

Week 7

ASSIGNMENTS

~~h5~~ due before 10pm on 3/4

~~h6~~ ungraded practice problems *(exam review)*.

x3 ~~x4~~ due before 10pm on Friday 3/8 (week 7)

~~p3~~ due before 10pm on Thursday 3/14 (week 8)

~~x4~~ available soon due before 10pm on Friday 3/15 (week 8)

Peer Mentors: will help student teams with hashing and p3

writing question.

MIDTERM EXAM: THURSDAY MARCH 7th, 5:00 pm – 7:00 pm

Lecture 001 students: Room 6210 of Social Sciences Building

Lecture 002 students: Room B10 of Ingraham Hall Building

Lecture 004 students: Room 125 of Agriculture Building

Bring to exam:

- UW ID
If you do not have your UW ID, you will be asked to wait until after students with ID have been admitted
- #2 Pencils and a good eraser

At Exam:

- arrive early
- get ID scanned and get scantron form
- find seat directly behind another student (all students must be in "columns")

See <https://pages.cs.wisc.edu/~deppeler/cs400/exams>

Read: Module Week 7 (start on week 8 before next week)

THIS WEEK

- Finish Hashing:
 - Collision Resolution
 - Java Support for Hashing
- Intro to Graphs
 - terminology
 - operations
 - edge representations
- Exam Review
- Exam (THURSDAY NIGHT 5-7pm)

NEXT WEEK

- Exam Results (Tuesday and Wednesday)
- Graphs
 - traversals
 - topological ordering
 - Dijkstra's shortest path
 - Spanning trees

Java API Support for Hashing

hashCode method

- method of Object class *~ reference type only*
- returns an int *-2B → 2Billion*
- default hash code for many objects is computed from object's memory address *except:*

Guidelines for overriding hashCode:

- **must be deterministic**
- **if your class overrides .equals(), it should also override .hashCode()**

not do that

$S_1 = \text{"CAT"}$
 $S_2 = \text{"CAT"}$ } hashCode Integer → Double, String.

Hashtable<K, V> and HashMap<K, V> classes

- in java.util package
- implement Map<K, V> interface
- K type parameter for the key *(not required to be Comparable)*
- V type parameter for the value

Operations

look up / find
V get (K key)*insert / add*
V put (K key, V value)*delete*
boolean remove(K key)*distinguish the duplicate key.*
V remove(K key , V value)

- constructors allow you to set
 - initial capacity (default = 16 for HashMap, 11 for Hashtable)
 - load factor (default = 0.75) *load factor threshold (Pg) 0.7 ~ 0.8*

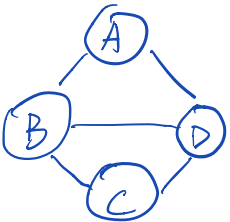

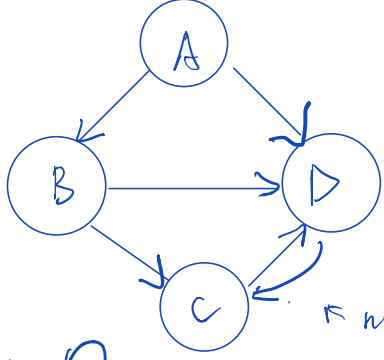

- handle collisions with chained buckets

- HashMap only: allows null key and null values, non-synchronized *processes single uses*

- Hashtable only: no null key – throws exception, synchronized *1.7k/1.8*

Synchronization covered in cs537 (operating systems)

Graph Terminology

<p>Undirected Graph</p>  <p>go either way $A \rightarrow B / B \rightarrow A$</p> <p>No Dup Edges</p> <p>Can have self-edge  eq. print key.</p>	<p>Directed Graph</p>  <p>no duplicate</p> 
<p>vertex</p> <p>degree: # edges for vertex</p> <p>path: seq of connected vertices</p> <p>ABDC length: # edges.</p> <p>ABC</p> <p>cycles: a seq returns to an earlier vertex</p> <p>ABCD A</p>	<p>in-degree: # edges coming in</p> <p>out-degree: --- going out.</p> <p>degree: in-degree + out-degree</p> <p>path: seq of connected vertices</p> <p>respect edge directions</p> <p>ABC</p> <p>ABDC</p> <p>ADC</p> <p>cycles: CDC / DCD (mostly case about cycle with 3+ edges)</p>
<p>Order: # vertices</p>	
<p>Size: # edges</p>	

