CS230 Labs

Lab: yEd and DiGraphs

Goals:

1. review and use java's ArrayList and LinkedList classes
2. get familiar with the yEd application
3. practice graph layout
4. start the DiGraph (**Di**rected **Graph**) interface implementation

---

PRE-LAB TASK: Working with yEd

---

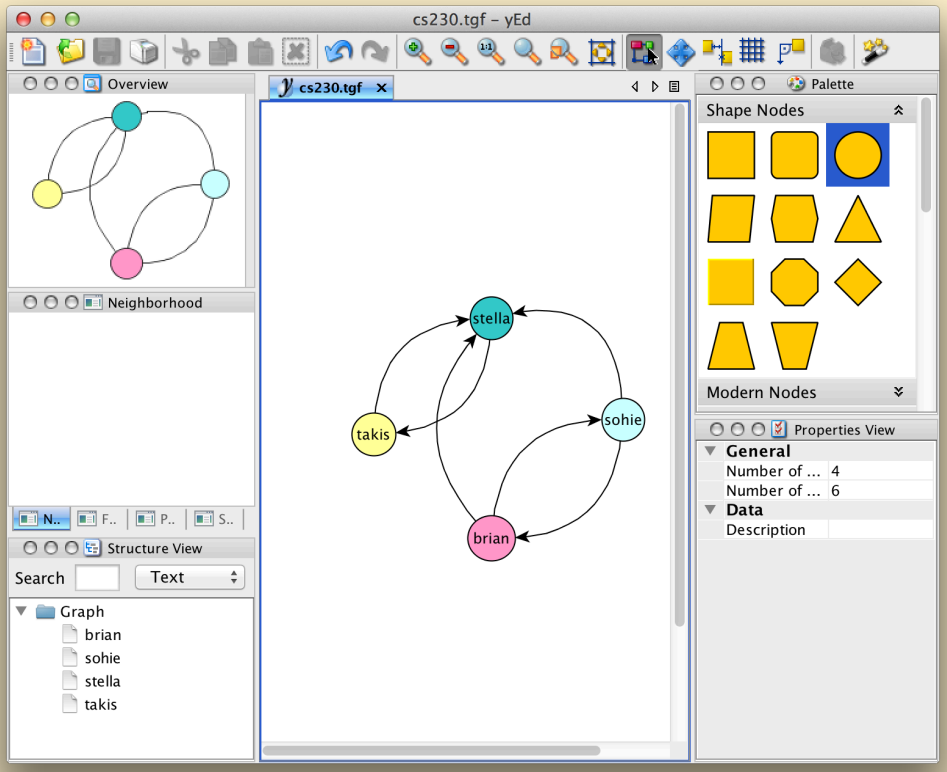
yEd and Graphs 

yEd is a tool for creating, importing, and sharing graphs.

Download yEd to your computer. NOTE: Your yEd screens may look somewhat different to what you see below, as our versions may differ.

Today, we will start writing an **AdjListsDiGraph** class, as an implementation of the **DiGraph** (Directed Graph) Interface. We'll use *yEd* to create graphs and then save these graphs as **.tgf** files (tgf stands for "trivial graph format").

The **AdjListsDiGraph** class will interact with **.tgf** files. Below is a snapshot of **yEd** with a graph with 4 nodes and some arcs.



A **.tgf** file is a text file containing the list of vertices, one per line, followed by the # symbol in a single line, followed by a list of arcs, one per line. What do you think the TGF file for the graph above looks like? How about this?

```
1 brian
2 sohie
3 stella
4 takis
#
1 2
2 1
3 4
4 3
1 3
```

