CISC 4090
Theory of Computation

Complexity

Professor Daniel Leeds
dleeds@fordham.edu
JMH 332

Computability

Are we guaranteed to get an answer?

Complexity

How long do we have to wait for an answer? (Ch7)

How much resources do we need to find the answer? (Ch8)

Time complexity example: Deciding \(\{0^k1^k | k \geq 0\}\)

How much time does a TM take to decide \(A=\{0^k1^k | k \geq 0\}\)?

Computation steps:
• Scan left to right and confirm no 0’s after 1’s
• Repeat if both 0s and 1 left on tape
  • Scan across tape removing a single 0 and a single 1
• If neither 0’s or 1’s remain on tape, accept; otherwise, reject

Characterizing run-time

If TM M halts on all inputs, there exists f: N->N
f(n) = max # steps on any input of length n

• “M runs in time f(n)”
• “M is an f(n) time Turing machine”
Asymptotic analysis – “Big O” and “Small O”

Assess runtime as input grows large
• Only consider highest-order term
• Ignore constant co-efficients

Example: \( f(n) = 5n^4 + 3n^2 + 10n + 5 \)
\( f(n) = O(?) \)

Big-O Defined

\( f(n) = O(g(n)) \) if positive integers \( c \) and \( n_0 \) exist such that for every \( n \geq n_0 \)
• \( f(n) \leq c \cdot g(n) \)
• \( g(n) \) is “asymptotic upper bound” for \( f(n) \)

Big-O does not require the upper bound to be “tight”

Beyond polynomial

Exponential bounds, like \( O(2^n) \), much bigger
Logarithmic bounds, like \( O(\log n) \), much smaller

\( O(\log_2 n) = O(\log_{10} n) = O(\ln n) \) – only constant difference

Small-O defined

\( f(n) = o(g(n)) \) if for any real number \( c > 0 \), \( n_0 \) exists such that for every \( n \geq n_0 \)
• \( f(n) < c \cdot g(n) \)

In other words:
\( \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 \)
Time complexity example: Deciding \( \{0^k1^k | k \geq 0\} \)

How much time does a TM take to decide \( A = \{0^k1^k | k \geq 0\} \)?

Computation steps:
• Scan left to right and confirm no 0’s after 1’s
• Repeat if both 0s and 1 left on tape
  • Scan across tape removing a single 0 and a single 1
• If neither 0’s or 1’s remain on tape, accept; otherwise, reject

Time complexity example

Computation steps:
• Scan left to right and confirm no 0’s after 1’s \( O(n) \)
• Repeat if both 0s and 1 left on tape \( O(\frac{n}{2}) \)
  • Scan across tape removing a single 0 and a single 1 \( O(n) \)
• If neither 0’s or 1’s remain on tape, accept; otherwise, reject \( O(n) \)

Total complexity: \( O(n) + O(n/2)O(n) + O(n) = 2O(n) + O(n)O(n) \)
\( = 3O(n) + O(n^2) = O(n^2) \)

Time complexity class

Let \( t: \mathbb{N} \rightarrow \mathbb{R}^+ \) be a function

\( \text{TIME}(t(n)) \) is collection of all languages decidable by an \( O(t(n)) \) time Turing machine

\( \text{TIME}(t(n)) \) is a time complexity class

Tighter bound for \( \{0^k1^k\} \) on TM

Scan across tape, reject if 0 found right of 1
Repeat as long as some 0s and 1s remain
• Scan across tape and confirm total number of 0s and 1s is even
• Scan again, crossing off every other 0 starting with the first 0 and every other 1, starting with first 1
If no 0s and no 1s left, accept; else reject
Computing the tighter bound

Scan across tape, reject if 0 found right of 1
Repeat as long as some 0s and 1s remain
  • Scan across tape and confirm total number of 0s and 1s is even
  • Scan again, crossing off every other 0 starting with the first 0
  and every other 1, starting with first 1
If no 0s and no 1s left, accept; else reject

\[ O(n) + O(\log_2 n) (O(n)+O(n)) + O(n) = O(n) + O(\log_2 n) O(n) \]

\[ = O(n \log n) \]

