P sınıfı algoritmalar

- **P:** Polinom sürede çözülebilen problemler.
- Bu problemler deterministik algoritmalar tarafından polinom zamanda çözülebilirler.
- Bu problemler Determinisitk bir Turing makinesi üzerinde çözülebilirler.
- Deterministik algoritma daima doğru cevanı hesaplar.

Polynomial time: $O(n^2)$, $O(n^3)$, $O(1)$, $O(n \lg n)$
Not in polynomial time: $O(2^n)$, $O(n^n)$, $O(n!)$

NP sınıfı algoritmalar

- **NP:** Bir problemi çözen algoritmanın sonucu polinom zamanda doğrulanabiliyorsa, problem NP sınıfındadır.
- Non-deterministic Polynomial, burada "non-determinism" ile algoritmanın sonucu kastedilir, doğrulama işlemi değil
- Başka bir deyişle: Nondeterministik turing makinesi ile sonuç polinom zamanda elde edilebilir.
- Nondeterministik makine bütün olasıklıkları dener ve böylece polinom zamanda ulaşılabilen sonucu veren olasılık hesaplandığında problem çözülmüş olur. Ancak nondeterministik turing makinesi teorik bir makinedir.
- NFA'nın DFA'ya çevrilmesi örneği
- Non-deterministik polinom sürenin deterministik versiyonu en kötü durumda üssel olarak çalışacaktır $O(2^n)$

P, NP

Bir algoritma ancak eğer süresel karmaşıklığı büyüklük parametrelerinin bir polinomu ise iyi kabul edilmektedir. Örneğin $O(n)$, $O\left(n^2\right)$, $O\left(ln + m^3\right)$, $O\left(n^3 + m^3 + 5\right)$, vb. polinom sınırlı süresel karmaşıklıkları ifade ederler, $O\left(6^n\right)$, $O\left(n!m!\right)$, $O\left(n^m\right)$, ise polinom sınırlı olmayan süre karmaşıklıklarının, dolayısıyla kötü algoritmaları temsil eder. Polinom zamanlı çözer yöntemi bulunmuş problem sınıfı P ile gösterilir. Problemler sınıfını formal olarak tanımlamak için *gerekirci (deterministik) (DTM)* ve *gerekirci olmayan (nondeterministik) Turing makinesi (NTM)* kavramları kullanılır. Burada DTM'yi bildiğimiz bilgisayarlara eş bir makine, NTM'yi ise henüz eşi dünyada olmayan üstün bir makine olarak kabul edebiliriz. Bu öyle bir üstün makenedir ki, bir anda sonlu bir k kadar çözüm tahminlerinde bulunup her tahmin için ayrı bir bilgisayar gibi hesaplamalar yapabilir.

Polinom sınırlı DTM ile çözülebilen tüm karar problemleri P ***sınıfını*** oluşturur. Polinom sınırlı NTM ile çözülebilen karar problemleri ise *NP sınıfındandır*. P, NP'nin bir alt kümesidir. Çünkü NTM'ler polinom sınırlı DTM ile çözülebilen her problemi çözebilir. Asıl sorun yalnız polinom sınırlı NTM ile çözülebilen karar problemleminin DTM ile polinom sınırlı sürede çözülüp çözülemeyeceğidir. Yani $P \stackrel{?}{=} NP$ sorusunun yanıtı Kapmaşıklık kuramının çözülmemiş en önemli problemidir.

The Complexity Class **NP**

- The complexity class **NP** (**nondeterministic polynomial time**) contains all problems that can be solved in polynomial time by an NTM.
- Formally:

$$\mathbf{NP} = \{ L \mid \text{There is a nondeterministic TM that decides } L \text{ in polynomial time.} \}$$

Örnek NP problem: Hamilton Döngüsü

- Hamilton Döngüsü (Her düğümü ziyaret et fakat her kenarı bir kere)
- Yönlü bir grafta hamilton döngüsünün olup olmadığına karar verilmesi polinomik bir çözüm halen bulunamamıştır. Fakat aynı graftan verilen bir düğüm serisinin hamilton döngüsünün olup olmadığına karar verilmesi polinomik sürede çözülür.
- Dolayısıyla döngüleri NP sınıfındadır.

