04-17-2023

CS051A

INTRO TO COMPUTER SCIENCE WITH TOPICS IN AI

Midterm II Study Guide

Alexandra Papoutsaki

she/her/hers

Lectures

Zilong Ye

he/him/his

Labs

How to study

- You can bring in two pages of notes (either two pieces of paper, single-side or one piece, double-sided).
- Go over the slides/notes slowly and deliberately.
- Practice writing code on paper. Once you are done, consider that as your submission and transfer your code to PyCharm. Is it syntactically correct? Test it with various inputs. Does it do what you thought it would?
- Do the practice problems/exam WITHOUT looking at the solutions.
- Open all the provided python files; look at the docstrings of the functions and make yourself implement them before you compare your response with the provided code.
- Review the assignments and feedback.
- Ask class staff questions.

Up to midterm I

- Review basic Python syntax
  - Different types: int, float, str, bool, list, tuple
    - Lists are mutable! Be careful with aliasing problems
    - Know how to slice and index a list and how to make a deep copy
  - Functions, parameters, and arguments
    - Local vs global variables
  - For and while loops
    - for i in range(num), vs for-each loop, for (index, value) in enumerate(sequence)
  - If/elif/else statements
  - Turtle module
  - Open files to read them line by line, trim line, split into words
  - input function to read input from user

Lecture 10

- How to create a dictionary, add a key-value pair, update an existing key value pair, access a value given a key.
- Dictionary methods:
  - `pop`, `clear`, `keys`, `values`, `items`
- How to iterate through dictionaries (both simple for loop and (key, value) pair for all items)

Lecture 11

- Look at recursive functions we saw together and practice writing recursive functions using our usual recipe:
  - 1. Define what the function the name and parameters of the function
  - 2. Define the recursive case
    - Pretend you had a working version of your function, but it only works on smaller versions of your current problem.
    - The recursive problem should be getting "smaller", by some definition of smaller.
      - E.g., for smaller numbers (like in factorial), lists that are smaller/shorter, strings that are shorter
  - 3. Define the base case ‣ What is the smallest (or simplest) problem? This is often the base case
  - 4. Put it all together

Lecture 11

- Look at recursive functions we saw together and practice writing recursive functions using our usual recipe:
  - 1. Define what the function the name and parameters of the function
  - 2. Define the recursive case
    - Pretend you had a working version of your function, but it only works on smaller versions of your current problem.
    - The recursive problem should be getting "smaller", by some definition of smaller.
      - E.g., for smaller numbers (like in factorial), lists that are smaller/shorter, strings that are shorter
  - 3. Define the base case ‣ What is the smallest (or simplest) problem? This is often the base case
  - 4. Put it all together

Lecture 12

- The idea that a single neuron/perceptron has connecting inputs, $x_1, x_2, \ldots$ each contributing $x_i * w_i$
- The sum $\sum_i x_i * w_i$ will be compared against a threshold function.
  - If greater than or equal to threshold, the output is 1.
  - If less than than threshold, the output is 0.
- Go over practice example in slide 33-38.

MIDTERM II STUDY GUIDE

Lecture 13

- Perceptron learning algorithm
  - Assume we want to train a neuron with n inputs, $x_1, x_2, \ldots, x_n$.
  - Start with adding an extra input neuron x_{n+1}whose input will always be 1.
  - Set threshold at 0.
  - Repeat until you get all examples right:
    - For each "training" example:
      - Calculate current prediction on example
      - If wrong:
      - Update weights and threshold towards getting this example correct.
        - $wi = wi + \Delta wi$
        - $\Delta wi = \lambda * (actual - predicted) * xi$, where λ is the learning rate.
- See practice example in slides 27–50.
- The perceptron learning algorithm will work if the training data is linearly separable (you can separate your data with a straight line). Counter-example: XOR

Lecture 14

- Given a training dataset with two labels (e.g., positive and negative reviews), know how to apply Naïve Bayes to classify a new data point (here, a review).
  - Start by calculating the conditional probabilities for each feature (here, a word) given a label.
  - Calculate the probability of the new data point given a label and pick the highest as the most likely label.
  - See example from slide 74 to 91.

Lecture 15

- An object is a software bundle of state (data/variables) and behavior (methods)
- A class is a blueprint for what data and methods objects should have
- You need to create an object, that is an instance of a class, using a constructor (through the magic method `__init__`)
  - `object_name = ClassName(potential_parameters)`
- `__str__` returns (doesn't print!) the string representation of the state of an object.
- `self` is a reference to the current object.
- We also use `self.variable` and `method()` to access an object's instance variables and methods *within* the class. `object.variable` and `object.method()` *outside* the class.
- `self` should be the first parameter in all methods

MIDTERM II STUDY GUIDE

Lecture 16

- Optional parameters and how to use them.
- Look at the code and be familiar with how queues (FIFO) and stacks (LIFO) work and their methods
- Identity vs equality
  - `id` function and `is` keyword vs `__eq__` method and == operator.
- Look at the fruit.py as an example of how to practice writing classes and creating and using objects.

Lecture 17

- add the start state to to_visit
- Repeat
  - take a state off the to_visit list
  - if it's the goal state
  - we're done!
  - if it's not the goal state
  - Add all of the next possible states to the to_visit list
- Depth first search (DFS): to_visit is a stack
- Breadth first search (BFS): to_visit is a queue
- Know how to apply DFS and BFS on a graph

Lecture 18

- Look into implementation of BFS
- Look into implementation of DFS
  - Iterative variant using a stack
  - Recursive variant returning A solution
  - Recursive variant returning ALL solutions
- Syntax for creating, accessing, and manipulating matrices (see files in last slide).
  - Be careful with aliasing issues
  - Practice with visualization tool https://pythontutor.com/visualize.html#mode=edit

Lecture 19–20

- Trick to avoid repeats in DFS
- DFS and BFS are uninformed search algorithms
- We can use heuristics to bias our search towards the solution.
- Best first search is an example of informed search algorithms where the to_visit list is sorted based on some evaluation function and the most desirable state (rather than the first- or last-added like in BFS and DFS, respectively) is picked.
    - Best first search for sudoku example demonstrate their difference.
- Coming up with good heuristics is hard and can be computationally expensive.