Graph Implementations II

Instructors: Sam McCauley and Dan Barowy

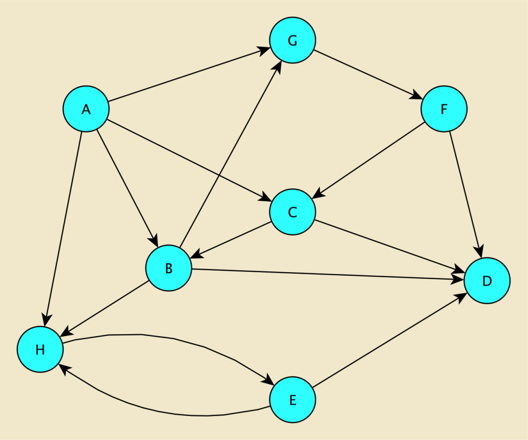
May 7, 2022

Admin

- Final review next Friday (no quiz!)
- If you have an “exam hardship” let me know as soon as possible
- Talk today at 2:35pm in Wege
  - On equity of access in algorithms
- Any questions?

Adjacency Matrix Representation

Adjacency Matrix



	A	B	C	D	E	F	G	H
A	0	1	1	0	0	0	1	1
B	0	0	0	1	0	0	1	1
C	0	1	0	1	0	0	0	0
D	0	0	0	0	0	0	0	0
E	0	0	0	1	0	0	0	1
F	0	0	1	1	0	0	0	0
G	0	0	0	0	0	1	0	0
H	0	0	0	0	1	0	0	0

If there's an `Edge` between *i* and *j*, Entry(i,j) stores it. Else, Entry(i,j) stores `null`.

(We use `1` in the picture, but in reality it will be a reference to some `Edge` object)

How to use the adjacency matrix

- How can we find the neighbors of a vertex v?
- Go to corresponding row of matrix
- Scan through the row. Each time we see a non-null `Edge` e, look at the two vertices of e. The non-v vertex is a neighbor!
- Let's look at the code for `Edge`, and the node for `neighbors()` in `GraphMatrix`

Making the adjacency matrix work

- How can I look up a vertex in the matrix?
- We look up by label, but we need a specific *row* in the matrix
- Each `GraphMatrixVertex` object stores its own index for its row (in addition to label, visited, etc.)
- How can we get the `GraphMatrixVertex` object that corresponds to a given label (of type `V`)?
- Answer: a hash table!

Maintaining rows

- Let's say we add a new vertex. What row should it be assigned? How can we keep track of that?
- One option: keep track of how many vertices there are. Assign any new vertices to the next empty row.
- What about deletes? Those cause an issue.
- Solution: keep track of unused rows in a `List`
    - Specifically, a `SinglyLinkedList`
    - Called `freeList`
    - Adding a new row, and removing the first vertex, are both $O(1)$.

Graph Matrix Classes

- `GraphMatrixVertex` and `Vertex`: classes for holding vertices
- `Edge`: class for holding edges
- `GraphMatrix`: abstract class for graphs stored using adjacency matrix
- `GraphMatrixDirected` and `GraphMatrixUndirected`: any remaining methods (that differ between directed and undirected graphs)
- Let’s take a look!

Analyzing Adjacency Matrix Representation

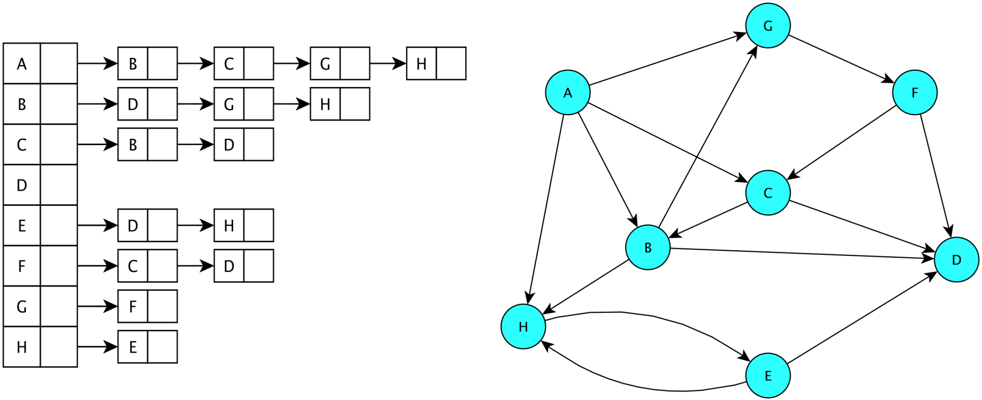
- Let's say we have a graph with n vertices and m edges
- How long does it take to find all neighbors of a vertex?
  - $O(n)$ (need to scan through all columns—coresponding to all vertices)
- How long does it take to find the edge between vertices v_1 and v_2? To add a new edge between two vertices?
  - $O(1)$! Just need to look it up in the matrix
- Space?
  - $O(n^2)$ (Can be very large!)

