Graph Algorithms

15-121

Introduction to Data Structures

Ananda Gunawardena

Review

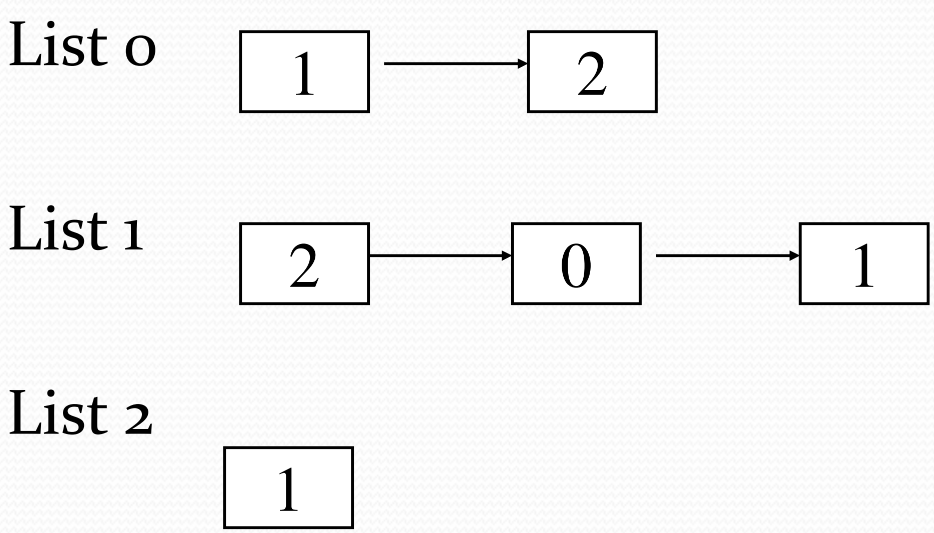
Graph Representations

Adjacency Matrix

- Adjacency Matrix
  - For each edge (v,w) in E, set A[v][w] = edge_cost
  - Non existent edges with logical infinity
- Cost of implementation
  - $O(|V|^2)$ time for initialization
  - $O(|V|^2)$ space
    - ok for dense graphs
    - unacceptable for sparse graphs

Adjacency List

- Adjacency List
  - Ideal solution for sparse graphs
  - For each vertex keep a list of all adjacent vertices
  - Adjacent vertices are the vertices that are connected to the vertex directly by an edge.
  - Example



Basic Graph Algorithms

Breadth First Traversal

- Algorithm
  - Start from any node in the graph
  - Traverse to its neighbors (nodes that are directly connected to it) using some heuristic
  - Next traverse the neighbors of the neighbors etc.. Until some limit is reach or all the nodes in the graph are visited
  - Use a queue to perform the breadth first traversal

Depth First Traversal

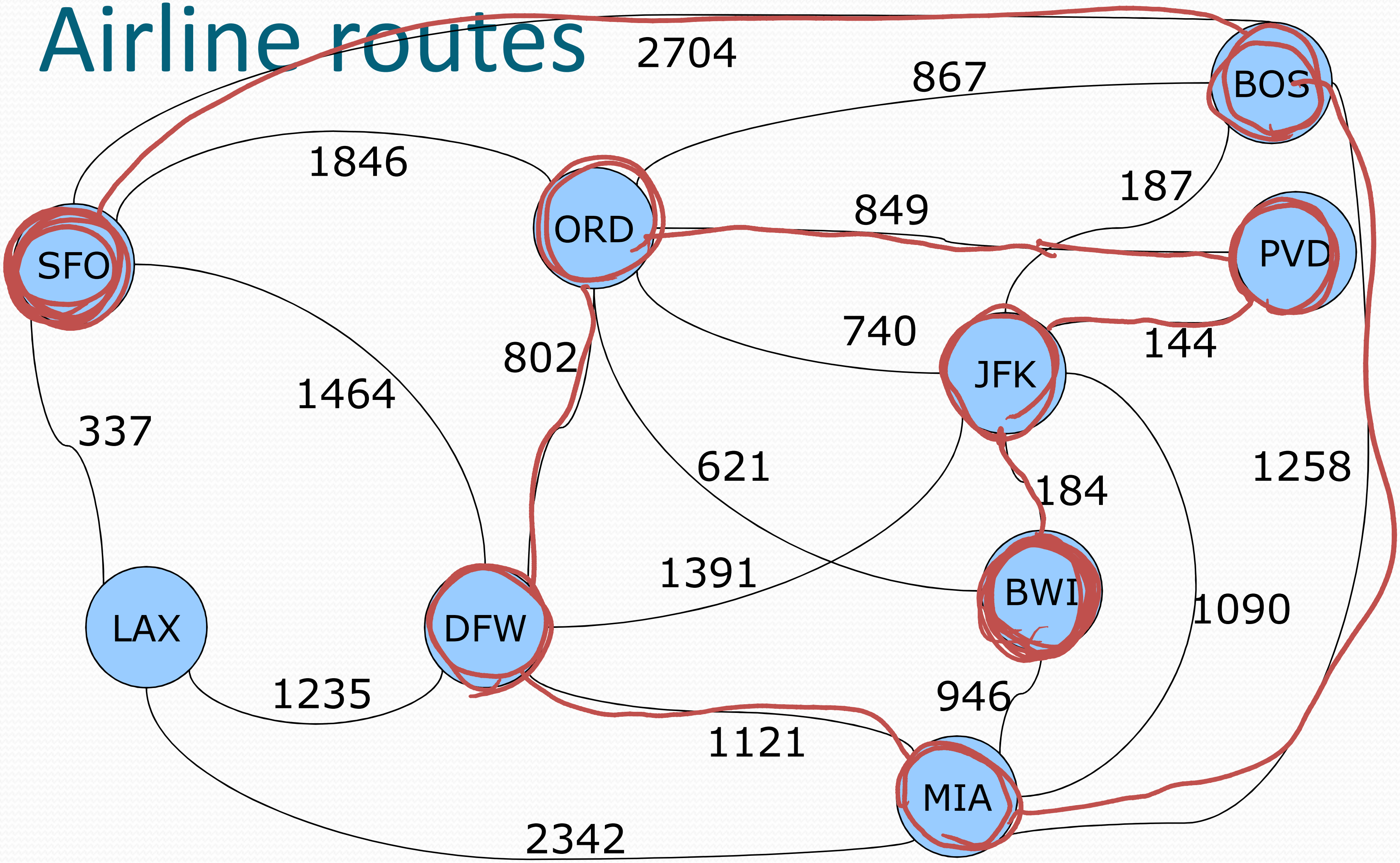
- Algorithm
  - Start from any node in the graph
  - Traverse deeper and deeper until dead end
  - Back track and traverse other nodes that are not visited
  - Use a stack to perform the depth first traversal