Implementing Graphs

Graph ADT Operations

constructor(s) — 0 vertices, 0 edges

insert(V vertex) — add a single new vertex (no dupes)

insert(V vertexA, V vertexB) — add an edge btw VA, VB — throw exceptions or add missing vertex

delete(V vertex) — remove a vertex and any edges it is a part of.

delete(V vertexA, V vertexB) — remove an edge from VA to VB — undirected or directed

lookup(V vertex) — return a list of adjacent vertices

lookup(V vertexA, V vertexB) — true if edge exists.

isEmpty() — return true if no vertices + no edges

getSize() — return number of edges (connections).

getOrder() — — — — — vertices

Graph class

```
public class Graph<T> {
    List<Graphnode<T>> vertices;
```

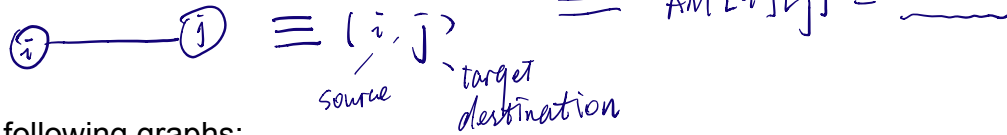
 edges;

Graphnode class

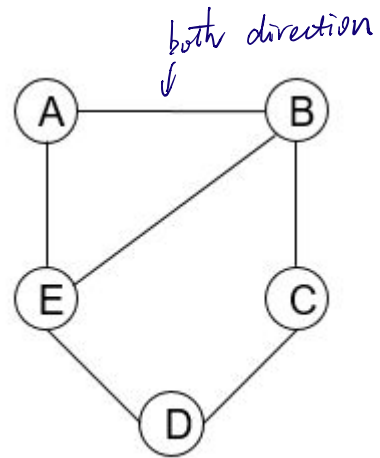
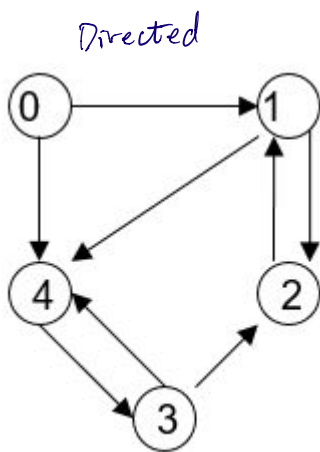
```
class Graphnode<T> {
    private T data; // node level
    private boolean visited;
    private List<Graphnode<T>> neighbors; // adjacent vertices
    private Color color;
```

Representing Edges: Adjacency Matrix

store edge information in 2D array
add array to Graph class
add AM index to graphnode



Given the following graphs:



Show the adjacency matrix representation of the edges for each graph

T → true

col To ↓

rows from →

	0	1	2	3	4
0		T			T
1			T		T
2		T			
3			T		T
4				T	

	A	B	C	D	E
A		T			T
B	T		T		T
C		T		T	
D			T		T
E	T	T		T	

check reflection

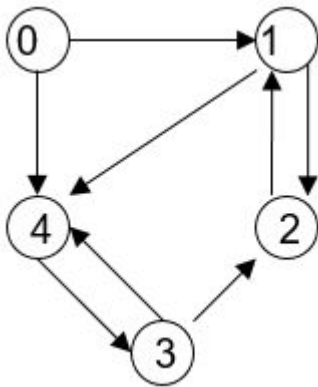
(A.L.)

