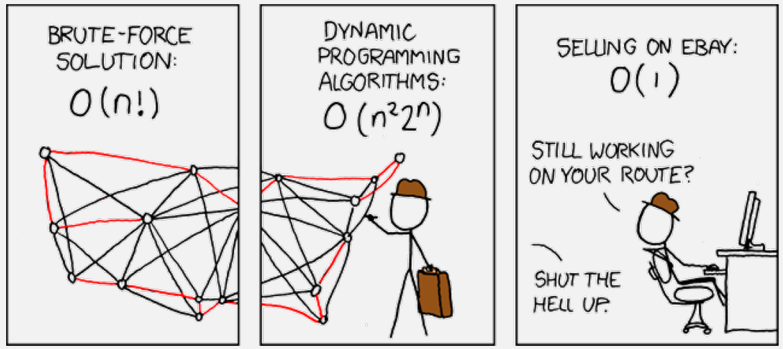
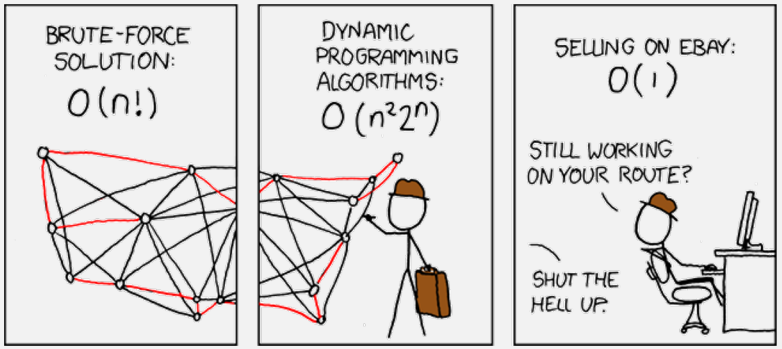


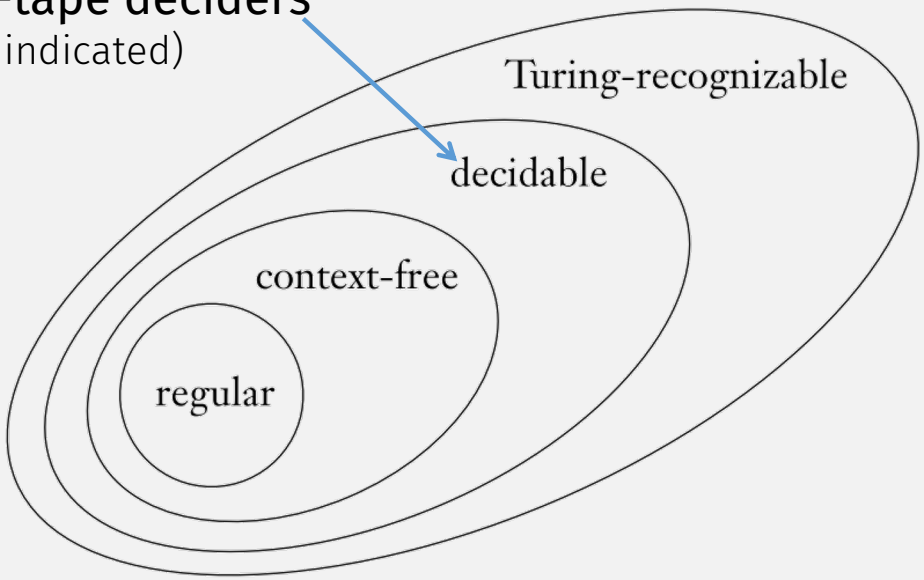
Polynomial Time (P)

Announcements



Where Are We Now?

