Polynomial Time Reduction

- Polynomial Time Reduction Definition
- Reduction by Equivalence
- Reduction from Special Cases to General Case
- Reduction by Encoding with Gadgets
- Transitivity of Reductions
- Decision, Search and Optimization Problem
- Self-Reducibility

Imdad ullah Khan

Efficiently Solvable and Hard (Intractable) Problems

Efficiently Solvable Problem

$\exists$ an $O(n^k)$ worst case time algorithm for instances of size n, constant k

- In complexity theory we study negative results
- Characterize problems for which we don't have good news
- Cannot say they are not efficiently solvable (just don't know yet)
- We might need to focus on approximation or special cases

Hard (Intractable) Problems

- No known $O(n^k)$ algorithm
- Exponential time is sufficient $O(n^n), O(n!), O(k^n)$

We establish that these "hard problems" are in some sense are equivalent

Polynomial time reduction is a way to compare hardness of problems

- To explore the class of computationally hard problems, we define a notion of comparing the hardness of two problems
- Measures the relative difficulty of two problems

Problem A is polynomial time reducible to Problem B, $A \leq_p B$

If any instance of problem A can be solved using a polynomial amount of computation plus a polynomial number of calls to a solution of problem B

- B is at least as hard as problem A (w.r.t polynomial time)
- Extremely important (a building block) for complexity theory
- Generally confused, make sure you understand it the right way

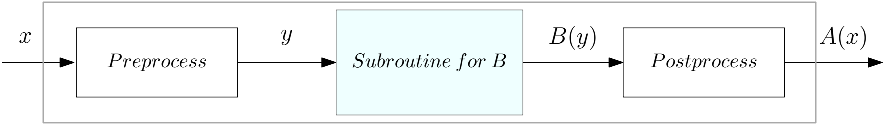
Polynomial time reduction is a way to compare hardness of problems

Problem A is polynomial time reducible to Problem B, $A \leq_p B$

If any instance of problem A can be solved using a polynomial amount of computation plus a polynomial number of calls to a solution of problem B

If any algorithm for problem B can be used [called (once or more) with *'clever'* legal inputs] to solve any instance of problem A

Subroutine for B takes an instance y of B and returns the solution $B(y)$



Algorithm for A transforms an instance x of A to an instance y of B. Then transforms $B(y)$ to $A(x)$

Polynomial time reduction can be used to design algorithms

Problem A is polynomial time reducible to Problem B, $A \leq_p B$

If any instance of problem A can be solved using a polynomial amount of computation plus a polynomial number of calls to a solution of problem B

- FINDMIN $\leq_p$ SORTING
- SORTING $\leq_p$ FINDMIN
- MEDIAN $\leq_p$ SORTING
- SORTING $\leq_p$ MEDIAN
- CYCLE-DETECTION $\leq_p$ DFS
- ALL-PAIRS-PHORTEST-PATHS $\leq_p$ SINGLE-SOURCE-SHORTEST-PATHS
- SINGLE-SOURCE-SHORTEST-PATHS $\leq_p$ ALL-PAIRS-PHORTEST-PATHS
- BIPARTITE-MATCHING $\leq_p$ MAXIMIMUM-FLOW

Complete details of these (toy) reductions-calls (with inputs), extra computation