CS 61B
Fall 2025

MSTs, DAGs, and Hashing

Exam-Level Discussion 09: October 27, 2025

1 Multiple MSTs

Recall a graph can have multiple MSTs if there are multiple spanning trees of minimum weight.

(a) For each subpart below, select the correct option and justify your answer. If you select "never" or "always," provide a short explanation. If you select "sometimes," provide two graphs that fulfill the given properties.

1. If **some** of the edge weights are **identical**, there will

   ◯ never be multiple MSTs in G.

   ◯ sometimes be multiple MSTs in G.

   ◯ always be multiple MSTs in G.

   Justification:

2. If **all** of the edge weights are **identical**, there will

   ◯ never be multiple MSTs in G.

   ◯ sometimes be multiple MSTs in G.

   ◯ always be multiple MSTs in G.

   Justification:

(b) Suppose we have a connected, undirected graph (G) with (N) vertices and (N) edges, where all the **edge weights are identical**. Find the maximum and minimum number of MSTs in (G) and explain your reasoning.

Minimum:

Maximum:

Justification:

(c) It is possible that Prim's and Kruskal's find different MSTs on the same graph G (as an added exercise, construct a graph where this is the case!).

Given any graph G with integer edge weights, modify the edge weights of G to *ensure* that

(1) Prim's and Kruskal's will output the same results, and

(2) the output edges still form a MST correctly in the original graph.

You may not modify Prim's or Kruskal's, and you may not add or remove any nodes/edges.

Hint: Look at subpart 1 of part (a).

2 Class Enrollment

You're planning your CS classes for the upcoming semesters, but it's hard to keep track of all the prerequisites! Let's figure out a valid ordering of the classes you're interested in. A valid ordering is an ordering of classes such that every prerequisite of a class is taken before the class itself. Assume we're taking one CS class per semester.

(a) The list of prerequisites for each course is given below (not necessarily accurate to actual courses!). Draw a graph to represent our scenario.

- CS 61A: None
- CS 61B: CS 61A
- CS 61C: CS 61B
- CS 70: None
- CS 170: CS 61B, CS 70
- CS 161: CS 61C, CS 70

(b) Suppose we added a new prerequisite where the student must take CS 161 before CS 170 and CS 170 before CS 61C. Is there still a valid ordering of classes such that no prerequisites are broken? If no, explain.

(c) With the original graph, perform a topological sort to find a valid ordering of the 6 classes. Break ties by going to the lower course number first.

3 Hashing Gone Crazy

For this question, use the following `TA` class for reference.

```
public class TA {
    int semester;
    String name;
    TA(String name, int semester) {
        this.name = name;
        this.semester = semester;
    }
    @Override
    public boolean equals(Object o) {
        TA other = (TA) o;
        return other.name.charAt(0) == this.name.charAt(0);
    }
    @Override
    public int hashCode() { return semester; }
}
```

Assume that the `ECHashMap` is a `HashMap` implemented with **external chaining** as depicted in lecture. The `ECHashMap` instance begins with 4 buckets.

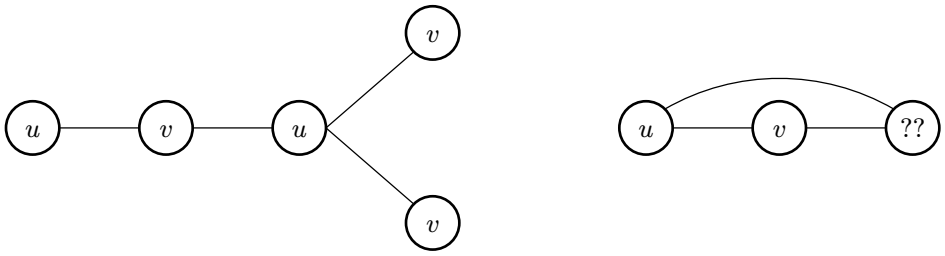
Resizing Behavior If an insertion causes the load factor to reach or exceed 1, we resize by doubling the number of buckets. During resizing, we traverse the linked list that correspond to bucket 0 to rehash items one by one, and then traverse bucket 1, bucket 2, and so on. Duplicates are **not** checked when rehashing into new buckets.

Draw the contents of map after the executing the insertions below:

```
ECHashMap<TA, Integer> map = new ECHashMap<>();
TA jasmine = new TA("Jasmine the GOAT", 10);
TA noah = new TA("Noah", 20);
map.put(jasmine, 1);
map.put(noah, 2);

noah.semester += 2;
map.put(noah, 3);

jasmine.name = "Nasmine";
map.put(noah, 4);

jasmine.semester += 2;
map.put(jasmine, 5);

jasmine.name = "Jasmine";
TA cheeseguy = new TA("Sam", 24);
map.put(cheeseguy, 6);
```

4 Extra: Graph Algorithm Design

(a) An undirected graph is said to be bipartite if all of its vertices can be divided into two disjoint sets U and V such that every edge connects an item in U to an item in V. 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 (ie. graph traversal)?



(b) Consider the following implementation of DFS, which contains a crucial error:

```
create the fringe, which is an empty Stack
push the start vertex onto the fringe and mark it
while the fringe is not empty:
    pop a vertex off the fringe and visit it
    for each neighbor of the vertex:
        if neighbor not marked:
            push neighbor onto the fringe
            mark neighbor
```

First, identify the bug in this implementation. Then, give an example of a graph where this algorithm may not traverse in DFS order.

(c) *Extra:* Provide an algorithm that finds the shortest cycle (in terms of the number of edges used) in a directed graph in $O(EV)$ time and $O(E)$ space, assuming $E > V$.

5 Extra: Buggy Hash

The following classes may contain a bug in one of its methods. Identify those errors and briefly explain why they are incorrect and in which situations would the bug cause problems.

(a) The `Timezone` class below:

```
class Timezone {
    String timeZone; // "PST", "EST" etc.
    boolean dayLight;
    String location;
    ...
    public int currentTime() {
        // return the current time in that time zone
    }

    public int hashCode() {
        return currentTime();
    }

    public boolean equals(Object o) {
        Timezone tz = (Timezone) o;
        return tz.timeZone.equals(timeZone);
    }
}
```

(b) The `Course` class below:

```
class Course {
    int courseCode;
    int yearOffered;
    String[] staff;
    ...
    public int hashCode() {
        return yearOffered + courseCode;
    }

    public boolean equals(Object o) {
        Course c = (Course) o;
        return c.courseCode == courseCode;
    }
}
```