Administrative Issues

EECS2011 N & Z:
Fundamentals of Data Structures
Winter 2022

CHEN-WEI WANG

Instructor

- How may you call me?

  "Jackie" (most preferred),

  "Professor Jackie", "Professor", "Professor Wang", "Sir", "Hey", "Hi", "Hello"
- When you need ***advice*** on the course, speak to me!
- There will be a bonus opportunity for you to fill out an informal, anonymous ***midterm course survey*** during the reading week.
- Throughout the semester, feel free to suggest ways for helping your learning.

If You Are Not Enrolled Yet

- Send me an email ASAP requesting access to the course eClass site, with your *name*, *student number*, *Passport York ID*.
- Still keep up with the study materials.
- Still complete assignments and tests (***no extension***).

Writing E-Mails to Your Instructor

- Think of me as your ***colleague*** who is happy to help you learn.
  - ***formality*** is unnecessary
  - ***courtesy*** is expected
- This sounds ***very rude*** (and may be delayed, if not ignored):

```
On the link you sent us for our mark
my mark for lab0 did not appear on it
and i submitted lab0 during my lab session
```

- This sounds ***much nicer***:

```
Hello Jackie, the link you sent didn’t work.
I did submit my lab0. Could you please look into this?
Thanks! Jim
```

Course Information

- A single eClass site:
  - *LE/EECS2011 N&Z - Fundamentals of Data Structures (Winter 2021-2022)*
    - Announcements common for both Sections N & Z
    - Assignments [instructions only]
    - Programming Tests [instructions & submissions]
    - Written Tests [instructions & submissions]
    - Exam
- Check your emails regularly!

Required Study Materials

- Study materials (lecture recordings, iPad notes, slides, example codes) will be posted on my website:

  `https://www.eecs.yorku.ca/~jackie/teaching/lectures/index.html#EECS2011_W22`

- The *course syllabus* is posted in the above site.

Course Syllabus

Let’s go over the ***course syllabus***.

Need Accommodation?

- Please contact me via email as soon as possible, so we can make proper arrangements for you.
- We will work out a way for you to gain the most out of this course!

Becoming a Software Engineer

- One useful mindset is to treat this course as a training course for ***programming interviews***.
- How a real ***software developer*** works:
  - Programming ***problems*** are explained via the expected methods' ***API*** (input and output types) and some ***use cases***, without visualization!
  - A set of ***tests*** must be *re-run automatically* upon changes.
- Thinking *abstractly* without seeing changes on a physical device is an important skill to acquire before graduating.

  e.g., Watch *interviews at Google*: Given problems described in English, solve it on a whiteboard.
- Take advantage of the ***Q&A sessions***: I will bring ***problems***.

Study Tips

- Plan steady, gradual study of:
  - Lecture videos [≈ 3 hours]
  - Optional Q&A sessions [≈ 1.5 hours – 3 hours]
- *Ask questions!*
- Take (even incomplete) notes, which will help when re-iterating lectures.

General Tips about Studying in a University

- To do well, ***inspiration*** is more important than ***perspiration***.
- Hard work does not necessarily guarantee success, but no success is possible without ***hard work***

  ⇒

  - Don’t be too satisfied just by the fact that you work hard.
  - Make sure you work hard both on ***mastering “ground stuffs”*** and, more importantly, on ***staying on top of what’s being taught***.
  - Go ***beyond*** lectures (e.g., CodingBat, LeetCode).
  - Be *curious* about why things work the way they do.
  - Always *reflect* yourself on *how things are connected* .

What is this course about?

- ***Data Structure*** [WHAT]

  Systematic way of organizing and accessing data
  e.g., arrays, linked-lists, stacks, queues, maps, trees, graphs, *etc.*

- ***Algorithm*** [HOW]

  Step-by-step procedure, using the appropriate data structure(s), for solving a computational problem
  e.g., inserting, deleting, sorting, searching

- ***Analysis*** [HOW GOOD?]

  Determining, mathematically, the **correctness** and **efficiency** of algorithms

Example (1): A Searching Problem

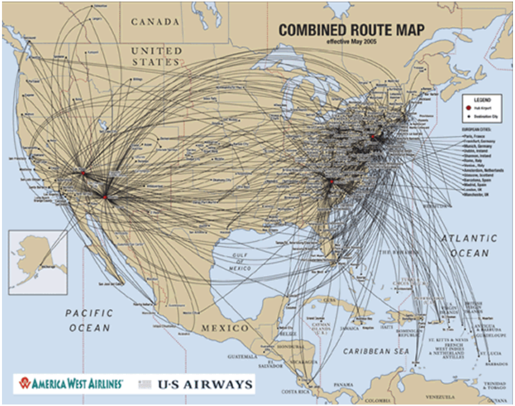
Problem: How would you save the records of a megacity with ***10 million residents***? Given a particular resident's social insurance number (ID), how ***fast*** can you locate his/her record?

```
ResidentRecord find(int sin) {
  for(int i = 0; i < database.length; i ++) {
    if(database[i].sin == sin) {
      return database[i];
    }
  }
}
```

- How many times will you have to run the loop?
  Best case? [1]
  Worst case? [10 million]
- You will learn about the appropriate data structure and algorithm to solve this problem (i.e., ***searching***), in the ***worst case***, within **24 iterations** of the loop!