Adjacency List Representation

Adjacency Lists

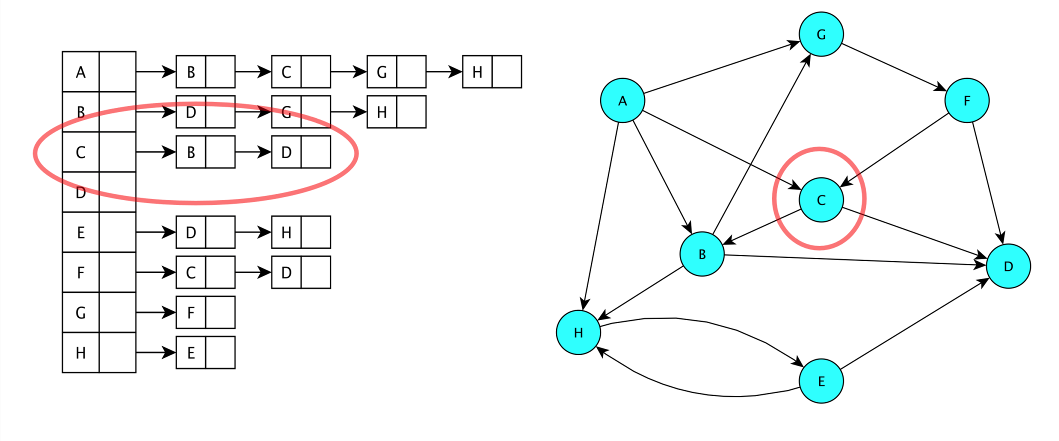
- The adjacency matrix was very wasteful of space, and finding the neighbors of a vertex was very slow
  - But, finding if there was an edge between two vertices was very fast
- Adjacency list representation: maintain a list of all edges that are indicent to each vertex
  - Only keep *outgoing* edges for directed graphs
  - Usually going to be a singly linked list
- Abstract class `GraphList`, concrete classes `GraphListDirected` and `GraphListUndirected`; also a new vertex class `GraphListVertex`

Adjacency List Visualization



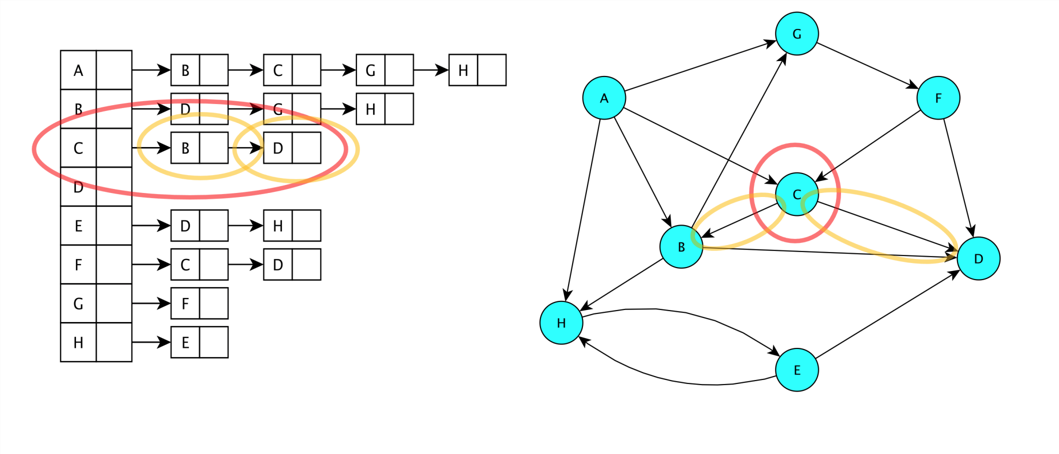
The vertices are stored in a `Map<V, GraphListVertex<V,E>>`. Each `GraphListVertex<V,E>` contains a linked list of all edges with a given *source*

