Name: Alex Book
ID: 108073300
Collaboration:
Varun Narayanswamy, Michael Hasenkamp, Jamie Foster, Irvin Carbajal, Jacob Reed, Simon Koeten

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 5b (48 points) **Fall 2019, CU-Boulder**

Instructions for submitting your solution:

- The solutions **should be typed** and we cannot accept hand-written solutions. Here's a short intro to Latex.
- You should submit your work through **Gradescope** only.
- If you don't have an account on it, sign up for one using your CU email. You should have gotten anhttps://www.overleaf.com/project/5d9127618927ee0001beb810 email to sign up. If your name based CU email doesn't work, try the identikey@colorado.edu version.
- Gradescope will only accept **.pdf** files (except for code files that should be submitted separately on Gradescope if a problem set has them) and **try to fit your work in the box provided**.
- You cannot submit a pdf which has less pages than what we provided you as Gradescope won't allow it.
- Verbal reasoning is typically insufficient for full credit. Instead, write a logical argument, in the style of a mathematical proof.
- For every problem in this class, you must justify your answer: show how you arrived at it and why it is correct. If there are assumptions you need to make along the way, state those clearly.
- You may work with other students. However, **all solutions must be written independently and in your own words.** Referencing solutions of any sort is strictly prohibited. You must explicitly cite any sources, as well as any collaborators.

Name: Alex Book
ID: 108073300
Collaboration:
Varun Narayanswamy, Michael Hasenkamp, Jamie Foster, Irvin Carbajal, Jacob Reed, Simon Koeten

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 5b (48 points) **Fall 2019, CU-Boulder**

1. (25 pts) For this question, you are going to implement Kruskal's algorithm and union-find to build an MST from supplied data. Refer to the python starter code **MST_Q1_starter_code.p** on Canvas that generates a graph of US cities, where the cities are the vertices and the edges are the distances between them. The code requires **miles_dat.txt.gz** file as the graph data source so keep it in the same folder as the code. Before you start writing any code, make sure you can build the code that's been supplied. The code uses the networkx library. You may need to install this library for the code to run.

   Read all instructions for this question carefully.

   (a) (5 pts) Complete the code to find the edges that are part of the MST. You should add these edges in the list *kruskal_selected_edges*. Do not change the existing format of the edges. They are represented as a tuple of vertices and a vertex is represented like $v =$ "Waukegan, IL". Read the comments in the code for more information. You don't need to read/understand the *miles_graph*() and *draw_graph*() functions.

   (b) (10 pts) Implement the *union*() function to implement Kruskal's.

   (c) (10 pts) Modify your code slightly so that you can produce disconnected components. Let's call these components clusters. The "spacing" of any particular clustering (group of clusters) is defined as the smallest edge between vertices in any pair of different clusters. If we stop Kruskal's k iterations before the algorithm completes, what is the spacing value? Run your code for $k = 2...10$ to generate spacing for all these k values. Your code needs to have this calculation for your answer to receive credit.

   (d) In the pdf that you submit for this assignment, please include the following:

      i. One of the generated graphs **MST.png** that your code produces that shows the MST for that run. Note that on each run, you can get a different number of edges to begin with. Thus, you can expect a different answer each time you run.

      ii. The spacing values for each k value that you use.

