Foundations of Computing

Lecture 20

Arkady Yerukhimovich

April 4, 2024

Outline

1. Lecture 19 Review
2. Verifying vs. Deciding
3. Nondeterministic Polynomial Time

Lecture 19 Review

- Polynomial Time Computation
- The Complexity Class $\mathcal{P}$

$$\mathcal{P} = \bigcup_k TIME(n^k)$$

Outline

1. Lecture 19 Review
2. Verifying vs. Deciding
3. Nondeterministic Polynomial Time

The Class $\mathcal{P}$

- $\mathcal{P}$ is the class of languages decidable in polynomial time

The Class $\mathcal{P}$

- $\mathcal{P}$ is the class of languages decidable in polynomial time
- Many examples of such (efficiently decidable) languages:
  - PATH
  - RELPRIME
  - Pretty much everything you studied in algorithms class

The Class $\mathcal{P}$

- $\mathcal{P}$ is the class of languages decidable in polynomial time
- Many examples of such (efficiently decidable) languages:
  - PATH
  - RELPRIME
  - Pretty much everything you studied in algorithms class
- But, some problems have resisted our efforts to find efficient algorithms

The Class $\mathcal{P}$

- $\mathcal{P}$ is the class of languages decidable in polynomial time
- Many examples of such (efficiently decidable) languages:
  - PATH
  - RELPRIME
  - Pretty much everything you studied in algorithms class
- But, some problems have resisted our efforts to find efficient algorithms
- Today we will study one important class of such problems

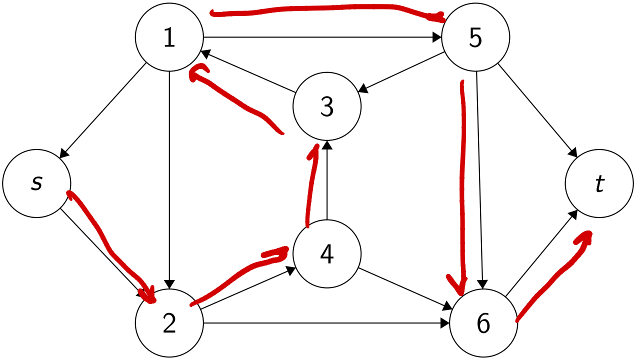
Hamiltonian Path

Hamiltonian Path

A Hamiltonian path in directed graph G is a path that goes through each node exactly once.

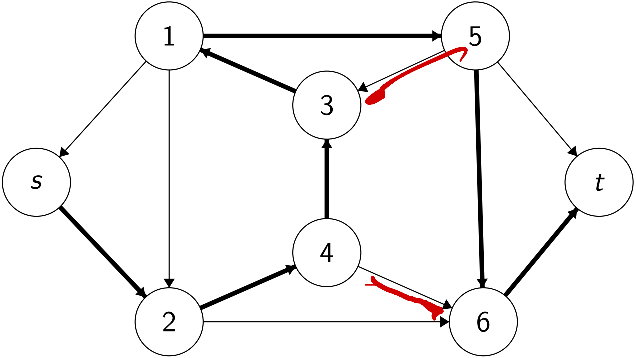
Hamiltonian Path

A Hamiltonian path in directed graph G is a path that goes through each node exactly once.



Hamiltonian Path

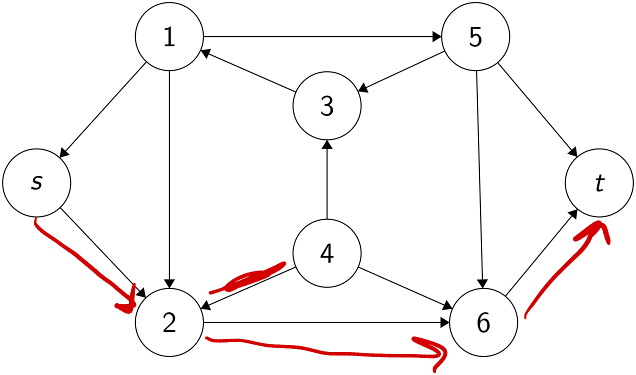
A Hamiltonian path in directed graph G is a path that goes through each node exactly once.



Hamiltonian Path

Hamiltonian Path

A Hamiltonian path in directed graph G is a path that goes through each node exactly once.
But, not every graph has a Hamiltonian Path.



