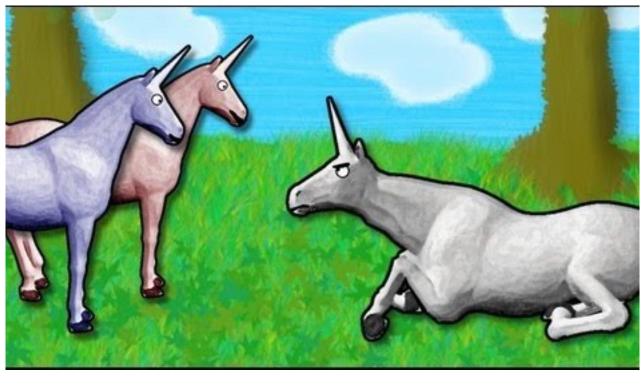
CMSC 420, Fall 2018
Homework #1

First & Last Name: ______________________________

UID (9 digits): ______________________

Problem 1: Charlie the Unicorn *(20 pts)*



Charlie! We're going on an adventure, Charlie!

Please note: This is a real homework assignment, worth 20% of your homework points.

Watch the entire saga of *Charlie the Unicorn* on the YouTube channel Filmcow and answer the following questions. We will be referring to the other two annoying unicorns in the video(s) as Red and Blue, respectively.

(a) What is the name of the **location** that the unicorns are traveling to in their first adventure?

____

(b) According to Charlie, what did the **magical Liopleurodon** say? ______________________

(c) What did the unicorns steal from Charlie at the end of the first adventure? ______________

(d) What kind of **fish** are the two annoying unicorns ostensibly imagining in the beginning of the second adventure? ______________________________

(e) To whom must the unicorns return **the amulet**? ______________________

(f) And, if they don't deliver the amulet, what will **the vortex** supposedly let out? ______________

____

(g) What is the name of the **giant sneakertrain** that the unicorns must board to reach their second adventure destination? ____________________

(h) According to the song sung by the creature in the end of the video, what must Charlie put **in his ear** in order to alleviate his general distress and everyday suffering? ____________________

(i) What did the unicorns steal from Charlie at the end of the second adventure? ____________________

(j) In the beginning of the third adventure, where do the annoying unicorns claim to be coming from?

____

(k) What do Red and Blue want to employ Charlie's help with? ____________________

(l) Why do the unicorns need to be **silent**? ____________________

(m) Fill - in the following dialogue between Charlie, Red and Blue.

- *Blue: Ring ring!*
- *Red: Hello?*
- *Blue: Ring ring!*
- *Red: He... he... hello?*
- *Blue:* ____________________
- *Red:* ____________________
- *Blue:* ____________________
- *Red:* ____________________
- *Charlie:* ____________________

(n) While underwater, what is **the first obstacle** that our unicorns have to pass in order to reach their final destination? ____________________

(o) In the song that essentially culminates the third adventure, what kind of **sea creature** is **madly in love** with Charlie? ______

(p) What did the unicorns steal from Charlie at the end of the third adventure? ______

(q) What is the destination of the fourth and, as of the time of this writing, final adventure of our unicorns?

(r) How did Charlie "defeat" **the millepede**? ______

(s) What did Charlie find inside the **Cavern of the Red Wind**? ______

(t) What did **Starfish** say right at the end of the adventure? ______

(u) *(1 pt extra credit)* In your opinion, what is the **meaning** behind Charlie the Unicorn, if any?

Problem 2: Syllabus / Deadlines *(18 pts)*

To answer some of the questions of this problem, you will need to navigate our course's ELMS page and study our syllabus.

(a) What is the due date of your **first homework**? ______________________

(b) What is the date of your **first** midterm? ______________________

(c) What is the date of your **second** midterm? ______________________

(d) Where are **both** midterms taking place? ______________________

(e) For what **percentage** of your **final score** do **all** of the **projects** count? ___________

(f) For what **percentage** of your **final score** do **all** of the **midterm and final exams** count? ___________

Problem 3: Git basics *(22 pts)*

To pull the starter code for our projects, you will need to familiarize yourselves with the VCS **Git**, since the entire skeleton code will be hosted on a GitHub repository that we maintain for you. To answer this problem effectively, you will need to study up on Git. Please navigate to the page named "More Resources" on our ELMS website if you would like a starting point. Of course, Google is always your friend.