We are back in here now:
deterministic, single-tape deciders
(unless otherwise indicated)



Definition: Big-O Notation

DEFINITION 7.2

Let f and g be functions $f, g: \mathcal{N} \longrightarrow \mathcal{R}^+$. Say that $f(n) = O(g(n))$ if positive integers c and n_0 exist such that for every integer $n \geq n_0$,

$$f(n) \leq c\, g(n).$$

When $f(n) = O(g(n))$, we say that $g(n)$ is an ***upper bound*** for $f(n)$, or more precisely, that $g(n)$ is an ***asymptotic upper bound*** for $f(n)$, to emphasize that we are suppressing constant factors.

- In English: Keep only highest order term, drop all coefficients
- Machine M_1 that decides $A = \{0^k 1^k \mid k \geq 0\}$
  - Is an $\mathbf{n^2 + 3n}$ time Turing machine
  - Is an $O(\mathbf{n^2})$ time Turing machine
  - Has asymptotic upper bound $O(\mathbf{n^2})$

Last Time: Time Complexity

DEFINITION 7.1

Let M be a deterministic Turing machine that halts on all inputs. The ***running time*** or ***time complexity*** of M is the function $f: \mathcal{N} \longrightarrow \mathcal{N}$, where $f(n)$ is the maximum number of steps that M uses on any input of length n. If $f(n)$ is the running time of M, we say that M runs in time $f(n)$ and that M is an $f(n)$ time Turing machine. Customarily we use n to represent the length of the input.

NOTE: exact units of n not specified, it's only *roughly* "length" of the input

But n can be #characters, #states, #nodes, etc, whatever is more convenient, so long as it's correlated with length of input

It doesn't matter because we only care about large n (so constant factors are ignored)

Last Time: Time Complexity *Classes*

DEFINITION 7.7

Let $t: \mathcal{N} \longrightarrow \mathcal{R}^+$ be a function. Define the ***time complexity class***, $\mathbf{TIME}(t(n))$, to be the collection of all languages that are decidable by an $O(t(n))$ time Turing machine.

Remember: **TMs** have a time complexity (ie, running time), **languages** are in a complexity class

The complexity class of a **language** is determined by the time complexity (ie, running time) of their deciding **TMs**

A Faster Machine? $A = \{0^k 1^k \mid k \geq 0\}$

M_2 = "On input string w:

1. Scan across the tape and *reject* if a 0 is found to the right of a 1.
2. Repeat as long as some 0s and some 1s remain on the tape:
3. Scan across the tape, checking whether the total number of 0s and 1s remaining is even or odd. If it is odd, *reject*.
4. Scan again across the tape, crossing off every other 0 starting with the first 0, and then crossing off every other 1 starting with the first 1.
5. If no 0s and no 1s remain on the tape, *accept*. Otherwise, *reject*."

M_1 = "On input string w:

1. Scan across the tape and *reject* if a 0 is found to the right of a 1.
2. Repeat if both 0s and 1s remain on the tape:
3. Scan across the tape, crossing off a single 0 and a single 1.
4. If 0s still remain after all the 1s have been crossed off, or if 1s still remain after all the 0s have been crossed off, *reject*. Otherwise, if neither 0s nor 1s remain on the tape, *accept*."

Number of steps (worst case), **n = length of input:**

A Faster Machine? $A = \{0^k 1^k \mid k \geq 0\}$

M_2 = "On input string w:

1. Scan across the tape and *reject* if a 0 is found to the right of a 1.
2. Repeat as long as some 0s and some 1s remain on the tape:
3. Scan across the tape, checking whether the total number of 0s and 1s remaining is even or odd. If it is odd, *reject*.
4. Scan again across the tape, crossing off every other 0 starting with the first 0, and then crossing off every other 1 starting with the first 1.
5. If no 0s and no 1s remain on the tape, *accept*. Otherwise, *reject*."

