CS 20B: Data Structures with Java

Week 15

Today's Lecture

- Graph Problems
- Course Review

Graph Problems

- Graphs and graph algorithms can be used to model and provide solutions to many important problems
- Some graph problems are easy, some are pretty challenging, some are practically impossible to solve
- We will give a brief overview of some well known and useful graph problems

Topological Sorting: Problem Motivation

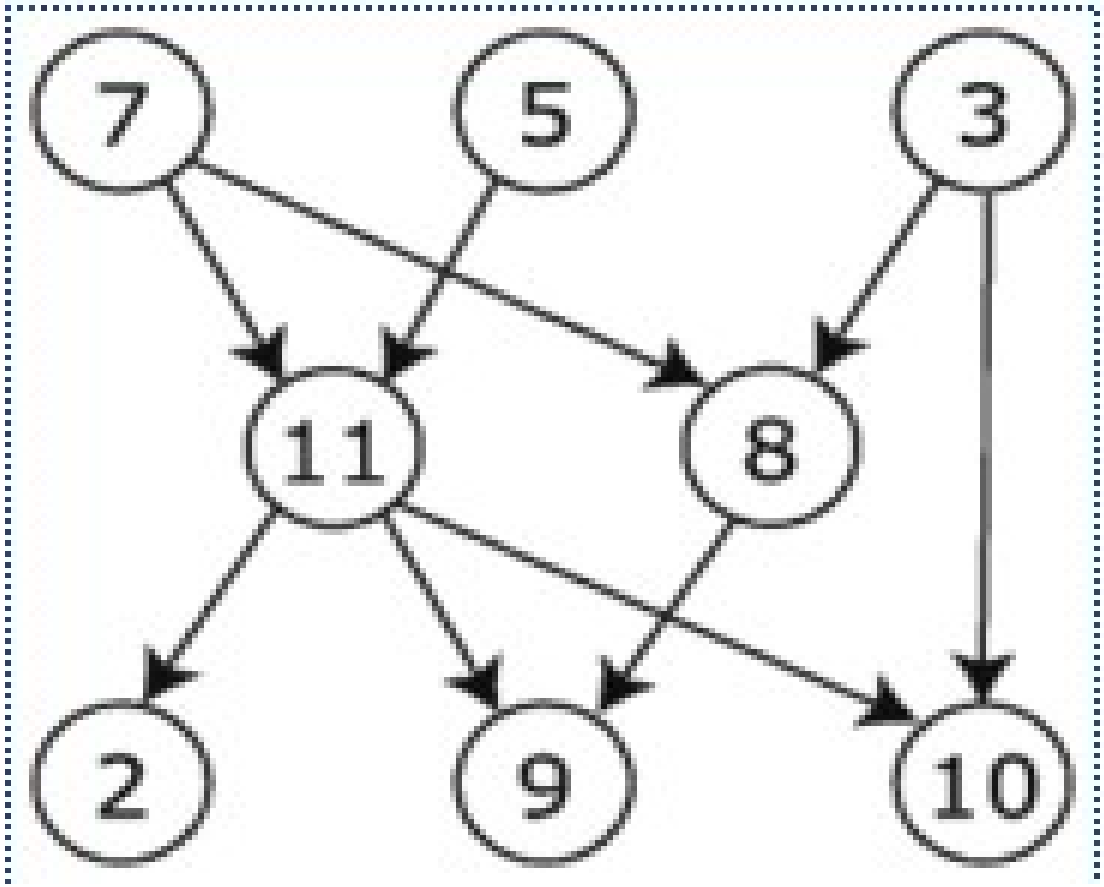
- There are n separate steps that need to be completed to complete a task. Some steps are prerequisite to others (i.e. a step cannot begin until all its prerequisite steps have been completed).
- Given all the dependencies between steps, in which order do we execute them to complete the task?
- e.g.: Order of courses to get a degree

  Order of assembling a car from parts

Topological Sorting

- Model the problem using a directed graph
- The vertices are the steps
- The dependencies are the edges
- An edge X → Y means that X has to be completed before Y (Y depends on X).
- Given this graph, we need to find an order for the vertices such that no vertex appears before any vertex it depends on
- Q: Does such an order always exist?

Topological Sorting



Topological Sorting

- A: No. The graph must not contain any cycles.
- DAG: Directed Acyclic Graph

- Q: How do we find such an order (or find a cycle → does not exist)?