While a large subset of Git's functionality can be implemented using various graphical tools, there is **no substitute** for the modularity, ease and cross-platform compatibility of using simple command line instructions from the shell. In this class, you will need to familiarize yourselves with a **very basic** subset of available Git commands.

For every one of the following tasks, write the **full** command that would be necessary to type on a command line in order for you to achieve the relevant task. Your answer should include:

- The system program `git`.
- The command to be passed to the program `git` (e.g. `add`, `checkout`, `push`, etc).
- Any flags to be passed to the command, e.g `-b`, `--cached`, `-u`.
- Arguments that the command itself may need (e.g branch names, file paths, names of remotes, entire quoted strings, etc).

You are **quite encouraged** to try the commands out in your console before writing your responses below. In fact, the first four commands are a fine example of how you could start implementing your first project! We will provide you with the answer to the first question to help start you off. Note that this command does not have any **flags**, but it does, of course, have a URL argument.

(a) Clone the repository `https://github.com/JasonFil/CMSC420-Fall2018.git` into the current directory, **using HTTPS**.

`git clone https://github.com/JasonFil/CMSC420-Fall2018.git`

(b) Rename the remote `origin` to `classrepo`.

____

(c) Rename the current branch to `skeletoncode`.

____

(d) Switch from the current branch to a **new** branch called `myimplem`.

____

(e) **Stage** a Java source file which you just created and which you named `MyStack.java`.

____

(f) Make a **commit** from the current branch (`myimplem`) with message "Fixed stack popping issue.".

____

(g) Switch back to the branch `skeletoncode`.

(h) **Merge** the branch `myimplem` with the current branch, `skeletoncode`.

(i) **Push** your latest commit upstream.

(j) List all your remotes.

(k) Delete a remote called `server2`.

(l) Assuming you have applied changes to 10,000 different files (e.g by having your IDE automatically create **Scaladocs** for the entire code base of **Twitter**), type a **single command** that will stage the files tracked by `git` and to which the changes have been made.

(m) Tweak your **global, system-wide** Git configuration so that instead of typing `git log -1 HEAD` to show your last commit's information, you could instead type the shorthand `git last`.

Problem 4: Sparse Matrix Representations *(20 pts)*

Whether stored in column or row - major order, two-dimensional matrices can be a big bottleneck in terms of memory if they are **sparse**, that is, they have lots of `null` cells. For example, the matrix shown in Table 1 is quite sparse; only about 14% of it is occupied!

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
X	**11**	X	X	X	X	X	X	X
1	X	X	X	X	X	**3**	X	X
2	X	**-3**	**4**	X	X	X	X	X
3	X	X	X	X	X	X	X	X
4	X	X	**11**	**6**	X	X	X	$\mathbf{e^2}$
5	**-1**	X	X	X	X	X	X	X
6	X	X	X	X	X	X	$\sqrt{\mathbf{3}}$	X
7	X	X	X	X	X	X	X	X

Table 1: A sparse matrix. X= `null` for brevity.

Programming languages such as `MATLAB` offer built-in functionality (methods or operators included in the "standard library" of the language) to *sparsify* a matrix. Sparsifying a matrix consists of transforming it into a l**inked list** which contains information *only for those cells of the matrix that have been filled in.* Every node of this list contains the (i, j) coordinates of the relevant cell, as well as the data originally held in the cell.

To answer some of the following questions, you will have to review **asymptotic notation**. We have a resource page on ELMS (with CMSC250 slides linked) where you can review "big-Oh" ($\mathcal{O}(\cdot)$), "big-Theta" ($\Theta(\cdot)$) and "big-Omega" ($\Omega(\cdot)$).

For example, if you believe that a certain operation has time complexity *at most proportional to* M (e.g, $2M, 2.5M, 4M-7$), you should answer with $\mathcal{O}(M)$. If instead you believe that it has time complexity at most proportional to the **square** of M multipled by N (e.g, $10M^2N+2N$), you should answer with $\mathcal{O}(M^2N)$. If you believe that the operation runs in time **exactly** proportional to $M^2 \cdot N$, you should answer with $\Theta(M^2 \cdot N)$.

