DataStructures-Winter-2016 Documentation

Release 0.1

Brian McMahan

Mar 12, 2017

Getting Started

CHAPTER 1

Course Description

Computer Science is about computing data and solving problems. In the "Introduction to Python" course, students learned how the syntax of Python worked. In this course, we cover the basic and fundamental tools used to solve computational problems. Students learn how to turn that syntax into a finely honed tool.

The course will start with looking at the computational concerns Computer Scientists have about algorithms: space and time complexity. We will use this as the basis for discussing recursion, linked lists, stacks, queues, trees, search strategies, and sorting techniques. The course will conclude with a practical tour of Python's data structures, grounding them in the theoretical underpinnings covered in the first 8 weeks.

CHAPTER 2

How to Browse This Document

This document is intended to be a companion to the Data Structures course taught at Heroes Academy. For more information about HEROES Academy, please visit it here.

Below and to the left you will find the sections of this document. Each week there will be exercises to complete at home, as well as supplementary materials for further understanding and learning. Data Strutures is a well-studied topic that consists of abstract data structures and their various implementations. We will cover the broad range of abstract data structures without delving too deeply into any specific implementation.

Course Information

What is HEROES Academy?

HEROES Academy is an intellectually stimulating environment where students' personal growth is maximized by accelerated learning and critical thinking. Our students enjoy the opportunity to study advanced topics in classrooms that move at an accelerated pace.

When does this course meet?

Data Structures will meet from 11:30 to 1:30 pm on the following Sundays: January 10, 17, 24, 31; February 7, 21, 28; March 6, 13, 20.

How do I register for this course?

This course has already begun, but new courses are started at regular intervals! The list of courses are listed on the HEROES website. If you have any questions about the process, you can check out the HEROES Frequently Asked Questions.

What are the expectations of this course?

I expect that...

1. You will ask questions when you do not get something.
2. You will keep up with the work.
3. You will fail fast:
   - Failing is good
   - We learn when we fail
   - We only find bugs when code fails; we rarely hunt for bugs when code is working
4. You will not copy and paste code from the internet
   - You are only cheating yourself.
   - It won't bother me if you do it, but you will not learn the material.
5. You will try the homework at least once and email me with solutions or questions by Wednesday

How do I contact you?

You can contact me through the following email: bmcmahan@njgifted.org

Installing Python

Python Distribution

There are several ways to get Python. You should have a flash drive with an installer. In case you don't, I recommend the Anaconda distribution. It has a bunch of things packaged with it above and beyond Python that make it useful.

An Editor

There are many good editors and IDEs (Integrated Development Environments). I personally prefer editors, which are more minimalistic. However, for beginners, I would highly recommend PyCharm. If you download PyCharm, make sure you download the Community Edition.

Other than PyCharm, Sublime Text is very good and what I use. Then, Github has their own editor that is very comparable to Sublime. It is called Atom.

General Resources

Awesome stuff

1. Project Euler

Online Books

- Problem Solving with Algorithms and Data Structures
- How to think like a Computer Scientist
- How to think like a Computer Scientist: Interactive Edition
- A collection of links to Python guides

Debugging Help

- 16 common Python runtime errors for Beginners

Interactive Coding Websites

I really enjoy websites that let you code and compete. My favorites are:

- Hackerrank
- Codewars
- CodinGame

Online Code Environments

There are several places to run code online. I really like Trinkets and will use them a lot in the curriculum.

I also like C9 as a more advanced environment.

[Week 1] Initialization

Summary

The general idea of data structures are to represent information in ways that make certain operations easier or faster. We briefly talked about why this matters: the complexity of algorithms. Complexity is a very important concept to computer scientists and should always be thought about while programming.

The important terms to remember are listed below. Part of your homework is to write down what these mean in your own words.

In addition to introducing these topics, we also looked at hackerrank.com. This is a website that provides a variety of programming problems. You should complete at least 2 before next week. For the 2 that you complete, you should think about why complexity is important to those problems.

Terms

1. Algorithm
2. Data Structure
3. Search
4. Sort

5. Complexity

Homework

1. Define the terms above in your own words
2. Complete 2 programming problems on hackerrank.com, taking note of why complexity is important

Bonus: Get at least rank 5000 in the Python section.

Lecture Slides

[Week 2] Complexity

Summary

Today, we are going through the ideas underlying complexity. We will look at some python code and how to time it to test for speed.

For fun, we will also be looking at this in the context of finding prime numbers!

Exercises

1. How to time things
2. Prime Numbers
3. Bonus: Python structure basics

Lecture Slides

[Week 3] Recursion

Today we cover recursion!

Exercises

1. Recursion exercises

Bonus

Solve some of the problems at Project Euler.

Extra Resources

You will have the best luck if you run the following trinket full screen.

