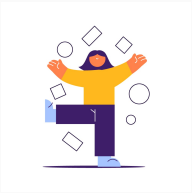
Lec 33: Graph Implementations

May 13, 2026

Admin



- Practice final posted (more details next slide)
- Lab today
- Review Friday; come with questions!
- Any questions?

Final/Practice Final

Practice Final

- Posted on Glow
- In 3 parts (mirrors the final; I'll discuss in a minute) Some questions could use a little improvement.
    - I think they're in the ballpark at least
    - Let me know if you find significant issues
    - Hopefully will not carry over to the final—I took the better setups for the real finals
- Solutions posted soon

Format of the Final

- 150 minutes total; time per question a bit less than on midterms
- More short/medium questions per long question
- In *three parts*
  - If you need to leave the room (go to the bathroom, whatever), turn in the previous part before leaving. When you come back you'll get the next part
  - Can take a few minutes to yourself, walk around, etc.
  - Or can just start on the next part right away
- Lida and I will give a reminder when you're roughly 1/3 or 2/3 through

Implementing Graphs

Graph Implementation

- Two methods for storing a graph:
  - Adjacency List (we'll use this one in the lab; similar to linked list/tree)
  - Adjacency Matrix (different approach; closer to an array/arraylist)
- Let's discuss both, and the tradeoffs between them

What do we want out of a Graph implementation?

- Each vertex has a *label*
  - Could be just a letter or number (like in our examples so far)
  - Name of a city
  - A word in the `WordLadder` lab
- Our graph operations should use *labels*
  - Example: get me the vertex with a given label
- Can we store a graph like a linked list/tree?
  - Data structure stores some metadata
  - Nodes store the structure itself

Adjacency List Representation

- Store a class `Vertex<E>`
  - Generic type `E` stores the type of the vertex label
- Each `Vertex<E>` stores a list of its neighbors
  - We'll use an `ArrayList` in the lab, though a `LinkedList` is more common
- What is the type of the `ArrayList`?
  - It stores a list of `Vertex<E>`. So it is an `ArrayList<Vertex<E>>`

Adjacency List Representation

`Vertex<E>`:

- Methods to get the name of the vertex
- Method to get all neighbors of a vertex
- Keep a `boolean visited` for BFS
- Some getter/setter methods, including a method to add a new neighbor

Adjacency List Representation: Graph class

- How can we add a new edge in our graph, given the *labels* of two vertices in our graph?
- Need a way to get from `E label` to the `Vertex<E>` that has the matching label.
- What data structure should we use?
  - A hash table! Keys are the labels; value of a key is the `Vertex<E>` with that label
  - Keys of the hash table have type `E`, values have type `Vertex<E>`
- To add an edge, look up each `Vertex<E>` using the hash table. Add each to the others' neighbor list

Adjacency List Representation: Graph Class

- How do we add a new vertex?
    - Create the `Vertex<E>`; add an entry to the hash table
- How do we determine if a vertex of a given label exists in the graph?
    - Check if a key with the label exists in the hash table

When Does an Adjacency List Fall Short?

- How do we check if a given edge (between a vertex with label `label1` and another with label `label2`) is contained in the graph?
- Use the hash table to look up the vertex with label `label1`. Check through all of its neighbors to see if `label2` is one of them.
- How much time does this take?
  - $O(\#\text{neighbors})$
  - This *can* be as bad as $O(n)$
- Takeaway: adjacency list is very slow if you want to check if a given edge exists, and there are many edges in the graph (vertices have many neighbors)

Adjacency Matrix Representation

- Let's store a graph all in one place!
- An $n \times n$ matrix of integers, we'll call it `adj`
- Entry $[i][j]$ has a 1 if the ith vertex shares an edge with the jth verted; 0 otherwise
- Let's do an example on the board

Adjacency Matrix Representation

- How do we keep track of vertex labels?
  - Need to map each vertex label to a row in the graph
  - A hash map from `E` to `int`!
  - We'll call it `indexMap`
- On the board: how do we tell if there is an edge between `label1` and `label2`?
- How do we get all neighbors of `label1`? How long does this take? Answer: $O(n)$ where n is the number of vertices

Adjacency Matrix Representation

- How much space does this take for n vertices and m edges?
- $O(n^2)$ space! (Can be a lot) How much space does the adjacency *list* take?
- Each vertex is stored once; each edge is stored *twice*
- $O(n + m)$ space

Adjacency Matrix vs Adjacency List

Adjacency List:

- $O(n + m)$ space
- Find neighbors of vertex in $O(\#\text{neighbors})$ time
- Finding if two vertices share an edge takes $O(\#\text{neighbors})$ time

Adjacency Matrix:

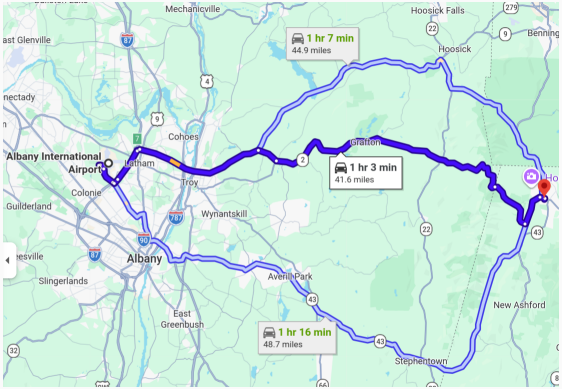
- $O(n^2)$ space
- Find neighbors of a vertex in $O(n)$ time
- Finding if two vertices share an edge takes $O(1)$ time

Adjacency List vs Adjacency Matrix

- Adjacency list is often better
  - The best choice if we are doing BFS on the graph
- Adjacency Matrix works well when:
  - We are OK with n^2 space, or m is close to n^2
  - We want to do lots of queries asking if two vertices share an edge, and we don't care much about neighbors

Dijkstra’s Algorithm

Shortest Paths in Real-World Graphs



- Nodes: intersections; edges: roads connecting intersections
- Does BFS give us the answer we want here?
- *No!* We want the minimum time (or perhaps distance); not the minimum number of roads.

Graphs with Edge Weights

- We can assign *weights* to each edge of a graph
- For roads: the weight of the road is the average travel time along that road
- Shortest path: given vertices `start` and `end`, find the sequences of edges with *smallest total weight* from `start` to `end`
- In pairs: can you come up with a graph where BFS does not give us the answer we want here?

Dijkstra's Algorithm

- Goal: a shortest path algorithm that takes edge weights into account
- Basic idea: like BFS, but rather than exploring the first vertex added, we instead explore the vertex in the data structure with shortest path
- For BFS, we used a Queue: First in/First out. What data structure gets the *smallest* out?
  - A priority queue!
  - Dijktra's algorithm: works exactly like BFS, but with a priority queue instead of a queue
  - One of the main priority queue applications!
- Let's look at an example, and then look a little bit at the code

SCS Forms