Number of steps (worst case), **n = length of input:**

➢ Line 1:

- **n** steps to scan + **n** steps to return to beginning = $O(\mathbf{n})$ steps

A Faster Machine? $A = \{0^k 1^k \mid k \geq 0\}$

M_2 = "On input string w:

1. Scan across the tape and *reject* if a 0 is found to the right of a 1.
2. Repeat as long as some 0s and some 1s remain on the tape:
3. Scan across the tape, checking whether the total number of 0s and 1s remaining is even or odd. If it is odd, *reject*.
4. Scan again across the tape, crossing off every other 0 starting with the first 0, and then crossing off every other 1 starting with the first 1.
5. If no 0s and no 1s remain on the tape, *accept*. Otherwise, *reject*."

Number of steps (worst case), **n = length of input:**

- Line 1:
  - **n** steps to scan + **n** steps to return to beginning = $O(\mathbf{n})$ steps

➢Lines 2, 3, 4 (loop):

- Steps for 1 iter: a scan takes $O(\mathbf{n})$ steps
- # iters: Each iter crosses off *half* the chars, so at most $O(\mathbf{\log n})$ scans
- Total: $O(\mathbf{n}) * O(\mathbf{\log n}) = O(\mathbf{n \log n})$ steps

A Faster Machine? $A = \{0^k 1^k \mid k \geq 0\}$

M_2 = "On input string w:

1. Scan across the tape and *reject* if a 0 is found to the right of a 1.
2. Repeat as long as some 0s and some 1s remain on the tape:
3. Scan across the tape, checking whether the total number of 0s and 1s remaining is even or odd. If it is odd, *reject*.
4. Scan again across the tape, crossing off every other 0 starting with the first 0, and then crossing off every other 1 starting with the first 1.
5. If no 0s and no 1s remain on the tape, *accept*. Otherwise, *reject*."

Number of steps (worst case), **n = length of input:**

- Line 1:
  - **n** steps to scan + **n** steps to return to beginning = $O(\mathbf{n})$ steps
- Lines 2, 3, 4 (loop):
  - Steps for 1 iter: a scan takes $O(\mathbf{n})$ steps
  - # iters: Each iter crosses off *half* the chars, so at most $O(\mathbf{\log n})$ scans
  - Total: $O(\mathbf{n}) * O(\mathbf{\log n}) = O(\mathbf{n \log n})$ steps

➢ Line 5:

- $O(\mathbf{n})$ steps to scan input one more time

A Faster Machine? $A = \{0^k 1^k \mid k \geq 0\}$

M_2 = "On input string w:

1. Scan across the tape and *reject* if a 0 is found to the right of a 1.
2. Repeat as long as some 0s and some 1s remain on the tape:
3. Scan across the tape, checking whether the total number of 0s and 1s remaining is even or odd. If it is odd, *reject*.
4. Scan again across the tape, crossing off every other 0 starting with the first 0, and then crossing off every other 1 starting with the first 1.
5. If no 0s and no 1s remain on the tape, *accept*. Otherwise, *reject*."

Number of steps (worst case), n = length of input:

- Line 1:
  - **n** steps to scan + **n** steps to return to beginning = $O(\mathbf{n})$ steps
- Lines 2, 3, 4 (loop):
  - Steps for 1 iter: a scan takes $O(\mathbf{n})$ steps
  - # iters: Each iter crosses off *half* the chars, so at most $O(\mathbf{\log n})$ scans
  - Total: $O(\mathbf{n}) * O(\mathbf{\log n}) = O(\mathbf{n \log n})$ steps
- Line 5:
  - $O(\mathbf{n})$ steps to scan input one more time
- Total: $O(\mathbf{n}) + O(\mathbf{n \log n}) + O(\mathbf{n}) = O(\mathbf{n \log n})$ steps

Interlude: Logarithms

- $2^x = y$
- $\log_2 y = x$
- $\log_2 n = O(\mathbf{\log n})$
  - "divide and conquer" algorithms = O(**log n**)
  - E.g., binary search
- In computer science, **base-2 is the only base!**

