Topological sort and shortest path finding

Comp Sci 1575 Data Structures

Missouri S&T | Computer Science

Debugging

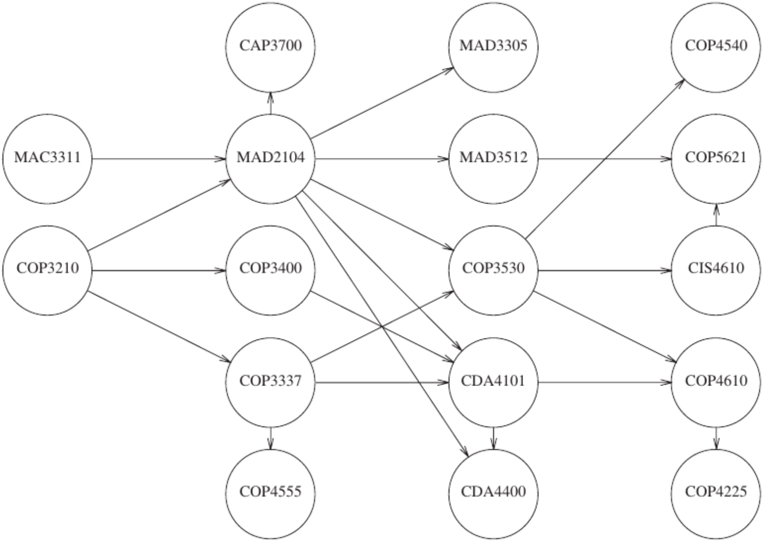
Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.

–Brian W. Kernighan

Outline

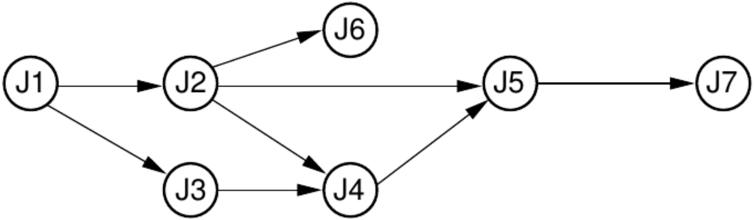
1. Topological sort
   - Printing nodes
     - DFS
     - BFS
2. Shortest path problems
   - High level
   - Code

Problem: Plan your course schedule



Task planning

Goal: organize the tasks into a linear order to complete them one at a time without violating any prerequisites.

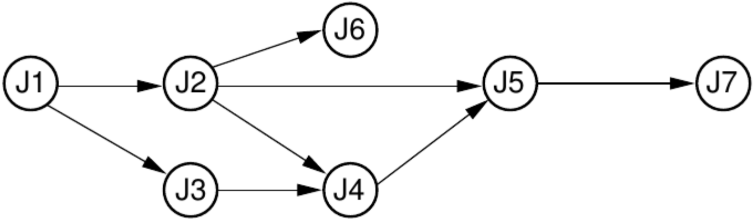


- Model the problem using a directed acyclic graph (DAG)
- One task is a prerequisite of another, represented by vertices with a directed relationship.
- Cycle would indicate a conflicting series of prerequisites that could not be completed without violating at least one prerequisite.

Outline

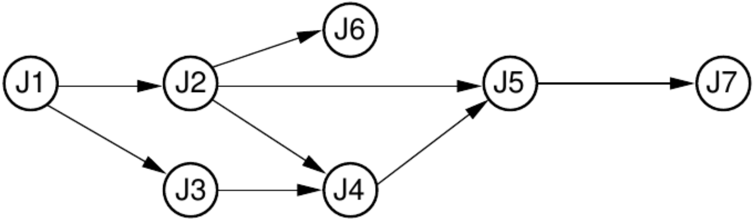
1. Topological sort
   - Printing nodes
     - DFS
     - BFS
2. Shortest path problems
   - High level
   - Code

How to print the nodes in a valid task order?



- Can we use DFS of BFS?
- Do we use recursion?
- If so, when do we print?
- Which tasks can be done immediately?
- Which can be done after that?

Option 1: DFS with post-print

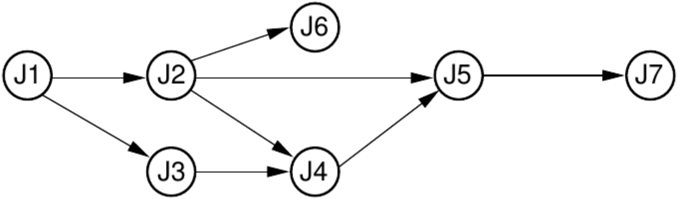


- Process of laying out the vertices of a DAG in a linear order to meet the prerequisite rules is called a topological sort.
- Found by performing a DFS on the graph (then reversed).
- When a vertex is visited, no action is taken (i.e., function PreVisit does nothing).
- When the recursion pops back to that vertex, function PostVisit prints the vertex.
- Yields a topological sort in reverse order.