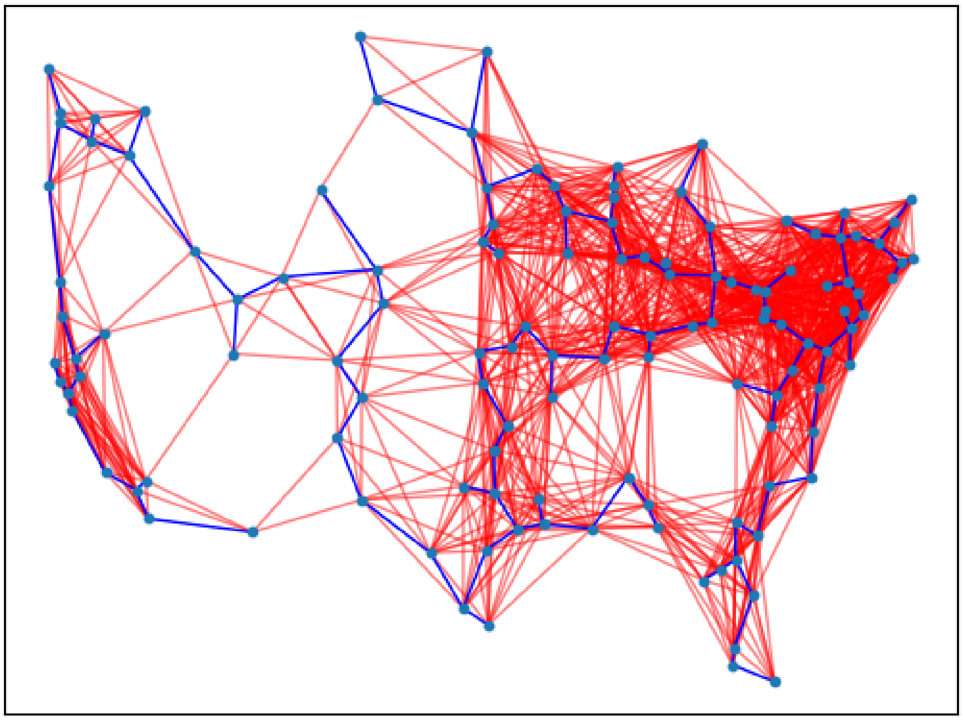
      iii. Your .py file for this question needs to be submitted to Canvas.

Name: Alex Book
ID: 108073300
Collaboration:
Varun Narayanswamy, Michael Hasenkamp, Jamie Foster, Irvin Carbajal, Jacob Reed, Simon Koeten

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 5b (48 points) **Fall 2019, CU-Boulder**

(Space for Q1 image and spacing values)

Threshold = 560, Edges in the MST = 127



k	spacing
2	418
3	358
4	357
5	344
6	324
7	320
8	278
9	271
10	236

Name: Alex Book
ID: 108073300
Collaboration:
Varun Narayanswamy, Michael Hasenkamp, Jamie Foster, Irvin Carbajal, Jacob Reed, Simon Koeten

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 5b (48 points) **Fall 2019, CU-Boulder**

2. (3 pts) How many disconnected components are there when you stop Kruskal's k round before you complete the MST? Justify your answer.

Solution.
Take an arbitrary base case to be k=0 (to aid in a more straightforward justification). There is 1 disconnected component, since you stop 0 rounds before completion (so at completion). If you stop one round sooner, you would have one less edge in the MST, and therefore, one more disconnected component (as an MST can't have cycles, so taking away an edge disconnects a part of the MST). Each further round you go back, you remove one more edge, adding one more disconnected component. Therefore, when you stop k rounds before the completion of the MST by Kruskal's algorithm, there are k+1 disconnected components.

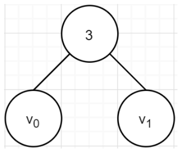
3. (5 pts) Consider the recurrence $F_n = 2F_{n-1} + F_{n-2}$, with the base cases $F_0 = 1$ and $F_1 = 2$. Suppose we have letters $v_0, \ldots, v_7$; where for $i \in \{0, \ldots, 7\}$, the frequency of v_i is given by F_i. Draw a Huffman tree for $v_0, \ldots, v_7$.

Solution.
The frequencies are as follows:

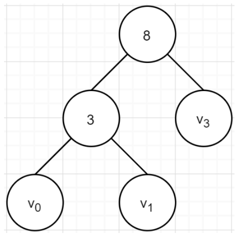
vertex	frequency
v_0	1
v_1	2
v_2	5
v_3	12
v_4	29
v_5	70
v_6	169
v_7	408

Huffman's would make each vertex a single-node binary tree, put them in a minimum priority queue, then pull the front two vertices each iteration. It then makes a sub tree out of those two trees, the smaller one being the left branch and the larger one being the right tree (with weight being defined by the sum of the frequencies of all nodes in that sub tree), then inserts that subtree back into the queue. Each step of the algorithm (with the corresponding sub tree(s) at each step) is detailed on the following page.