Representing Edges: Adjacency List

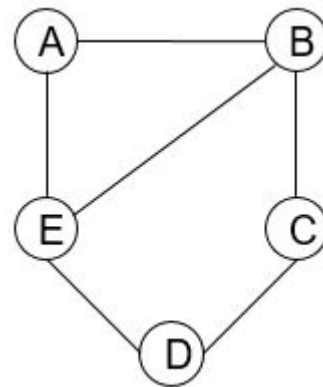
store a list of successors for each vertex
 add A.L. to Graphnode class
 add accessors + mutators

Given the following graphs:

Graph 1



Graph 2



Show the adjacency list representation of the edges for each graph

Graph 1

0	1, 4.
1	2, 4.
2	1
3	2, 4
4	3.

Graph 2

A	B, E
B	A, C, E
C	B, D
D	C, E
E	A, B, D

* CS 400 convention - list in alphanumeric order

Midterm Exam Review

Scantron Example

Last name, first five letters of your first name
 UW PHOTO ID NUMBER
 Lecture number
 Version number

Question Forms

- | | | |
|---------------------------------|---------|-------|
| I. Simple Choice (1pt each) | 12 | } 87. |
| II. Multiple Choice (3pts each) | 57 (x9) | |
| III. Written (3-6pts each) | 18 (x6) | |

Topic reminders: see <https://pages.cs.wisc.edu/~deppeler/cs400/exams> for latest information

- **Command Line Linux (UNIX):** navigating file system (pwd,ls,cat,diff,grep,echo,tree), creating, moving, removing files and directories (cp,mkdir,rmdir), compiling and running Java programs (javac, java, jar), redirecting output to a file (>),
- **Testing:** Black-box testing, White-box testing, Unit testing, JUnit tests, testing exceptions
- **Team Building:** Stages of team development, team roles, team rules, communication and conflict resolution strategies
- **Version Control:** Git commands (init, log, status, clone, add, commit, diff, pull, push), GitHub
- **Balanced Search Trees**
 - **binary search trees (BSTs)**, key value, implementing BSTs, algorithms for BST operations (print, lookup, insert, delete), in-order successor, in-order predecessor, complexities of BST operations, balanced search tree terminology
 - **AVL Trees:** algorithms for operations (print, lookup, insert, delete)
 - **Red-black tree:** red-black tree properties, root property, red property, black property, algorithms for red-black tree operations (print, lookup, insert, delete (for all cases except from black leaf node))
 - **B-Trees:** branching factor, k-ary trees, BTree nodes, 3-Nodes, 4-Nodes, B-Tree algorithms for operations (print, lookup, insert, delete)
- **Hashing:** hashing terminology: key, hash function, Java's hashCode() vs hash index, hash table, load factor, collision, ideal hashing; hash table operations (algorithms & complexities); choosing table size; defining a good hash function, hashing techniques: mod by table size, folding, extraction, weighting, exclusive-OR, bit-shifting, handling non-integer keys using Java's hashCode(); collision handling strategies: open addressing (linear & quadratic probing, double hashing) and buckets (fixed-sized arrays, linked-lists, balanced search trees)

Data Structure Visualizations

This web site is great place to practice your understanding of insert and delete from various data structures: <https://www.cs.usfca.edu/~galles/visualization/Algorithms.html>

Computer Sciences 400

Midterm Exam 1 Information and Sample Questions

The following information and questions are meant to familiarize you with the CS 400 exam instructions and give you a few examples of midterm exam 1 questions. This sample does not represent the length or difficulty of an actual midterm 1 exam. The actual exam will have more questions and cover a broader range of topics than what is shown here. For a list of topics for the exam and more information about CS 400 exams, see the course website.

Solutions to these sample questions will be covered during the lecture before the exam. You are welcome to discuss these questions with classmates, but don't post solutions on Piazza since that makes it harder for students to find their own solutions.

Cover Sheet Information

1. **LAST NAME (Print):** _____ **FIRST NAME (Print):** _____

2. **Circle your Lecture:**

(Deb) Lec 001 9:30 TR

(Deb) Lec 002 2:30 TR

3. **Fill in these fields and their bubbles on the scantron form (use #2 pencil).**

- (a) LAST NAME - write in your last (family) name starting at left column.
- (b) FIRST NAME - write the first five letters of your first (given) name.
- (c) IDENTIFICATION NUMBER - write your UW Student ID number.
- (d) Under ABC of SPECIAL CODES, write your lecture number as a three digit value 001, 002, or 003.
- (e) Under F of SPECIAL CODES, write the letter P and fill in the number (1) bubble.