Shortest Paths

Many applications

- Shortest paths model many useful real-world problems.
  - Minimization of latency in the Internet.
  - Minimization of cost in power delivery.
  - Job and resource scheduling.
  - Route planning.
    - MapQuest, Google Maps

Airline routes



Single-source shortest path

- Suppose we live in Baltimore (BWI) and want the shortest path to San Francisco (SFO).
  - Naïve Approach
- A Better way to solve this is to solve the *single-source shortest path problem*:
  - That is, find the shortest path from BWI to every city.

Why Need to Find ALL Shortest Paths?

- *While we may be interested only in BWI-to-SFO, there are <u>no known algorithms</u> that are asymptotically faster than solving the single-source problem for BWI-to-every-city.*

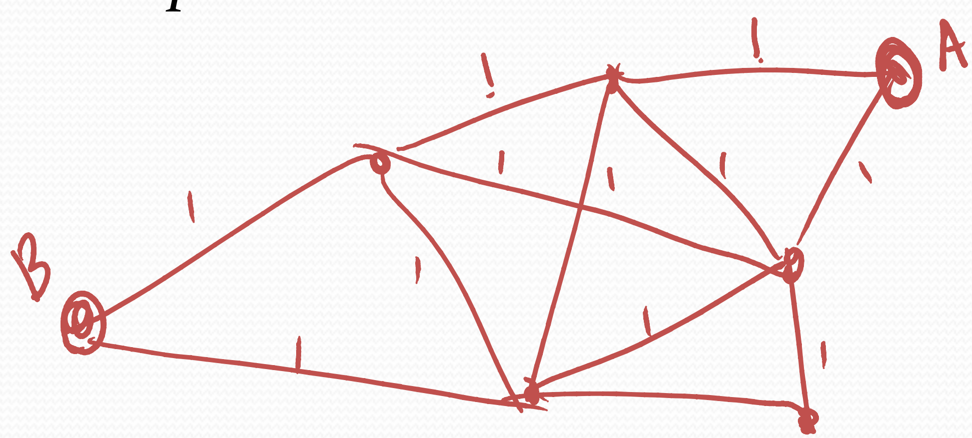
Shortest paths

- What do we mean by “shortest path”?
  - Minimize the number of layovers (i.e., fewest hops).
    - *Unweighted shortest-path problem.*
  - Minimize the total mileage (i.e., fewest frequent-flyer miles ;-).
    - *Weighted shortest-path problem.*