Lecture Slides

[Week 4] Linked Lists

Today we are covering linked lists! We will look at what they are, how they function conceptually, and how a simple one can be implemented.

Exercises

We will go over things on the slide and in the exercises together.

1. Test-Driven Code, Introductions, and Refreshers
2. Linked Lists, Part 1
3. Linked Lists, Part 2

Take home exercise to come either before, during, or after class.

Linked Lists

1. Think CS chapter
2. Harvard CS50

Recursion

1. Think CS chapter
2. MIT Intro to CS
3. Harvard CS50

Lecture Slides

[Week 5] More Linked Lists!

Today we are practicing linked lists! There are no slides (see yesterday if you want to see the material again)

- Start here for the Week 5 Linked List Tutorial!

[Week 6]: Stacks and Queues

Summary

Today, we are going through the slides first.

Then, when we are done with the lecture, there is a series of practice examples to go through:

- Week 6 Stack and Queue examples!

Lecture Slides

[Week 7] Binary Trees

Summary

Today we covered the basics of binary trees.

Important terms

- **Graph: a data structure made out of nodes and edges**
  - you can think of the edges like roads
  - the edges could have a direct (like one way streets)
- **Node: In a graph, this is like a variable, it represents a piece of information**
  - you can also think of it as cities
- **Edge: In a graph, edges connect two nodes**
  - you can also think of edges as roads
- **Tree: a data structure with the following properties:**
  - there are no cycles (you can't follow edges in a cirlce)
  - there is a root node (the top most part of the tree)
  - edges are called branches
  - each node can have children
  - the node with children is the children's parent
- **Root node**: the topmost part of the tree
- **Binary Trees**: Each node can have at most 2 children
- **Leaves**: A leaf is the nodes in a tree with no children
- **Binary Search Trees**: each parent is larger than its left child and smaller than its right

Extra Resources

Online Books

1. Think like a Computer Scientist

Videos

1. Harvard CS50

Lecture Slides

[Week 8] Problem Solving and Searching

Summary

Today we talked about how trees can be used to solve problems! As you click through the tutorials below, you will see how you can think about each node in a tree as being a setting of variables and each child is 1-step beyond the state.

For the take home work, you should work on the project work below. You don't have to have a working system by next week, but you should have a `class` which can represent the problem (in the same way we represented Missionaries and Cannibals in the tutorial). It should also have the set of possible moves, and a function which can generate the next legal states.

Also, don't forget, that there is a Binary Search Tree exercise on repl.it. It's ok if you don't get it done, but please look at it and ask me questions if there's anything you don't know.

Tutorial Pages

- Part 1: Introduction
- Part 2: State
- Part 3: Transitions
- Part 4: Searching

Project Work

During the next week, you should be thinking about what kinds of problems you could solve with these techniques. The properties the problem should have will be:

1. Can be represented by a couple of variables
2. Involves a series of steps to get to a solution

Here are some options to get you thinking about it:

1. **Towers of Hanoi**
    - There are pegs and a certain number of disks
    - The goal is to have the tree be the sequences of moves that you can do

    to solve the game
2. **Tic-Tac-Toe**
    - The blank game board has 9 open spaces
    - Each move is a selection of a square and a mark
    - The top root has 9 children, the second layer each has 8, and so on
3. **Nim or Chomp**
    - These are games where you have a set amount of things—in Nim, it's stones or sticks,

    and in Chomp it is pieces of candy. + You and another person are playing the game + Each person takes turns taking 1, 2, or 3 things + Whoever is forced to take the last thing loses + So, the problem solving programming for this problem would be to find the series of selections you can use to win!

You can also look through these links and see if there is something you'd like to try:

1. River Crossing Puzzles
2. Mathematical Games

[Week 9]: Projects

Summary

We will start class by reviewing what we've covered these last 9 weeks. Then, you will work on your projects! If you didn't get a working prototype over the week, the goal is to get one by the end of the class time. See below for the project prototype requirements.

Project Prototype

The project is to implement a search for a problem of your choosing. Last week we did Missionaries and Cannibals in class and I provided links to some other options. These included tic-tac-toe, nim, and some others.

The important parts of the search are:

1. **The search state representation**
    - this should be some set of variables which represent the state of the game
    - You should define both your start and your goal states.
    - **In the missionaries and cannibals, it was a tuple with three things:**
        (a) "how many missionaries on the right side"
        (b) "how many cannibals on the right side"
        (c) "how many boats on the right side"
    - The start state was `(3, 3, 1)`
    - The goal state was `(0,0,0)`
    - In general, representing with tuples is a good easy first solution.
2. **The set of possible actions**
    - Given the way you represented the state, what are all the ways the state can change?
    - This is usually determined by the rules of the game
    - For example, in Missionaries and Cannibals, the rules were the you can only

    have two people in a boat at one time. - So, we could do the following actions in one boat trip:

