# CS 6363: Homework 3

Version 1.0

Instructor: Benjamin Raichel

Due November 9, at 11:59 pm

Homework is to be submitted online through eLearning by the above deadline, and on paper in the following class. Typeset solutions (in LaTeX) are strongly preferred, but not required. If you chose to hand wright and scan your solutions, if what is written is illegible you will be marked down accordingly. Please make sure your name is on each page of the submission. You do not need to restate the problem, just the problem number.

Explanations are to be given for each problem, unless the problem specifically states it is not necessary. Explanations should be clear and concise. Rambling and/or imprecise explanations will be marked down accordingly.

## Problem 1. Fractional Jewelery Collection (20 Points)

Recall the jewelery collection problem from homework 2. There is a set of jewels labeled $\{1, \ldots, n\}$ for which the sizes are given in an array $S[1 \ldots n]$ and the values are given in an array $V[1 \ldots n]$. The robber also has a bag of size $b$ to place the stolen jewels in. You can assume $b$ as well as the entries in $S$ are all positive integers, and the values in $V$ are positive real numbers. To make things simple, you can also assume all $V[i]/S[i]$ values are distinct.

Here we consider the fractional version of this problem. Specifically, the robber has brought a jewel cutter, and can either take the whole jewel or any desired fraction of the jewel. Specifically, if for any $c \geq 1$, the robber takes a $1/c$ fraction of the $i$th jewel then the size is $S[i]/c$ and the value is $V[i]/c$. Note that in an optimal solution $S[i]/c$ will be an integer, and if it helps simplify your solution, you can restrict to only allowing $c$ such that $S[i]/c$ is an integer. As before, the goal of the robber is to get the largest total value (i.e. sum of values) of the jewels that are stolen, subject to those jewels fitting in his bag.

**a)** Give a short description of the greedy algorithm for this fractional variant.

**b)** Prove that your greedy algorithm is correct.

## Problem 2. Counting Paths (20 points)

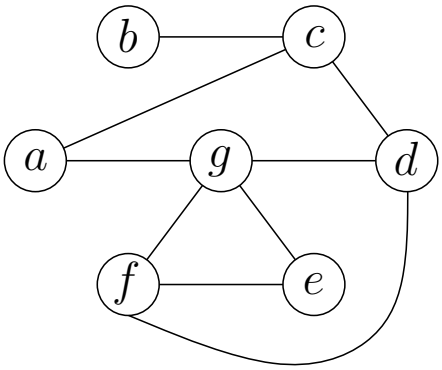
You are given a directed acyclic graph, $G = (V, E)$, and two labeled vertices $s, t \in V$. Give an $O(|V| + |E|)$ time algorithm to count the number of directed paths in $G$ which start at $s$ and end at $t$.

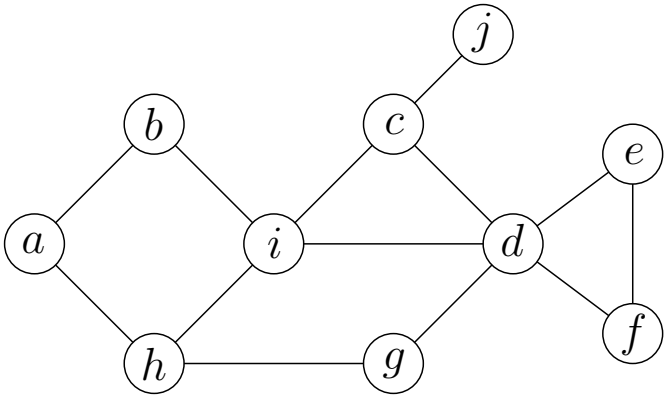
## Problem 3. Running the Algorithms (36 points)

No explanation is required for any part of this problem.

Some parts of this problem require drawing graphs. The figures below were created using the ipe drawing editor, and I recommend this for your figures since it is free, easy to use, and makes nice figures. Otherwise, feel free to hand draw and scan your graphs.

**(a)** (12 points) Draw the DFS and BFS trees of the following two graphs, when called from vertex $a$. (So you should submit four trees in total.) There is ambiguity in the order which neighboring vertices are processed, so to make the trees consistent, process neighbors in alphabetical order.



