جامعة
الملك سعود
King Saud University
1957

Theory of Computation
CSC 339 – Spring 2021

Chapter-7: part2
The Class P

King Saud University

Department of Computer Science

Dr. Azzam Alsudais

Introduction

- **Decidable problems can be classified in terms of their running time growth rate.**

Introduction

- **Decidable problems can be classified in terms of their running time growth rate.**
- **Two main classes**
  - **Polynomial time algorithms (P)**
  - **Nondeterministic polynomial time algorithms (NP)**
- **In this part, we focus on the class P.**

The Class P

- **What does it mean for a given algorithm to have polynomial running time?**

The Class P

- **What does it mean for a given algorithm to have polynomial running time?**

- **What does polynomial running time mean?**

The Class P

- **What does it mean for a given algorithm to have polynomial running time?**

- **What does polynomial running time mean?**

- **It is crucial to understand the difference between <u>polynomial</u> and (<u>exponential</u> and <u>factorial</u>).**

The Class P: Polynomial vs. Exponential & Factorial

Polynomial

$O(1)$, $O(\log n)$,
$O(n \log n)$, $O(n^2)$,
$O(n^3)$, …

Exponential & Factorial

$O(2^n)$, $O(3^n)$, …,
$O(m^n)$, $O(n!)$, $O(n^n)$

The Class P: Polynomial vs. Exponential & Factorial

Polynomial time algorithms are
fast enough for many purposes

Polynomial

$O(1)$, $O(\log n)$,
$O(n \log n)$, $O(n^2)$,
$O(n^3)$, …

Exponential & Factorial

$O(2^n)$, $O(3^n)$, …,
$O(m^n)$, $O(n!)$, $O(n^n)$

The Class P: Polynomial vs. Exponential & Factorial

Polynomial time algorithms are fast enough for many purposes

Exponential time algorithms are rarely useful!

Polynomial

$O(1)$, $O(\log n)$,
$O(n \log n)$, $O(n^2)$,
$O(n^3)$, …

Exponential & Factorial

$O(2^n)$, $O(3^n)$, …,
$O(m^n)$, $O(n!)$, $O(n^n)$

The Class P: Polynomial vs. Exponential & Factorial

Polynomial time algorithms are fast enough for many purposes

Exponential time algorithms are rarely useful!

Polyn

$O(1)$, O(

O(n log n

$O(n^3)$, …

Keep in mind that we're talking about decidable (solvable) problems!

ntial & Factorial

), $O(3^n)$, …,

), $O(n!)$, $O(n^n)$

The Class P: Polynomial vs. Exponential & Factorial

Theoretically, ***exponential*** algorithms decide some problems, but they are so slow that they cannot be used in practice.

Poly
fast

tial time algorithms
rarely useful!

Keep in mind that we're talking about decidable (solvable) problems!

Polyn

O(1), O(

O(n log n

$O(n^3)$, …

ntial & Factorial

), $O(3^n)$, …,

), $O(n!)$, $O(n^n)$

The Class P: Polynomial vs. Exponential & Factorial

Let n = 10

$O(1)$	= 1
$O(\log_2 n)$	= 3.3
$O(n)$	= 10
$O(n \log_2 n)$	= 33
$O(n^2)$	= 100
$O(n^3)$	= 1000

$O(2^n)$	= 1024
$O(3^n)$	= 59049
$O(n!)$	= 3628800
$O(n^n)$	= 10000000000

Polynomial

$O(1)$, $O(\log n)$,
$O(n \log n)$, $O(n^2)$,
$O(n^3)$, …

Exponential & Factorial

$O(2^n)$, $O(3^n)$, …,
$O(m^n)$, $O(n!)$, $O(n^n)$

The Class P: Polynomial vs. Exponential & Factorial

Let n = 20

$O(1)$	= 1
$O(\log_2 n)$	= 4.3
$O(n)$	= 20
$O(n \log_2 n)$	= 86
$O(n^2)$	= 400
$O(n^3)$	= 8000