Unweighted Single-Source Shortest Path Algorithm

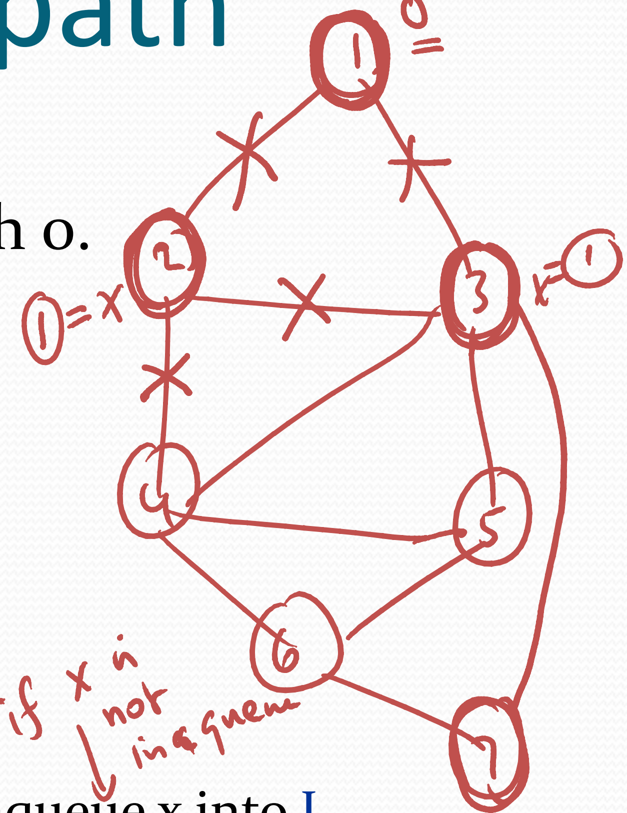
Unweighted shortest path

- In order to find the unweighted shortest path, we will mark vertices and edges so that:
  - vertices can be marked with an integer, giving the number of hops from the source node, and
  - edges can be marked as either *explored* or *unexplored*. Initially, all edges are *unexplored*.



Unweighted shortest path

- Algorithm:
  - Set i to 0 and mark source node v with 0.
  - Put source node v into a queue L_o.
  - While L_i is not empty:
    - Create new empty queue L_{i+1}
    - For each w in L_i do:
      - For each *unexplored* edge (w,x) do:
        - mark (w,x) as explored
        - if x not marked, mark with i+1 and enqueue x into L_{i+1}
    - Increment i.



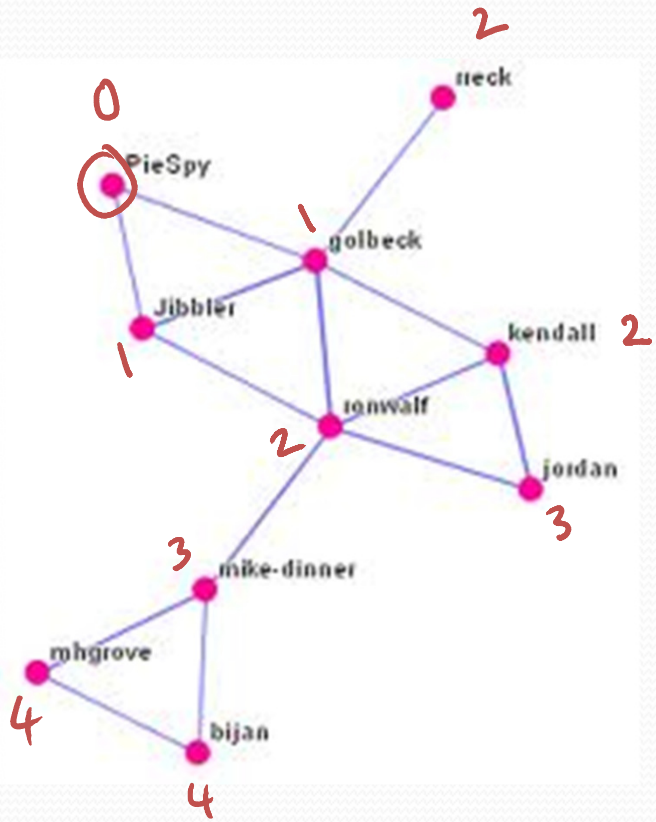
if x is not in queue

$O(|V|+|E|)$

i
~~0~~
~~1~~
2

$L_0=\{1\}$
$L_1=\{2,3\}$
$L_2=\{4,5,7\}$
$L_3=\{$

Example



Source: jibble.org

Complexity

- This algorithm is a form of *breadth-first search*.
- Performance: O(|V|+|E|). Why?

- *Q*: Use this algorithm to find the shortest route (in terms of number of hops) from BWI to SFO.
- *Q*: What kind of structure is formed by the edges marked as *explored*?

Use of a queue

- It is very common to use a queue to keep track of:
  - nodes to be visited next, or
  - nodes that we have already visited.
- Typically, use of a queue leads to a *breadth-first* visit order.
- Breadth-first visit order is “cautious” in the sense that it examines every path of length i before going on to paths of length i+1.

Next
Dijkstra’s Shortest Path Algorithm