    (a) 2 Missionaries, 0 Cannibals
    (b) 1 Missiionary, 0 Cannibals
    (c) 1 Missionary, 1 Cannibal
    (d) 0 Missionaries, 1 Cannibal
    (e) 0 Missionaries, 2 Cannibals
    - We don't mention the boat because it just changes sides to whichever side it wasn't on
    - **To represent this as changes to our state, we can say each of these three moves is a tuple:**

(a) `(2,0)`

(b) `(1,0)`

(c) `(1,1)`

(d) `(0,1)`

(e) `(0,2)`

3. **A rule for applying the actions to the state that results in a new state**

    - Given that you have your state and set of actions, there should be a rule

    that you can turn into a function that determines the new state. - For example, in missionaries and cannibals:

    - If the boat is on the left side, then the action is to bring the specified people to the right
    - If the boat is on the right side, then the action is to bring the specified people to the left
    - So if the state is `(3,3,1)`, the boat is on the right, and any actions will bring people to the left
    - To bring people to the left, we subtract our move numbers
    - So, the move `(1,1)`, which is 1 missionary and 1 cannibal, will result in the state `(2,2,0)`
    - If the boat were on the left side, we **add** our move numbers to the state because we are bringing people to the right

    and the state represents the number of people on the right

4. **A test for illegal actions**

    - When are actions illegal?
    - You might have to apply the actions to your state a couple of time to notice the possible illegal states
    - It usually comes from possibilities that are outside of realistic situations
    - **For example, in Missionaries and Cannibals:**
        - if we were in the state `(2,2,0)`, we still have the full set of actions available
        - So we could pick the action `(2,0)` which means "2 missionaries use the boat"
        - Since the boat is on the left side, we would add the number to our state
        - This results in state `(4,2,1)`.
        - This state is clearly impossible!
    - Given this example, you should just have a rule that tells you when things are impossible

5. **A test for losing states**

    - when is the state a losing state?
    - **In Missionaries and Cannibals, the state is a losing state when:**
        - there are more cannibals than missionaries on either side.

6. **A test for the winning state**

    - when is the state a winning state?
    - In missionaries and cannibals, the winning state is `(0,0,0)`

So, to sum that up, you need:

1. State Representation
2. Action Representation
3. Rule for applying actions to states
4. Test for illegal actions
5. Test for losing states
6. Test for winning states

Given these things, the search is fairly simple. Below is the example code from Missionaries and Cannibals. When you implement your state code, this search code should also work for you. It is the following steps:

1. Create the initial root state
2. Create the python data structures that are useful (`to_search, seen-states, solutions`)
3. Loop until the `to_search` stack/queue is empty
4. Get the next state
5. Check to see if it's a solution
6. If it's not a solution, look at the states that can follow it
7. **Add in any states we haven't seen yet**
    - an example of a state that could repeat is just bringing 1 person back

   and forth forever.

```
### this is the stack/queue that we used before
from collections import deque

### create the root state
root = MCState.root()


### we use the stack/queue for keeping track of where to search next
to_search = deque()

### use a set to keep track fo where we've been
seen_states = set()

### use a list to keep track of the solutions that have been seen
solutions = list()

### start the search with the root
to_search.append(root)


### while the stack/queue still has items
while len(to_search) > 0:

    ### get the next item
    current_state = to_search.pop()

    ### Test for Winning State
    if current_state.is_solution():
        ## this is a successful state!
        ## the state vars should be (0,0,0)

```

```
        ## Save it into our solutions list
        solutions.append(current_state)

        ## we don't really want to go through the rest of this loop
        ## continue will skip the rest of the loop and start at the top again
        continue

    ## look at the current state's children
    ## this uses the rule for actions and moves to create next states
    ## it is also removing all illegal states
    next_states = current_state.get_next_states()

    ## next_states is a list, so iterate through it
    for possible_next_state in next_states:

        ## to see if we've been here before, we look at the state variables
        possible_state_vars = possible_next_state.state_vars

        ## we use the set and the "not in" boolean comparison
        if possible_state_vars not in seen_states:

            # the state variables haven't been seen yet
            # so we add the state itself into the searching stack/queue

            #### IMPORTANT
            ## which side we append on changes how the search works
            ## why is this?
            to_search.append(possible_next_state)

            # now that we have "seen" the state, we add the state vars to the set.
            # this means next time when we do the "not in", that will return False
            # because it IS in
            seen_states.add(possible_state_vars)

## finally, we reach this line when the stack/queue is empty (len(to_searching==))
print("Found {} solutions".format(len(solutions)))
```

Examples of Missionaries and Cannibals in iPython Notebooks

Presentation Template

You will give a presentation to your parents when we meet next week. You will have time at the beginning of class to finish things up, but your presentation is due to me that Friday.

Here is the presentation template:

CHAPTER 3

Indices and tables

- genindex
- modindex
- search