Complexity Classes P and NP

Lecture 36
Sections 14.3 - 14.5

Robb T. Koether

Hampden-Sydney College

Mon, Nov 28, 2016

Outline

1. Multitape Turing Machines
2. The Classes $DTIME(T(n))$ and $NTIME(T(N))$
3. Examples
   - The Satisfiability Problem
   - The Hamiltonian Path Problem
   - The Clique Problem
   - The Vertex Cover Problem
4. Assignment

Multitape Turing Machines

Theorem

If a two-tape Turing machine can carry out a computation in n steps, then a one-tape Turing machine can simulate that computation in $O(n^2)$ *steps.*

Multitape Turing Machines

Theorem

If a two-tape Turing machine can carry out a computation in n steps, then a one-tape Turing machine can simulate that computation in $O(n^2)$ *steps.*

Corollary

If a two-tape Turing machine can carry out a computation in $T(n)$ *steps, then a one-tape Turing machine can simulate that computation in* $O(T(n)^2)$ *steps.*

Multitape Turing Machines

Proof.

- For each move of the two-tape machine, the one-tape machine must scan its tape to read the contents stored on Tape 2 of the two-tape machine.
- Each step of the two-tape machine can add at most 2 cells, one on each tape.
- After n moves, there are $O(n)$ cells to scan.
- Thus, the one-tape machine can simulate the computation in $n \cdot O(n) = O(n^2)$ moves.

$\square$

Multitape Turing Machines

Corollary

If a k-tape Turing machine can carry out a computation in n steps, then a one-tape Turing machine can simulate that computation in $O(n^2)$ steps.

Multitape Turing Machines

Corollary

If a k-tape Turing machine can carry out a computation in n steps, then a one-tape Turing machine can simulate that computation in $O(n^2)$ steps.

Proof.

Note that $2n$ and kn are both in $O(n)$. The rest of the proof is the same. $\square$

Multitape Turing Machines

Theorem

If a nondeterministic Turing machine M can carry out a computation in n steps, then a standard Turing machine can simulate that computation in $O(k^{an})$ steps, for some constants k and a dependent on M, but not on n.

Multitape Turing Machines

Theorem

If a nondeterministic Turing machine M can carry out a computation in n steps, then a standard Turing machine can simulate that computation in $O(k^{an})$ steps, for some constants k and a dependent on M, but not on n.

- Note that what the nondeterministic Turing machine can do in linear time may take exponential time on a deterministic Turing machine.

Outline

Deciding in time $T(n)$

Definition

A Turing machine M **decides a language L** in time $T(n)$) if M decides every $w \in L$ in at most time $T(n)$, where $n = |w|$.

Deciding in time $T(n)$

Definition

A Turing machine M decides a language L in time $T(n)$) if M decides every $w \in L$ in at most time $T(n)$, where $n = |w|$.

What about strings not in L?

Deciding in time $T(n)$

Definition

A Turing machine M decides a language L in time $T(n)$) if M decides every $w \in L$ in at most time $T(n)$, where $n = |w|$.

What about strings not in L?

Definition

A nondeterministic Turing machine M decides a language L in time $T(n)$ if for every $w \in L$, there is at least one path to acceptance and M halts on all inputs w in at most $T(n)$ moves, where $n = |w|$.

The Classes $DTIME(T(n))$ and $NTIME(T(N))$

Definition

A language L is in the class $DTIME(T(n))$ if there exists a deterministic multitape Turing machine that decides L in at most time $T(n)$.

The Classes $DTIME(T(n))$ and $NTIME(T(N))$

Definition

A language L is in the class $DTIME(T(n))$ if there exists a deterministic multitape Turing machine that decides L in at most time $T(n)$.

Definition

A language L is in the class $NTIME(T(n))$ if there exists a nondeterministic multitape Turing machine that decides L in at most time $T(n)$.