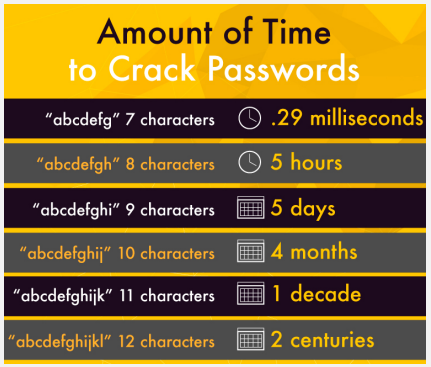
Terminology: Categories of Bounds

- Exponential time
  - $O(\mathbf{2^{n\hat{}c}})$, for $c > 0$ (always base 2)
- Polynomial time
  - $O(\mathbf{n^c})$, for $c > 0$
- Quadratic time (special case of polynomial time)
  - $O(\mathbf{n^2})$
- Linear time (special case of polynomial time)
  - $O(\mathbf{n})$
- Log time
  - $O(\mathbf{\log n})$

Today: Polynomial Time (**P**) Complexity Class

- Corresponds to **tractable** vs **intractable** problems; roughly:
  - Problems in **P** = "tractable"
  - Problems outside **P** = "intractable"

- Problems can be "decidable" in theory, but "intractable" in practice



- Intractable problems usually only have "brute force" solutions
  - "try all possible inputs"

Brute-force attack

From Wikipedia, the free encyclopedia

In cryptography, a **brute-force attack** consists of an attacker submitting many passwords or passphrases with the hope of eventually guessing a combination correctly. The attacker systematically checks all possible passwords and passphrases until the correct one is found. Alternatively, the attacker can attempt to guess the key which is typically created from the password using a key derivation function. This is known as an **exhaustive key search**.

Today: Polynomial Time, Formally

DEFINITION 7.12

P is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine. In other words,

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

Today: 2 Problems in **P**

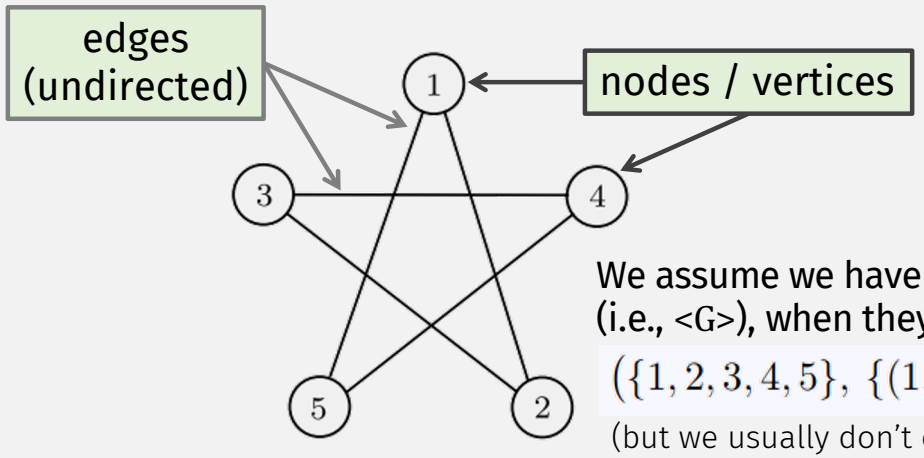
- A Graph Problem:

$$PATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph that has a directed path from } s \text{ to } t\}$$

- A CFL Problem:

Every context-free language is a member of P

Interlude: Graphs (see Chapter 0)