$O(2^n)$	= 1048576
$O(3^n)$	= 3486784401
$O(n!)$	= 2432902008176640000
$O(n^n)$	= 104857600000000000000000000

Polynomial

$O(1)$, $O(\log n)$,
$O(n \log n)$, $O(n^2)$,
$O(n^3)$, …

Exponential & Factorial

$O(2^n)$, $O(3^n)$, …,
$O(m^n)$, $O(n!)$, $O(n^n)$

The Class P: Polynomial vs. Exponential & Factorial

Let n = 100

$O(1)$	$= 1$
$O(\log_2 n)$	$= 6.64$
$O(n)$	$= 100$
$O(n \log_2 n)$	$= 664$
$O(n^2)$	$= 10000$
$O(n^3)$	$= 1000000$

Polynomial

$O(1)$, $O(\log n)$,
$O(n \log n)$, $O(n^2)$,
$O(n^3)$, …

$O(2^n)$ $=$ 1267650600228229401496703205376

$O(3^n)$ $=$ 515377520732011331036461129765621272702107522001

$O(n!)$ $= 9.33e+157$

$O(n^n)$ $= 1e+200$

Exponential & Factorial

$O(2^n)$, $O(3^n)$, …,
$O(m^n)$, $O(n!)$, $O(n^n)$

The Class P: Polynomial vs. Exponential & Factorial

Let n = 100

$O(1)$ = 1
$O(\log_2 n)$ = 6.64
$O(n)$ = 100
$O(n \log_2 n)$ = 664
$O(n^2)$ = 10000
$O(n^3)$ = 1000000

$O(2^n)$ = 1267650600228229401496703205376

$O(3^n)$ = 515377520732011331036461129765621272702107522001

$O(n!)$ = 9.33e+157

$O(n^n)$ = 1e+200

Polynomial

$O(1)$, $O(\log n)$

$O(n \log n)$, $O(n$

$O(n^3)$, …

Exponential & Factorial

Number of atoms in the universe is ~10^{80}

The Class P

- **What does it mean for a given algorithm to have polynomial running time?**

- **What does polynomial running time mean?**

The Class P

- **What does it mean for a given algorithm to have polynomial running time?**

- **What does polynomial running time mean?**

- **It is crucial to understand the difference between <u>polynomial</u> and (<u>exponential</u> and <u>factorial</u>).**

The Class P

- **What does it mean for a given algorithm to have polynomial running time?**

- **What does polynomial running time mean?**

- **It is crucial to understand the difference between <u>polynomial</u> and (<u>exponential</u> and <u>factorial</u>).**

    - **“Exponential time algorithms typically arise when we solve problems by exhaustively searching a space of solutions (brute-force search).”**

The Class P

- **What does it mean for a given algorithm to have polynomial running time?**

- **What does polynomial running time mean?**

- **It is crucial to understand the difference between <u>polynomial</u> and (<u>exponential</u> and <u>factorial</u>).**

  - **“Exponential time algorithms typically arise when we solve problems by exhaustively searching a space of solutions (brute-force search).”**
  - **This means we try all possible combinations to reach a solution.**

The Class P

- **Can we avoid exponential time algorithms?**
  - **Yes**

The Class P

- **Can we avoid exponential time algorithms?**
  - **Yes, but not always…**

The Class P

- **Can we avoid exponential time algorithms?**
  - **Yes, but not always…**
  - **A deep understanding of the problem can help us design a polynomial time algorithm.**

The Class P

- **Can we avoid exponential time algorithms?**
  - **Yes, but not always…**
  - **A deep understanding of the problem can help us design a polynomial time algorithm.**

- **From now on, we will ignore polynomial differences in the running time of algorithms.**

The Class P

- **Can we avoid exponential time algorithms?**
  - **Yes, but not always…**
  - **A deep understanding of the problem can help us design a polynomial time algorithm.**

- **From now on, we will ignore polynomial differences in the running time of algorithms.**
  - **e.g., $O(n)$, $O(n^2)$, and $O(n^3)$ will be treated equally: they are all members of the Class P.**
  - **Just like we did when we ignored constants for *big-O*.**

The Class P

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,

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

The Class P

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,

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

P corresponds to the class of problems that are *realistically* solvable on a computer.