Hamiltonian Path

Hamiltonian Path Problem

$$HAMPATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t\}$$

Hamiltonian Path

Hamiltonian Path Problem

$$HAMPATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t\}$$

- Easy to find an exponential time algorithm for *HAMPATH*

$n!$

Hamiltonian Path

Hamiltonian Path Problem

$$HAMPATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t\}$$

- Easy to find an exponential time algorithm for *HAMPATH*
- But, no one knows a polynomial time algorithm for it

Hamiltonian Path

Hamiltonian Path Problem

$$HAMPATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t\}$$

- Easy to find an exponential time algorithm for *HAMPATH*
- But, no one knows a polynomial time algorithm for it

Polynomial Verifiability

However, given a path from s to t, can easily verify whether it is Hamiltonian in polynomial time.

Satisfiability

Boolean Formula

A Boolean formula is an expression inolving Boolean variables and logic operations AND ($\wedge$), OR ($\vee$), and NOT ($\neg$ or $\overline{x}$).

$$\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$$

Satisfiability

Boolean Formula

A Boolean formula is an expression inolving Boolean variables and logic operations AND ($\wedge$), OR ($\vee$), and NOT ($\neg$ or $\overline{x}$).

$$\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$$

- A satisfying assignment is an assignment of 0 or 1 to the variables such that the formula evaluates to 1
- Example: $x = 0$, $y = 1$, $z = 0$ is a satisfying assignment for ϕ

Satisfiability

Boolean Formula

A Boolean formula is an expression inolving Boolean variables and logic operations AND ($\wedge$), OR ($\vee$), and NOT ($\neg$ or $\overline{x}$).

$$\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$$

- A satisfying assignment is an assignment of 0 or 1 to the variables such that the formula evaluates to 1
- Example: $x = 0$, $y = 1$, $z = 0$ is a satisfying assignment for ϕ
- We say that formula ϕ is satisfiable if it has a satisfying assignment

Satisfiability

Boolean Formula

A Boolean formula is an expression inolving Boolean variables and logic operations AND ($\wedge$), OR ($\vee$), and NOT ($\neg$ or $\overline{x}$).

$$\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$$

- A satisfying assignment is an assignment of 0 or 1 to the variables such that the formula evaluates to 1
- Example: $x = 0$, $y = 1$, $z = 0$ is a satisfying assignment for ϕ
- We say that formula ϕ is satisfiable if it has a satisfying assignment
- Not all formulas are satisfiable

$$\phi' = (\overline{x} \wedge y) \wedge (x \wedge \overline{z})$$

if $x = 0$

if $x = 1$

Satisfiability

Satisfiability Problem

$$SAT = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$$

Satisfiability

Satisfiability Problem

$$SAT = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$$

- Easy to find an exponential time algorithm for *SAT*

Satisfiability

Satisfiability Problem

$$SAT = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$$

- Easy to find an exponential time algorithm for *SAT*
- But, it is widely believed no polynomial time algorithm exists

Satisfiability

Satisfiability Problem

$$SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$$

- Easy to find an exponential time algorithm for *SAT*
- But, it is widely believed no polynomial time algorithm exists

Polynomial Verifiability

However, given an assignment (i.e., values for all the variables), can easily verify whether ϕ is satisfied by this assignment in polynomial time.

Verifiability of a Language

A verifier for a language L is an algorithm V, where

$$L = \{x \mid V \text{ accepts } \langle x, w \rangle \text{ for some string } w\}$$

Verifiability of a Language

A verifier for a language L is an algorithm V, where

$$L = \{x \mid V \text{ accepts } \langle x, w \rangle \text{ for some string } w\}$$

- Runtime of V is measured as a function of $|x|$

Verifiability of a Language

A verifier for a language L is an algorithm V, where

$$L = \{x \mid V \text{ accepts } \langle x, w \rangle \text{ for some string } w\}$$

- Runtime of V is measured as a function of $|x|$
- V is a polynomial time verifier if it runs in time $poly(|x|)$

$|w| \leq poly(|x|)$

Verifiability of a Language

A verifier for a language L is an algorithm V, where

$$L = \{x \mid V \text{ accepts } \langle x, w \rangle \text{ for some string } w\}$$

- Runtime of V is measured as a function of $|x|$
- V is a polynomial time verifier if it runs in time $poly(|x|)$
- L is polynomially verfiable if it has a polynomial time verifier

Verifiability of a Language

A verifier for a language L is an algorithm V, where

$$L = \{x \mid V \text{ accepts } \langle x, w \rangle \text{ for some string } w\}$$

- Runtime of V is measured as a function of $|x|$
- V is a polynomial time verifier if it runs in time $poly(|x|)$
- L is polynomially verfiable if it has a polynomial time verifier
- String w is called a witness that $x \in L$

SAT   x = formula   w = assignment

The Class $\mathcal{NP}$

Definition

$\mathcal{NP}$ is the class of languages that have polynomial time verifiers.

