CS-204: Design and Analysis of Algorithms

220001025, 220001026, 220001027

February 9, 2024

1 Topological Sorting

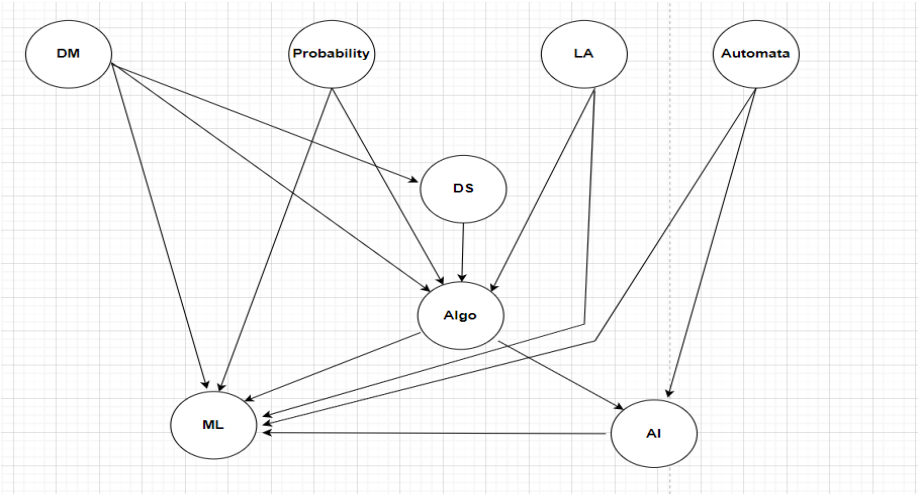
Topological sorting is an algorithm for linear ordering of vertices in a Directed Acyclic Graph (DAG) such that for every directed edge u-v, vertex u comes before v in the ordering.

1.1 Example

For example we have a set of courses that have some prerequisite courses.
Set of courses=DM, Probability, Linear Algebra, Data Structures, Automata, Algorithm, AI, ML

Prerequisites:

1. DS(DM)
2. Algo(DS, Probability)
3. AI(Algorithm, Automata)
4. ML(AI, Algorithm, Probability, LA, DM, DS)



Here each course in bracket is a prerequisite for the course. How can we take these courses such that all prerequisite courses are completed before taking the course?
We do this by using Directed Acyclic Graph(DAG).
A Directed Acyclic Graph (DAG) is a data structure that has allowing features:

- Directed: The edges of the graph have a direction, indicated by arrows. This implies a one-way dependency or flow between nodes.
- Acyclic: The graph does not contain any cycles. This means you cannot start at a node, follow the direction of the edges, and end up back at the same node.

In graph we can represent directed edges(i,j) where course i is a prerequisite for course j.

Note: The above example is a partial ordering(reflexive,anti-symmetric,transitive) not a total ordering as directed edge from i to j means no directed edge from j to i.

Question: Prove that in a DAG a situation with no node with indegree=0 is not possible and prove that topological sorting exist for DAG.
Hint: use contradiction

1.2 Algorithm of Topological Sorting

Steps:

1. push all vertex with indegree=0 in a queue.
2. remove a vertex and reduce the indegree of all its neighbour vertices by 1. If the indegree of the vertices become 0 push into queue.
3. Longest path of graph calculation: we use recursion to find longest path.
   Base case: if indegree=0, then LP[i]=0
   Recursive step:

$$LP[i] = 1 + max[LP(j)] \forall j : A[i,j] = 1$$

Algorithm:

```
Topological-Sort(G)
for each vertex u in G.V
    for each vertex v in G.Adj[u]
        in_degree[u]=in_degree[u]+1
for each vertex u in G.V
    if in_degree[u]==0
```

```
        Enqueue(Q,u)
while Q is not empty
    u=Dequeue(Q)
    for each v in G.Adj[u]
        in_degree[v]--
        if in_degree[v]==0
            Enqueue(v)
```

1.3 Difference between Dependency and General Graph

Normal Graph:

- A general graph is a broad term that encompasses any type of graph, including both directed and undirected graphs with any possible configuration of edges and vertices.
- Normal graphs are studied extensively in graph theory, where properties such as connectivity,Shortest paths, cycles, and other structural characteristics are analyzed.

Dependency Graph:

- A dependency graph is a directed graph that represents the dependencies between various elements in a system or a set of entities.
- In a dependency graph, nodes typically represent components or tasks, and directed edges represent dependencies between them. For example, if task A must be completed before task B can start, there would be a directed edge from node A to node B in the dependency graph.

