COMP 285 (NC A&T, Spr '22) Homework 10

Due. Friday, April 29th, 2022 @ 11:59 PM!

Homework Expectations: Please see Homework.

Exercises The following questions are exercises. We encourage you to work with a group and discuss solutions to make sure you understand the material.
Points This assignment is graded out of 30 points.

Fun with MSTs, Flows, and Cuts

Written Problems The following questions are to be submitted in written/typed form to gradescope.

1 Learning for Fun (0 pt.)

Watch 1 (<30 minute video) or read 2 before you start the following two questions:

1. If you prefer a lecture format, this is a really good video resource (as a general note, this person's lectures are high quality across the board). Slight terminology differences in this video from the HW and textbook: he uses "weighted union" which is similar to the "union by rank", and he says "collapsing find" which is essentially "path compression".
2. If you prefer a written resource, Ch 21 in the CLRS textbook is fully dedicated to the topic. You can skip over the exercises and detailed math proofs to understand the main ideas, and they are very similar to the ideas explained in the video.

2 Understanding Kruskal's Algorithm (8 pt.)

In class, we showed an abbreviated version of Kruskal's pseudocode. The actual pseudocode looks like this:

```
algorithm kruskal(G, w)
  Input: A connected undirected graph G = (V, E) with edge weights w(u, v)
    for all (u,v) in E
  Output: A minimum spanning tree defined by the edges X

  for all u in V:
    makeset(u)
  X = {}
  Sort the edges E by weight
  for all edges {u, v} in E, in increasing order of weight:
    if find(u) != find(v):
      add edge {u, v} to X
      union(u, v)
  return X
```

Recall during the Kruskal's lecture (Lecture 32) how we would make sure that adding an edge between two nodes would not create a cycle. A data structure called "Disjoint Sets" allow us to do this efficiently by keeping track of / updating a "set" for each connected component.

We have 3 functions related to Disjoint Sets:

1. `makeset`(x), which takes in a node x and creates a new set with just x in it.
2. `find`(x), which takes in a node and returns which set it belongs to.
3. `union`(x, y), which takes in two sets, and combines them into one big set.

2.1 (2 pt.)

How many times will we call `makeset` within Kruskal's? Put your answer in terms of $n = |V|$ and $m = |E|$.

[We are expecting: A mathematical expression using n and m.**]**

2.2 (2 pt.)

How many times will we call `find` within Kruskal's? Put your answer in terms of $n = |V|$ and $m = |E|$.

[We are expecting: A mathematical expression using n and m.**]**

2.3 (2 pt.)

How many times will we call `union` within Kruskal's? Put your answer in terms of $n = |V|$ and $m = |E|$.

[We are expecting: A mathematical expression using n and m.**]**

2.4 (2 pt.)

Describe using your own words why `find(u) != find(v)` is the same as "if this edge doesn't cause a cycle".

[We are expecting: One sentence explaining the above.**]**

3 Disjoint-Set Data Structure (8 pt.)

Now here's the code for the Disjoint Set functions above:

```
algorithm makeset(x)
  Input: a graph node x
  Output: modify x such that it has a "rank" and a "parent"

  x.rank = 0
  x.parent = x

algorithm find(x)
  Input: a graph node x
  Output: the ancestor

  if x != x.parent
    x.parent = find(x.parent)
  return x.parent

algorithm union(x, y)
  Input: graph nodes x and y
  Output: modify x and y so that they are now connected in the same "set"

  x = find(x)
  y = find(y)

  if x.rank > y.rank
    y.parent = x
  else
    x.parent = y
    if x.rank == y.rank
      y.rank = y.rank + 1
```

3.1 (2 pt.)

Suppose we make the following sequence of calls: '`makeset`(A); `makeset`(B); `makeset`(C); `makeset`(D); `union`(C, D); `union`(A, B); `union`(B, D);'. Draw what the set(s) look like at this point (either a picture or text representation is fine).

[We are expecting: Include an image or text representation of what the data structure looks like.**]**

3.2 (2 pt.)

You call ‘`find`(x)‘ on the above and it takes multiple recursive calls, then you call ‘`find`(x)‘ again on the same node and it takes fewer. What is ‘x‘? This phenomena is called "path compression".

[We are expecting: A short explanation of what ‘x’ stands for.**]**

3.3 (2 pt.)

Draw what the set(s) look like after the additional ‘`find`‘ calls.

[We are expecting: Include an image or text representation of what the data structure looks like.**]**

3.4 (2 pt.)

What does rank signify for each node?

[We are expecting: A short explanation of what the ‘rank’ represents**]**.

4 Network Flow Cuts (14 pt.)

An **s-t cut** of a flow network is a partitioning of nodes into two groups, one which contains the source s and the other which contains the sink t.

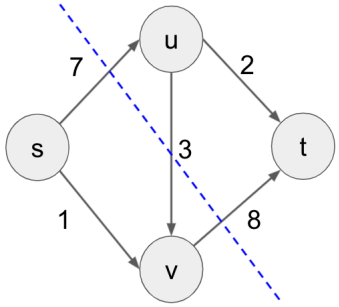

Figure 1: An example of an s-t cut.

The image in Figure 1 represents the cut $\{s, v\}/\{u, t\}$. The **cut capacity** is the sum of all edge capacities that **go from the set containing s to the set containing t**. So the cut capacity shown above is $7 + 8 = 15$ (note that we do not include 3).

4.1 (9 pt.)

Define all 4 s-t cuts in the graph above and calculate their capacities. The states that the max flow in a network is equal to the capacity of the $s - t$ cut with minimum capacity. Using this, what is the max flow of this network?

[We are expecting: The 4 s-t cuts along with their capacities and the value of the max flow.**]**

4.2 (5 pt.)

Find and list flow values for each edge that will give this max-flow. What do you notice about the flows of the edges along the minimum s-t cut? Use this observation to explain why the max-flow min-cut theorem makes intuitive sense.

[We are expecting: Flows for each edge and a short explanation of why the max-flow min-cut theorem makes sense.**]**

5 Negative Prim? (5 pt.)

We want to now consider a similar algorithm to Prim's called Negative-Prim for computing minimum spanning tree in graphs with negative edge weights.

This algorithm adds some number to all of the edge weights to make them all nonnegative, then runs Prim's algorithm on the resulting graph, and argues that the Minimum Spanning Tree in the new graph are the same as the MST in the old graph. You can assume that all the edge weights are unique integers.

```
Negative-Prim(G, s):
  minWeight = minimum edge weight in G
  for e in E: # iterate through all edges in G
    modifiedWeight(e) = w(e) - minWeight
    modifiedG = G with weights modifiedWeight
  T = Prim(modifiedG, s) # run Prim's algorithm starting from s
  update T with edges that corresponds to graph G
  return T
```

[We are expecting: Either an informal explanation of why Negative-Prim computes the correct MST, or a counter-example of an undirected graph with negative edge weights where Negative-Prim does not output**]**

Submitting the Assignment

The assignment should be submitted through Gradescope.

The "Homework 10: Fun with MSTs, Flow, and Cuts" assignment is the written portion, for which you should submit a **typed** response to questions 1-5. Each response should clearly be marked with its corresponding number. You are free to use the provided templates, print the questions and write your answers, or to simply type your responses on a blank document (whatever works for you).