$$\mathbf{P} = \{ L \mid \text{There is a polynomial-time decider for } L \}$$

$$\mathbf{NP} = \{ L \mid \text{There is a nondeterministic polynomial-time decider for } L \}$$

$$\mathbf{P} \subseteq \mathbf{NP}$$

$\mathbf{P \stackrel{?}{=} NP}$

- The $\mathbf{P \stackrel{?}{=} NP}$ question is the most important question in theoretical computer science.
- With the verifier definition of **NP**, one way of phrasing this question is

If a solution to a problem can be **verified** efficiently, can that problem be **solved** efficiently?

- An answer either way will give fundamental insights into the nature of computation.

Why This Matters

- The following problems are known to be efficiently verifiable, but have no known efficient solutions:
  - Determining whether an electrical grid can be built to link up some number of houses for some price (Steiner tree problem).
  - Determining whether a simple DNA strand exists that multiple gene sequences could be a part of (shortest common supersequence).
  - Determining the best way to assign hardware resources in a compiler (optimal register allocation).
  - Determining the best way to distribute tasks to multiple workers to minimize completion time (job scheduling).
  - **And many more**.
- If **P = NP**, **all** of these problems have efficient solutions.
- If **P ≠ NP**, **none** of these problems have efficient solutions.

What We Know

- Resolving **P** $\stackrel{?}{=}$ **NP** has proven ***extremely difficult.***
- In the past 35 years:
  - Not a single correct proof either way has been found.
  - Many types of proofs have been shown to be insufficiently powerful to determine whether **P = NP**.
  - A majority of computer scientists believe **P ≠ NP**, but this isn't a large majority.
- Interesting read: Interviews with leading thinkers about **P** $\stackrel{?}{=}$ **NP**:

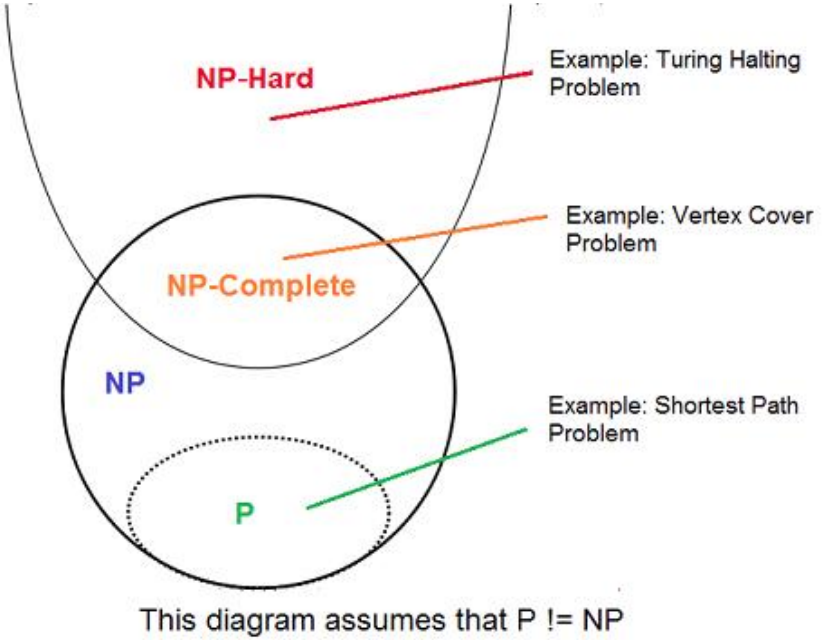
NP-Zor (NP-Hard)

- **NP-Zor (NP-Hard):** $L \in NP$ olmak üzere, $L \leq_p H$ ise H dili NP-Zor (NP-hard) olarak adlandırılır.
- Bir başka deyişle: NP içindeki her L için L den H'ye polinom zamanda bir indirgeme işlemi yapılabiliyorsa, H dili NP-hard sınıfındadır.
- NP-Hard problemlerinden elde edilen sonuç polinom zamanda doğrulamaz.

NP-Tam (NP-Complete)

- **NP-Tam (NP-complete):** Bir L dili NP-hard ve aynı zamanda NP ise, L dili NP-tam olarak adlandırılır.
- NP-tam problemler NP'deki zor problemler sınıfıdır.

