of steps.

We have coP=P $\sqsubseteq$ coNP.

We don’t know if coNP=NP. Most researchers believe NP $\neq$ coNP.

# Example for Complement

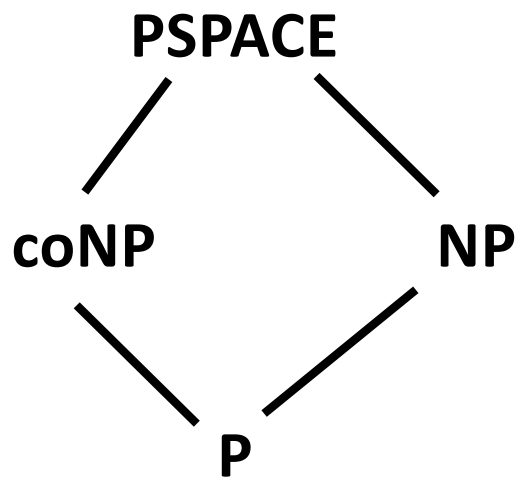
- Given a graph G, does it contain a Hamiltonian cycle? (in NP, accept if G contains HC)
- Given a graph G, is there no Hamiltonian cycle in G? (in co-NP as complement is in NP, accept if there is no HC in G).

Note that the statement that a graph G doesn't contain a Hamiltonian cycle is hard to verify.

# PSPACE

Complexity class PSPACE: PSPACE is the set of problems that can be solved by a deterministic Turing machine using a polynomial number of space on the tape.

We have NP $\sqsubseteq$ PSPACE and coNP $\sqsubseteq$ PSPACE.



# Probabilistic Turing Machines

- In probabilistic computations there is a random choice among severval possible transitions during the computation.
- A probabilistic Turing machine is a Turing machine that randomly performs one of several tasks at each time step.
- Formally a probabilistic Turing machine is a 6-tuple $M = (Q, \Sigma, q_{start}, q_{accept}, q_{reject}, \delta)$

  where $\delta : Q \times \Sigma \to \tilde{[0, 1]}^{Q \times \Sigma \times \{L,R\}}$

$\tilde{[0, 1]}$ is the set of real numbers in [0,1] such that a deterministic Turing machine can calculate their nth digit in polynomial time

# Probabilistic Turing Machines

- We can also write the transition function as

$$\delta' : Q \times \Sigma \times Q \times \Sigma \times \{L, R\} \to \{0\tilde{,}1\}$$

Where

$$\delta'(q, \sigma, q', \sigma', D) = r \in [0\tilde{,}1]$$

We have the requirement:

$$(\forall q \in Q)(\forall \sigma \in \Sigma) \sum_{q' \in Q, \sigma' \in \Sigma, D \in \{L,R\}} \delta'(q, \sigma, q', \sigma', D) = 1$$

(sum of probabilities of all possible moves is 1)

# Probabilistic Complexity Class BPP

Complexity class BPP (bounded-error probabilistic polynomial time): BPP is the set of problems that can be solved by a probabilistic Turing machine M in a polynomial number of steps with the possibility of some error. Precisely we have:

If $x \in L$ then Prob(M accepts x)> 2/3
If $x \notin L$ then Prob(M rejects x)> 2/3

# Probabilistic Complexity Class RP

Complexity class RP (randomized polynomial time): RP is the set of problems that can be solved by a probabilistic Turing machine M in a polynomial number of steps with only the possibility false negatives. Precisely we have:

If $x \in L$ then Prob(M accepts x)> 2/3
If $x \notin L$ then Prob(M rejects x)= 1

# Probabilistic Complexity Class coRP

Complexity class coRP: coRP is the set of problems that can be solved by a probabilistic Turing machine M in a polynomial number of steps with only the possibility false positives. Precisely we have:

If $x \in L$ then Prob(M accepts x)= 1
If $x \notin L$ then Prob(M rejects x)> 2/3

