Lecture 2 Discussion

Instructor: Karteek Sreenivasaiah

19th April 2020

Computability Theory

The main topics covered in the second NPTEL lecture:

- Two definitions of NP.
- Examples of languages in NP.

Two definitions of NP

Definition 1: Guess and Verify

A language L is in NP if there is a non-deterministic Turing machine that decides L in polynomial time.

Two definitions of NP

Definition 1: Guess and Verify

A language L is in NP if there is a non-deterministic Turing machine that decides L in polynomial time.

More formally:
A language L is in NP $\iff \exists$ TM $M, \exists k \in \mathbb{N},$:

$$\forall x \in \{0,1\}^n, x \in L \iff M \text{ accepts } x$$

and M runs in time n^k on inputs of length n.

Definition 2: Certificate and Verifier

A language L is in NP if there is a deterministic Turing machine V such that for strings $x \in L$, there is a certificate y of polynomial length, and V accepts (x, y). Further, V runs in polynomial time.

Definition 2: Certificate and Verifier

A language L is in NP if there is a deterministic Turing machine V such that for strings $x \in L$, there is a certificate y of polynomial length, and V accepts (x, y). Further, V runs in polynomial time.

More formally:
A language L is in NP $\iff \exists$ TM $V, \exists k_1, k_2 \in \mathbb{N},$:

$$\forall x \in \{0,1\}^n, x \in L \iff \exists y \in \{0,1\}^{n^{k_1}} : V \text{ accepts } (x, y)$$

and V runs in time n^{k_2} on inputs of length n.

Intuitively: For a string $x \in L$, the statement "$x \in L$" is a *theorem*, and the string y is a *proof* of the theorem. The verifier checks if the theorem and proof given are valid.

Equivalence of the two definitions

Defn 1 $\implies$ Defn 2:

We have a non-deterministic TM M deciding L in polynomial time.

We want to convert it to:

- a deterministic Turing machine V,
- show a certificate y for each $x \in L$

Equivalence of the two definitions

Defn 1 $\implies$ Defn 2:

We have a non-deterministic TM M deciding L in polynomial time.
We want to convert it to:

- a deterministic Turing machine V,
- show a certificate y for each $x \in L$

The idea is this:

- A non-deterministic TM has several branches.
- For a string $x \in L$, there is an accepting branch.
- Define certificate y to be the non-det choices along an accepting branch.

The *verifier* TM V simply simulates M along the branch indicated by y and accepts if M accepts, and rejects otherwise.
Note: if $x \notin L$, then no branch accepts. Hence no certificate y exists.

Equivalence of the two definitions

Defn 2 $\implies$ Defn 1:

Let $L \in NP$ via a deterministic verifier TM V.

We want to construct:

- a non-deterministic TM N that decides L
- N should run in polynomial time.

Equivalence of the two definitions

Defn 2 $\implies$ Defn 1:

Let $L \in NP$ via a deterministic verifier TM V.

We want to construct:

- a non-deterministic TM N that decides L
- N should run in polynomial time.

The idea is:

- Let input be x
- Non-det *guess* a string y of length $|x|^{k_1}$
- Run V on $\langle x, y \rangle$
- Accept if V accepts, else reject.

Note: Details on how to guess the string y is important!

Guessing a string

By "guess a string y of length m", we mean: Explore every possible y in parallel using non-determinism. We do this by working bit by bit

Assume we have a two tape non-det machine:

- First tape is the work tape.
- Second tape will be used for a length counter.

Guessing a string

By "guess a string y of length m", we mean: Explore every possible y in parallel using non-determinism. We do this by working bit by bit

Assume we have a two tape non-det machine:

- First tape is the work tape.
- Second tape will be used for a length counter.

The idea is to non-det branch into two copies:

- The first copy writes 0 on the next tape cell
- Increment the length counter
- If length counter is less than m, then repeat. Else stop.

The second copy is identical except it writes 1 on the tape cell.

Guessing a string

The key observation is:
To guess a string, you need to know the length beforehand!

In the case of "Defn 2 $\implies$ Defn 1", we should do:

- Read input x from left to right
- Maintain a counter on tape 2 to measure the length n of x
- Compute $m = n^{k_1}$ on tape 2

Guessing a string

The key observation is:
To guess a string, you need to know the length beforehand!

In the case of "Defn 2 $\implies$ Defn 1", we should do:

- Read input x from left to right
- Maintain a counter on tape 2 to measure the length n of x
- Compute $m = n^{k_1}$ on tape 2

Guessing string y of length m is done bit by bit while decrementing the value on tape 2.

kCLIQUE

Exercise:

$$k\text{CLIQUE} = \{\langle G \rangle \mid G \text{ is a graph with a clique of size } k\}$$

Show that $k-$CLIQUE is in NP using definitions 1 and 2.

coNP

Towards the end of this lecture, he defines coNP:

$$\mathsf{coNP} = \{L \mid \overline{L} \in \mathsf{NP}\}$$

Important: coNP is **not** the complement of NP!!
coNP is the set of languages *whose complements* are in NP.

Easy to see that $\mathsf{P} \subseteq \mathsf{NP} \cap \mathsf{coNP}$.
Exercise: Show that if $\mathsf{NP} \neq \mathsf{coNP}$, then $\mathsf{P} \neq \mathsf{NP}$.

Exercise: Running time

Condsider the BTech Ist year algorithm to test if a number is *prime*:

- Input n
- FOR $i = 0; i \leq n/2$
  - If i divides n, output "Not prime"
- Output "Prime"

What is the running time of the above algorithm?

Exercise: Running time

Condsider the BTech Ist year algorithm to test if a number is *prime*:

- Input n
- FOR $i = 0$; $i \leq n/2$
  - If i divides n, output "Not prime"
- Output "Prime"

What is the running time of the above algorithm?
(Hint: first write down the input size)

Thank you!