P, NP, NP-Complete, NP-Hard arasındaki ilişki



NP-Tam (NP-complete)

- Bir problem NP-tam ise onun için verimli bir algoritma yoktur (P≠NP olduğu düşünülürse)
- NP-Tam problemler, NP sınıfındaki oldukça zor problemlerdir.
  - NP-Tam problemleri için bilinen bütün algoritmalar en kötü durumuda üstel zamanda çalışırlar.
  - NP-complete problemler için bir çok algoritma makul büyüklükteki girişler için kullanışsızdır.

EK: NP-Zor, NP-Tam

Bir X problemi eğer, Y problemini çözen iyi bir algoritmadan yararlanılarak elde edilen diğer iyi bir algoritma ile çözülebiliyorsa, X, Y'ye (polinom) *indirgenebilir* denir.

Örnek-3: X ve Y problemi aşağıdaki gibi olsun:

$$Y: \ ax^2 + bx + c = 0, \qquad X: \ dx + q = 0.$$

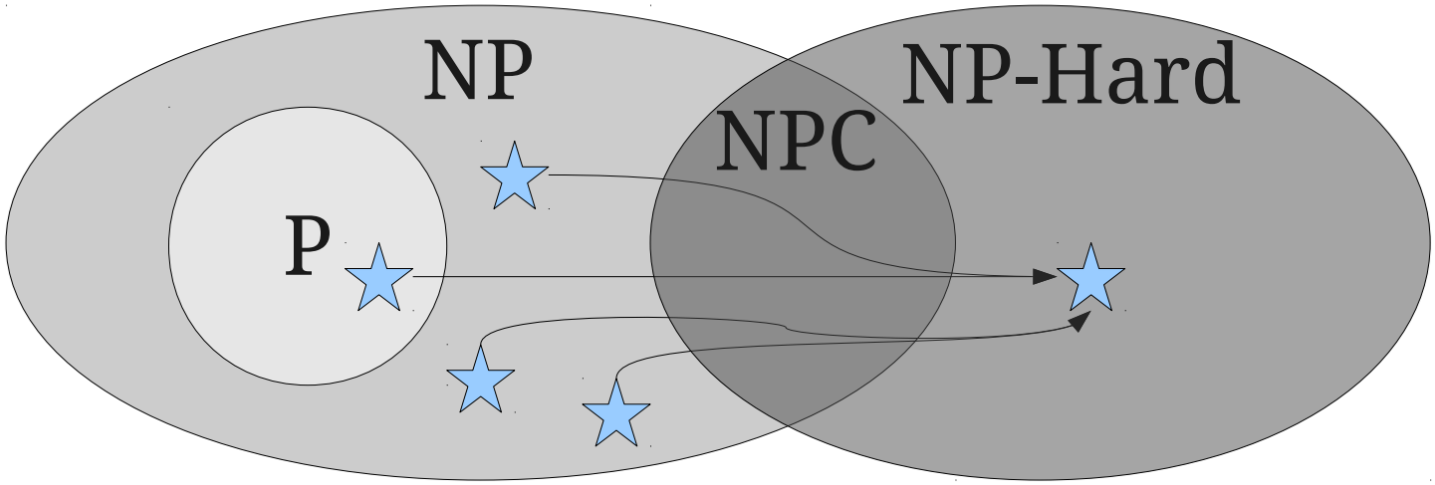
X problemi Y probleminin özel durumu olduğu için Y problemini çözen algoritmadan yararlanarak X'i de çözebiliriz. Yani X, Y'ye indirgenebilir.

Burada X ya da Y için polinom sınırlı bir algoritmanın varlığı gerekli değildir. Yalnızca birini çözen bir iyi algoritma varsa, bunun diğeri için de iyi algoritma elde etme anlamına geleceğini söylüyoruz. Bunun için de X'i Y'nin özel bir durumuna polinom sınırlı sürede indirgeyecek bir algoritma bulmak yeterlidir. Eğer NP sınıfındaki her problem Y probleminin özel durumlarına indirgenebiliyorsa, Y problemi *NP-zor* ve aynı zamanda Y'nin kendisi de NP sınıfından ise Y'ye *NP-tam problem* denir.