The Class P: Analyzing Algorithms for Polynomiality

- **To simplify the analysis of algorithms, we will describe them in a way that's irrelevant to the underlying computational model (FA, CFG, PDA, TM, etc).**

The Class P: Analyzing Algorithms for Polynomiality

- **To simplify the analysis of algorithms, we will describe them in a way that's irrelevant to the underlying computational model (FA, CFG, PDA, TM, etc).**

- **Describe algorithms as a sequence of ordered stages.**

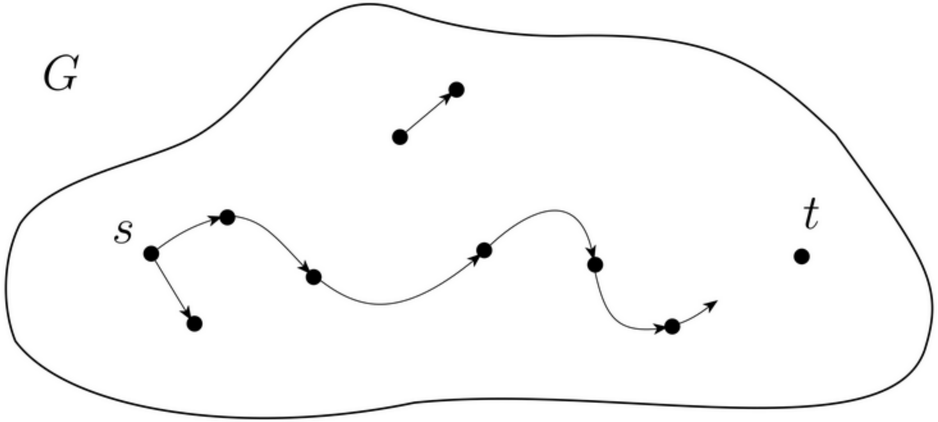
- **Doing so allows us to analyze these stages for polynomiality.**

The Class P: Example Problems (PATH)

- **In a directed graph G, is there a path from node s to node t?**

The Class P: Example Problems (PATH)

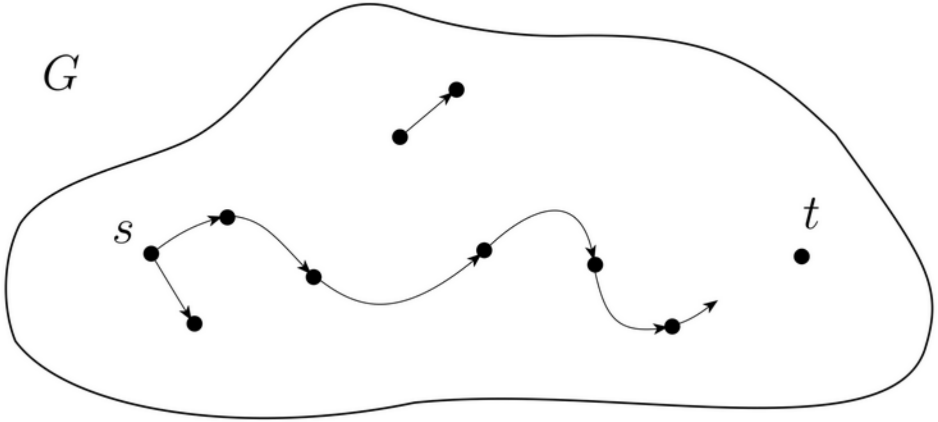
- **In a directed graph G, is there a path from node s to node t?**



The Class P: Example Problems (PATH)

