Creator: Keith Nicholas Pallo

MSAI Data Structures Project Overview

This document provides a high-level overview of all the assignments that I completed for my data structures project which I utilized as a supplement for my Masters of Science in Artificial Intelligence application. These assignments were modeled after the Fall 2017 EECS 214: Data Structures course homework assignments at Northwestern University, which I was able to find online. All relevant links are included at the end of this document for reference.

Before moving on to the description of each assignment and problem, I wanted to specifically acknowledge that code for all of these data structures is readily available online without much searching. So, what I have done throughout my code for these assignments in particular is to comment aggressively, in a fashion far more than typical or even necessary. My intent in doing so was to show understanding, and to authenticate my work.

In both the Python and C++ subdirectories of the current directory, one can run the main function to see specific outputs from test cases that I have created based on the homework descriptions. I recommend starting there as it presents a logical path to check associate header / source / module files.

Homework 1: Basic Structures (C++ Implementation)

Implement a data structure called an account that has the following properties:

- ID Number, Owner Name, and Balance
  Include the following functions:
    - Credit: Increases the account balance
    - Transfer: Moves money from one account to another

Note: Since vector / dynamic array utility has been shown throughout my MOOC code, vector components of HW 1 and 2 have been omitted from this project.

Homework 2: Dictionaries (C++ Implementation)

Implement a dictionary that holds accounts (created in HW1) using the following data structures shown below. Implement a lookup / search function for each to determine if an account is present or not present within the structure.

- Linked List (Added function not in homework: insert_position)
- Binary Search Tree

Creator: Keith Nicholas Pallo

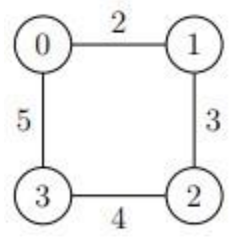
<u>Homework 3: Graphs (C++ and Python Implementation)</u>

Implement a graph data structure with functions to make a graph, get information about the graph, and add edges to the graph.

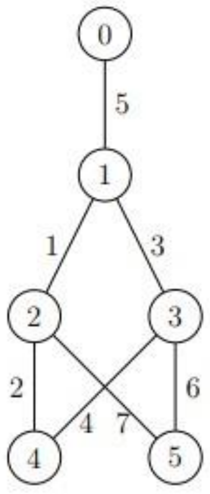
Once these functions are complete, create two sample graphs as shown in the homework (images shown below for simplicity). Then, Implement a depth first search on the graph.

I have created my core graph data structure in python using an adjacency matrix. I have also laid out the core functions for a dynamic unweighted graph in C++, to show usage of the C++ library vector module. For this implementation both the vertex list, and the adjacency list are vector objects as defined in C++. Hence, one could implement additional functions to add or remove vertices from the graph.

<u>Graph 1</u>



<u>Graph 2</u>



<u>Homework 4: Binary Heaps (C++ Implementation)</u>

Implement a binary heap data structure that has an explicit maximum number of nodes determine at the time of creation. Once the binary heap is created, implement the following functions:

- heap_insert (insert a new node)
- find_min (find the minimum value)
- remove_min (remove the minimum value and keep all heap properties in tact)

In order to achieve desired function complexity requirements, this has been done using an array.

Creator: Keith Nicholas Pallo

Homework 5: Union find and MST (Python Implementation)

Implement a union find data structure class that creates a union find instance with n number of singleton disjoint sets predefined at the time of creation. The classical disjoint set functions should be available, which are the following:

- size (returns the number of nodes in the disjoint set)
- union (merges two sets together if they separate)
- find (finds the current set of a desired node)
- same_set (determines if two objects are within the set or not)

Then, using the graph data structure created in problem 3 (adjacency list representation in python) create a function that takes in any graph and performs the Kruskal Algorithm to find the minimum spanning tree in a weighted undirected graph. This has been performed on an example, whose solution has been shown following the relevant links section of this document.

Relevant Links:

Course Website: http://users.eecs.northwestern.edu/~jesse/course/eecs214-fa17/#homework

Homework 1: http://users.eecs.northwestern.edu/~jesse/course/eecs214-fa17/hw/1/warmup.pdf

Homework 2: http://users.eecs.northwestern.edu/~jesse/course/eecs214-fa17/hw/2/dictionaries.pdf

Homework 3: http://users.eecs.northwestern.edu/~jesse/course/eecs214-fa17/hw/3/graph.pdf

Homework 4: http://users.eecs.northwestern.edu/~jesse/course/eecs214-fa17/hw/4/binheap.pdf

Homework 5: http://users.eecs.northwestern.edu/~jesse/course/eecs214-fa17/hw/5/unionfind.pdf

Kruskal Algorithm Detail: https://en.wikipedia.org/wiki/Kruskal%27s_algorithm