The Class $\mathcal{NP}$

Definition

$\mathcal{NP}$ is the class of languages that have polynomial time verifiers.

- We already saw that *HAMPATH* and *SAT* are in $\mathcal{NP}$

Definition

$\mathcal{NP}$ is the class of languages that have polynomial time verifiers.

- We already saw that *HAMPATH* and *SAT* are in $\mathcal{NP}$
- Every $L \in \mathcal{P}$ is also in $\mathcal{NP}$:

$\exists$ D s.t. D decides L in poly-time

V(x, w)

1. Run D(x)
   output that

The Class $\mathcal{NP}$

Definition

$\mathcal{NP}$ is the class of languages that have polynomial time verifiers.

- We already saw that *HAMPATH* and *SAT* are in $\mathcal{NP}$
- Every $L \in \mathcal{P}$ is also in $\mathcal{NP}$: $\mathcal{P} \subseteq \mathcal{NP}$

The Class $\mathcal{NP}$

Definition

$\mathcal{NP}$ is the class of languages that have polynomial time verifiers.

- We already saw that *HAMPATH* and *SAT* are in $\mathcal{NP}$
- Every $L \in \mathcal{P}$ is also in $\mathcal{NP}$: $\quad \mathcal{P} \subseteq \mathcal{NP}$

Intuition

- $\mathcal{P}$ is the class of problems where you can find a solution in poly-time

The Class $\mathcal{NP}$

Definition

$\mathcal{NP}$ is the class of languages that have polynomial time verifiers.

- We already saw that *HAMPATH* and *SAT* are in $\mathcal{NP}$
- Every $L \in \mathcal{P}$ is also in $\mathcal{NP}$: $\mathcal{P} \subseteq \mathcal{NP}$

Intuition

- $\mathcal{P}$ is the class of problems where you can find a solution in poly-time
- $\mathcal{NP}$ is the class of problems where you can verify a solution in poly-time

The Class $\mathcal{NP}$

Definition

$\mathcal{NP}$ is the class of languages that have polynomial time verifiers.

- We already saw that *HAMPATH* and *SAT* are in $\mathcal{NP}$
- Every $L \in \mathcal{P}$ is also in $\mathcal{NP}$: $\mathcal{P} \subseteq \mathcal{NP}$

Intuition

- $\mathcal{P}$ is the class of problems where you can find a solution in poly-time
- $\mathcal{NP}$ is the class of problems where you can verify a solution in poly-time
- Question: $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$

Outline

The Class $\mathcal{NP}$ – Another Formulation

- $\mathcal{NP}$ stands for non-deterministic polynomial time

The Class $\mathcal{NP}$ – Another Formulation

- $\mathcal{NP}$ stands for non-deterministic polynomial time
- $\mathcal{NP}$ is the set of languages decided by poly-time NTMs

The Class $\mathcal{NP}$ – Another Formulation

- $\mathcal{NP}$ stands for non-deterministic polynomial time
- $\mathcal{NP}$ is the set of languages decided by poly-time NTMs

Theorem

The two definitions of $\mathcal{NP}$ are equivalent:
For any language L,

$$L \text{ is poly-time verifiable} \iff L \text{ is decided by a poly-time NTM}$$

The Class $\mathcal{NP}$ – Another Formulation

- $\mathcal{NP}$ stands for non-deterministic polynomial time
- $\mathcal{NP}$ is the set of languages decided by poly-time NTMs

Theorem

The two definitions of $\mathcal{NP}$ are equivalent:
For any language L,

$$L \text{ is poly-time verifiable} \iff L \text{ is decided by a poly-time NTM}$$

Proof Idea:

The Class $\mathcal{NP}$ – Another Formulation

- $\mathcal{NP}$ stands for non-deterministic polynomial time
- $\mathcal{NP}$ is the set of languages decided by poly-time NTMs

