CS 61B Spring 2026

Heaps, Tree Traversals, and Graphs

Exam-Level Discussion 09: March 16, 2025

1 Heaps and Graphs Summary

Heap. A min-heap is a complete binary tree where every node is smaller than its children. When used to implement a Priority Queue, then operations are as follows:

- **Add:** Add to the end of the array, then *swim up* (repeatedly swap with parent while smaller). Runtime: $O(\log N)$.
- **Remove Smallest:** Swap root with last element, remove last, then *sink down* (repeatedly swap with the smaller child while larger). Runtime: $O(\log N)$.

Tree Traversals. Suppose we take some action on every node of a tree, e.g. printing. Suppose current node is v, left subtree L, and right subtree R:

Pre-order:	action(v), L, R
In-order:	L, action(v), R
Post-order:	L, R, action(v)
Level-order:	Top to bottom, left to right

Graphs. A graph $G = (V, E)$ consists of vertices V and edges E. Representations:

- **Adjacency list:** For each vertex, store a list of its neighbors. Space: $\Theta(V + E)$.
- **Adjacency matrix:** $V \times V$ grid; entry $(i, j) = 1$ if edge exists. Space: $\Theta(V^2)$.

Graph Traversals.

- **DFS (Depth-First Search):** Recursively explore vertices by going as deep as possible before backtracking. Use a `marked` array to avoid infinite recursion. Runtime: $\Theta(V + E)$.

```
dfs(Graph G, int v) {
  marked[v] = true;                  // LINE 1
  for (int w : G.adj(v)) {
      if (!marked[w]) {              // LINE 2
          edgeTo[w] = v;
          dfs(G, w);
      }
  }
```

  - DFS Pre-order: The order in which DFS calls are made.
  - DFS Post-order: The order in which DFS calls complete.
- **BFS (Breadth-First Search):** Visit vertices in order of number of edges from source. Uses a `Queue`. Runtime: $\Theta(V + E)$. Covered on next worksheet.
- **Dijkstra's (Best-First Search):** Visit vertices in order of total distance from source (based on edge weights). Uses a `Priority Queue`. Runtime: $\Theta((V + E) \log V)$. Covered on next worksheet.

2 Heap Mystery

We are given the following array representing a min-heap where each letter represents a **unique** number. Assume the root of the min-heap is at index one, i.e. `A` is the root. Our task is to figure out the numeric ordering of the letters. Therefore, there is **no** significance of the alphabetical ordering. i.e. just because B precedes C in the alphabet, we do not know if B is less than or greater than C.

Array: [-, A, B, C, D, E, F, G]

Four unknown operations are then executed on the min-heap. An operation is either a `removeMin` or an `insert`. The resulting state of the min-heap is shown below.

Array: [-, A, E, B, D, X, F, G]

(a) Determine the operations executed and their appropriate order. The first operation has already been filled in for you!

Hint: Which elements are gone? Which elements are newly added? Which elements are removed and then added back?

1. `removeMin()`
2. _____
3. _____
4. _____

(b) Fill in the following comparisons with either >, <, or ? if unknown. We recommend considering which elements were compared to reach the final array.

1. X _____ D
2. X _____ C
3. B _____ C
4. G _____ X

3 Graph Conceptuals

(a) Answer the following questions as either **True** or **False** and provide a brief explanation:

1. If a graph with n vertices has $n-1$ edges, it **must** be a tree.

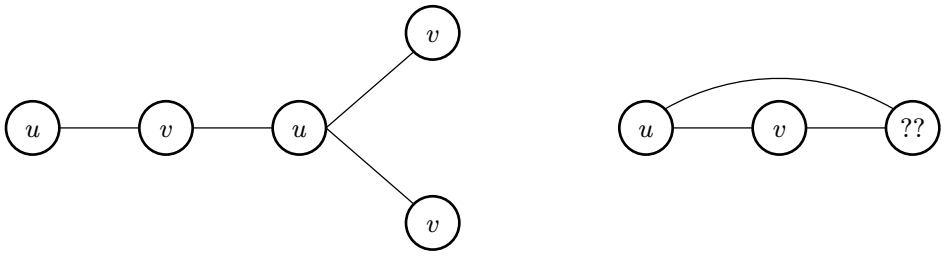
2. Every edge is looked at exactly twice in each full run of DFS on a connected, undirected graph.

3. 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 (recall that the fringe in BFS is a queue), $|d(u) - d(v)|$ is **always less than** 2.

(b) Given an undirected graph, provide an algorithm that returns true if a cycle exists in the graph, and false otherwise. Also, provide a Θ bound for the worst case runtime of your algorithm.

4 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 $E > V$.