# Basic Information

- Instructor: Hu Fu
- Office: ICICS-X539. Email: hufu@cs.ubc.ca
- Course website: http://fuhuthu.com/CPSC420S2019/index.html
  - Contains links to Piazza and Gradescope.
- Office Hours: Wednesday 2-3pm (starting Jan 9)
- Teaching Assistants:
  - Jack Spalding-Jamieson s4x0b@ugrad.cs.ubc.ca
  - Da Wei (David) Zheng zhengdw@cs.ubc.ca
  - Yihan Zhou yihan95@cs.ubc.ca

# What's covered

- We will cover topics selected from the second half of the textbook **Algorithm Design** by Kleinberg and Tardos.
- The textbook is required. It is the same one as required by CPSC 320.

# What's covered

- We will cover topics selected from the second half of the textbook **Algorithm Design** by Kleinberg and Tardos.
- The textbook is required. It is the same one as required by CPSC 320.
- Network flows and NP-completeness will be covered in details, followed by topics selected from approximation algorithms and randomized algorithms. Additional topics may be supplemented.

# Prerequisites

- Prerequisite: CPSC 320. You should be proficient with asymptotic running time analysis (e.g. big $O(\cdot)$ notations), basic data structures (e.g. linked lists, trees) and basic graph algorithms (e.g. DFS, BFS, minimum spanning trees).

# Prerequisites

- Prerequisite: CPSC 320. You should be proficient with asymptotic running time analysis (e.g. big $O(\cdot)$ notations), basic data structures (e.g. linked lists, trees) and basic graph algorithms (e.g. DFS, BFS, minimum spanning trees).
- You should be very comfortable with the design methods covered in 320 (e.g. greedy, dynamic programming).

# Prerequisites

- Prerequisite: CPSC 320. You should be proficient with asymptotic running time analysis (e.g. big $O(\cdot)$ notations), basic data structures (e.g. linked lists, trees) and basic graph algorithms (e.g. DFS, BFS, minimum spanning trees).
- You should be very comfortable with the design methods covered in 320 (e.g. greedy, dynamic programming).
- Familiarity with linear algebra and probability theory will be helpful. I will try my best to provide the basics.

# Prerequisites

- Prerequisite: CPSC 320. You should be proficient with asymptotic running time analysis (e.g. big $O(\cdot)$ notations), basic data structures (e.g. linked lists, trees) and basic graph algorithms (e.g. DFS, BFS, minimum spanning trees).
- You should be very comfortable with the design methods covered in 320 (e.g. greedy, dynamic programming).
- Familiarity with linear algebra and probability theory will be helpful. I will try my best to provide the basics.
- In terms of what you knew and what you didn't, feedback is always welcome, and helpful.

# Course work

- Problem sets (30%) + Midterm (30%) + Final (40%)
- Midterm date: Feb 13, Wednesday, in class.

# Course work

- Problem sets (30%) + Midterm (30%) + Final (40%)
- Midterm date: Feb 13, Wednesday, in class.
- Both exams are open book. The textbook and any written notes are allowed. No other printed materials are allowed.

# Course work

- Problem sets (30%) + Midterm (30%) + Final (40%)
- Midterm date: Feb 13, Wednesday, in class.
- Both exams are open book. The textbook and any written notes are allowed. No other printed materials are allowed.
- There is a problem set roughly every two weeks.

# Course work

- Problem sets (30%) + Midterm (30%) + Final (40%)
- Midterm date: Feb 13, Wednesday, in class.
- Both exams are open book. The textbook and any written notes are allowed. No other printed materials are allowed.
- There is a problem set roughly every two weeks.
- Expect curving to happen. Most likely we will not use the original scores as your grade. (More on this later.)

# Homework Policies

- You may form groups of up to three people for each assignment, although a group of two is encouraged. Each group needs to turn in only one solution. Each member of the group should completely understand the solution turned in.

# Homework Policies

- You may form groups of up to three people for each assignment, although a group of two is encouraged. Each group needs to turn in only one solution. Each member of the group should completely understand the solution turned in.
- Typesetting solutions using LaTex is encouraged. If, by the end of the semester, the majority of your assignments are typeset with LaTeX, your lowest assignment mark will be dropped from the calculation of your course grade. (A LaTeX template is provided.)

# Homework Policies

- You may form groups of up to three people for each assignment, although a group of two is encouraged. Each group needs to turn in only one solution. Each member of the group should completely understand the solution turned in.
- Typesetting solutions using LaTex is encouraged. If, by the end of the semester, the majority of your assignments are typeset with LaTeX, your lowest assignment mark will be dropped from the calculation of your course grade. (A LaTeX template is provided.)
- We use Gradescope for assignments.

# Homework Policies

- You may form groups of up to three people for each assignment, although a group of two is encouraged. Each group needs to turn in only one solution. Each member of the group should completely understand the solution turned in.
- Typesetting solutions using LaTex is encouraged. If, by the end of the semester, the majority of your assignments are typeset with LaTeX, your lowest assignment mark will be dropped from the calculation of your course grade. (A LaTeX template is provided.)
- We use Gradescope for assignments.
- Some problems in the assignments are more challenging than the others. You are encouraged to discuss problems among yourselves, on Piazza, and/or come to office hours. You should start thinking about the problems early and not wait till the last day or two. Allow yourself time to think and to seek help.