- **In a directed graph** G**, is there a path from node** s **to node** t**?**

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



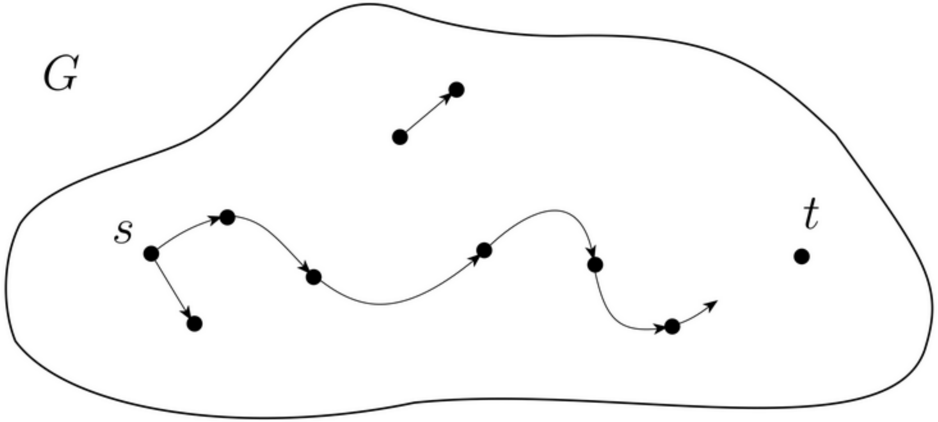
The Class P: Example Problems (PATH)

- **In a directed graph *G*, is there a path from node *s* to node *t*?**

PATH={⟨G, s, t⟩| G is a directed graph that has a directed path from s to t}.

Theorem 7.14

$PATH \in P$



The Class P: Example Problems (PATH)

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

- **How to solve this problem?**

The Class P: Example Problems (PATH)

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

- **How to solve this problem?**
- **Brute-force approach**
  - **Examine ALL potential paths in** G**, and determine whether there is a path from** s **to** t**.**

The Class P: Example Problems (PATH)

PATH={⟨*G*, *s*, *t*⟩| *G is a directed graph that has a directed path from s to t*}.

- **How to solve this problem?**
- **Brute-force approach**
  - **Examine ALL potential paths in** *G***, and determine whether there is a path from** *s* **to** *t***.**
  - **A potential path is a sequence of nodes in** *G* **having a length of at most** *m* **(***m* **is the number of nodes).**

The Class P: Example Problems (PATH)

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

- **How to solve this problem?**
- **Brute-force approach**
  - **Examine ALL potential paths in** *G***, and determine whether there is a path from** *s* **to** *t***.**
  - **A potential path is a sequence of nodes in** *G* **having a length of at most** *m* **(***m* **is the number of nodes).**
  - **How many potential paths are there in** *G***?**

The Class P: Example Problems (PATH)

PATH={⟨*G*, *s*, *t*⟩| *G is a directed graph that has a directed path from s to t*}.

- **How to solve this problem?**
- **Brute-force approach**
  - **Examine ALL potential paths in** G**, and determine whether there is a path from** s **to** t**.**
  - **A potential path is a sequence of nodes in** G **having a length of at most** m **(**m **is the number of nodes).**
  - **How many potential paths are there in** G**?**

m^m**, where** m **is the number of nodes.**

The Class P: Example Problems (PATH)

PATH={⟨G, s, t⟩| G is a directed graph that has a directed path from s to t}.

- **How to solve this problem?**
- **Brute-force approach**
  - **Examine ALL p ne whether there is a path from**
  - **A potential pa ving a length of at most *m* (*m* is**
  - **How many potential paths are there in *G*?**

m^m, **where *m* is the number of nodes.**

*Can we design a better algorithm with a **polynomial** running time?*

The Class P: Example Problems (PATH)

PATH={⟨*G*, *s*, *t*⟩| *G is a directed graph that has a directed path from s to t*}.