Questions on opposite page...

(i) Make a drawing of the sparsification of matrix shown in Table 1, assuming **row-major** order. Recall what your nodes should contain.

BEGIN YOUR ANSWER BELOW THIS LINE

(ii) Perform the same task, this time assuming **column-major** order.

BEGIN YOUR ANSWER BELOW THIS LINE

(iii) Suppose that our original matrix has dimensions $M \times N$. As a function of those two variables, and using **"big-Theta" ($\Theta()\cdot$) notation**, what is the **space** that the sparse matrix occupies in memory, in its **original** representation? ____________

(iv) Using the same notation, what is the space that the matrix occupies in its **sparsified** format? ____________

(v) Suppose that $0 \leq i < M$ and $0 \leq j < N$. In its *original* representation, what is the **time complexity** of assigning the cell $((i, j))$ a value of, say, 3 in the matrix's **original** representation? ____________

(vi) What is the time complexity of the same operation in the **sparsified** representation? ____________

Problem 5: Memory representations of graphs *(20 pts)*

Observe the graph of figure 1.

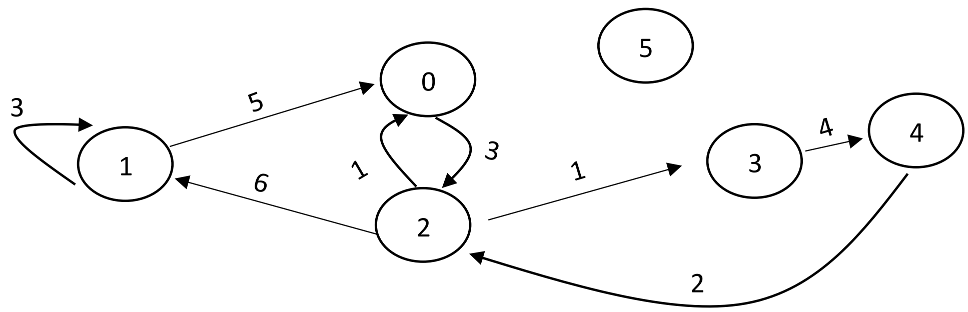


Figure 1: A conceptual image of a graph. Nodes are enumerated from 0 (zero) for consistency with later drawings.

There are two common ways in which graphs, directed or undirected, are represented in computer memory: the **adjacency matrix** (Table 2) and the **adjacency list** (Figure 2).

	0	**1**	**2**	**3**	**4**	**5**
0	0	0	3	0	0	0
1	5	3	0	0	0	0
2	1	6	0	1	0	0
3	0	0	0	0	4	0
4	0	0	2	0	0	0
5	0	0	0	0	0	0

Table 2: The **adjacency matrix** representation of the graph of Figure 1.

Let V be the number of **nodes** (or vertices) in a given graph G and E be the number of its **edges**. The adjacency matrix is a $V \times V$ matrix M where:

$$M(i,j) = \begin{cases} w, & \textit{where } w > 0 \textit{ is the weight of the edge } i \to j \\ 0, & \textit{if there is no edge } i \to j \end{cases}$$

In **unweighted** graphs, $w = 1$ for all edges.

Undirected graphs, on the other hand, have the interesting property that their adjacency matrix is **symmetric**:

$$(\forall i, j \in \{1, 2, \ldots, V\})[M(i,j) = M(j,i)]$$

.

The **adjacency list** representation of this graph is a single-dimensional array A of length V, defined as follows:

$$A(i) = \{v, \ \textit{if vertex } v \textit{ is connected to vertex } i \textit{ in } G\}$$