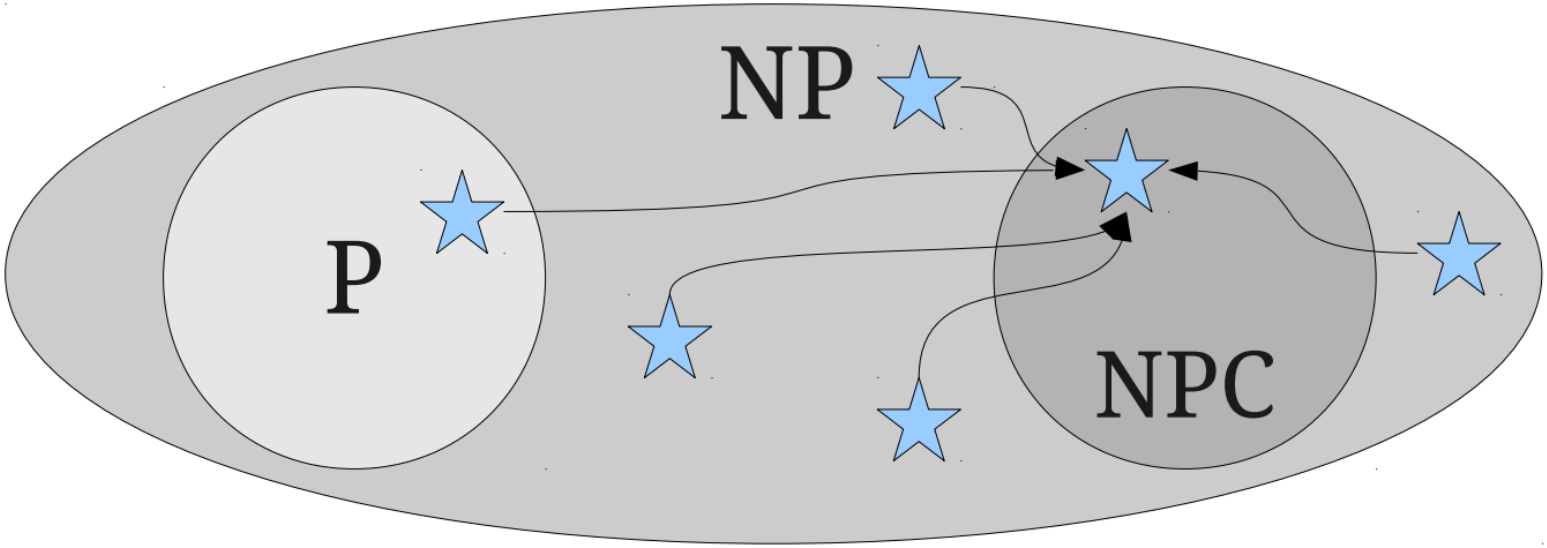
Bir NP tam probleme verimli bir çözüm yöntemi bulunması halinde diğer NP tam problemler de verimli bir şekilde çözülebileceklerdir.

NP-Zor, NP-Tam

- A language L is called **NP-hard** iff for *every* $L' \in \mathbf{NP}$, we have $L' \leq_P L$.
- A language in L is called **NP-complete** iff L is **NP**-hard and $L \in \mathbf{NP}$.
- The class **NPC** is the set of **NP**-complete problems.

