Design and Analysis of Algorithms

L47: NP Class of Problems

NP-Complete and NP-Hard

Dr. Ram P Rustagi
Sem IV (2019-H1)
Dept of CSE, KSIT/KSSEM
rprustagi@ksit.edu.in

Resources

- Text book 2: Horowitz
  – Sec 8 . 2
- R1: Introduction to Algorithms
  - Cormen et al.
- URLs
  – https://www.slideshare.net/narayanagalla/np-cooks-theorem?from_action=save

Overview

- `P`: The the class of problems which can be solved by a deterministic polynomial algorithm.
- `NP`: the class of decision problems which can be solved by a non-deterministic polynomial algorithm
- `NP-Hard`: the class of problems to which every NP problem reduces
- `NP-Complete`: the class of problems which are NP-hard and belong to NP

P Problem

- Bubblesort
  - Time complexity $O(n^2)$
- Heapsort
  - Time complexity $O(n^2)$
- Strassen's matrix multiplication
  - Time complexity $O(n^{2.8})$
- Topological Sort
  - Time complexity $O(|V|+|E|)$
- Prim's/Kruskal's algorithm
  - Time complexity $O(|E|.lg|V|)$, $O(|E|lg*n|$
- Warshall-Floyd Algorithm
  - Time complexity $O(n^3)$

Overview NP Complete Problems

- Definition of reduction: Problem A reduces to problem B ($A \propto B$) iff A can be solved by a deterministic polynomial time algorithm using a deterministic algorithm that solves B in polynomial time.
- NP-Complete problems:
  - Up to now, none of the NPC problems can be solved by a deterministic polynomial time algorithm in the worst case
  - It does not seem to have any polynomial time algorithm to solve the NPC problems
  - The lower bound ofN PC seems to be in the order of an exponential function

NP Complete

- If A, B $\in$ NP-Complete, then A $\propto$ B and B $\propto$ A.
- If any NP-Complete problem can be solved in polynomial time, then all NP problems can be solved in polynomial time. (NP = P)

Decision Problems

- Decision problems are those for which solution is a `Yes` or `No`.
- Optimization problems are diificult but can be derived from decision problems.
  - Consider each possible value starting from highest possible answer going by one at a time to treat it as a decision problem
- Example: Knapsack Problem
  - Optimization: Find the max value of items in the knapsack
  - Decision: Is there a set of items whose value is greater than or equal to constant `c`?

Solving Optimization by Decision

- Solving Knapsack problem with decision algo
  - Take a value c_1, and check if knapsack value $>= c_1$
  - Take a value c_2, and check if knapsack value $>= c_2$
  - :
  - :
  - Take a value c_k, and check if knapsack value $>= c_k$
  - Find the smallest c_i.
  - Smallest c_i is the optimization value.

Satisfiability Problem

- Logical operators: AND ($\wedge$), OR ($\vee$)
- Consider logical expression $E = x_1 \vee x_2 \vee x_3$
  - Evalute of the assignment with values
  $x_1 \leftarrow F;\ x_2 \leftarrow F;\ x_3 \leftarrow T;$
  - Answer: True
- For a given logical expression, if there exists an assignment of boolean variables which evalutes expression to be True,
  - then expression is satisfiable, otherwise
  - expression is unsatisfiable.
- Consider following expression

$$(x_1 \vee x_2) \wedge (x_1 \vee \sim x_2) \wedge (\sim x_1 \vee x_2) \wedge (\sim x_1 \vee \sim x_2)$$

  - This expression is unsatisfiable.

Satisfiability Problem

- Satisfiability problem definition:
  - Given a boolean expression, determine if the expression is `satisfiable` or not.
- Some terms
  - Literal: x_i or $\sim x_i$
  - Clause: $C_i \equiv x_1 \vee x_2 \vee \sim x_3$
  - Conjunctive Normal Form (CNF):

    $C_1 \wedge C_2 \wedge C_3 \wedge \ldots\ldots \wedge C_m$
- Any boolean expression can be converted into CNF
- Time complexity for satisfiability problem
  - 2^n (Try all possible combinations of n variables)

Nondeterministic Algorithms

- A nondeterministics algorithm involves two phases
  – Making a choice (i.e. making the correct guess)
  – Using the choice to check the problem answer
- Nondeterministic problems are used only for *decision* problems
- If checking time is of polynomial time complexity, then
  – The algorithm is called NP (Nondeterministic Polynomial) problem
- Examples of NP problems (includes P problems as well)
  - Sorting of N numbers (given numbers, check if in order)
  - Satisfiability problem (given variable values, evaluate)
  - TSP problem (given a tour, check if cost is c)