Complexity on 2-tape Turing machine computed

Scan across tape and reject if 0 right of 1
Scan across 0s on tape 1, writing each onto tape 2
Scan across the 1s on tape 1, removing a 0 from tape 2 for each 1
If run out of 0s while still reading 1s, reject
If 0s left after 1s finished, reject
If no 0s left, accept

\[ 4 O(n) = O(n) \]

Relationship between Single and Multi-tape TM

Theorem: Let \( t(n) \) be function, where \( t(n) \geq n \). Then every \( t(n) \) time multitape TM has equivalent \( O(t^2(n)) \) time single-tape TM

Convert any multi-tape TM M to single-tape TM S

Across full runtime, M makes visits at most \( t(n) \) tape locations
In S simulation, include \( t(n) \) locations from each of the \( k \) tapes
For each S step, need to read/write each tape – \( k t(n) \) steps to traverse all tapes in one direction
S takes \( t(n) \) computation loops, one for each step of M

\[ O(t(n)) O(t(n)) = O(t^2(n)) \]

Relationship between DTM and NDTM

Let N be NDTM that is a decider. Run time of N is max number of steps that N uses on any branch of its computation on input of length \( n \)

Does not correspond to real-world computer
  • Except maybe a quantum computer!
NDTM -> DTM

If NDTM N decides language A in t(n) steps, can construct DTM D to decide A in $O(b^{t(n)})$ steps, where b is the maximum number of possible branches for a state-input pair.

D must simulate each branch of N, using breadth first search

At each step of computation, N chooses one branch down tree of possible branches

If b possible branches taken at each step, and there are t(n) steps, there are a total of $b^{t(n)}$ possible terminal points – exponential!!!

Polynomial time

Difference between polynomial times considered small compared to exponential time

Big picture:
• Exponential: brute force trying every solution to see what fits
• Polynomial: more efficient computation

“Reasonable” computational models are polynomial-time equivalent

Class P

P is the class of languages decidable in polynomial time on a deterministic single-tape Turing Machine

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

P is class of problems realistically solvable on a computer*
Example: Path problem

Is there a path from s to t in graph G?
- \( \text{PATH} = \{ <G,s,t> \mid G \text{ is directed graph with directed path from } s \text{ to } t \} \)

Brute force
- If G has m nodes, path cannot be more than m
- Upper bound on possible paths \( m^m \)
- Try each “path” one by one for legality and for linking s-to-t

Path – Breadth first search complexity
- Place mark on node s
- Repeat until no new nodes marked
  - Scan all edges; If edge \((a,b)\) found from marked node a to unmarked b, mark b
  - If t is marked, accept; Otherwise, reject

Measure complexity based on number of nodes m
- Place mark on node s \(- O(m)\) (find node s in list of m nodes)
- Repeat until no new nodes marked \(- O(m)\) (if only mark one new node per loop)
  - For each edge \((a,b)\) with a already marked, mark b also \(- O(m^2)\)
    - \(O(m^2)\) max total edges, for each edge, search for a to see if marked \(O(m)\), then mark b in list if needed \(O(m)\)
    - In total: \(O(m^2) \times (O(m) + O(m)) = O(m^3)\)
- If t marked, accept; else, reject \(- O(m)\) (find node t in list of m nodes)
- In total: \(O(m) + O(m) \times O(m^3) + O(m) = O(m^4) + 2O(m) = O(m^4)\) – POLYNOMIAL!
**Example: RELPRIME**

Two numbers are relatively prime of 1 is the largest number that evenly divides them both
- 10 and 21 are relatively prime
- 10 and 22 are not relatively prime

Solution: search all divisors from 2 until \( \text{min}(x,y)/2 \)
- \( \text{min}(x,y)/2 \) numbers tried, \( \text{min}(x,y)/2 \) steps
- size of input \( n = \text{length of binary encoding} = \log_2(\text{max}(x,y)) \)
- \( 2^n \) steps – exponential complexity!