Adjacency List Visualization



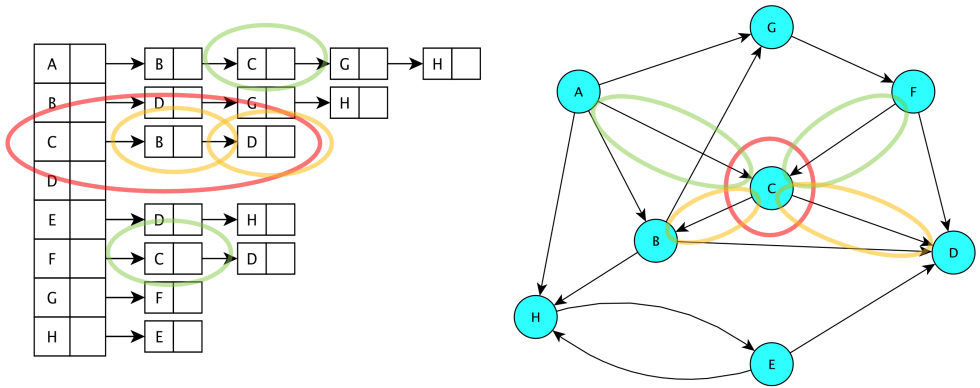
The vertices are stored in a `Map<V, GraphListVertex<V,E>>`. Each `GraphListVertex<V,E>` contains a linked list of all edges with a given *source*

Adjacency List Visualization



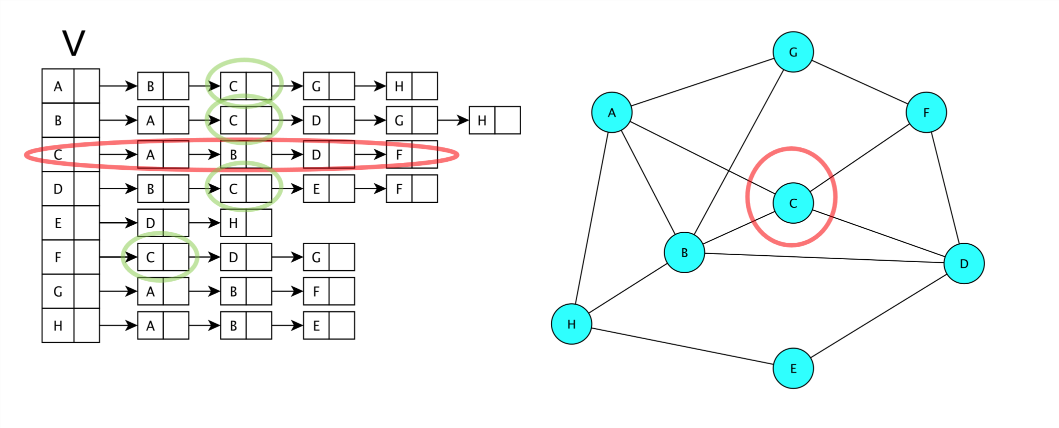
The vertices are stored in a `Map<V, GraphListVertex<V,E>>`. Each `GraphListVertex<V,E>` contains a linked list of all edges with a given *source*

Adjacency List Visualization



The vertices are stored in a `Map<V, GraphListVertex<V,E>>`. Each `GraphListVertex<V,E>` contains a linked list of all edges with a given *source*

Adjacency List Visualization: Undirected



The vertices are stored in a `Map<V, GraphListVertex<V,E>>`. Each `GraphListVertex<V,E>` contains a linked list of all edges *incident* to that vertex.

Creating adjacency list classes

- What does `GraphListVertex` need on top of `Vertex`?
  - Linked list of incident edges
- What is the difference between `GraphList` and `GraphMatrix`?
  - Do not need a free list of remaining vertices
  - Do not need to know number of vertices ahead of time
- `GraphList` is an abstract class for common methods; `GraphListUndirected` and `GraphListDirected` are concrete
- Let's take a look

Operations on an adjacency list for a graph?

- Let's say we have a graph with n vertices and m edges
- Getting all neighbors of a vertex?
  - $O(\#\text{ neighbors})$
- The *degree* of the vertex is its number of neighbors. So we can say $O(\text{degree})$.
- Adding a vertex or an edge?
  - $O(1)$
- Removing a vertex?
  - Expensive! Up to $O(n + m)$
- Getting an edge?
  - $O(\text{degree of vertex})$. Could be as bad as $O(n)$!
- Space?
  - $O(1)$ space per vertex or edge. Total: $O(n + m)$

Adjacency List vs Adjacency Matrix

- Adjacency List is (often) much faster for listing neighbors of a vertex:
  - Adjacency Matrix gives time proportional to the total number of vertices, Adjacency List gives time proportional to the degree.
- Adjacency Matrix is much faster for looking up if there is an edge bettween two vertices
- Adjacency List is much more space efficient if $m < n^2$