- **How to solve this problem?**
- **Brute-force approach**
  - **Examine ALL** … **ne whether there is a path from** …
  - **A potential pa** … **ving a length of at most** m **(**m **i** …
  - **How many potential paths are there in** G**?**

m^m, **where** m **is the number of nodes.**

Can we design a better algorithm with a ***polynomial*** *running time?*

YES

The Class P: Example Problems (PATH)

Algorithm M for PATH

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*."

The Class P: Example Problems (PATH)

Algorithm M for PATH

M = "On input ⟨G, s, t⟩, where G is a directed graph with nodes s and t:

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

Executed only once

The Class P: Example Problems (PATH)

Algorithm M for PATH

Runs at most *m* times

M = “On input ⟨G, s, ph 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*.”

The Class P: Example Problems (RELPRIME)

- **Two numbers are relatively prime if 1 is the largest integer that evenly divides them both.**
- **e.g., 21 and 10 are relatively prime**

The Class P: Example Problems (RELPRIME)

- **Two numbers are relatively prime if 1 is the largest integer that evenly divides them both.**

- **e.g., 21 and 10 are relatively prime**

$RELPRIME = \{\langle x, y\rangle \mid x \text{ and } y \text{ are relatively prime}\}.$

The Class P: Example Problems (RELPRIME)

- **Two numbers are relatively prime if 1 is the largest integer that evenly divides them both.**
- **e.g., 21 and 10 are relatively prime**

Theorem 7.15

$RELPRIME \in P$

$$RELPRIME = \{\langle x, y\rangle \mid x \text{ and } y \text{ are relatively prime}\}.$$

The Class P: Example Problems (RELPRIME)

- **Two numbers are relatively prime if 1 is the largest integer that evenly divides them both.**

Theorem 7.15

$RELPRIME \in P$

- **e.g., 21 and 10 are relatively prime**

$RELPRIME = \{\langle x, y \rangle \mid x \text{ and } y \text{ are relatively prime}\}.$

- **A brute-force approach would be to examine all possible divisors of both numbers, and accept if none are greater than 1.**

The Class P: Example Problems (RELPRIME)

- **Two numbers are relatively prime if 1 is the largest integer that evenly divides them both.**

- **e.g., 21 and 10 are relatively prime**

Theorem 7.15

$$RELPRIME \in P$$

$$RELPRIME = \{\langle x, y\rangle \mid x \text{ and } y \text{ are relatively prime}\}.$$

- **A brute-force approach would be to examine all possible divisors of both numbers, and accept if none are greater than 1.**

- **Clearly, this approach goes through an exponential number of potential divisors, and has an exponential running time.**

The Class P: Example Problems (RELPRIME)

- **Two numbers are relatively prime if 1 is the largest integer that evenly divides them both.**

- **e.g., 21 and 10 are relatively prime**

Theorem 7.15

$RELPRIME \in P$

$RELPRIME = \{\langle x, y \rangle \mid x \text{ and } y \text{ are relatively prime}\}.$

- **A brute-force approach would be to examine all possible divisors of both numbers, and accept if none are greater than 1.**

- **Clearly, this approach goes through an exponential number of potential divisors, and has an exponential running time.**

- **Use the Euclidean algorithm to solve this in polynomial time.**

The Class P: Example Problems (RELPRIME)

- Two numbers are relatively prime if 1 is the largest integer that evenly divides them both.

Theorem 7.15

$RELPRIME \in P$

- e.g., 21 and 10 are relatively prime

$RELPRIME = \{\langle x, y\rangle \mid x \text{ and } y \text{ are relatively prime}\}.$

- A brute-force approach would be to examine all possible divisors of both numbers, a … than 1.

See page 289 for algorithm

- Clearly, this appro … tial number of potential divisors, and has an exponential running time.

- Use the Euclidean algorithm to solve this in polynomial time.

