CS 209
Data Structures and Mathematical Foundations

04 / 26 / 2024

Instructor: Michael Eckmann

Today's Topics

- Questions?/Comments?
- Graphs

Graphs

- Graphs consist of a set of vertices and a set of edges.
- An *edge* connects two *vertices*.
- Edges can be *directed* or *undirected*.
- Directed graphs' edges are all directed. Undirected graphs' edges are all undirected.
- Edges can be weighted or unweighted

Graphs

- Graph's Edges can be represented in programs in several ways. Two common ways to represent them are:
  - adjacency matrix --- each row number r represents a vertex and the value at column c is true if there's an edge from vertex r to vertex c, false otherwise
    - Notice that the type stored in the adjacency matrix is boolean
    - Could we use this for weighted graphs?
    - Could we use this for directed graphs?
  - edge lists

  which is a list of linked lists
    - store a linked list for each vertex, v_i
    - items in the list are those vertices v_j for which there's an edge from v_i to v_j

Graphs

- Graph traversals
  - what are traversals? --- we used that word when we worked on binary trees

Graphs

- Graph traversals
  - Breadth First Search (BFS) and Depth First Search (DFS) are traversals

Graphs

- Graph traversal
  - Breadth first search (BFS)
    - Pick a vertex at which to start
    - Visit all of the adjacent vertices to the start vertex
    - Then for each of the adjacent vertices, visit their adjacent vertices
    - And so on until there are no more adjacent vertices
    - Do not visit a vertex more than once
  - Only vertices that are reachable from the start vertex will be visited --- example on the board.
  - The order that vertices in a BFS are visited are in increasing order of length of path from starting vertex.
  - Those that have the same path length from the start vertex can be visited in any order.
  - Example of BFS on the board.

Graphs

- Implementation of breadth first search
  - Have a flag for each vertex to mark it as unvisited, waiting, or visited – so we don't visit vertices more than once.
  - Keep a queue which will hold the vertices to be visited
  - Output a visited list of vertices
  - BFS algorithm:
    - Mark all vertices as unvisited
    - Initially enqueue a vertex into the queue, mark it as waiting
    - While the queue is not empty
      - Dequeue a vertex from the queue
      - Put it in the visited list, mark it as visited
      - Enqueue all the adjacent vertices that are marked as unvisited to the vertex just dequeued.
      - Mark the vertices just enqueued as waiting
    - return the visted list

Graphs

- Graph traversal
  - Depth first search (DFS)
    - Pick a vertex at which to start
    - Visit one of its adjacent vertices then visit one of that one's adjacent vertices, and so on until there is no unvisited adjacent vertex of the one we're working on.
    - Then backtrack one level and visit another adjacent vertex from that one and repeat.
    - Do this until we're at the start vertex and there's no more unvisited adjacent vertices
    - Do not visit a vertex more than once
  - Only vertices that are reachable from the start vertex will be visited
  - Those vertices that are adjacent to a vertex can be visited in any order.
  - Example of DFS on the board.

Graphs

- Recall that the BFS used a Queue.
- DFS
  - Any thoughts on how DFS could be implemented?
  - What data structure allows us to “backtrack”?

Graphs

- DFS
  - set all vertices to UNVISITED
  - push start vertex
  - visit start vertex and set start vertex to visited
  - while (stack is not empty)
    - peek to get vertex at top of stack
    - try to get an unvisited adjacent vertex to the peeked one
    - if there isn't one
      - pop the stack
    - else
      - push that unvisited adj v to the stack
      - Put it in the visited list and set it to visited
  - return the visited list

Graphs

- Shortest path algorithms
  - problem is to find the shortest path from one given vertex to each of the other vertices.
  - output is a list of paths from given vertex to all other vertices
  - what real world examples might ever want to find the shortest path?

Graphs

- Shortest path algorithms
  - problem is to find the shortest path from one given vertex to each of the other vertices.
  - output is a list of paths from given vertex to all other vertices
  - the shortest path could be in terms of path length (number of edges between vertices)
    - e.g. a direct flight has path length 1, flights with connecting flights have path length > 1
  - the shortest path could be in terms of minimum weight for weighted graphs (example on the board.)
    - e.g. finding the lowest cost flights
    - Dijkstra's algorithm solves this problem

Graphs

- the shortest path could be in terms of path length (number of edges between vertices)
    - e.g. a direct flight has path length 1, flights with connecting flights have path length > 1
  - Initialize all lengths to infinity
  - Process the graph in a BFS order starting at the given vertex
  - but when visit a node, also replace its length with the current length.
- Example on the board
- This is just BFS while also keeping track of path lengths.