Complexity Classes

Problem classes

Tractable

Solvable in polynomial time. That is, a polynomial time algorithm exists for finding a solution to the problem

Intractable

No polynomial time algorithm exists for solving the problem

Polynomial Time

- Problems having polynomial time algorithms are called tractable
- In practice, a polynomial of high order will be as bad as a non-polynomial-time algorithm but we keep with this idea for the following reasons:
  - For most practical problems that we encounter that have a polynomial time algorithm, the polynomial is found to be of lower order only
  - If a higher order polynomial algorithm is discovered for a problem, lower order ones follow. Therefore, if a problem exists for which we have a higher order polynomial solution it is highly likely that we will discover a lower order one
  - Polynomial class problems have the nice closure property i.e. if the output of one polynomial-time algorithm is fed to other the resulting algorithm will also be polynomial class

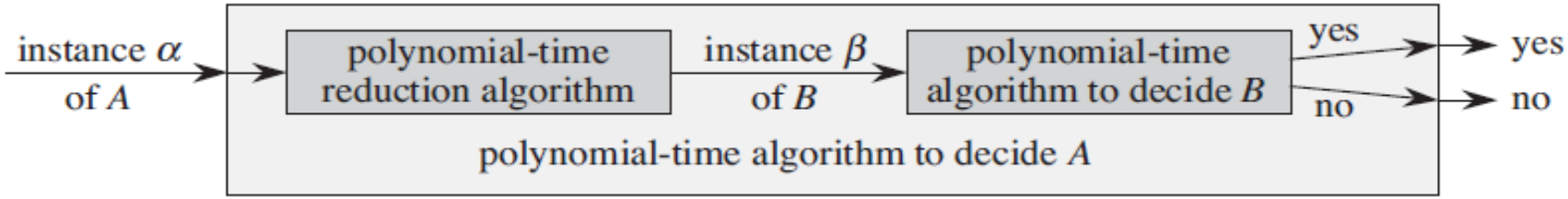
Decision Problems v/s Optimization Problems

- An important class of problems that we will consider are NP-complete problems
- NP-complete problems are confined to the realm of Decision problems
- However, optimization problems can be recast into decision problem
- For example, the problem of finding shortest path between two given vertices in a graph can be recast into a decision problem by formulating a new problem as: given a graph G, two vertices u and v and an integer k, does there exist a path between u and v having at most k edges?
- The corresponding decision problem is no harder than the optimization problem in the sense that if optimization problem can be solved in polynomial time, the decision problem is also solvable in polynomial time. Conversely, if we prove that the decision problem is hard then it gives sufficient evidence that the optimization problem is hard

Reduction

- The recasting of problems can also be used for two different problems
- This technique is useful in showing that one problem is no easier than the other or it is at least as hard
- In reduction, we use a decision problem we know about. Suppose, we know how to solve a decision problem B in polynomial time. Given, a decision problem A, we device a transformation function that converts every instance α of A into an instance β of B with the following characteristics:

- The transformation takes polynomial time
- If the solution for instance α in A is yes then it is also yes for instance β in B and vice-versa
- We call such a procedure as *polynomial-time reduction algorithm*
- For solving A we do the following:
  - Use the polynomial time reduction algorithm to convert the given instance α of A to instance β of B
  - Use the polynomial time algorithm of B to solve the instance β
  - Use the answer of β as answer of α



Proving NP-Completeness

- We can use the above procedure in opposite-way to show how hard a problem is
- Suppose, there is a problem A that we know that no polynomial time algorithm exists for and let there be a new problem B to which A can be reduced in polynomial time
- Then, we can simply prove by contradiction that no polynomial time algorithm exists for problem B because if it were not so then we could use the reduction algorithm to solve the problem A in polynomial time
- For proving NP-completeness, we need to have a “first” NP-complete problem that can be used to prove NP-completeness of other problems through reduction though for NP-complete problems we might not be able to prove that no polynomial time algorithm exists

P-Class

- We try to analyze resources associated by an algorithm for solving a decision problem
- P Class is the class of problems that can be solved by a deterministic machine in polynomial time
- An algorithm can solve the decision problem in $O(n^k)$, where k is a small constant and n is the input size
- Also known as tractable problems

NP-Class

- Non deterministic polynomial class
- Set of decision problems that can be solved non deterministically in polynomial time
- Nondeterministic implies that the solution can be guessed out of polynomially many options in constant time
- If any guess leads to “yes” answer then get such a guess
- verified by a deterministic algorithm in polynomial time
- Such problems cannot be solved in polynomial time but can be verified in polynomial time
- That is, if a correct solution of the problem is given, it can be verified in polynomial time that it is a solution to the problem
- Also called as intractable problems