The Class P: Example Problems (Context-free Langs)

- **Since context-free languages are solvable, can we decide CFLs in polynomial time?**

The Class P: Example Problems (Context-free Langs)

- **Since context-free languages are solvable, can we decide CFLs in polynomial time?**
  - Yes!

The Class P: Example Problems (Context-free Langs)

- **Since context-free languages are solvable, can we decide CFLs in polynomial time?**
  - Yes!

Theorem 7.16

Every context-free language $\in P$

The Class P: Example Problems (Context-free Langs)

- **Since context-free languages are solvable, can we decide CFLs in polynomial time?**
  - Yes!

Theorem 7.16

Every context-free language $\in P$

- **Any derivation of a string *w* has *2n-1* steps (*n* is the length of *w*).**

The Class P: Example Problems (Context-free Langs)

- **Since context-free languages are solvable, can we decide CFLs in polynomial time?**
  - Yes!

Theorem 7.16

Every context-free language $\in P$

- **Any derivation of a string *w* has *2n-1* steps (*n* is the length of *w*).**
- **One idea:**
  - **A decider can try all possible derivations of length *k* (*2n-1*)**

The Class P: Example Problems (Context-free Langs)

- **Since context-free languages are solvable, can we decide CFLs in polynomial time?**

Theorem 7.16

Every context-free language $\in P$

  - Yes!

- **Any derivation of a string** *w* **has** *2n-1* **steps (***n* **is the length of** *w***).**
- **One idea:**
  - **A decider can try all possible derivations of length** *k* **(***2n-1***)**
  - **If any of these is a derivation of** *w***, the decider** *accepts***. Otherwise, it** *rejects***.**

The Class P: Example Problems (Context-free Langs)

- **Since context-free languages are solvable, can we decide CFLs in polynomial time?**

Theorem 7.16

Every context-free language $\in P$

  - Yes!

- **Any derivation of a string *w* has *2n-1* steps (*n* is the length of *w*).**
- **One idea:**
  - **A decider can try all possible derivations of length *k* (*2n-1*)**
  - **If any of these is a derivation of *w*, the decider *accepts*. Otherwise, it *rejects*.**
  - **But, the number of derivations with *k* steps may be <u>exponential</u> in *k*.**

The Class P: Example Problems (Context-free Langs)

- **To solve this in polynomial time, we leverage <u>dynamic programming</u>.**

Theorem 7.16

Every context-free language $\in P$

The Class P: Example Problems (Context-free Langs)

- **To solve this in polynomial time, we leverage <u>dynamic programming</u>.**

Theorem 7.16

Every context-free language $\in P$

- **<u>Dynamic programming</u> involves accumulating information about smaller problems to solve larger problems.**

The Class P: Example Problems (Context-free Langs)

- **To solve this in polynomial time, we leverage dynamic programming.**

Theorem 7.16

Every context-free language $\in P$

- **Dynamic programming involves accumulating information about smaller problems to solve larger problems.**

- **We achieve this by recording the solution to any smaller subproblem so that we solve it only once.**

The Class P: Example Problems (Context-free Langs)

- **To solve this in polynomial time, we leverage <u>dynamic programming</u>.**

> **Theorem 7.16**
>
> *Every context-free language* $\in P$

- **<u>Dynamic programming</u> involves accumulating information about smaller problems to solve larger problems.**

- **We achieve this by recording the solution to any smaller subproblem so that we solve it only once.**

- **To solve *CFLs*, the subproblems are determining whether each variable of a *CFG* generates a substring of *w*.**

The Class P: Example Problems (Context-free Langs)

- **To solve this in polynomial time, we leverage <u>dynamic programming</u>.**

Theorem 7.16

Every context-free language $\in P$

- **<u>Dynamic programming</u> involves accumulating information about smaller problems to solve larger problems.**

- **We achieve this b … y smaller subproblem so tha**

See page 291 for algorithm

- **To solve *CFLs*, the subproblems are determining whether each variable of a *CFG* generates a substring of *w*.**