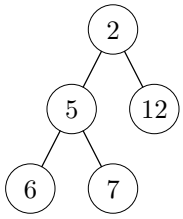
CS 61B Fall 2026

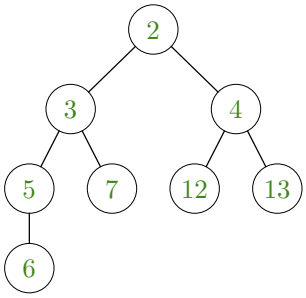
Heaps, Graphs

Discussion 08: October 19, 2026

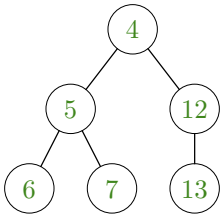
1 Heaperations



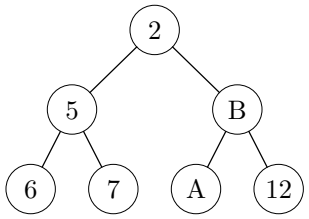
(a) Draw the heap after inserting 4, 13, and 3 in that order.



(b) Now, draw the heap after 2 calls of `removeMin`.



(c) Suppose we recover the original heap and insert two elements of unknown value, A and B, **in that order.**



For each of the following conditional statements, write whether they are **always** true, **sometimes** true, or **never** true, and briefly explain why.

$A < B$

Never. The min-heap property states that a node's children must be greater than or equal to itself. $A \geq B$ implies that A cannot be less than B.

$B > 5$

Sometimes. Without any other information, a node's position relative to other nodes at the same depth does not give us any information about the node's value.

$A \geq 12$

Always. If A were less than 12, then it would have bubbled up to a depth of 1, with 12 as a left child. There are two cases for what happens next:

- If B is less than A, then B would swap with A, but 12 would remain as a left child. This is not the case.
- If B is greater than or equal to A, then A would remain at depth 1. This is not the case in the given heap.

(d) Fill in the following blanks related to min-heaps. Let N be the number of elements in the min-heap. For the entirety of this question, assume the elements in the min-heap are distinct.

`removeMin` has a best-case runtime of

$\Theta(1)$

and a worst-case runtime of

$\Theta(\log N)$

`insert` has a best-case runtime of

$\Theta(1)$

and a worst-case runtime of

$\Theta(\log N)$

A

DFS preorder

or

BFS

traversal on a min-heap may output the elements in sorted order. *(Assume there are at least 3 elements in the min-heap)*

The smallest item of a min heap is at the top, so whatever traversal we choose must output the top element first in a complete binary tree. Only preorder and level-order have this property.

The fourth smallest element in a min-heap with 1000 distinct elements can appear in

14

different places in the heap.

The 4th smallest item can be on the 2nd, 3rd, or 4th level of the heap (1-indexed)

Given a min-heap with 2^{N-1} **distinct** elements, for an element

- to be on the second level it must be less than

  $2^{N-1} - 2$

  element(s) and greater than

  1

  element(s).
- to be on the bottommost level it must be less than

  0

  element(s) and greater than

  $N - 1$

  element(s).

An element on the second level must be larger than the root and less than the elements in its subtree. There are $2^{N-1} - 2$ elements in the subtree of an element on the second level: half the elements in the tree minus the root, then subtracting off the node itself.

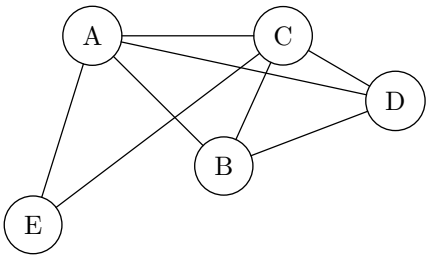
An element on the bottom level must be greater than all elements on the path from itself to the root. A min heap with 2^{N-1} elements has N levels, so there are $N - 1$ items above it on a path to the root.

Hint: A complete binary tree (with a full last-level) has $2^N - 1$ *elements, with N being the number of levels.*

2 Write Your Representative

For each scenario below, justify whether an **adjacency list** or **adjacency matrix** would make more sense to represent the graph. Then, draw your chosen representation for the sample graph provided.

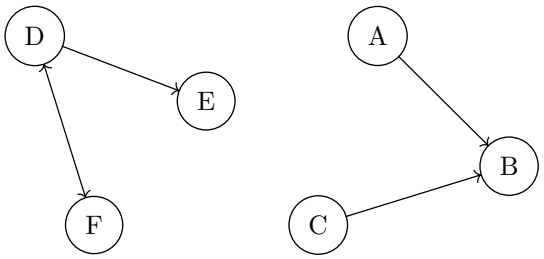
(a) A transit agency wants to use a graph to represent connections between bus stops in a **densely connected** bus system. The agency wants to be able to check if two stations are linked **as quickly as possible**.



An adjacency matrix would provide the quick station connection lookups that the agency wants.

	A	B	C	D	E
A	0	1	1	1	1
B	1	0	1	1	0
C	1	1	0	1	1
D	1	1	1	0	0
E	1	0	1	0	0

(b) A social media platform wants to represent which users are following other users using a directed graph. The graph is relatively **sparse**, and the platform values **memory efficiency** over all, because the site has a lot of users.