Decision Problems

- Decision version of following problems
- Sorting problem,
  - Given a n numbers: $a_1, a_2, \ldots, a_n$.
  - Is there a permutation $(a_1, \ldots, a_n)$ such that $a_i \le a_{i+1}$
- Max clique problem:
  - Clique: a complete subgraph of given graph G.
  - Max Clique: max complete subgraph of G
  - Decision problem: Does $\exists$ a clique of size $\geq k$
- Not all decision problems are NP problems
- Example: Halting problem
  - Given a program with some input data, will the program ever terminate

Nondeterministic Algorithms

- Three functions
  - `Choice(S)`: arbitrarily choses one of set elements
  - `Failure`: an unsuccessful completion
  - `Success`: a successful completion
- A simple nondeterministic algorithm for searching

```
j←Choice(1:n)   // making a guess
if A[j]==x,  then
   Success
else
   Failure
fi
```

Nondeterministic Algorithms

- A nondeterministic algo terminates unsuccessfully if
    - there does not exist a set of choices
    - that leads to a success result.
- – The time required for `Choice(1:n)` is `O(1)`.
- – A deterministic interpretation (or theoretical implementation) of non-deterministic algorithm
    - – Achieved by creating parallel number of executions
        - – Equal to number of choices.
- – Note: Nondeterministic algorithm is for theoretical study.
    - – An algorithm requires decisive step.

Nondeterministic algorithm: Sorting

```
Algo NSort(A[], n)
 B[] // initialized to 0.
 for i←1 to n
   j←Choice(1:n)
   if B[j]≠0 //incorrect choice
     Failure; return
   B[i]=A[j]
 for i←1 to n-1 //verify order
   if B[i]>B[i+1] //not ascending order
     Failure; return
 for i←1 to n
   print B[i]
 Success
```

Nondeterministic algo: Sum of Subsets

```
Algo SumSubsets(A[], n, m)
  // check if sum of some subset of A[] equals m
  s←0
  for i←1 to n
     j←Choice(0,1) //makes a correct choice
     if j==1
        s←s+A[i]
  if s≠m
        Failure; return
  Success
```

Nondeterministic algo: Knapsack Problem

```
Algo Knapsack(p[],w[],n,m,v)
  // check if sum of some subset of w[] <= m, and
  // profit >=v
  wt←0; val ←0
  for i←1 to n
     j←Choice(0,1) //makes a correct choice
     if j==1
        wt←wt+w[i]
        val←val+[i]
  if wt>m or val <v
        Failure
  Success
```

Nondeterministic algo: Satisfiability Problem

```
Algo Satisfiability()
  // check if expression evaluates to True
  wt←0; val ←0
  for i←1 to n
     xi←Choice(True, False) //a correct choice?
  if evaluate(Expr(x1,…,xn)) is False
       Failure
  Success
```

NP-Hard and NP-Complete Classes

- Definitions

  P class:

  Set of all decision problems solvable by deterministic algorithms in polynomial time.