4. **DOUBLE-CHECK THAT BUBBLES ARE FILLED IN FOR EACH OF ABOVE FIELDS.**

5. **Read, agree to, and sign this ACADEMIC CONDUCT STATEMENT.**

I will keep my answers covered so that they may not be viewed by another student during the exam or prior to completion of their exam. I will not view or in any way use another's work or any unauthorized devices. I understand that I may not make any type of copy of any portion of this exam. I understand that being caught doing any of these or other actions that permit me or another student to submit work that is not our own will result in automatic failure of the course. All such penalties are reported to the Deans Office for all involved.

Signature: _____

Parts	Number of Questions	Question Format	Possible Points
I	24	Simple Choice	24
II	17	Multiple Choice	51
	41	Total	75

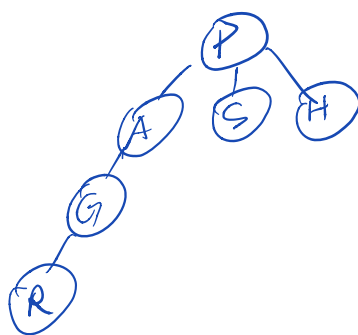
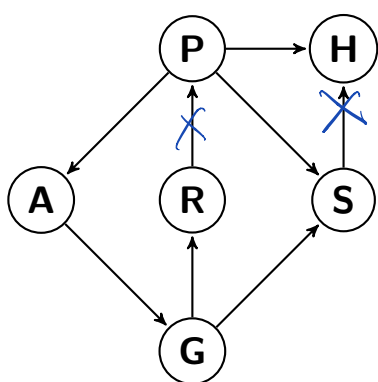
Turn off and put away all electronic devices and wait for the proctor to signal the start of the exam.

Part I: Simple Choice (5 Questions 1 point each)

Mark the corresponding letter on your scantron sheet.

Unless otherwise specified, assume the ADTs, interfaces, classes, and algorithms mentioned are those discussed in lecture and in the readings. Mark the corresponding letter on your scantron sheet.

For the next three questions, consider the following graph with nodes containing character labels:



pre-order: P A G R S H
 post-order: R G A S H P
 in-order: ~~X~~ BT: ✓
 level-order: P A S H G R
 height: 4

For each of the following, is the statement true or false about the above graph?

Mark **A** for true and **B** for false.

1. The graph is *directed*. ☒ A. true B. false (1)
2. The graph is *acyclic*. A. true ☒ B. false (1)
3. The graph is *weakly connected*. ☒ A. true B. false (1)
4. A *pre-order traversal* on a **general tree** visits the nodes in the same order as: (1)
 - A. a *depth-first search* starting at the root of that tree. level-order
 - ☒ B. a *breadth-first search* starting at the root of that tree. pre-order-
5. Which of the following has a lower complexity to sort an array of N objects that *is already in sorted order*? (1)
 - ☒ A. merge sort $O(N \log N)$
 - B. quick sort using the first item in each part as the pivot $O(N^2)$ worst case

Part II: Multiple Choice (6 Questions, 3 points each)

For questions 6 through 11, choose the best answer after reading all of the choices. Mark the corresponding letter on your answer sheet.

6. Which one of the following abstract data types *must* have a linear structure? (3)

~~A. binary search tree~~ ~~B. graph~~ C. hash table ~~D. heap~~ E. stack

7. Consider implementing a hash table to be used to store 20-digit integer keys. Which one of the following **hash functions** is *most likely* to return different hash indexes for two different keys? (3)

- ~~A. A hash function based on the first 5 digits of the key.~~
~~B. A hash function based on the last 5 digits of the key.~~
 C. A hash function based on the product of the digits of the key.
~~D. A hash function based on the sum of the digits of the key.~~
E. A hash function based on the sum of two numbers: one number contains the first 10 digits of the key and the second number contains the last 10 digits of the key.