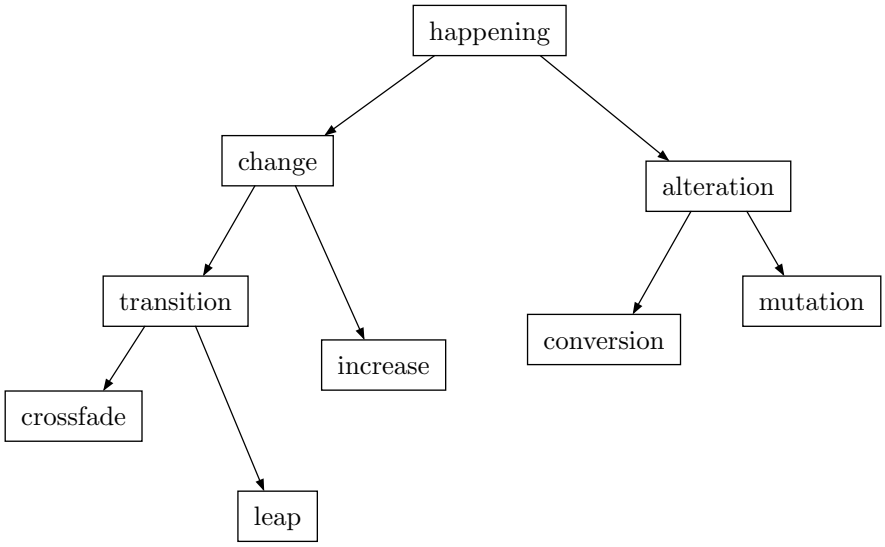


An adjacency list is more memory-efficient in this case.

A -> B
B
C -> A
D -> E F
E
F -> D

3 K Equals Zero

Consider the simplified hyponym graph below.



(a) Write the order in which hyponyms would be returned if we run **DFS preorder** on **change**. Break ties alphabetically.

change, increase, transition, crossfade, leap

(b) Write the order in which hyponyms would be returned if we run **DFS postorder** on **change**. Break ties alphabetically.

increase, crossfade, leap, transition, change

(c) Write the order in which hyponyms would be returned if we run **BFS** on **happening**. Break ties alphabetically.

happening, alteration, change, conversion, mutation, increase, transition, crossfade, leap

(d) As you may know, in Project 4, our hyponym graph is guaranteed to be directed and acyclic. If we changed all edges in the above graph to be undirected instead, would any of your answers change? Why?

Yes, our answers for DFS preorder and DFS postorder would change. Since the edge from **happening** to **change** is now bidirectional, **happening** could be reached from the start node, and the traversal would eventually print every node in the graph.

The BFS traversal would remain the same, since every besides **happening** is a child of **happening**.

4 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, and the graph is connected, it **must** be a tree.

   Solution: True. First we remind ourselves that a tree is just a graph without any cycles. Then we consider what a graph with n vertices and n - 1 edges looks like. As an example, suppose we have 5 nodes that are initially disconnected. Suppose we add 4 edges, connecting a new node each time. At this point the graph is fully connected. However there are no cycles (since we can only get a cycle by adding edge between two vertices that are already connected).

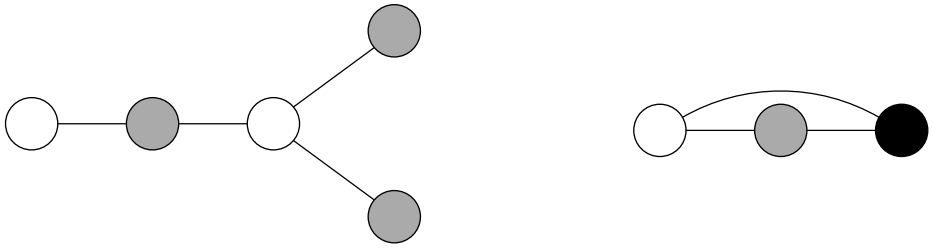
2. Every edge is looked at exactly twice in each full run of DFS on a connected, undirected graph.

   Solution: True. Say an edge connects u and v. Both u and v will look at the other one through this edge when it's their turn.

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.

   Solution: True. Suppose this was not the case. Then, we could have a vertex 2 edges away and a vertex 4 edges away in the fringe at the same time. But, the only way to have a vertex 4 edges away is if a vertex 3 edges away was removed from the fringe. We see this could never occur because the vertex 2 edges away would be removed before the vertex 3 edges away!

(b) 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?

To solve this problem, we run a special version of a traversal from any vertex. This can be implemented using either DFS and BFS as the underlying traversal that we will modify. Our special version marks the start vertex as "white", then each of its neighbors "red", and each of their neighbors as "white", and so forth. If at any point in the traversal we want to mark a node as white but it is already marked with as red (or vice versa), then the graph is not bipartite.

If the graph is not connected, we repeat this process for each connected component. If the algorithm completes, successfully marking every vertex in the graph, then it is bipartite.

The runtime of the algorithm is the same whether you use BFS or DFS: $\Theta(E+V)$.