Theorem

The two definitions of $\mathcal{NP}$ are equivalent:
For any language L,

$$L \text{ is poly-time verifiable} \iff L \text{ is decided by a poly-time NTM}$$

Proof Idea:

- Need to prove both directions

The Class $\mathcal{NP}$ – Another Formulation

- $\mathcal{NP}$ stands for non-deterministic polynomial time
- $\mathcal{NP}$ is the set of languages decided by poly-time NTMs

Theorem

The two definitions of $\mathcal{NP}$ are equivalent:
For any language L,

$$L \text{ is poly-time verifiable} \iff L \text{ is decided by a poly-time NTM}$$

Proof Idea:

- Need to prove both directions
- ($\Rightarrow$) An NTM simulates the verifier by guessing the witness w

The Class $\mathcal{NP}$ – Another Formulation

- $\mathcal{NP}$ stands for non-deterministic polynomial time
- $\mathcal{NP}$ is the set of languages decided by poly-time NTMs

Theorem

The two definitions of $\mathcal{NP}$ are equivalent:
For any language L,

$$L \text{ is poly-time verifiable} \iff L \text{ is decided by a poly-time NTM}$$

Proof Idea:

- Need to prove both directions
- ($\Rightarrow$) An NTM simulates the verifier by guessing the witness w
- ($\Leftarrow$) A verifier simulates the NTM by using the accepting branch as the witness

Equivalence of $\mathcal{NP}$ Definitions

Theorem

L is poly-time verifiable $\iff$ L is decided by a poly-time NTM

Equivalence of $\mathcal{NP}$ Definitions

Theorem

L is poly-time verifiable $\iff$ L is decided by a poly-time NTM

Proof:

1. ($\Rightarrow$) Verifiability implies decidability by NTM

Equivalence of $\mathcal{NP}$ Definitions

Theorem

L is poly-time verifiable $\iff$ L is decided by a poly-time NTM

Proof:

1. ($\Rightarrow$) Verifiability implies decidability by NTM
   - Let V be a verifier for L running in time n^k

Equivalence of $\mathcal{NP}$ Definitions

Theorem

L is poly-time verifiable $\iff$ L is decided by a poly-time NTM

Proof:

1. ($\Rightarrow$) Verifiability implies decidability by NTM
    - Let V be a verifier for L running in time n^k
    - Construct NTM N as follows: On input x of length n

Equivalence of $\mathcal{NP}$ Definitions

Theorem

L is poly-time verifiable $\iff$ L is decided by a poly-time NTM

Proof:

1. ($\Rightarrow$) Verifiability implies decidability by NTM
    - Let V be a verifier for L running in time n^k
    - Construct NTM N as follows: On input x of length n
        1. Nondeterministically select string w of length n^k

Equivalence of $\mathcal{NP}$ Definitions

Theorem

L is poly-time verifiable $\iff$ L is decided by a poly-time NTM

Proof:

1. ($\Rightarrow$) Verifiability implies decidability by NTM
   - Let V be a verifier for L running in time n^k
   - Construct NTM N as follows: On input x of length n
     1. Nondeterministically select string w of length n^k
     2. Run V on input $\langle x, w \rangle$

Equivalence of $\mathcal{NP}$ Definitions

Theorem

L is poly-time verifiable $\iff$ L is decided by a poly-time NTM

Proof:

1. ($\Rightarrow$) Verifiability implies decidability by NTM
   - Let V be a verifier for L running in time n^k
   - Construct NTM N as follows: On input x of length n
     1. Nondeterministically select string w of length n^k
     2. Run V on input $\langle x, w \rangle$
     3. Accept if V accepts and reject otherwise

→ if $x \in L$, $\exists w$ s.t. $V(x, w) = 1$

if $x \notin L$, $\forall w$ $V(x, w) = 0$

Equivalence of $\mathcal{NP}$ Definitions

Theorem

L is poly-time verifiable $\iff$ L is decided by a poly-time NTM

Proof:

1. ($\Rightarrow$) Verifiability implies decidability by NTM
   - Let V be a verifier for L running in time n^k
   - Construct NTM N as follows: On input x of length n
     1. Nondeterministically select string w of length n^k
     2. Run V on input $\langle x, w \rangle$
     3. Accept if V accepts and reject otherwise
2. ($\Leftarrow$) Decidability by NTM implies verifiability