# Probabilistic Complexity Class ZPP

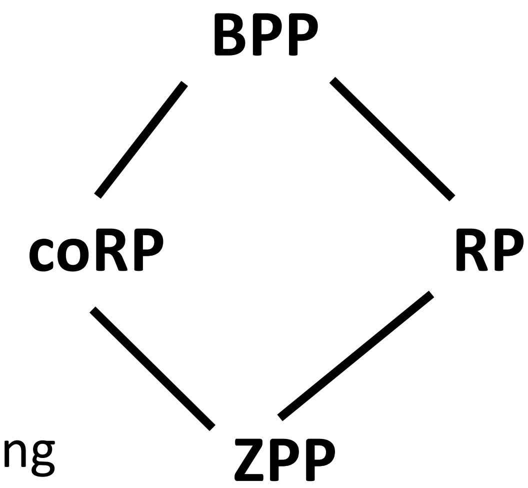
Complexity class ZPP (zero error probabilistic polynomial time): ZPP is the set of problems that can be solved by a probabilistic Turing machine M in an expected polynomial number of steps with zero error. Precisely we have:

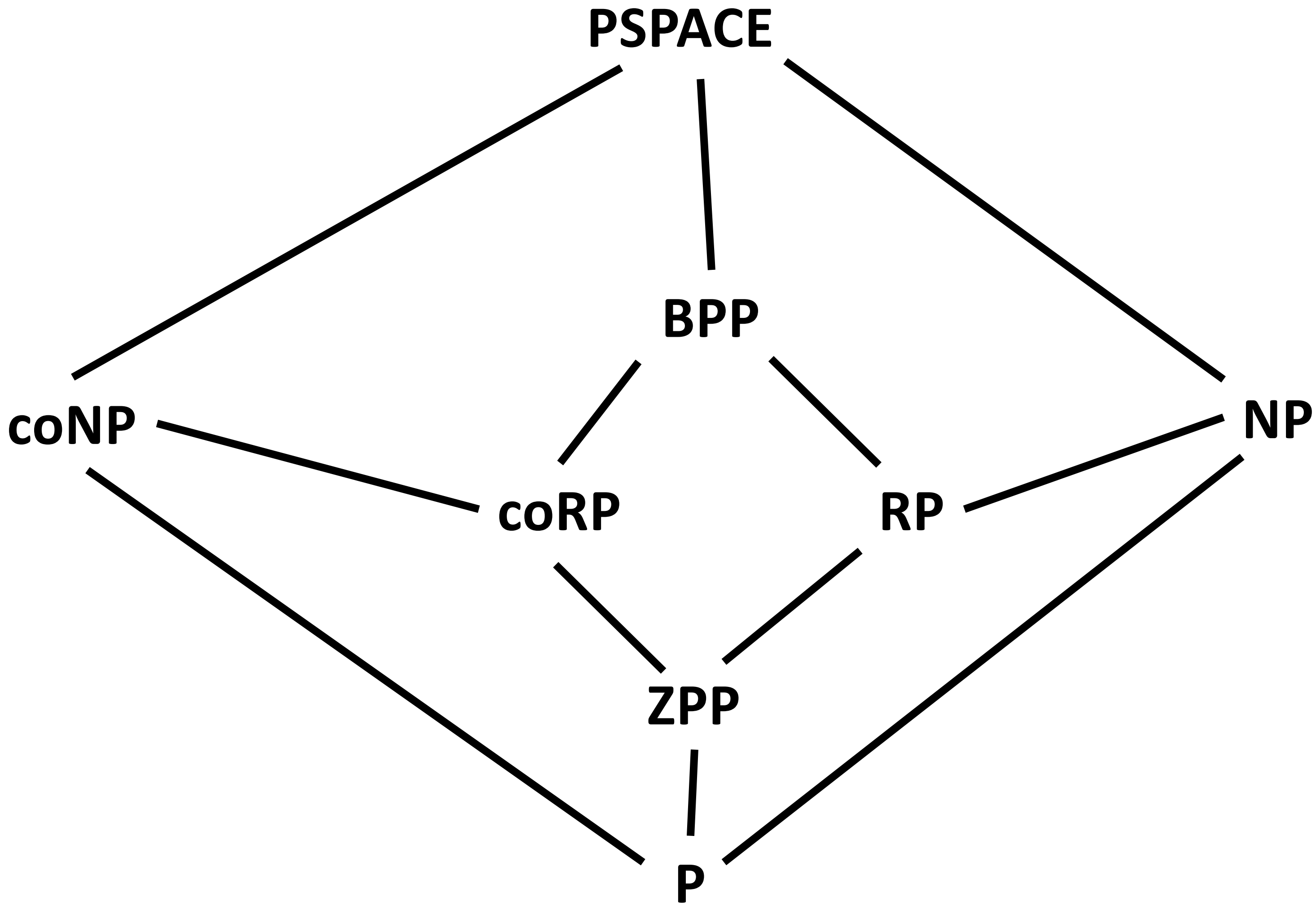
If $x \in L$ then Prob(M accepts x)= 1
If $x \notin L$ then Prob(M rejects x)=1