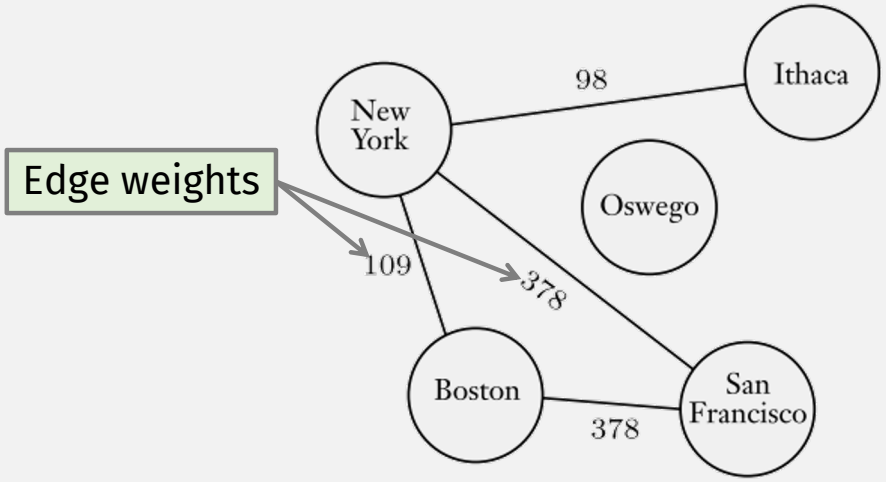
We assume we have *some* **string encoding of a graph** (i.e., <G>), when they are args to TMs, e.g.:

$$(\{1,2,3,4,5\},\ \{(1,2),\ (2,3),\ (3,4),\ (4,5),\ (5,1)\})$$

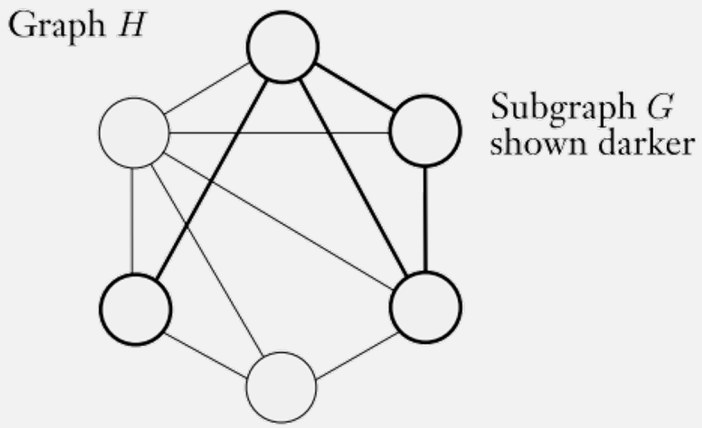
(but we usually don't care about the actual details)

- Edge defined by two nodes (order doesn't matter)
- Formally, a graph = a pair (V, E)
  - Where V = a set of nodes, E = a set of edges