Equivalence of $\mathcal{NP}$ Definitions

Theorem

L is poly-time verifiable $\iff$ L is decided by a poly-time NTM

Proof:

1. ($\Rightarrow$) Verifiability implies decidability by NTM
   - Let V be a verifier for L running in time n^k
   - Construct NTM N as follows: On input x of length n
     1. Nondeterministically select string w of length n^k
     2. Run V on input $\langle x, w \rangle$
     3. Accept if V accepts and reject otherwise
2. ($\Leftarrow$) Decidability by NTM implies verifiability
   - Let N be an NTM deciding L

Equivalence of $\mathcal{NP}$ Definitions

Theorem

L is poly-time verifiable $\iff$ L is decided by a poly-time NTM

Proof:

1. ($\Rightarrow$) Verifiability implies decidability by NTM
   - Let V be a verifier for L running in time n^k
   - Construct NTM N as follows: On input x of length n
     1. Nondeterministically select string w of length n^k
     2. Run V on input $\langle x, w \rangle$
     3. Accept if V accepts and reject otherwise
2. ($\Leftarrow$) Decidability by NTM implies verifiability
   - Let N be an NTM deciding L
   - Construct verifier V as follows: On input $\langle x, w \rangle$,

Equivalence of $\mathcal{NP}$ Definitions

Theorem

L is poly-time verifiable $\iff$ L is decided by a poly-time NTM

Proof:

1. ($\Rightarrow$) Verifiability implies decidability by NTM
   - Let V be a verifier for L running in time n^k
   - Construct NTM N as follows: On input x of length n
     1. Nondeterministically select string w of length n^k
     2. Run V on input $\langle x, w \rangle$
     3. Accept if V accepts and reject otherwise
2. ($\Leftarrow$) Decidability by NTM implies verifiability
   - Let N be an NTM deciding L
   - Construct verifier V as follows: On input $\langle x, w \rangle$,
     1. Simulate N on input x, treating each symbol of w as a description of the nondeterministic choice to make at each step

Equivalence of $\mathcal{NP}$ Definitions

Theorem

L is poly-time verifiable $\iff$ L is decided by a poly-time NTM

Proof:

1. ($\Rightarrow$) Verifiability implies decidability by NTM
   - Let V be a verifier for L running in time n^k
   - Construct NTM N as follows: On input x of length n
     1. Nondeterministically select string w of length n^k
     2. Run V on input $\langle x, w \rangle$
     3. Accept if V accepts and reject otherwise
2. ($\Leftarrow$) Decidability by NTM implies verifiability
   - Let N be an NTM deciding L
   - Construct verifier V as follows: On input $\langle x, w \rangle$,
     1. Simulate N on input x, treating each symbol of w as a description of the nondeterministic choice to make at each step
     2. If this branch of N's computation accepts, accept, otherwise reject

The Class $\mathcal{NP}$

We can define the class of languages decided by poly-time NTMs

Definition

$$NTIME(t(n)) = \{L \mid L \text{ is a language decided by a } O(t(n)) \text{ time NTM}\}$$

The Class $\mathcal{NP}$

We can define the class of languages decided by poly-time NTMs

Definition

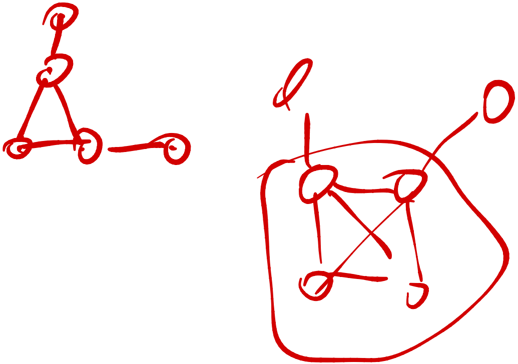
$$NTIME(t(n)) = \{L \mid L \text{ is a language decided by a } O(t(n)) \text{ time NTM}\}$$

$$\mathcal{NP} = \bigcup_k NTIME(n^k)$$

Problems in $\mathcal{NP}$ – Example 1

Clique

A clique in and undirected graph is a subset of nodes s.t. every two nodes are connected by an edge. A k-clique is a clique containing k nodes



Problems in $\mathcal{NP}$ – Example 1

Clique

A clique in and undirected graph is a subset of nodes s.t. every two nodes are connected by an edge. A k-clique is a clique containing k nodes

$$CLIQUE = \{\langle G, k\rangle \mid G \text{ is an undirected graph with a } k\text{-clique}\}$$

Problems in $\mathcal{NP}$ – Example 2

Subset Sum

Given a collection of integers $\{x_1, \ldots, x_k\}$ is there a subset of them that adds up to k?