$$RP \cap coRP = ZPP$$

# Probabilistic Complexity Classes

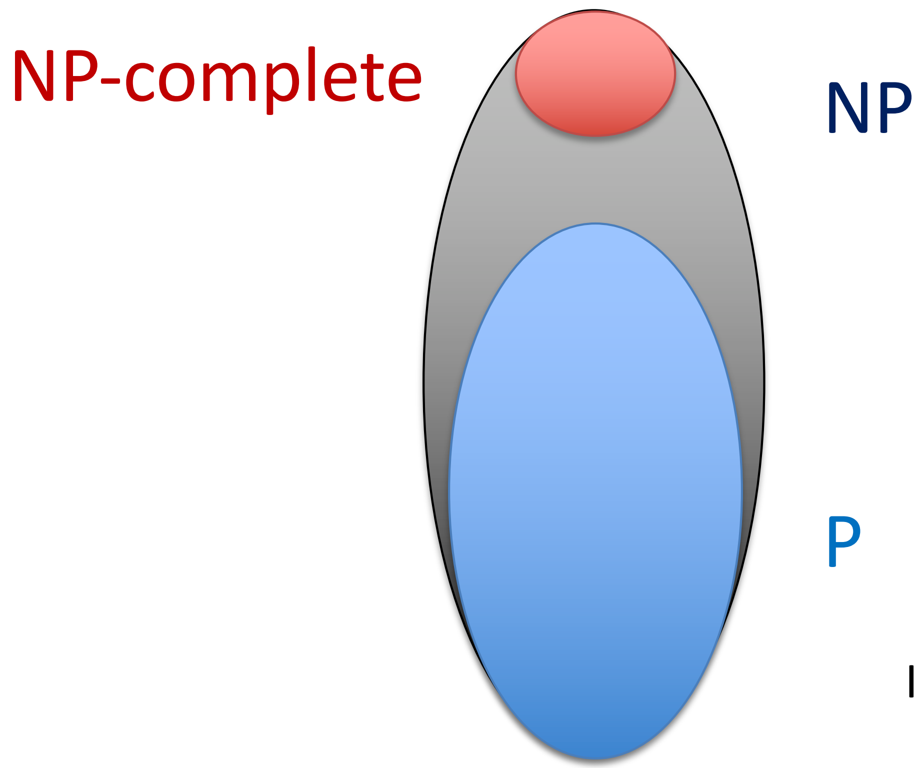
- If we can solve a problem with no error (ZPP), then we can solve problem permitting false negatives (RP) or false positives (coRP).
- If we can solve a problem permitting only false Negatives (RP) then we can solve the problem permitting both false negatives and false positives (BPP) (similar argument for coRP and BPP)




PSPACE
BPP
coNP
NP
coRP
RP
ZPP
P

# P and NP

We want to zoom in on P and NP.




It is widely assumed that P!=NP

Many important decision problems are NP-complete (are the hardest problems in NP)