# Homework Policies

- You may form groups of up to three people for each assignment, although a group of two is encouraged. Each group needs to turn in only one solution. Each member of the group should completely understand the solution turned in.
- Typesetting solutions using LaTex is encouraged. If, by the end of the semester, the majority of your assignments are typeset with LaTeX, your lowest assignment mark will be dropped from the calculation of your course grade. (A LaTeX template is provided.)
- We use Gradescope for assignments.
- Some problems in the assignments are more challenging than the others. You are encouraged to discuss problems among yourselves, on Piazza, and/or come to office hours. You should start thinking about the problems early and not wait till the last day or two. Allow yourself time to think and to seek help.
- If you work with someone outside your group or use some outside source, you must acknowledge them in your write-up.

# This course is "proof-based"

- For assignments and exams, unless stated otherwise, for all questions that ask to design an algorithm, you need to provide justification for, (that is, to prove) the correctness of your algorithm.
- When the question asks for a certain running time (e.g. polynomial time, or $O(n^2)$), you should analyze the running time of your algorithm.

# This course is "proof-based"

- For assignments and exams, unless stated otherwise, for all questions that ask to design an algorithm, you need to provide justification for, (that is, to prove) the correctness of your algorithm.
- When the question asks for a certain running time (e.g. polynomial time, or $O(n^2)$), you should analyze the running time of your algorithm.
- Both the lectures and the assignments are more focused on proofs. Things will be more abstract and mathematical. (More on this later.)

# Other Differences from 320

- You are almost never asked to write pseudo-code. (It suffices to say, e.g., perform BFS.) You may resort to pseudo-code when you find it the most clear way to express your idea. Very often though there is a clearer way to say it in English.

# Other Differences from 320

- You are almost never asked to write pseudo-code. (It suffices to say, e.g., perform BFS.) You may resort to pseudo-code when you find it the most clear way to express your idea. Very often though there is a clearer way to say it in English.
- Think of it as "talking to humans" rather than "talking to compilers". Train yourself in clear writing.

# Other Differences from 320

- You are almost never asked to write pseudo-code. (It suffices to say, e.g., perform BFS.) You may resort to pseudo-code when you find it the most clear way to express your idea. Very often though there is a clearer way to say it in English.
- Think of it as "talking to humans" rather than "talking to compilers". Train yourself in clear writing.
- Proof skeleton is much less often given than in 320. You are usually required to come up with a proof from scratch.

# Other Differences from 320

- You are almost never asked to write pseudo-code. (It suffices to say, e.g., perform BFS.) You may resort to pseudo-code when you find it the most clear way to express your idea. Very often though there is a clearer way to say it in English.
- Think of it as "talking to humans" rather than "talking to compilers". Train yourself in clear writing.
- Proof skeleton is much less often given than in 320. You are usually required to come up with a proof from scratch.
- The course will take a slower pace than the last time I taught it.

# A little more philosophy

- A navigation analogy for mathematical maturity.

# A little more philosophy

- A navigation analogy for mathematical maturity.
- Compared with previous courses, the lectures may be more about big pictures and how one approaches problems at a high level, than about low-level details.
- Understanding things is more helpful than rote memorization.

# A little more philosophy

- A navigation analogy for mathematical maturity.
- Compared with previous courses, the lectures may be more about big pictures and how one approaches problems at a high level, than about low-level details.
- Understanding things is more helpful than rote memorization.
- On the other hand, it is crucial to grasp firmly basic definitions and to able to state results we prove precisely.

# How to write a proof

- A proof proves the validity of a statement by showing it as a logical consequence of other, more elementary, valid statements.

# How to write a proof

- A proof proves the validity of a statement by showing it as a logical consequence of other, more elementary, valid statements.
- More elementary statements may be from definitions, or may be proved previously.

# How to write a proof

- A proof proves the validity of a statement by showing it as a logical consequence of other, more elementary, valid statements.
- More elementary statements may be from definitions, or may be proved previously.
- Stating intermediate steps may help organize your thoughts and make the presentation clean.

# How to write a proof

- A proof proves the validity of a statement by showing it as a logical consequence of other, more elementary, valid statements.
- More elementary statements may be from definitions, or may be proved previously.
- Stating intermediate steps may help organize your thoughts and make the presentation clean.
- Setting up convenient notations may help with the writing trememndously.

# How to write a proof

- A proof proves the validity of a statement by showing it as a logical consequence of other, more elementary, valid statements.
- More elementary statements may be from definitions, or may be proved previously.
- Stating intermediate steps may help organize your thoughts and make the presentation clean.
- Setting up convenient notations may help with the writing trememndously.
- The logical steps could be some proof technique, e.g. induction.

# Example: Finding shortest paths in a graph

- Input: a directed graph $G = (V, E)$, with nonnegative cost $\ell_e \geq 0$ for each edge $e \in E$. A node $s \in V$.
- Output: for each node $t \in V$, the minimum cost of a path from $s$ to $t$.

# Example: Finding shortest paths in a graph

- Input: a directed graph $G = (V, E)$, with nonnegative cost $\ell_e \geq 0$ for each edge $e \in E$. A node $s \in V$.
- Output: for each node $t \in V$, the minimum cost of a path from $s$ to $t$.
- What if we remove the constraints $\ell_e \geq 0$?

# Example: Finding shortest paths in a graph

- Input: a directed graph $G = (V, E)$, with nonnegative cost $\ell_e \geq 0$ for each edge $e \in E$. A node $s \in V$.
- Output: for each node $t \in V$, the minimum cost of a path from $s$ to $t$.
- What if we remove the constraints $\ell_e \geq 0$?
- Input: a directed graph $G = (V, E)$, with cost $\ell_e \geq 0$ for each edge $e \in E$, and no negative cycle. A node $s \in V$.
- Output: for each node $t \in V$, the minimum cost of a path from $s$ to $t$.