Problems in $\mathcal{NP}$ – Example 2

Subset Sum

Given a collection of integers $\{x_1, \dots, x_k\}$ is there a subset of them that adds up to t?

$$SUBSET-SUM = \{\langle S, t\rangle \mid S = \{x_1, \dots, x_k\} \text{ and for some } \{y_1, \dots, y_l\} \subseteq \{x_1, \dots, x_k\}, \sum y_i = t\}$$

The Million Dollar Question

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

The Million Dollar Question

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

- Is it easier to verify a solution than to find that solution?
- This is the biggest open question in complexity theory

Let's Try to Answer It

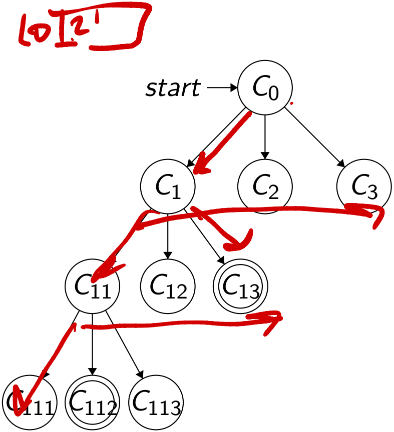
Theorem

Every nondeterministic TM has an equivalent deterministic TM.

Let's Try to Answer It

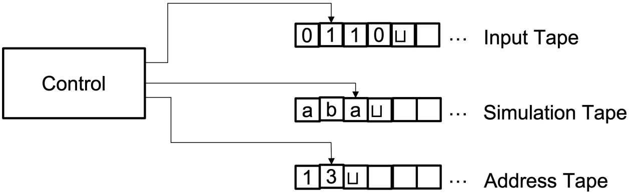
Theorem

Every nondeterministic TM has an equivalent deterministic TM.



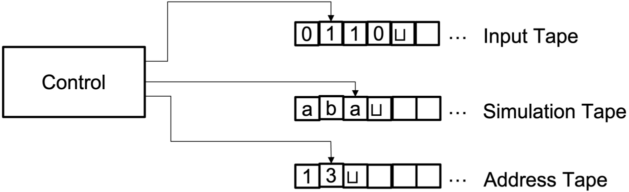
- Recall that an execution of a DTM is a sequence of configurations
- Execution of an NTM is a tree of configurations (branches correspond to non-deterministic choices)
- If any node in the tree is an accept node, the NTM accepts
- To simulate an NTM by a DTM, need to try all configurations in the tree to see if we find an accepting one

Simulating NTM on a 3-tape DTM



To simulate an NTM N by a DTM D, we use three tapes:

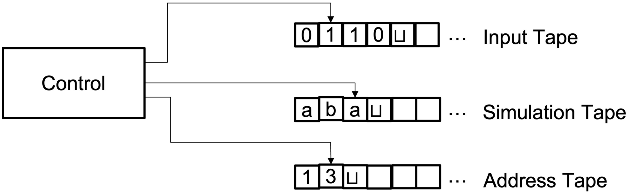
Simulating NTM on a 3-tape DTM



To simulate an NTM N by a DTM D, we use three tapes:

1. Input tape – stores the input and doesn't change

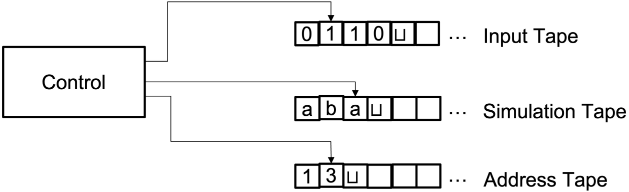
Simulating NTM on a 3-tape DTM



To simulate an NTM N by a DTM D, we use three tapes:

1. Input tape – stores the input and doesn't change
2. Simulation tape – work tape for the NTM on one branch of nondeterminism

Simulating NTM on a 3-tape DTM



To simulate an NTM N by a DTM D, we use three tapes:

1. Input tape – stores the input and doesn't change
2. Simulation tape – work tape for the NTM on one branch of nondeterminism
3. Address tape – use to store which nondeterministic branch you are on