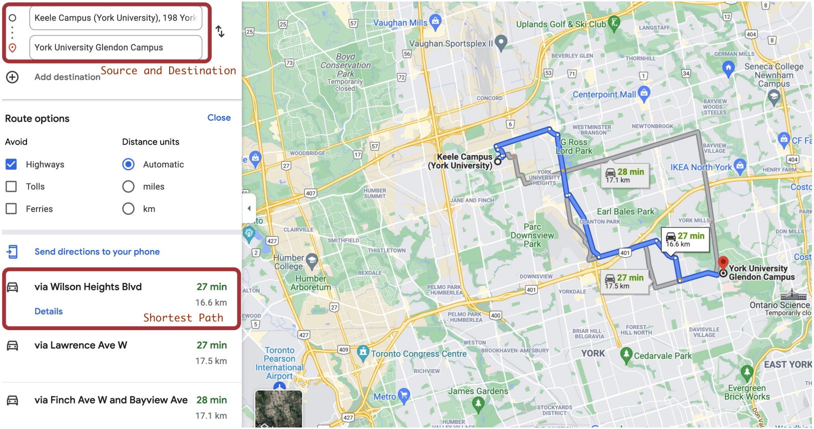
Example (2a): Flight Routing

Problem: Given the point-to-point connections of several airline companies, how do you plan an ***itinerary*** of flying from one city (origin) to another (destination)?



Example (2b): Car Routing

Problem: Plan a driving route which takes the ***minimum*** amount of time to arrive.



○ Keele Campus (York University), 198 York

York University Glendon Campus

⊕ Add destination

Source and Destination

Route options — Close

Avoid	Distance units
☑ Highways	● Automatic
☐ Tolls	○ miles
☐ Ferries	○ km

Send directions to your phone

via Wilson Heights Blvd — 27 min — 16.6 km

Details

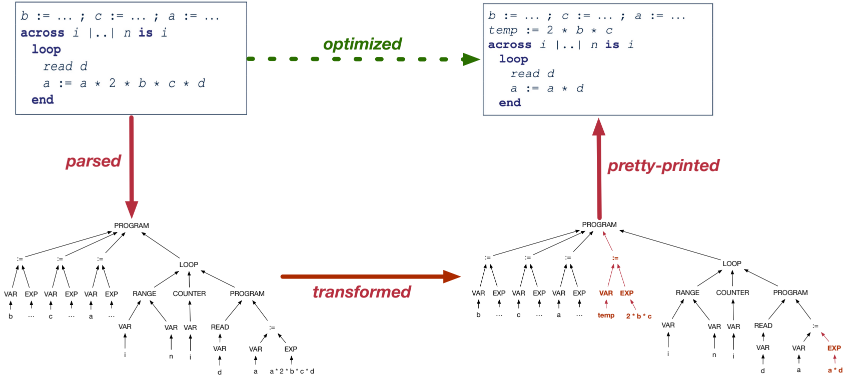
Shortest Path

via Lawrence Ave W — 27 min — 17.5 km

via Finch Ave W and Bayview Ave — 28 min — 17.1 km

Example (3a): Program Optimization

Problem: Given a user-written program, ***optimize*** it for best runtime performance.

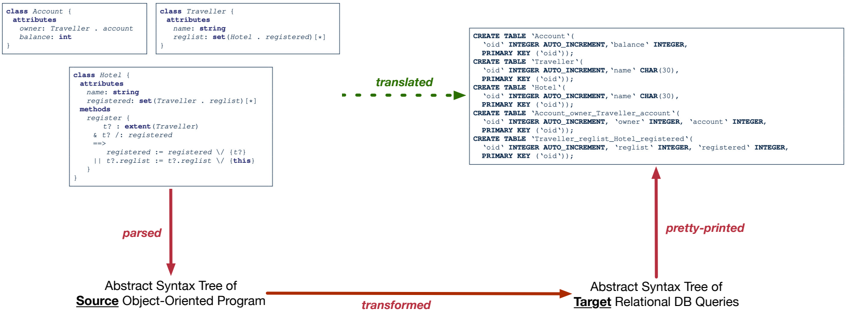


```
b := ... ; c := ... ; a := ...
across i |..| n is i
  loop
    read d
    a := a * 2 * b * c * d
  end
```

```
b := ... ; c := ... ; a := ...
temp := 2 * b * c
across i |..| n is i
  loop
    read d
    a := a * d
  end
```

Example (3b): Program Translation

Problem: Given a user-written object-oriented program, ***translate*** it into SQL tables/queries for persistent storage in a relational database.



```
class Account {
  attributes
    owner: Traveller . account
    balance: int
}
```

```
class Traveller {
  attributes
    name: string
    reglist: set(Hotel . registered)[*]
}
```

```
class Hotel {
  attributes
    name: string
    registered: set(Traveller . reglist)[*]
  methods
    register {
        t? : extent(Traveller)
      & t? /: registered
      ==>
         registered := registered \/ {t?}
      || t?.reglist := t?.reglist \/ {this}
    }
}
```

```
CREATE TABLE `Account`(
  `oid` INTEGER AUTO_INCREMENT,`balance` INTEGER,
  PRIMARY KEY (`oid`));
CREATE TABLE `Traveller`(
  `oid` INTEGER AUTO_INCREMENT,`name` CHAR(30),
  PRIMARY KEY (`oid`));
CREATE TABLE `Hotel`(
  `oid` INTEGER AUTO_INCREMENT,`name` CHAR(30),
  PRIMARY KEY (`oid`));
CREATE TABLE `Account_owner_Traveller_account`(
  `oid` INTEGER AUTO_INCREMENT, `owner` INTEGER, `account` INTEGER,
  PRIMARY KEY (`oid`));
CREATE TABLE `Traveller_reglist_Hotel_registered`(
  `oid` INTEGER AUTO_INCREMENT, `reglist` INTEGER, `registered` INTEGER,
  PRIMARY KEY (`oid`));
```

translated

parsed

pretty-printed

Abstract Syntax Tree of
Source Object-Oriented Program

transformed

Abstract Syntax Tree of
Target Relational DB Queries

Index (1)

Index (2)