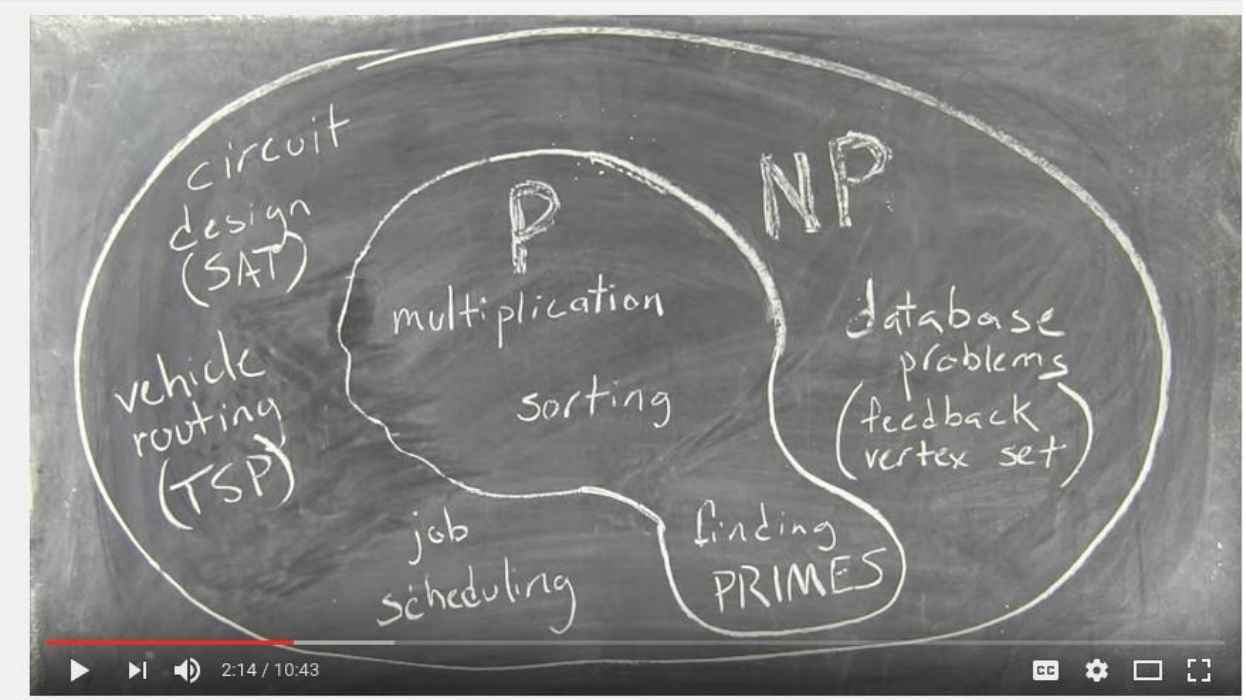


- To prove that some language L is **NP**-complete, show that $L \in$ **NP**, then reduce some known **NP**-complete problem to L.
- **Do not** reduce L to a known **NP**-complete problem.
  - We already knew you could do this; *every* **NP** problems is reducible to any **NP**-complete problem!



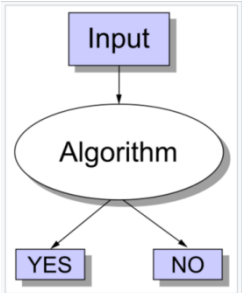
Konuyla ilgili video

- https://www.youtube.com/watch?v=YX40hbAHx3s&spfreload=10



Optimization/Decision Problems

- Optimization Problems:
  - » An optimization problem is one which asks, "What is the optimal solution to problem X?"
  - » Examples:
    - ❖ 0-1 Knapsack
    - ❖ Fractional Knapsack
    - ❖ Minimum Spanning Tree
- Decision Problems
  - » An decision problem is one which asks, "Is there a solution to problem X with property Y?"
  - » Examples:
    - ❖ Does a graph G have a MST of weight ≤ W?



A decision problem has only two possible outputs, *yes* or *no* (or alternately 1 or 0) on any input.

Optimization/Decision Problems

- An optimization problem tries to find an optimal solution
- A decision problem tries to answer a yes/no question
- Many problems will have decision and optimization versions.
  - » Eg: Traveling salesman problem
    - ❖ optimization: find hamiltonian cycle of minimum weight
    - ❖ decision: find hamiltonian cycle of weight < k
- Some problems are decidable, but *intractable*: as they grow large, we are unable to solve them in reasonable time
  - » *What constitutes "reasonable time"?*
  - » *Is there a polynomial-time algorithm that solves the problem?*

polynomial time

On an input of size n the worst-case running time is $O(n^k)$ for some constant k

Polynomial time: $O(n^2)$, $O(n^3)$, $O(1)$, $O(n \lg n)$

Not in polynomial time: $O(2^n)$, $O(n^n)$, $O(n!)$