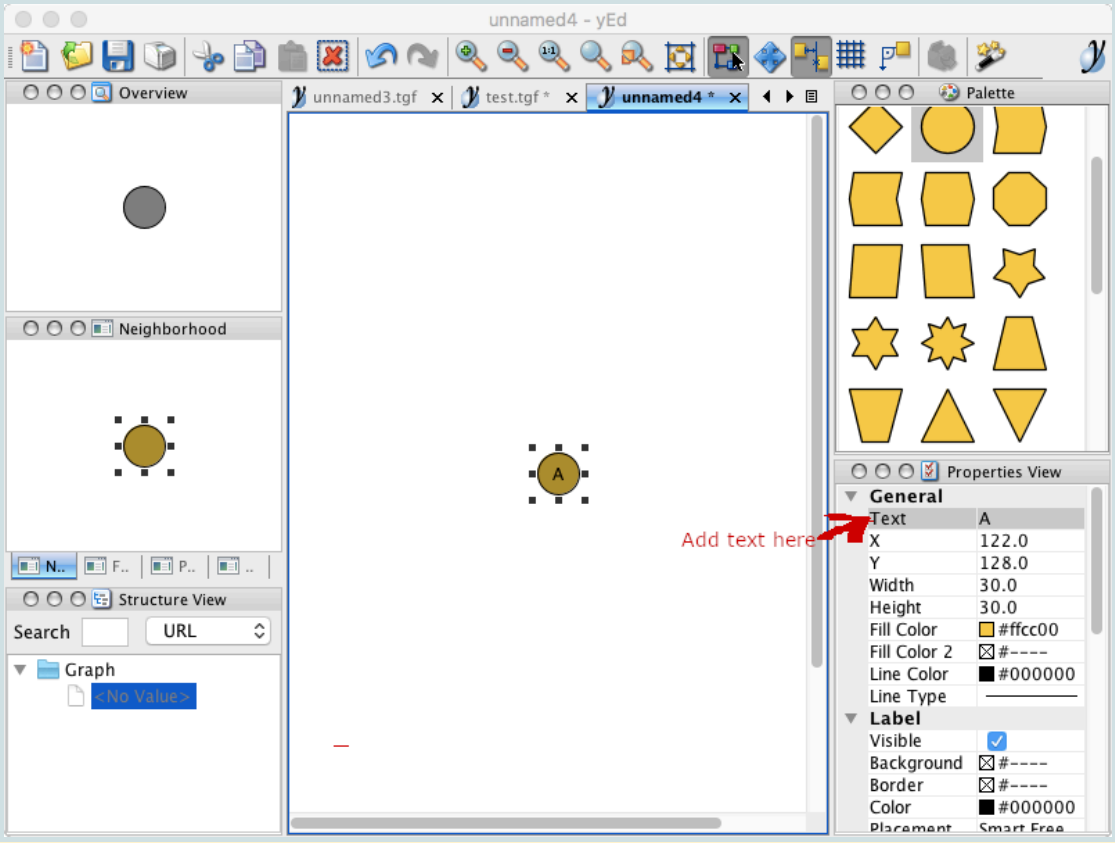
Actually, there is one more arc that is included in the TGF file. Which one?

Creating a graph in yEd

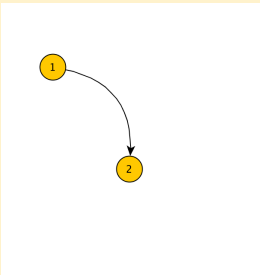
We have created a video to show you how to use **yEd**, feel free to look at it as you read the instructions below.

Follow these steps to get a feel for how **yEd** works. Start yEd. Click on the **New Document** icon.

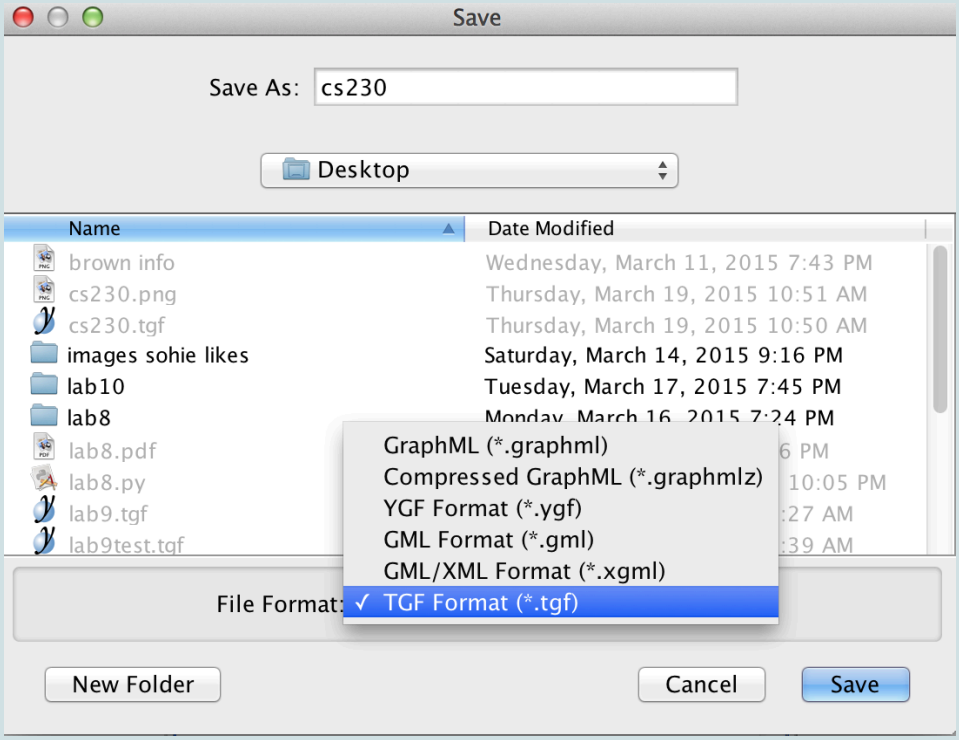
- Drag a node from the **Shape Nodes** palette to the main window.
- Double-click on the node and type "A" in the Text field of the Properties View:



- Repeat three more times, so you have four total nodes, labeling them B,C,D
- Add an edge between two nodes by clicking inside a node and dragging your mouse until it is inside the other node. For example:



- Save your graph as a **TGF** file.
  yEd allows you to save your graphs as .tgf (trivial graph format) files. To do this: **File > Save As > TGF Format (*.tgf)** from the drop down menu (see image below).



Opening a file in yEd

In the folder you downloaded there is a subdirectory named `tgf_graphs`, containing several .tgf files**.** Open one in **yEd**. It seems that there is only one vertex in there, right?. That's because the .tgf format contains no information about how the vertices are layed out, therefore yEd placed them all on top of each other. You can **click-and-drag** one vertex at a time and pull them to a different location on the yEd panel, revealing the other vertices below. Doing so you can create your own layout to have the graph look the way you like it.

PRE-LAB TASK: Review Java's APIs

Task A: Review java ArrayList class

Please access the java API and review the ArrayList class. For the purposes of this work we will need the following methods from that class:

- constructor with no inputs
- add() an object
- size() of the collection
- get() the element at a certain index
- indexOf() a certain element in the collection
- remove() the element at a certain index

In your notebook, make a list of the headers of these methods, so you can have a quick reference to them.

Task B: Review java LinkedList class

Access the java API and remind yourself on how to use this class, and some of its basic methods, like:

- size()
- get()
- add()
- remove()

In your notebook, make a list of the headers of these methods, so you can have a quick reference to them.

Set up

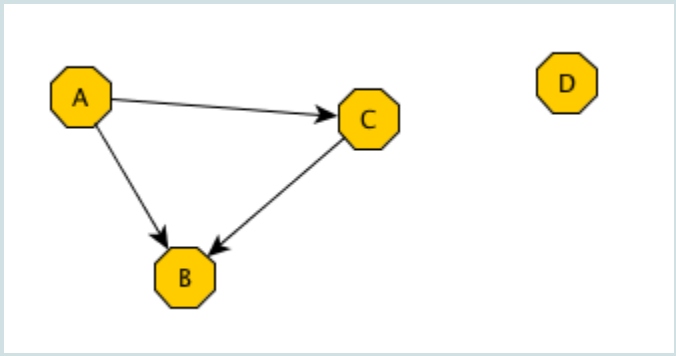
Download the **lab9** folder. Rename it to include the partners names. It contains:

1. the **DiGraph interface**
2. a starting version of the **AdjListsDiGraph** class, which eventually should implement the DiGraph interface.
3. a folder **tgf_graphs**

TASK: Writing the AdjListsDiGraph class

Once the **AdjListsDiGraph** class is defined, the user should be able to create an empty graph, add and remove vertices, and connections between them. In addition, the user should be able to write a graph into a file in **tgf** format, as well as read a graph from a tgf file.

Start by drawing a simple graph, like the one below. In your notebook, fill in the Data Structures to represent this graph.



Set up:

1. the **instance variables**:
    - an ArrayList, named **vertices** to hold the vertices of the graph, and
    - An ArrayList of LinkedLists, named **arcs**. In the Adjacent Lists Directed Graph implementation, each such list corresponds to a vertex, and holds the vertices adjacent to that vertex.
2. the **constructor** with no inputs, which creates an empty graph.
3. the **saveToTGF()** method, which writes the **AdjListsDiGraph** instance into the files whose name is passed as an input to the method.
4. the static **AdjListsGraphFromFile()** method, which takes as an input the name of a tgf file, and returns an instance of an **AdjListsDiGraph**.

As always, you should test your code incrementally, i.e every method as you write it, before you move on to the next one. For testing purposes, you can use the **saveToTGF()**. Alternatively, you can define the **toString()** early on in your implementation, and use this for verifying your results.

For the graph shown earlier, a reasonable String representation might be something like this:

```
Vertices:
[A, B, C, D]
Arcs:
from A: [B, C]
from B: []
from C: [B]
from D: []
```

We have found the above **String representation** to be quite convenient and easy to implement. Of course, it is up to you to decide on a "reasonable String representation".

SUBMITTING

Please submit **AdjListsDiGraph**`.java` by uploading to Gradescope.

SAMPLE SOLUTIONS