Definition (The Class **P**)

The class **P** is the class of all languages that can be decided *deterministically* in polynomial time. That is,

$$\mathbf{P} = \bigcup_{i=1}^{\infty} DTIME(n^i).$$

Definition (The Class **P**)

The class **P** is the class of all languages that can be decided *deterministically* in polynomial time. That is,

$$\mathbf{P} = \bigcup_{i=1}^{\infty} DTIME(n^i).$$

Definition (The Class **NP**)

The class **NP** is the class of all languages that can be decided *nondeterministically* in polynomial time. That is,

$$\mathbf{NP} = \bigcup_{i=1}^{\infty} NTIME(n^i).$$

Outline

Outline

SAT is in **NP**

Example (SAT is in NP)

- Set up the problem.

SAT is in **NP**

Example (SAT is in **NP**)

- Set up the problem.
  - Let n be the length of the boolean expression e (in CNF). (n = Number of literals.)

SAT is in **NP**

Example (SAT is in **NP**)

- Set up the problem.
    - Let n be the length of the boolean expression e (in CNF). (n = Number of literals.)
    - Encode e on the tape using the alphabet

$$\Sigma = \{x, 0, 1, \overline{x}, \wedge, \vee, (,)\}.$$

SAT is in **NP**

Example (SAT is in **NP**)

- Set up the problem.
  - Let n be the length of the boolean expression e (in CNF). (n = Number of literals.)
  - Encode e on the tape using the alphabet

$$\Sigma = \{x, 0, 1, \overline{x}, \wedge, \vee, (,)\}.$$

  - This would require $O(n \log n)$ cells.

SAT is in **NP**

Example (SAT is in **NP**)

- Set up the problem.
  - Let n be the length of the boolean expression e (in CNF). (n = Number of literals.)
  - Encode e on the tape using the alphabet

$$\Sigma = \{x, 0, 1, \overline{x}, \wedge, \vee, (,)\}.$$

  - This would require $O(n \log n)$ cells.
- Generate a solution.

SAT is in **NP**

Example (SAT is in **NP**)

- Set up the problem.
  - Let n be the length of the boolean expression e (in CNF). (n = Number of literals.)
  - Encode e on the tape using the alphabet

$$\Sigma = \{x, 0, 1, \overline{x}, \wedge, \vee, (,)\}.$$

  - This would require $O(n \log n)$ cells.
- Generate a solution.
  - Nondeterministically choose a boolean value for each of the variables.

SAT is in **NP**

Example (SAT is in **NP**)

- Set up the problem.
  - Let n be the length of the boolean expression e (in CNF). (n = Number of literals.)
  - Encode e on the tape using the alphabet

$$\Sigma = \{x, 0, 1, \overline{x}, \wedge, \vee, (,)\}.$$

  - This would require $O(n \log n)$ cells.
- Generate a solution.
  - Nondeterministically choose a boolean value for each of the variables.
  - This can be done in $O(n)$ time.

SAT is in **NP**

Example (SAT is in **NP**)

- Set up the problem.
  - Let n be the length of the boolean expression e (in CNF). (n = Number of literals.)
  - Encode e on the tape using the alphabet

$$\Sigma = \{x, 0, 1, \overline{x}, \wedge, \vee, (,)\}.$$

  - This would require $O(n \log n)$ cells.
- Generate a solution.
  - Nondeterministically choose a boolean value for each of the variables.
  - This can be done in $O(n)$ time.
  - Write the potential solution on Tape 2.

SAT is in **NP**

Example (SAT is in **NP**)

- Verify the solution.

SAT is in **NP**

Example (SAT is in **NP**)

- Verify the solution.
  - For each x_i, get its value from Tape 2.

SAT is in **NP**

Example (SAT is in **NP**)

- Verify the solution.
  - For each x_i, get its value from Tape 2.
  - Scan Tape 1, marking the clauses that it satisfies.

SAT is in **NP**

Example (SAT is in **NP**)

- Verify the solution.
  - For each x_i, get its value from Tape 2.
  - Scan Tape 1, marking the clauses that it satisfies.
  - This can be done in time $O(n \log n)$ for each variable.

SAT is in **NP**

Example (SAT is in **NP**)

- Verify the solution.
  - For each x_i, get its value from Tape 2.
  - Scan Tape 1, marking the clauses that it satisfies.
  - This can be done in time $O(n \log n)$ for each variable.
  - So it can be done for all the variables in $O(n^2 \log n)$ time.

