About HW2:

General:

Review how to walk a linked list, including maintaining previous pointer

Question #4:

Do we *have* to shift rest of array up, or is there an alternative?

What does answer to that depend upon?

Note: Design of best data structures and algorithms can very problem domain- and solution-specific.

Linked Lists:

Structure depends very much on what functions it needs to provide and what runtime (speed) these operations need

That in turn depends on how frequently those operations are needed

Some possible functional requirements:

Scan direction: forward, or bidirectional?

Insertion needs:

Just at end? Which end, or both?

Also insert in middle? At arbitrary pt, or as we traverse?

Deletion?

Traversal/iteration?

Allow modification of list as we are traversing?

Some properties:

singly- or doubly-linked

head ptr and/or node, tail ptr and/or node, dummy node

(Gibson's slides are available: show one specific type)

Study of Project1--pass 2:

Now, let's look at some more elements of the project:

1) What kind of LL are we using?

Structure set in project spec--very simple:

singly-linked, with just a single header pointer

List can be empty

List is unordered

Need insertion, but at any pos (i.e., position doesn't matter)

so you only need to support simplest: insertion at head

Don't need to support deletion

2) Observe existing class design from project spec and provided Graph.h:

"It is what it is"

3) Copy constructor: should create deep new copy

4) Overloaded operator=: must handle cleaning up previously existing data;

(after that, acts much like copy constructor)

5) Iterators:

Purpose is to encapsulate state of traversal, in a compact,

self-contained form.

Neighbor iterator: Simple

Edge iterator: Harder

Special things to consider:

iterator end (returneed by egEnd()) have some flexibility in design--just be consistent--match what operator!= is looking for.

must only return each edge once--i.e., shouldn't return both (1,2) and (2,1); this means skipping over one or more edges

Watch out for vertices with no neighbors, i.e., where m_adjLists[i] == NULL

when you hit end, must return with iterator in state such that "iterator != graph->