Topological Sorting

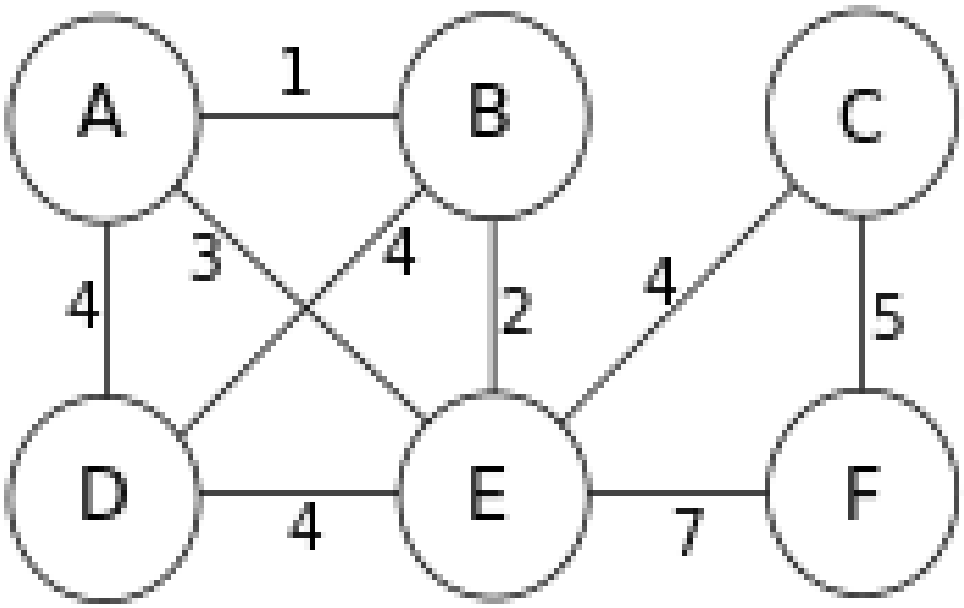
- A: Recursive algorithm based on DFS (pseudo code below from Wikipedia)

```
L ← Empty list that will contain the sorted nodes
while there are unmarked nodes do
    select an unmarked node n
    visit(n)
function visit(node n)
    if n has a temporary mark then stop (not a DAG)
    if n is not marked (i.e. has not been visited yet) then
        mark n temporarily
        for each node m with an edge from n to m do
            visit(m)
        mark n permanently
        add n to head of L
```

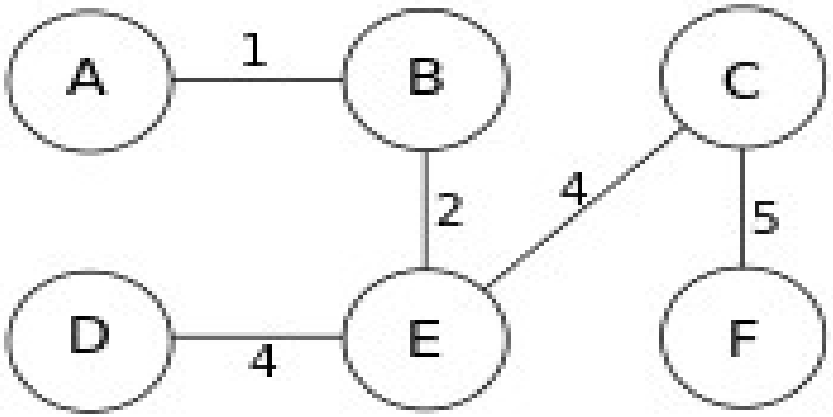
Minimum Spanning Trees

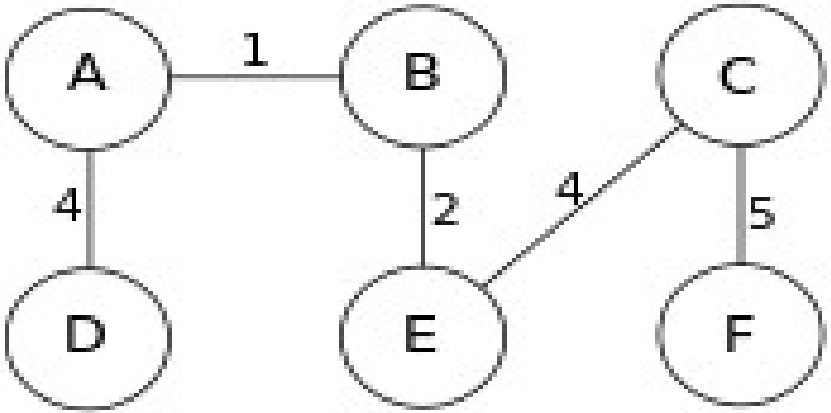
- We are given a connected undirected graph with weighted edges
- We need to find a subset of edges such that all the vertices are connected and that has the minimum total weight
- The graph formed by this subset of edges forms a tree called an MST
- Q: How do we go about finding the MST?

Minimum Spanning Trees
The Graph:



Minimum Spanning Trees
The MSTs:





Minimum Spanning Trees

- Idea: We know the MST has no cycles (there is only one way to reach a vertex). If we divide the MST into two groups of vertexes, we know that there is one and only one edge connecting the two groups. Furthermore, this edge is the lowest cost edge of all possible edges that could connect the two groups.
- Known as “the min cut property”
- We will use a “greedy” algorithm: repeatedly adding the min cost edge that does not form a cycle

Minimum Spanning Trees

- Prim's Algorithm

  1- Initialize a tree with a single vertex, chosen arbitrarily from the graph.