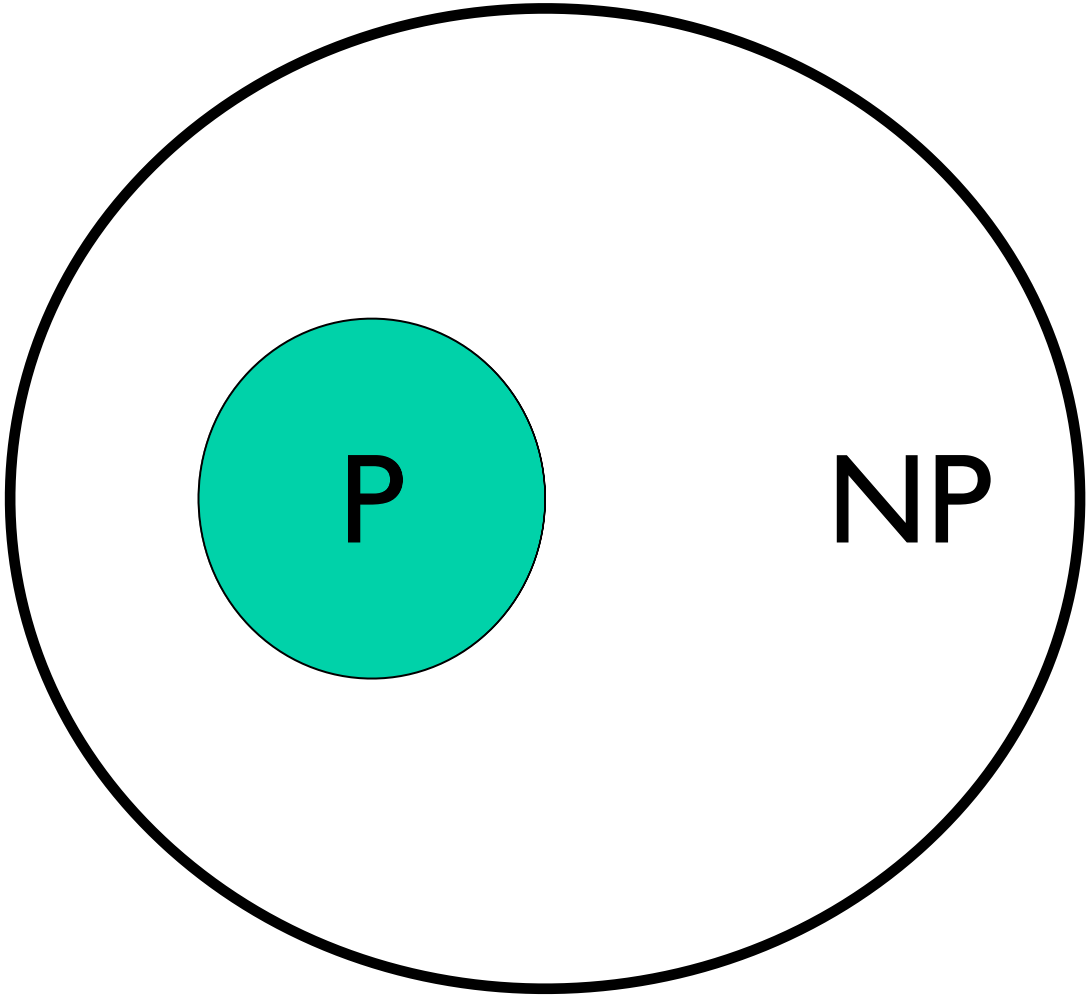
  NP class:

  Set of all decision problems solvable by nondeterministic algorithms in polynomial time.,

- P and NP
  - Deterministic algorithms are a special case of nondeterrministic algorithms, thus
    - $P \subseteq NP$
- Most famous unsolved problem in computer science:
  - Is P==NP or $P \neq NP$

NP-Hard and NP-Complete Classes

- Assuming $P \neq NP$ the relationship is given by



Cook's Theorem

- Cook's theorem:
  – Satisfiability is in `P` if and only if `P=NP`
- Cook formulated the following question
  – Is there a single problem in `NP` such that if we showed it to be in `P`, then that would imply that `P=NP`?
  – Cook's theorem answers it in affirmative