Option 1: DFS with post-print

Look at code

Option 2: BFS with queue



- Visit all edges, counting the number of edges that lead to each vertex (prerequisites). Store in an array.
- All vertices with no prerequisites are placed on a queue.
- When Vertex V is taken off of the queue and printed, all neighbors of V (V as a prerequisite) have counts decremented by one.
- Put any neighbor whose count becomes zero in queue
- If the queue becomes empty without printing all of the vertices, then the graph contains a cycle, and there is no valid ordering for the tasks.

Option 2: BFS with queue

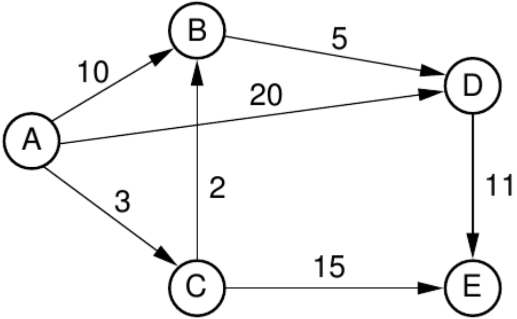
Look at code

Outline

1. Topological sort
    - Printing nodes
        - DFS
        - BFS

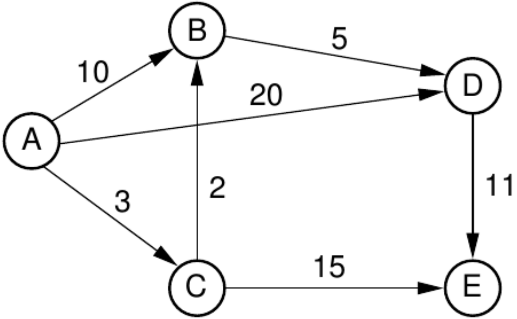
2. Shortest path problems
    - High level
    - Code

Single-source shortest-paths problem



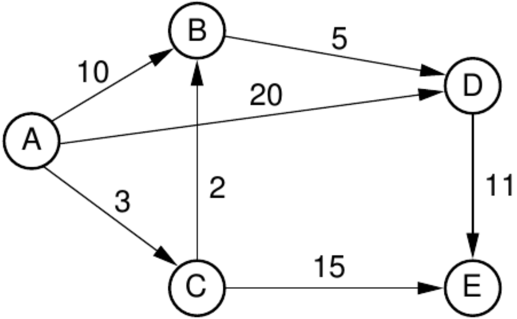
- Given Vertex S in Graph G, find a shortest path from S to every other vertex in G.
- Goal: shortest path between two vertices, d(S,T)
- But, in the worst case, while finding d(S,T), we might find the shortest paths from S to every other vertex as well.

Single-source shortest-paths problem



- Starting at node A, any ideas?

Check shortest neighbors first



- Dijkstra's algorithm finds the shortest paths between nodes in a graph, and it comes in several varieties.
- In AI, variants known as uniform-cost search, formulated as an instance of the more general idea of best-first search.

Outline

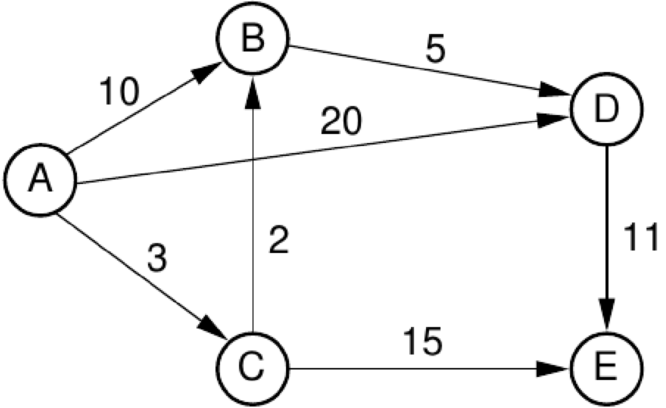
1. Topological sort
   - Printing nodes
     - DFS
     - BFS
2. Shortest path problems
   - High level
   - Code

Dijkstra's algorithm(s) overview

Let the node at which we are starting be called the initial node. Let the distance of node Y be the distance from the initial node to Y.

1. Assign all nodes a tentative distance: set to 0 for initial node and to infinity for all others.
2. Set initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes called the unvisited set.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.
4. After processing all neighbors of the current node, mark the current node as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited, or if the smallest tentative distance among the nodes in the unvisited set is infinity (when there is no connection between the initial node and remaining unvisited nodes), then stop.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as current, and go back to step 3 (How??)

Single-source shortest-paths problem



Outline

1. Topological sort
   - Printing nodes
     - DFS
     - BFS
2. Shortest path problems
   - High level
   - Code

Dijkstra's algorithm

Look at code, which is provided in two varieties, one with a heap and one without (speeds up step 6 on previous slide).