  2- Grow the tree by one edge: Of the edges that connect the tree to vertices not yet in the tree, find the minimum-weight edge, and transfer it to the tree.

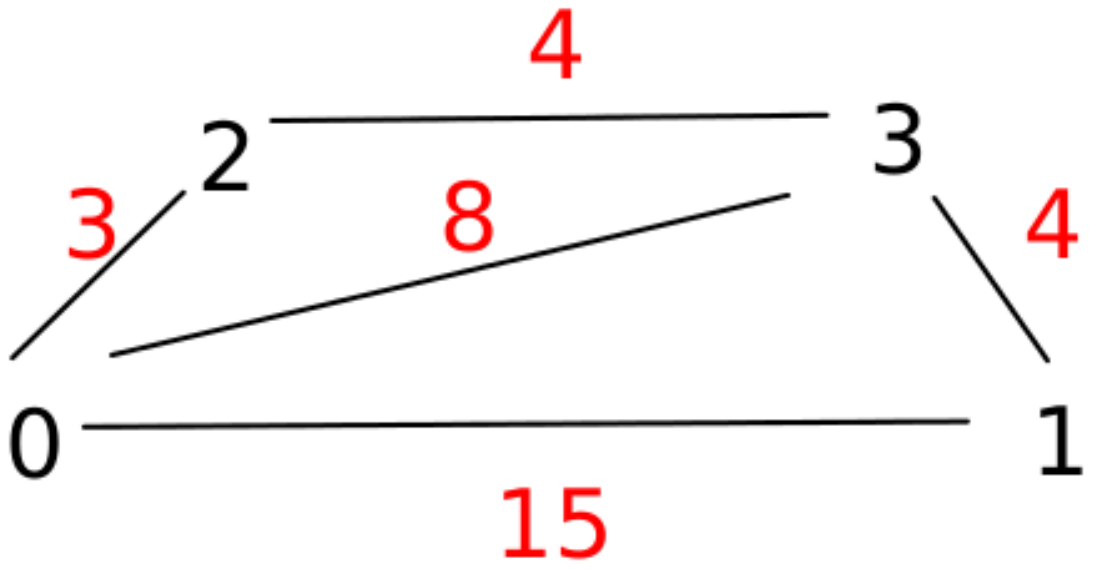
  3- Repeat step 2 (until all vertices are in the tree).

  (from Wikipedia)

Weighted Shortest Path

- Given a weighted graph and a start vertex, find the minimum cost path (shortest path) from that vertex to all other vertices
- e.g. Graph of cities and airline connections. What is the lowest cost path from LA to Rome? What are the lowest cost paths from LA to all possible destinations?

Weighted Shortest Path



Weighted Shortest Path

- Dijkstra's algorithm - idea:

  - The vertices will be in one of two categories: “settled” and “not yet settled”. Settled means we know the shortest path to that vertex. Not yet settled means we don't have a path yet or we are not sure it's the shortest.

  - Each time a new vertex is settled, we try to reach all not yet settled neighboring vertices from this new vertex and see if we can get a shorter path than what we had before

Weighted Shortest Path

- Start with distance 0 for start vertex, infinity for all others
- Keep an array boolean[ ] settled initially set to false for all vertices
- Repeatedly pick the closest to start unsettled vertex, update it's path cost and mark it settled. Update all its unsettled neighbors if the cost of the path going through this vertex is lower than previous cost.

Course Review

Where We Have Been

- We have seen the most fundamental ADTs and data structures in CS
- We have seen some of the most fundamental algorithms
- We have learned how to analyze the runtime/memory requirements of a solution

Where We Have Been

- We have learned to think of problems abstractly and translate them so we can make use of things we know (ADTs and algorithms) to get to a solution
- We have learned that when dealing with a large problem an inefficient solution is “no solution”.
- We have learned how to use fundamental data structures to handle large data sets and to solve hard problems efficiently and clearly

Where We Have Been

- Data structures:
  - Array
  - Array List, Array Deque, Linked List
  - Stack, Queue
  - PriorityQueue
  - Tree Set, Tree Map (Sorted Set/Map)
  - Hash Set, Hash Map (Unsorted Set/Map)
  - Graph

Where We Have Been

- Algorithms:
  - Linear search, Binary search
  - Sort
  - Depth/Breadth First Search/Traversal
  - Topological Sort
  - Minimum Spanning Tree
  - Shortest Path

Parting Thoughts

- Think before you code. Design, not debug.
- Pick the right tool for the job. Don't use a cannon to shoot a mosquito.
- Err towards too simple. Be clever only when it's really needed. *-The competent programmer is fully aware of the strictly limited size of his own skull; therefore he approaches the programming task in full humility, and among other things he avoids clever tricks like the plague. (E. Dijkstra)*

Where You Are Going

Best of Luck to All of You!

Next Class

Project 6 due

Assignment 7 Due Wedensday
on eCompanion

Final Exam