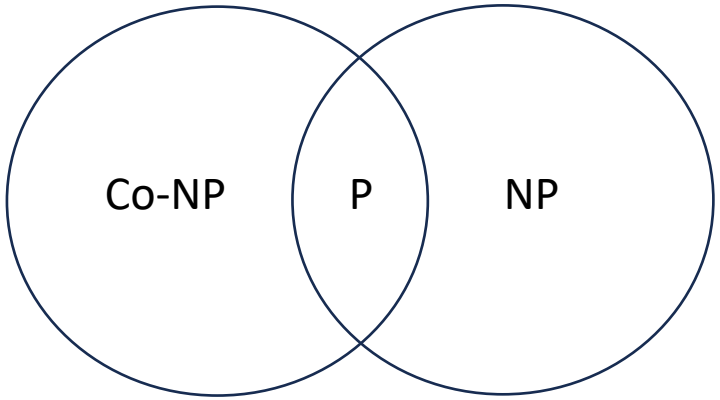
A non-deterministic algorithm

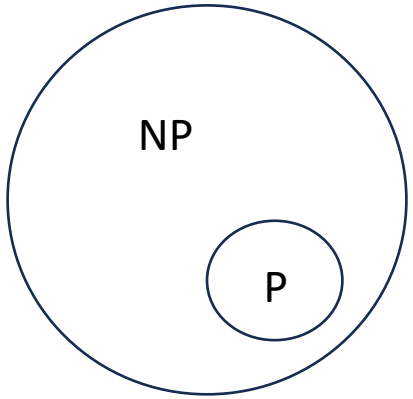
- Search (A, n, key) // Takes O(1) time

```
{
    j = choice();
    if (j==key)
      {
        write(j);
        success();
      }
    write(0);
    failure();
}
```

Co-NP Class

- Complement of NP class
- Only “yes” answers are polynomial time verifiable in NP class
- If we can verify “no” answers to a decision problem in polynomial time, then this class of problems is called Co-NP
- That is, if an incorrect solution is given to a decision problem, it can be verified in polynomial time that it is not a solution to the problem
- A problem that is solvable in polynomial time can also be verified for “yes” or “no” solutions as well in polynomial time
- Thus, we get a venn diagram of the following type:


Co-NP
P
NP


NP
P

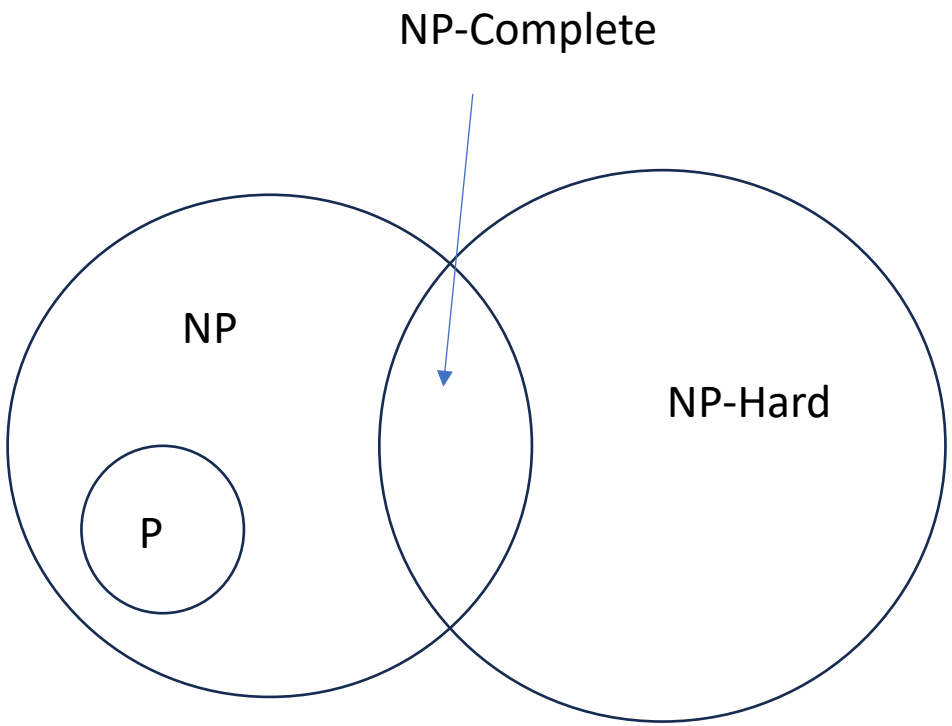
NP-Hard

- Stands for non deterministic polynomial time hard problems
- Described as at least as hard as the hardest problem in NP
- An NP problem is hard if it takes large amount of time in verification
- An NP problem will be called hardest if it takes the maximum time in verification among all NP problems

NP-Complete

- A decision problem is NP-complete if
  - It is in **NP** and
  - Every problem in NP is reducible to it in polynomial time
  - Circuit Satisfiability is the first NP-complete problem

- A problem is **NP-Hard** if it satisfies Property 2 but not necessarily Property 1


NP-Complete
NP
NP-Hard
P