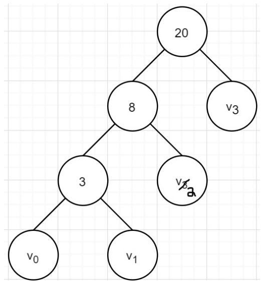
Iteration 1: Queue=(1,2,5,12,29,70,169,408)



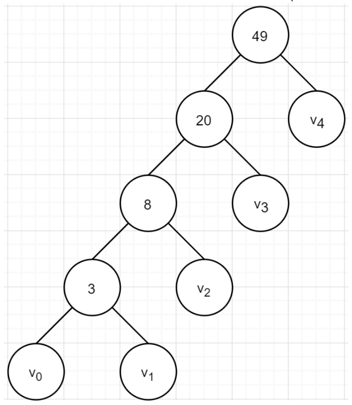
Iteration 2: Queue=(3,5,12,29,70,169,408)



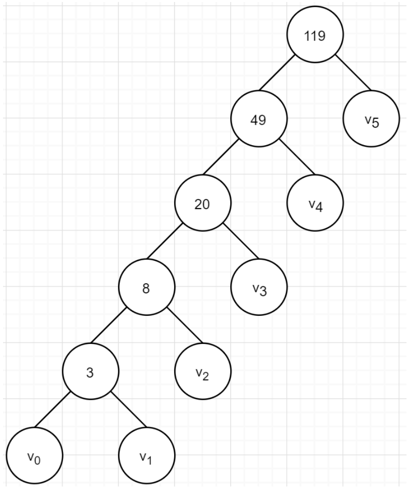
Iteration 3: Queue=(8,12,29,70,169,408)



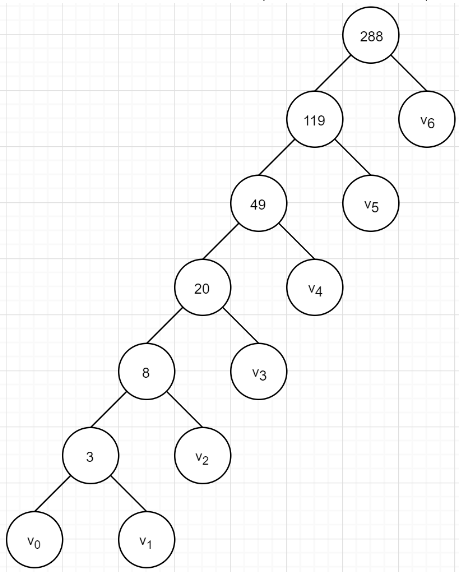
Iteration 4: Queue=(20,29,70,169,408)



Iteration 5: Queue=(49,70,169,408)



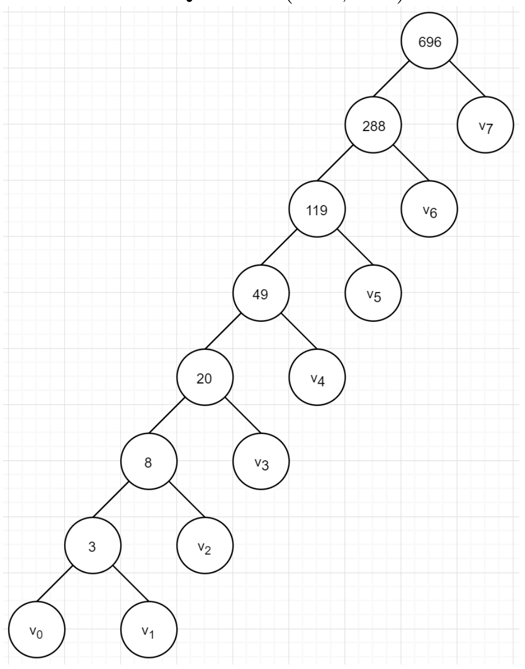
Iteration 6: Queue(119,169,408)



Name: Alex Book
ID: 108073300
Collaboration:
Varun Narayanswamy, Michael Hasenkamp, Jamie Foster, Irvin Carbajal, Jacob Reed, Simon Koeten

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 5b (48 points) **Fall 2019, CU-Boulder**

Iteration 7: Queue=(288,408)



Before iteration 8, Queue=(696) so Queue size = 1, therefore Huffman's returns the completed tree, which was shown above in iteration 7.

