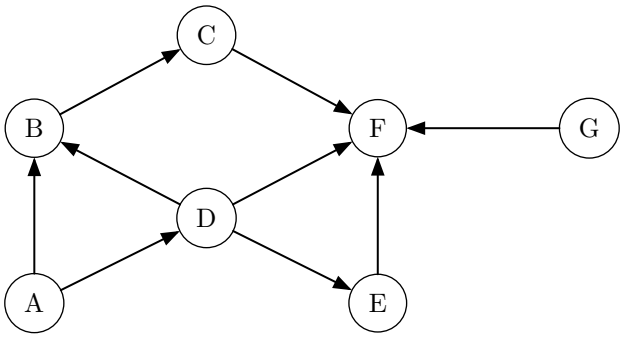
CS 61BL Summer 2026

Graphs, Hashing, and Heaps

Exam Prep 03: July 24, 2026

1 Graphs and Traversals

For the following graph…



(a) Write out the adjacency matrix and adjacency list.

(b) Write out the order in which nodes are visited by DFS pre-order and post-order, starting from A. Tiebreak by alphabetical order, where nodes that come earlier in the alphabet are higher priority.

(c) Give a valid topological sort of the graph.

Answer the following questions as either **True** or **False** and provide a brief explanation.

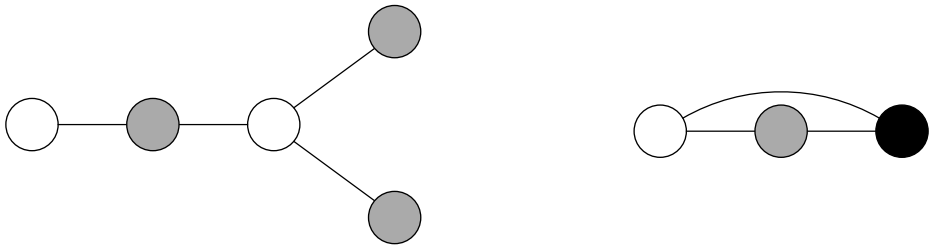
(d) If a graph with n vertices has n − 1 edges, it must be a tree.

(e) Every edge is looked at exactly twice in every iteration of DFS on a connected, undirected graph.

(f) In BFS, let d(v) be the minimum number of edges between a vertex v and the start vertex. For any two vertices u, v in the fringe, $|d(u) - d(v)|$ is always less than 2.

2 Painting Graphs

An undirected graph is said to be bipartite if each of its vertices can be painted with one of the two colors, such that every edge connects two vertices of different colors. For example below, the graph on the left is bipartite, whereas on the graph on the right is not.



Provide an algorithm which determines whether or not a graph is bipartite. What is the runtime of your algorithm?

Hint: Can you modify an algorithm we already know?

3 A Side of Hash Browns

We want to map food items to their yumminess. We want to be able to find this information in constant time, so we've decided to use Java's built-in **`HashMap`** class! Here, the key is an **`String`** representing the food item and the value is an **`int`** yumminess rating.

For simplicity, let's say that here a **`String`**'s hashcode is the first letter's position in the alphabet (A = 0, B = 1... Z = 25). For example, the **`String "Hashbrowns"`** starts with "H", and "H" is 7th letter in the alphabet (0 indexed), so the hashCode would be 7. Note that in reality, a **`String`** has a much more complicated **`hashCode`** implementation.

Our **`HashMap`** will compute the index as the key's hashcode value modulo the number of buckets in our **`HashMap`**. Assume the initial size is 4 buckets, and we double the size of our **`HashMap`** as soon as the load factor reaches 3/4. If we try to put in a duplicate key, simply replace the value associated with that key with the new value.

(a) Draw what the **`HashMap`** would look like after the following operations.

```
HashMap<String, Integer> hm = new HashMap<>();
hm.put("Hashbrowns", 7);
hm.put("Dim sum", 10);
hm.put("Escargot", 5);
hm.put("Brown bananas", 1);
hm.put("Burritos", 2);
hm.put("Buffalo wings", 8);
hm.put("Banh mi", 9);
hm.put("Burritos", 10);
```

(b) Do you see a potential problem here with the behavior of our **`HashMap`**? How could we solve this?

4 Hashing

(a) Here are five potential implementations of the `Integer` class's `hashCode()` method. Categorize each as having (1) Good spread, (2) Bad spread, and (3) No spread.

For the 2nd implementation, note that ***`intValue()`*** *will return that* ***`Integer`****'s number value as an* ***`int`****.*

```
public int hashCode() {
    return -1;
}
```

```
public int hashCode() {
    return super.hashCode(); // Object's hashCode() is based on memory location
}
```

```
public int hashCode() {
    return intValue() * intValue() * 2;
}
```

```
public int hashCode() {
    return (int) (new Date()).getTime(); // returns the current time as an int
}
```

(b) For each of the following questions, answer **Always**, **Sometimes**, or **Never**.

1. If you were able to modify a key that has been inserted into a `HashMap` would you be able to retrieve that entry again later? For example, let us suppose you were mapping ID numbers to student names, and you did a `put(303, "Dawn")` operation. Now, let us suppose we somehow went to that item in our `HashMap` and manually changed the key to be 304. If we later do `get(304)`, will we be able to find and return `"Dawn"`? Explain.

2. When you modify a value that has been inserted into a `HashMap` will you be able to retrieve that entry again? For example, in the above scenario, suppose we first inserted `put(303, "Dawn")` and then changed that item's value from `"Dawn"` to `"Michelle"`. If we later do `get(303)`, will we be able to find and return `"Michelle"`? Explain.

5 Absolutely Valuable Heaps

(a) Assume that we have a binary min-heap (smallest value on top) data structure called **`MinHeap`** that has properly implemented the **`insert`** and **`removeMin`** methods. Draw the heap and its corresponding array representation after each of the operations below:

```
MinHeap<Character> h = new MinHeap<>();
h.insert('f');
h.insert('h');
h.insert('d');
h.insert('b');
h.insert('c');
h.removeMin();
h.removeMin();
```

(b) Your friendly TA Benjamin challenges you to create an integer max-heap without writing a whole new data structure. Can you use your min-heap to mimic the behavior of a max-heap? Specifically, we want to be able to get the largest item in the heap in constant time, and add things to the heap in $\Theta(\log n)$ time, as a normal max heap should.

Hint: You should treat the **`MinHeap`** as a black box and think about how you should modify the arguments/ return values of the heap functions.