1.4 Time Complexity Analysis of Topological Sorting

1. Push the vertex with indegree 0
   - for Adj Matrix– $O(V^2)$
   - for Adj List– $O(E)$
2. Remove the vertex with indegree 0 and visit its neighbour and reduce its indegree by 1
   - for Adj Matrix– $O(V^2)$
   - for Adj List– $O(E+V)$
3. Longest path of the graph
   - for Adj Matrix– $O(V^2)$
   - for Adj List– $O(E+V)$

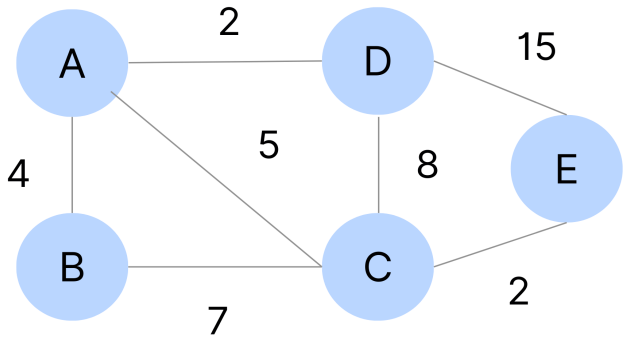
1.5 Why longest path not shortest?

Question: In most of the problems we look at finding the shortest path but for longest path problems like longest path in a DAG we use topological sort. Why?

- In a Directed Acyclic Graph (DAG), a topological sort is required for finding the longest path because the longest path can be determined by traversing the nodes in a specific order.
- Topological sort arranges the nodes of the DAG in such a way that for any directed edge from node A to node B, A comes before B in the ordering. This ordering ensures that all dependencies are resolved before processing a node.
- Since the longest path inherently relies on traversing nodes while respecting their dependencies, a topological order provides the necessary framework to identify and traverse the nodes in the right sequence.
- In addition to this, by definition, a DAG does not contain any cycles. This property is essential for finding the longest path because cycles can potentially create infinite-length paths.
- Topological sorting inherently avoids cycles, ensuring that the traversal of nodes proceeds without the risk of encountering cycles, thus enabling the determination of a well-defined longest path.

2 Dijkstra's algorithm

Dijkstra's algorithm is a graph traversal algorithm that finds the shortest path between a source node and all other nodes in the graph. It works by greedily selecting the unvisited node with the shortest tentative distance from the source node, and then updating the tentative distances of its neighbors. The algorithm can be used to find shortest paths in both directed and undirected graphs.



Iteration	Visited	Tentative distances	Priority queue
1	A	A:0	(0, A)
2	A	A:0, B:4, D:2	(0, A), (2, D), (4, B)
3	A, D	A:0, B:4, D:2, C:5	(0, A), (2, D), (4, B), (5, C)
4	A, D	A:0, B:4, D:2, C:5	(0, A), (2, D), (4, B), (5, C)
5	A, D, C	A:0, B:4, C:5, D:2, E:7	(0, A), (2, D), (4, B), (5, C), (7, E)
6	A, D, C	A:0, B:4, C:5, D:2, E:7	(0, A), (2, D), (4, B), (5, C), (7, E)
7	A, D, C, B	A:0, B:4, C:5, D:2, E:7	(0, A), (2, D), (4, B), (5, C), (7, E)
8	A, D, C, B	A:0, B:4, C:5, D:2, E:7	(0, A), (2, D), (4, B), (5, C), (7, E)

Table 1: Dry Run of Dijkstra's Algorithm

2.1 Dry Run of the Algorithm

Below is a dry run for the example weighted graph with the following nodes and edges:
Nodes: A, B, C, D, E
Edges: A-B (4), A-C (5), A-D (2), D-C (8), B-C(7), C-E (2), D-E (15)
Consider vertex A as source vertex and find the shortest path distances of all other vertexes from vertex A using dijkstra's algoritm.

1. Initialize the distances dictionary and the priority queue.
    - distances dictionary is initialized with all vertices having a distance of positive infinity(except for the source vertex A)
    - Set the distance for source vertex A as 0
    - The priority queue is initialized with the source vertex A and its distance 0.
2. While the priority queue is not empty:
    - Get the vertex with the shortest tentative distance from the priority queue.
    - If the current vertex has already been visited, skip it.
    - Visit the current vertex and update the distances of its neighbors.
    - Add the unvisited neighbors to the priority queue with their updated distances.

The shortest path distances from vertex A are:
A to B: 4
A to C: 5
A to D: 2
A to E: 7