Interlude: Weighted Graphs

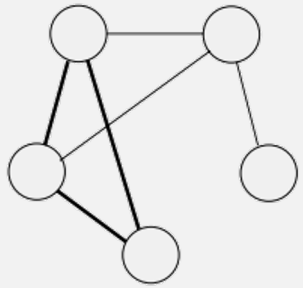


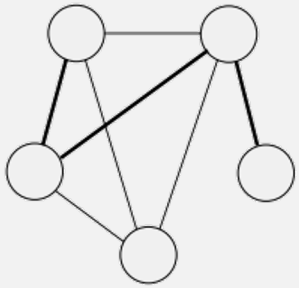
Interlude: Subgraphs

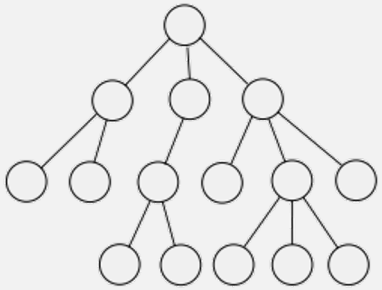


Interlude: Paths and other Graph Things

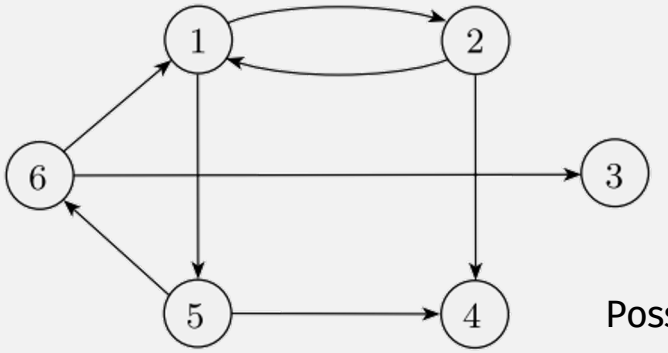
- Path
  - A sequence of nodes connected by edges
- Cycle
  - A path that starts/ends at the same node
- Connected graph
  - Every two nodes has a path
- Tree
  - A connected graph with no cycles







Interlude: Directed Graphs



Possible **string encoding** given to TMs:

$$(\{1,2,3,4,5,6\},\ \{(1,2),(1,5),(2,1),(2,4),(5,4),(5,6),(6,1),(6,3)\})$$

Each pair of nodes included twice

- Directed graph = (V, E)
  - V = set of nodes, E = set of edges
- An edge is a pair of nodes (u,v), **order now matters**
  - u = "from" node, v = "to" node
- "degree" of a node: number of edges connected to the node
  - Nodes in a directed graph have both indegree and outdegree

Interlude: Graph Encodings

$$(\{1,2,3,4,5\},\ \{(1,2),\ (2,3),\ (3,4),\ (4,5),\ (5,1)\})$$

- For graph algorithms, "length of input" n is usually # of vertices
  - (Not number of chars in the encoding)
- So given graph $G = (V, E)$, $n = |V|$
- Max edges?
  - $= O(|V|^2) = O(\mathbf{n}^2)$
- So if a set of graphs (call it lang L) is decided by a TM where
  - # steps of the TM = polynomial in the # of vertices
  - Then L is in **P**

Today: 2 Problems in **P**

- A Graph Problem:

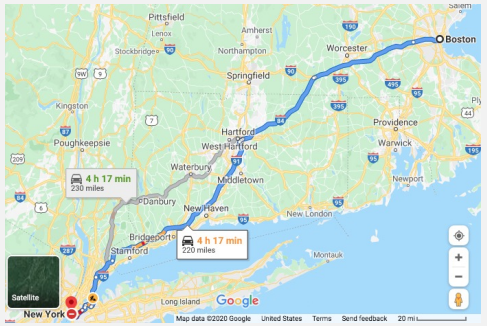
$$PATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph that has a directed path from } s \text{ to } t\}$$

- A CFL Problem:

Every context-free language is a member of P

A Graph Theorem: $PATH \in \mathrm{P}$

$$PATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph that has a directed path from } s \text{ to } t\}$$



- To prove that a language is in **P** ...
- ... we must construct a polynomial time algorithm deciding the lang
- A non-polynomial (i.e., exponential,"brute force") algorithm:
  - check all possible paths, and see if any connect s to t
  - If n = # vertices, then # paths $\approx n^n$

A Graph Theorem: $PATH \in \mathrm{P}$

$$PATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph that has a directed path from } s \text{ to } t\}$$

PROOF A polynomial time algorithm M for $PATH$ operates as follows.

M = "On input $\langle G, s, t\rangle$, where G is a directed graph with nodes s and t:

1. Place a mark on node s.
2. Repeat the following until no additional nodes are marked:
3. Scan all the edges of G. If an edge (a, b) is found going from a marked node a to an unmarked node b, mark node b.
4. If t is marked, *accept*. Otherwise, *reject*."

# of steps (worst case) **(*n* = # nodes):**

- Line 1: **1 step**

A Graph Theorem: $PATH \in \mathrm{P}$

$$PATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph that has a directed path from } s \text{ to } t\}$$

PROOF A polynomial time algorithm M for $PATH$ operates as follows.