Name: Alex Book
ID: 108073300
Collaboration:
Varun Narayanswamy, Michael Hasenkamp, Jamie Foster, Irvin Carbajal, Jacob Reed, Simon Koeten

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 5b (48 points) **Fall 2019, CU-Boulder**

4. (5 pts) Assume you run your Huffman tree algorithm and you produce the following pre-fix codes. Describe why there must be an error in your algorithm.

```
S = 00
c = 01
i = 001
e = 011
n = 101
```

Solution.
The codes for S and i overlap (S is 00, i starts with 00), as do the codes for c and e (c is 01, e starts with 01). If one were to read '00', they would not know if they were reading the entirety of 's' or are in the process of reading 'i' (the same goes for c and e, but with 01 instead of 00). The Huffman tree algorithm shouldn't allow for any ambiguous pre-fix code readings.

Name: Alex Book
ID: 108073300
Collaboration:
Varun Narayanswamy, Michael Hasenkamp, Jamie Foster, Irvin Carbajal, Jacob Reed, Simon Koeten

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 5b (48 points) **Fall 2019, CU-Boulder**

5. (10 pts) Assume you're given an integer matrix that represents a plot of land, where the value at that location in the matrix represents the height above sea level. A value of zero indicates water. A pond is a region of water connected vertically, horizontally, or diagonally. The size of the pond is the total number of connected water cells. Write an algorithm to compute the sizes of all ponds in the matrix.

   Example:

   ```
   0 2 1 0
   0 1 0 1
   1 1 0 1
   0 1 0 1
   ```

   would output 1, 2, 4.

   (a) (3 pts) Describe the graph data structure that your algorithm will use for this problem.

   Solution.
   NOTE: I submitted python code for this problem titled 'PS5bNumber5', as allowed by Piazza post @352. Please reference that code as my submission.

   The data structure used by my algorithm is shown at the top of the submitted code. The graph itself is represented by a 2-d array, with neighbors being those immediately touching it (this is explained more later in the data structure with DFS). For use later in the structure, the numbers of rows and columns that the graph has is also stored. Each graph also has a matching 2-d array of Booleans 'visited' that keeps track of which nodes in the graph have been visited so far.

   The structure also has a function 'exists' that simply checks if a vertex exists in the given graph (used to make sure only valid neighbors are searched, as opposed to getting an error for searching outside of the range of the graph).

   The structure also has a function 'DFS' that recursively searches through all neighbors of a given node, then all neighbors of that node, so on and so forth (following a depth-first-search) setting the corresponding place in the 'visited' array to True for each node that is searched. For each call, it looks at the 8 possible locations of neighboring nodes (up and to the left, straight up, up and to the right, straight left, straight right, down and to the left, straight down, and down and to the right). If the node exists in the graph, has not been visited, and has

Name: Alex Book
ID: 108073300
Collaboration:
Varun Narayanswamy, Michael Hasenkamp, Jamie Foster, Irvin Carbajal, Jacob Reed, Simon Koeten

an elevation value of 0, DFS is called on it. Additionally, the function takes in a value 'size' that is used to keep track of the size of the pond in question (relevant because DFS is only ever called on nodes having water - an elevation value of 0 - so when DFS is called in the pondSizes function, it will return the size of the entire pond that stems from the node in question during the traversal of the graph).

(b) (2 pts) Provide a 3-4 sentence description of how your algorithm works, including how the matrix is converted to the graph, how adjacent vertices are identified, and how the algorithm traverses the graph to identify connected vertices.

Solution.
As explained in part a, the matrix is converted to a graph by keeping it as a matrix/2-d array. Possible adjacent vertices are those in one of the 8 spots surrounding a vertex. Whether or not that adjacent vertex indeed exists (as opposed to being out of range) is checked by the 'exists' function, and used when calling DFS.

The pondSizes function itself works mainly by using the helper functions defined in the data structure. First, an empty list 'ponds' is initialized, which we eventually be returned, full of the sizes of any ponds. The 2-d array is then traversed with nested for loops, looking at the vertices of each row. If the value of the vertex in question is 0 and it is unvisited, the list of pond sizes is appended with the size of that pond, found by calling DFS on that vertex. Therefore, once an entire pond is found, none of the vertices in that pond will be analyzed again (double-counted), as their visited values will have all been set to True as they were analyzed by DFS. Once the array has been traversed, the list of pond sizes is returned.

(c) (5 pts) Write an algorithm to solve this problem.

Solution.
I submitted the algorithm for this problem via Canvas. Feel free to change the test case to check for correctness, and please reference the code in any explanations I gave in parts a and b.