SAT is in **NP**

Example (SAT is in **NP**)

- Verify the solution.
  - For each x_i, get its value from Tape 2.
  - Scan Tape 1, marking the clauses that it satisfies.
  - This can be done in time $O(n \log n)$ for each variable.
  - So it can be done for all the variables in $O(n^2 \log n)$ time.
  - Finally, scan Tape 1 to see whether every clause is satisfied.

Example (SAT is in **NP**)

- Verify the solution.
  - For each x_i, get its value from Tape 2.
  - Scan Tape 1, marking the clauses that it satisfies.
  - This can be done in time $O(n \log n)$ for each variable.
  - So it can be done for all the variables in $O(n^2 \log n)$ time.
  - Finally, scan Tape 1 to see whether every clause is satisfied.
  - This can be done in $O(n \log n)$ time.

SAT is in **NP**

Example (SAT is in **NP**)

- Verify the solution.
  - For each x_i, get its value from Tape 2.
  - Scan Tape 1, marking the clauses that it satisfies.
  - This can be done in time $O(n \log n)$ for each variable.
  - So it can be done for all the variables in $O(n^2 \log n)$ time.
  - Finally, scan Tape 1 to see whether every clause is satisfied.
  - This can be done in $O(n \log n)$ time.
- Thus, the problem can be decided in $O(n^2 \log n) \subset O(n^3)$ time.

Outline

The Hamiltonian Path Problem

Definition (Hamiltonian Path)

Given a graph G, a Hamiltonian path is a path that passes through every vertex of G exactly once.

The Hamiltonian Path Problem

Definition (Hamiltonian Path)

Given a graph G, a Hamiltonian path is a path that passes through every vertex of G exactly once.

Definition (The Hamiltonian Path Problem)

Given a graph G, the Hamiltonian Path Problem (HAMPATH) asks whether there exists a Hamiltonian path in G.

The Hamiltonian Path Problem

Definition (Hamiltonian Path)

Given a graph G, a Hamiltonian path is a path that passes through every vertex of G exactly once.

Definition (The Hamiltonian Path Problem)

Given a graph G, the Hamiltonian Path Problem (HAMPATH) asks whether there exists a Hamiltonian path in G.

Theorem

$HAMPATH \in \mathbf{NP}$

Outline

The Clique Problem

Definition (Clique)

Given a graph G, a clique is a complete subgraph $G' \subseteq G$. That is, every two vertices in G' are adjacent.

The Clique Problem

Definition (Clique)

Given a graph G, a clique is a complete subgraph $G' \subseteq G$. That is, every two vertices in G' are adjacent.

Definition (The Clique Problem)

Given a graph G and an integer k, the Clique Problem (CLIQ) asks whether there exists a clique in G of size k.

The Clique Problem

Definition (Clique)

Given a graph G, a clique is a complete subgraph $G' \subseteq G$. That is, every two vertices in G' are adjacent.

Definition (The Clique Problem)

Given a graph G and an integer k, the Clique Problem (CLIQ) asks whether there exists a clique in G of size k.

Theorem

$CLIQ \in \mathbf{NP}$

Outline

The Vertex Cover Problem

Definition (Vertex Cover)

Given a graph G with vertices V, a vertex cover is a set of vertices $V' \subseteq V$ such that every edge in G is adjacent to some vertex in V'.

The Vertex Cover Problem

Definition (Vertex Cover)

Given a graph G with vertices V, a vertex cover is a set of vertices $V' \subseteq V$ such that every edge in G is adjacent to some vertex in V'.

Definition (The Vertex Cover Problem)

Given a graph G and an integer k, the Vertex Cover Problem (VC) asks whether there exists a vertex cover in G of size k.

The Vertex Cover Problem

Definition (Vertex Cover)

Given a graph G with vertices V, a vertex cover is a set of vertices $V' \subseteq V$ such that every edge in G is adjacent to some vertex in V'.

Definition (The Vertex Cover Problem)

Given a graph G and an integer k, the Vertex Cover Problem (VC) asks whether there exists a vertex cover in G of size k.

Theorem

$VC \in \mathbf{NP}$

Outline

Assignment

Homework

- Section 14.3 Exercises 2, 3.