**RELPRIME – faster solution**

“Euclidean algorithm”

\[ E = \text{On input <} x,y \text{>} \]
- Repeat until \( y=0 \)
  - Assign \( x \leftarrow x \mod y \)
  - Exchange \( x \) and \( y \)
- Output \( x \)

\[ R = \text{On input <} x,y \text{>} \]
- Run \( E \) on \(< x,y \>)
- If result is 1, accept; Otherwise, reject

**Simulating the Euclidean algorithm**

\[
\begin{align*}
x &= 10 \quad y &= 21 \\
x &= 10 \quad y &= 21 \quad \text{MOD} \\
x &= 21 \quad y &= 10 \quad \text{SWAP} \\
x &= 1 \quad y &= 10 \quad \text{MOD} \\
x &= 10 \quad y &= 1 \quad \text{SWAP} \\
x &= 0 \quad y &= 1 \quad \text{MOD} \\
x &= 1 \quad y &= 0 \quad \text{SWAP}
\end{align*}
\]

\( x=1 \) when \( y=0 \), so original numbers relatively prime

**Euclidean Algorithm – complexity**

\[
x = x \mod y \leftarrow \text{new x always less than y}
\]
- If old \( x \) is twice \( y \) or more, new \( x \) will be cut at least in half
- If old \( x \) between \( y \) and \( 2y \), new \( x \) will be cut at least in half
  * new \( x = x \cdot y \)

Number of loops: \( 2\log_2(\text{max}(x,y)) \)
Length of input (in binary): \( \log_2(x) + \log_2(y) = O(\log_2(\text{max}(x,y))) \)
Number of loops: \( O(n) \)
Hamiltonian Path Example

Hamiltonian path in directed graph $G$: directed path that goes through each node exactly once

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

Brute force
- List all possible paths and confirm if it’s a valid path
- If $m$ nodes, $m!$ paths – exponential

Polynomial Verifiability – e.g., HAMPATH

- If given an answer, can determine if it is correct in polytime
- Path at most $m$ long, graph has at most $m^2$ edges

Verifier definition

Verifier for language $A$ is algorithm $V$ where:
- $A = \{ w \mid V \text{ accepts } <w,c> \text{ for some string } c \}$
- Verifiers use extra information in the “certificate” $c$ to verify string $w$ is in $A$

$A$ is polytime verifiable if it has a polytime verifier
- Complexity measured in terms of $w$

NP definition

NP is class of languages that have polytime verifiers

$\text{NP} \triangleleft \text{Nondeterministic Polynomial time}$

For HAMPATH
- Write list of $m$ nodes, nondeterministically selected
- Check for repeats in list; if repeats, reject
- Check if $s=\text{node}_1$ and $t=\text{node}_m$; if either fails, reject
- For each $i$ between 1 and $m-1$, check $(\text{node}_i, \text{node}_{i+1})$ is in $G$. If not, reject; else except.

$\text{HAMPATH} c$: proposed $m$-1 step path along nodes in graph $G$. 
NTIME and Class NP

NTIME(t(n)) = \{L \mid L \text{ is language decided by } O(t(n)) \text{ time nondeterministic Turing machine}\}

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

NP Example: CLIQUE

A clique in an undirected graph is a subgraph where 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-clique}\}

• k is a parameter. Determining clique for certain fixed k is in P

Another NP Example: Subset-Sum

• SUBSET-SUM problem: given a list of numbers \(S=\{x_1, x_2, \ldots, x_n\}\), determine if a subset of numbers add to the target \(t\)

• SUBSET-SUM = \{\langle S, t \rangle \mid S=\{x_1, \ldots, x_n\} \text{ and some } \{y_1, \ldots, y_k\} \subseteq \{x_1, \ldots, x_n\} \text{ and } \sum y_i = t\}

SUBSET-SUM \in NP

Polytime verifiable!

Does P = NP?

In other words: Are there polynomial time solutions to all algorithms that are polytime verifiable?

Probably not, but it’s an open question!

If P=NP, lots of “hard” problems become doable – including cracking encrypted networks!
Class coNP

• A language is coNP if its complement is NP
• Unknown if all coNP languages are also NP

NP-Complete problems

The hardest problems in NP are NP complete
If polynomial time algorithm for these problems, \( P=NP \)

• If any NP problem requires more than polytime, then NP-complete problems also require more than polytime

Since no polytime solution has been found for an NP-complete problem, if we determine new problem is NP-complete, reasonable to give up search for general polytime solution to this problem

Satisfiability: An NP-complete problem

Consider Boolean operator AND, OR, NOT
Consider set of Boolean variables

Boolean formula is satisfiable if some assignment of T’s and F’s makes the total formula True (T)

• Ex: \( (x’ \lor y) \land (x \lor z’) \)
• Ex: \( (x’ \lor y) \land (x \lor y) \land (x’ \lor y’) \)