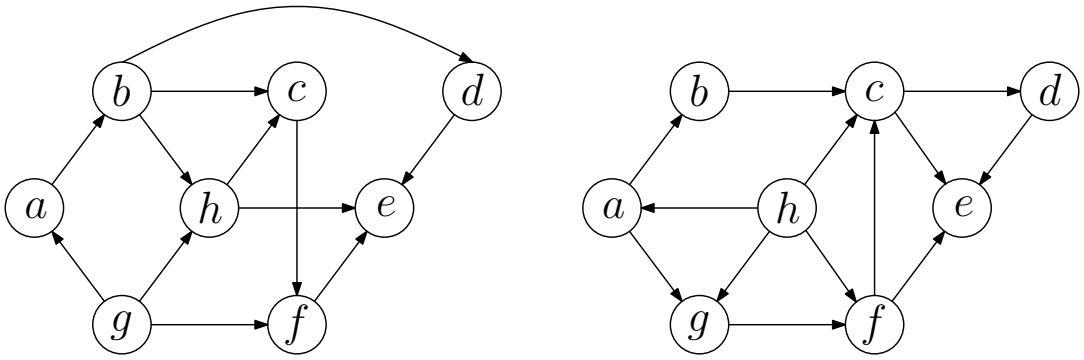




In case it was unclear from lecture the following algorithms should be used for BFS and DFS. The edges of the resulting trees are all the $(v, parent(v))$ pairs. Since neighbors are to be processed alphabetically, for BFS neighbors are put in the queue in alphabetical order, and for DFS recursive calls to neighbors are made in alphabetical order.

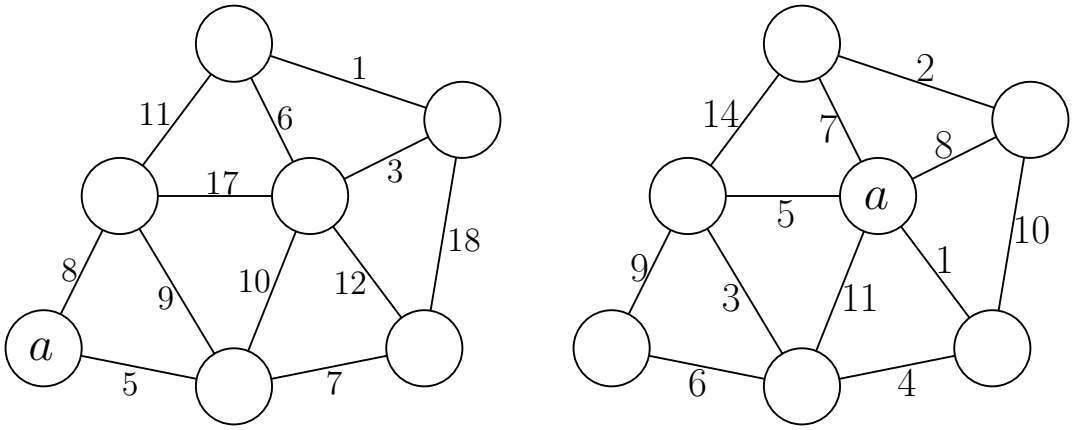
```
1:  procedure BFS(v)
2:      Initialize a queue, Q, with vertex v
3:      Mark v
4:      while Q not empty do
5:          h = Q.Pop()
6:          for all edges hw do
7:              if w not marked then
8:                  Q.Push(w)
9:                  Mark w
10:                 Parent(w) = h
```

```
1:  procedure DFS(v)
2:      Mark v
3:      for all edges vw do
4:          if w not marked then
5:              Parent(w) = v
6:              DFS(w)
```

**(b)** (8 points) Give a topological ordering of the vertices for each of the following two graphs. Preferably you will draw each graph on a horizontal line with vertices in topological order and all edges going from the left to right. However, simply listing the vertices in a correct topological order will get full credit.




**(c)** (8 Points) Give the weight of the fifth edge added when running Prim's algorithm for each of the two graphs below, when starting from vertex $a$.




**(d)** (8 points) Give the weight of the fifth edge added when running Kruskal's algorithm for each of the two graphs above.

## Problem 4. Minimum Spanning Tree Problems (24 points)

In the following $G$ is always edge-weighted, undirected, and connected. You can assume you have a correct implementation of one of the MST algorithms from class, say Kruskal's or Prim's, with running time say $O(m \log n)$. Rigorous proofs are not required though please explain your answers.

Problem 5 in Chapter 20 of Jeff's notes:

(a) (5 points) Describe and analyze an algorithm to compute the maximum weight spanning tree of a given graph $G$. [Hint: this is easy.]

(b) (9 points) A feedback edge set of an undirected graph $G$ is a subset $F$ of the edges such that every cycle in $G$ contains at least one edge in $F$. In other words, removing every edge in $F$ makes the graph $G$ acyclic. Describe and analyze a fast algorithm to compute the minimum weight feedback edge set of of a given edge-weighted graph. For simplicity you can assume the weights in $G$ are positive.

Problem 6b in Chapter 20 of Jeff's notes:
You are given an undirected graph $G$ with weighted edges, and an MST, $T$, of $G$.

(c) (10 points) Give an algorithm to update the MST when the weight of a single edge $e \in G$ is increased, producing a new graph $G'$. Specifically, the input is the edge $e$ and its new weight, and your algorithm should modify $T$ so that it is an MST in $G'$, and do so in $O(|V| + |E|)$ time.

Useful Fact from class: The min weight edge across any vertex bi-partition must be in the MST. Moreover, (the removal of) any edge $e$ in the MST induces a bi-partition, call it $B(e)$, and so $e$ must be the minimum weight edge across $B(e)$.