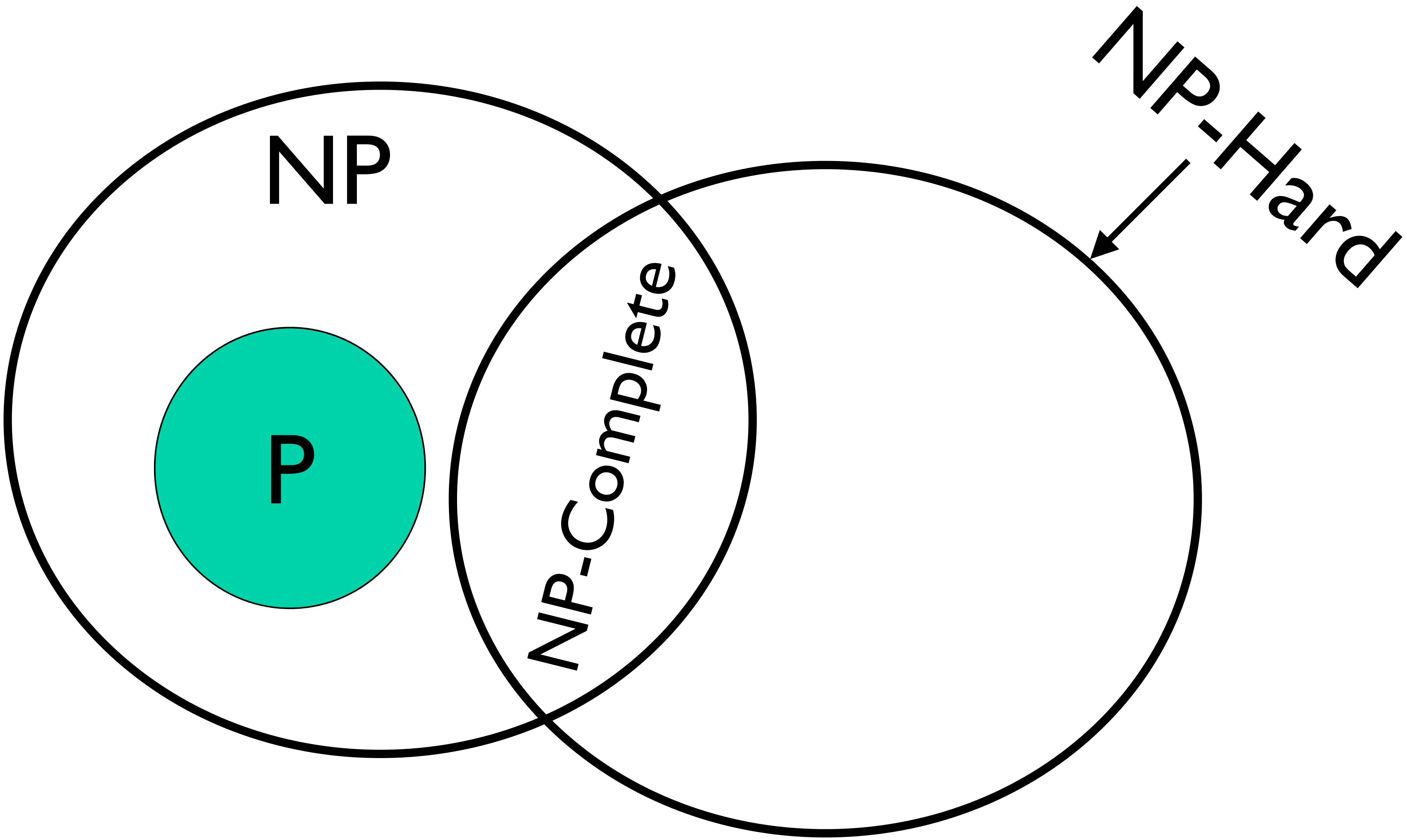
Reducibility

- Let L_1 and L_2 be two problems.
- Problem L_1 reduces to L_2, also written as $L_1 \propto L_2$,
  - if and only if there is a way to solve L_1 by a deterministic polynomial time algorithm using a deterministic algorithm that solves L_2 in polynomial time.
- The definition implies that if we have polynomial time algorithm for L_2 then we can solve L_1 in polynomial time.
- Reducibility is a transitive relation i.e.
  - if $L_1 \propto L_2$ and , $L_2 \propto L_3$ then $L_1 \propto L_3$

NP-Hard

- A problem is `L` is called `NP-Hard` if and only if Satisfiability problem can be reduced to `L` i.e. (Satisfiability $\propto$ `L`).
- A problem `L` is called `NP-Complete` if and only if `L` is `NP-Hard` and `L` $\in$`NP`.
- Implies that there are problems which are not in `NP`, but satisfiability problem can be reduced to these problems. Thus, all `NP-Hard` problems are not `NP-Complete`.
- Only a decision problem can be NP-Complete.
- An optimization problem may be NP-Hard.

P, NP, NP-Hard, NP-Complete



NP-Hard and NP-Complete

- If L_1 is a decision problem and L_2 is an optimization problem, it is quite possible that $L_1 \propto L_2$
- Examples:
  - Knapsack decision problem reduces to knapsack optimization problem
  - Clique decision problem can be easily reduced to clique optimization problem.
- Optimization problems can't be NP-Complete, whereas decision problems can be NP-Complete.
- However, $\exists$ NP-Hard decision problems that are not NP-Complete.

NP-Hard and NP-Complete

- Ex:NP-Hard decision problems that is not NP-Complete.
  - Consider halting problem, which is undecidable, i.e.
    - There exists no algorithm that can solve this problem.
  - Thus, this problem is not in NP.
    - Can’t be solved in a nondeterministic polynomial time.
  - Show that Satisfiability reduces to Halting problem.
  - Construct an algo A whose input is CNF proposition X
    - Algo tries out all 2n possible truth assignments and verifies if X is satisfiable.
    - If X is satisfiable, then A stops else runs for ever.
  - If Halting can be solved in polynomial time, so is satisfiability using A and X as input to algo A.
  - Thus, Halting is NP-Hard but not in NP.

NP-Hard and NP-Complete

- Two problems L_1 and L_2 are said to be polynomially equivalent if and only if
  - $L_1 \propto L_2$, and $L_2 \propto L_1$.
- To show that a problem L_2 is NP-Hard,
  - it is adequate to show that $L_1 \propto L_2$ and,
    - L_1 is already known as NP-Hard problem
  - Proof:
    - $\mathtt{Satisfiability} \propto L_1$, and $L_1 \propto L_2$,
    - Thus by transitive relation, $\mathtt{Satisfiability} \propto L_2$

Summary:

- P Problems
- NP Problems
- NP Complete
- NP Hard