M = "On input $\langle G, s, t\rangle$, where G is a directed graph with nodes s and t:

1. Place a mark on node s.
2. Repeat the following until no additional nodes are marked:
3. Scan all the edges of G. If an edge (a, b) is found going from a marked node a to an unmarked node b, mark node b.
4. If t is marked, *accept*. Otherwise, *reject*."

# of steps (worst case) **(*n* = # nodes):**

- Line 1: **1 step**
- Lines 2, 3 (loop):
  - Steps per loop: max # steps = max # edges = $O(n^2)$

A Graph Theorem: $PATH \in \mathrm{P}$

$$PATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph that has a directed path from } s \text{ to } t\}$$

PROOF A polynomial time algorithm M for $PATH$ operates as follows.

M = "On input $\langle G, s, t\rangle$, where G is a directed graph with nodes s and t:

1. Place a mark on node s.
2. Repeat the following until no additional nodes are marked:
3. Scan all the edges of G. If an edge (a, b) is found going from a marked node a to an unmarked node b, mark node b.
4. If t is marked, *accept*. Otherwise, *reject*."

# of steps (worst case) **(*n* = # nodes):**

- Line 1: **1 step**
- Lines 2, 3 (loop):
  - Steps per loop: max # steps = max # edges = $O(n^2)$
  - ➢ # loops: loop runs at most n times

A Graph Theorem: $PATH \in \mathrm{P}$

$$PATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph that has a directed path from } s \text{ to } t\}$$

PROOF A polynomial time algorithm M for $PATH$ operates as follows.

M = "On input $\langle G, s, t\rangle$, where G is a directed graph with nodes s and t:

1. Place a mark on node s.
2. Repeat the following until no additional nodes are marked:
3. Scan all the edges of G. If an edge (a, b) is found going from a marked node a to an unmarked node b, mark node b.
4. If t is marked, *accept*. Otherwise, *reject*."

# of steps (worst case) **(*n* = # nodes):**

- Line 1: **1 step**
- Lines 2, 3 (loop):
  - Steps per loop: max # steps = max # edges = $O(n^2)$
  - # loops: loop runs at most n times
  - ➢ Total: $O(n^3)$

A Graph Theorem: $PATH \in \mathrm{P}$

$PATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph that has a directed path from } s \text{ to } t\}$

PROOF A polynomial time algorithm M for $PATH$ operates as follows.

M = "On input $\langle G, s, t\rangle$, where G is a directed graph with nodes s and t:

1. Place a mark on node s.
2. Repeat the following until no additional nodes are marked:
3. Scan all the edges of G. If an edge (a, b) is found going from a marked node a to an unmarked node b, mark node b.
4. If t is marked, *accept*. Otherwise, *reject*."

# of steps (worst case) **(*n* = # nodes):**

- Line 1: **1 step**
- Lines 2, 3 (loop):
  - Steps per loop: max # steps = max # edges = $O(n^2)$
  - # loops: loop runs at most n times
  - Total: $O(n^3)$

➢ Line 4: **1 step**

A Graph Theorem: $PATH \in \mathrm{P}$

$$PATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph that has a directed path from } s \text{ to } t\}$$

PROOF A polynomial time algorithm M for $PATH$ operates as follows.

M = "On input $\langle G, s, t\rangle$, where G is a directed graph with nodes s and t:

1. Place a mark on node s.
2. Repeat the following until no additional nodes are marked:
3. Scan all the edges of G. If an edge (a, b) is found going from a marked node a to an unmarked node b, mark node b.
4. If t is marked, *accept*. Otherwise, *reject*."

# of steps (worst case) **(*n* = # nodes):**

- Line 1: **1 step**
- Lines 2, 3 (loop):
  - Steps per loop: max # steps = max # edges = $O(n^2)$
  - # loops: loop runs at most n times
  - Total: $O(n^3)$
- Line 4: **1 step**

➢ Total = 1 + 1 + $O(n^3)$ = $O(n^3)$

DEFINITION 7.12

P is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine. In other words,

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