8. Assume *linear probing* collision resolution is used on a **hash table** of size 11 having a hash function that is simply the key mod by the table's size. Which one of the following shows the hash table that results from inserting these keys in the order given: 26, 13, 6, 15, 19, 4, 16, 22 (3)

4 2 6 4 8 4 5 0

~~A.~~

22		13		4	16	6	26	19	15	
----	--	----	--	---	----	---	----	----	----	--

B.

22		13		26	15	6	4	19	16	
----	--	----	--	----	----	---	---	----	----	--

~~C.~~

22		13		4	16	6		19		
----	--	----	--	---	----	---	--	----	--	--

~~D.~~

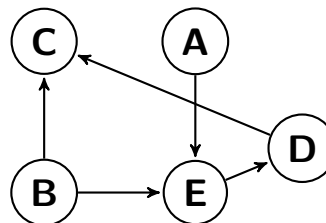
26	13	6	15	19	4	16	22			
----	----	---	----	----	---	----	----	--	--	--

E.

22		13	4	26	15	6		19	16	
----	--	----	---	----	----	---	--	----	----	--

9. Consider the following ordered lists of nodes and the **directed graph**: (3)

- i A B C D E
ii A B D C E
 iii A B E D C



Which of following lists the nodes in *topological order* for the graph above?

- A. *i* only
 B. *ii* only
 C. *iii* only
 D. *ii* and *iii* only
 E. *i*, *ii*, *iii*

10. Assume that you have the following *incomplete* definitions of **Graph** and **Graphnode** classes: (3)

```
class Graphnode<T> {
    private T data;
    private int index ; // for indexing into edge matrix or nodes array
    public T getData() { return data ; }
    public int getIndex() { return index; }
}
```

Consider writing the following method which calculates the *in-degree* of a given node **n**.
An incomplete version of the method is as follows:

```
private int inDegree(Graphnode<T> n) {
    int nodeNum = n.getIndex(); // for indexing into edge matrix
    int deg = 0;
    for (int i=0; i < nodes.length; i++ )
        STMT
    return deg;
}
```

Which of the following replacements for STMT correctly completes this method?

- A. if (edge[i][nodeNum]) deg++;
 B. if (edge[nodeNum][i]) deg++;
 C. if (edge[i][nodeNum]) deg++;
 if (edge[nodeNum][i]) deg++;
 D. if (edge[i][nodeNum] || edge[nodeNum][i]) deg++;
 E. if (edge[i][nodeNum] && edge[nodeNum][i]) deg++;

11. Given the following array: *0 8 3 5 9 7 1 0 6 4 2* (3)

8	3	5	9	7	1	0	6	4	2
---	---	---	---	---	---	---	---	---	---

Which one of the following represents what the array looks like after three passes (i.e., three iterations of the outer loop) of the **bubble sort** algorithm as covered in lecture?

- A.

0	1	2	3	5	4	6	7	8	9
---	---	---	---	---	---	---	---	---	---

 B.

0	1	2	8	3	5	7	9	4	6
---	---	---	---	---	---	---	---	---	---

 C.

0	1	2	8	3	5	9	7	4	6
---	---	---	---	---	---	---	---	---	---

 D.

0	1	2	8	3	5	9	7	6	4
---	---	---	---	---	---	---	---	---	---

 E.

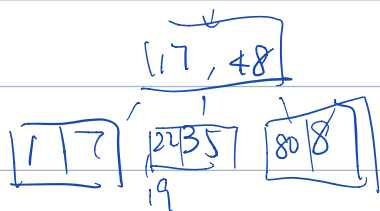
0	1	2	9	7	3	8	6	4	5
---	---	---	---	---	---	---	---	---	---

48, 35, 87, 17, 7, 1, 22, 80, 9, 97.

$\boxed{35} \boxed{48} \boxed{+87}$

48

7 $\boxed{17} \boxed{35}$ 87



→