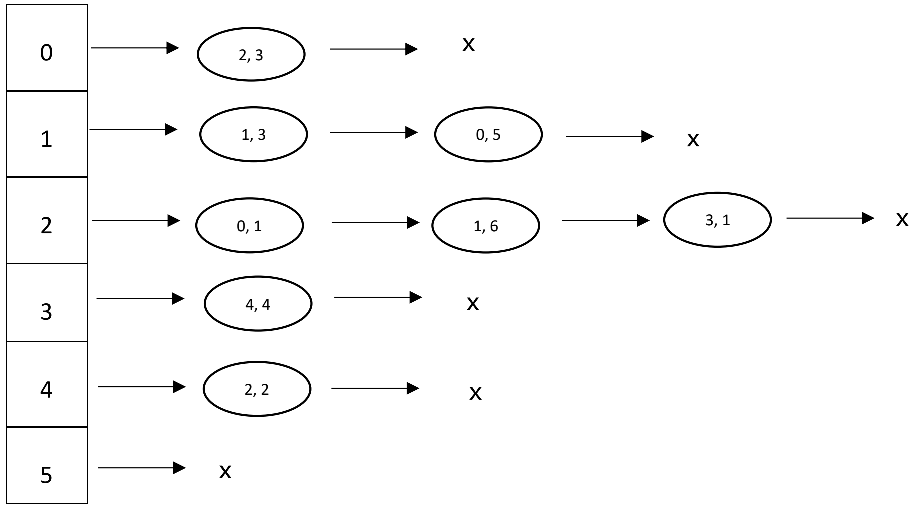


Figure 2: The **adjacency list** representation of the graph of Figure 1. Every node in the list contains a tuple (**target_node_id, weight**).

So $A(i)$ points to the **set** of all vertices v on which i is incident. These sets are typically organized as **linked lists**, as implied by Figure 2. These lists do **not have to be sorted in any way**. For this question, we will assume that:

- The heads of the sets are contained in a classic, one-dimensional contiguous storage array (**not** a "dynamic" array like Java's `ArrayList` and C++'s `Vector`). So the entire adjacency list representation will consist of an **array of lists** (see Figure 2).
- The lists themselves are organized as **classic, bare-bones linked lists**. This means that we will **not** endow these lists with a pointer to their last element, and we will also **not** have a link pointing to the previous node from every node (i.e the linked lists are **not** doubly-connected) (see Figure 2).

Furthermore, we add a little twist to these representations, based on information from problem 4. We will consider a **sparse adjacency matrix** representation of this graph (linked list of only non-`null` elements). Figure 3 shows how the graph of Figure 1 is represented in such a format.

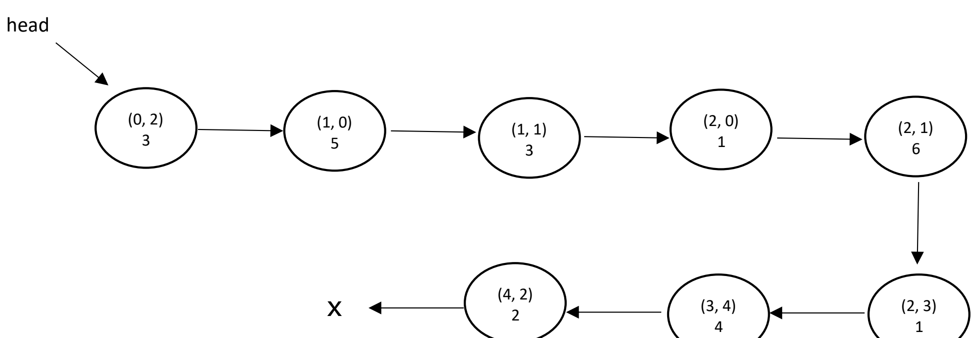


Figure 3: The **sparse adjacency matrix** representation of the graph of Figure 1. The matrix has been scanned in row-major order (but does it really matter which order we use?)

Your task is to fill-in Table 3 using appropriate complexity notation, coupled with the parameters V and E. For example, if you believe that a certain operation will take time *proportional* to the **cube** of the number of nodes V, you should answer with $\mathcal{O}(V^3)$. Or, if you believe that the operation will take time proportional to the product of vertices multiplied by the logarithm base 2 of the number of edges E, you should answer

$\mathcal{O}(V \cdot \log_2 E)$. Finally, also consider the fact that $0 \leq E \leq V^2$, since the "worst case" for the number of edges in a graph is V^2, for a V-clique.

Operation	**Adjacency Matrix**	**Adjacency List**	**Sparse Adjacency Matrix**
Is node i **connected** to node j?			
Add a new **node** k.			
Add the **edge** $i \to j$.			
Remove a **node** ℓ.			
Remove the **edge** $i \to j$.			
Spatial Complexity			

Table 3: Fill in this table with appropriate asymptotic notation.