Simulating NTM on a 3-tape DTM

Simulating an NTM N

1. Start with input w on tape 1, and tapes 2,3 empty

Simulating NTM on a 3-tape DTM

Simulating an NTM N

1. Start with input w on tape 1, and tapes 2,3 empty
2. Copy w to tape 2

Simulating NTM on a 3-tape DTM

Simulating an NTM N

1. Start with input w on tape 1, and tapes 2,3 empty
2. Copy w to tape 2
3. Use tape 2 to simulate a run of N. Whenever it needs to make a non-deterministic choice, see next symbol on tape 3 for which branch to take. If no symbols left, go to step 4

Simulating NTM on a 3-tape DTM

Simulating an NTM N

1. Start with input w on tape 1, and tapes 2,3 empty
2. Copy w to tape 2
3. Use tape 2 to simulate a run of N. Whenever it needs to make a non-deterministic choice, see next symbol on tape 3 for which branch to take. If no symbols left, go to step 4
4. Replace string on tape 3 with the lexicographically next one (move onto next non-deterministic branch)

Simulating NTM on a 3-tape DTM

Simulating an NTM N

1. Start with input w on tape 1, and tapes 2,3 empty
2. Copy w to tape 2
3. Use tape 2 to simulate a run of N. Whenever it needs to make a non-deterministic choice, see next symbol on tape 3 for which branch to take. If no symbols left, go to step 4
4. Replace string on tape 3 with the lexicographically next one (move onto next non-deterministic branch)
5. If N ever enters an accept state, stop and accept

Simulating NTM on a 3-tape DTM

Simulating an NTM N

1. Start with input w on tape 1, and tapes 2,3 empty
2. Copy w to tape 2
3. Use tape 2 to simulate a run of N. Whenever it needs to make a non-deterministic choice, see next symbol on tape 3 for which branch to take. If no symbols left, go to step 4
4. Replace string on tape 3 with the lexicographically next one (move onto next non-deterministic branch)
5. If N ever enters an accept state, stop and accept

What's the Problem?

Simulating NTM on a 3-tape DTM

Simulating an NTM N

1. Start with input w on tape 1, and tapes 2,3 empty
2. Copy w to tape 2
3. Use tape 2 to simulate a run of N. Whenever it needs to make a non-deterministic choice, see next symbol on tape 3 for which branch to take. If no symbols left, go to step 4
4. Replace string on tape 3 with the lexicographically next one (move onto next non-deterministic branch)
5. If N ever enters an accept state, stop and accept

What's the Problem?

- NTM running in time $t(n)$, makes $O(t(n))$ non-deterministic choices

Simulating NTM on a 3-tape DTM

Simulating an NTM N

1. Start with input w on tape 1, and tapes 2,3 empty
2. Copy w to tape 2
3. Use tape 2 to simulate a run of N. Whenever it needs to make a non-deterministic choice, see next symbol on tape 3 for which branch to take. If no symbols left, go to step 4
4. Replace string on tape 3 with the lexicographically next one (move onto next non-deterministic branch)
5. If N ever enters an accept state, stop and accept

What's the Problem?

- NTM running in time $t(n)$, makes $O(t(n))$ non-deterministic choices
- Above algorithm tries all possible values for these branches: $2^{O(t(n))}$

Simulating NTM on a 3-tape DTM

Simulating an NTM N

1. Start with input w on tape 1, and tapes 2,3 empty
2. Copy w to tape 2
3. Use tape 2 to simulate a run of N. Whenever it needs to make a non-deterministic choice, see next symbol on tape 3 for which branch to take. If no symbols left, go to step 4
4. Replace string on tape 3 with the lexicographically next one (move onto next non-deterministic branch)
5. If N ever enters an accept state, stop and accept

What's the Problem?

- NTM running in time $t(n)$, makes $O(t(n))$ non-deterministic choices
- Above algorithm tries all possible values for these branches: $2^{O(t(n))}$
- Resulting DTM runs in exponential time

Next Week

- We will study properties of languages in $\mathcal{NP}$

Next Week

- We will study properties of languages in $\mathcal{NP}$
- We will show that there are $\mathcal{NP}$-complete languages that are as hard as any other language in $\mathcal{NP}$

Next Week

- We will study properties of languages in $\mathcal{NP}$
- We will show that there are $\mathcal{NP}$-complete languages that are as hard as any other language in $\mathcal{NP}$
- We will show this using reductions – Yay!