Dijkstra's Algorithm

Given a source, find least cost to every destination

```
Function SSSP(graph, src) : Dictionary<V, int>          (int: total cost)
    frontier ← new PQ<int, V>          ← low priority is next
    frontier.insert(0, src)
    cost ← new Dictionary<V, int>
    cost.insert(src, 0)
    While frontier is not empty:                                   O(|V|)
        current ← frontier.remove()        ← by the time a vertex is dequeued, we have found its best path
        For each edge leaving current:                             O(|E|)
            O(1) (avg)  total ← cost.get(current) + edge.weight
            O(1) (avg)  If (not cost.contains(neighbor)):
            O(1) (avg)      cost.insert(neighbor, total)
            O(log |V|)      frontier.insert(total, neighbor)
            Else If cost.get(neighbor) > total:
                cost.update(neighbor, total)
                frontier.insert(total, neighbor)        ← update
            End If
        End For
    End While
    Return cost
End Function
```

O(|E|) · O(|V|)

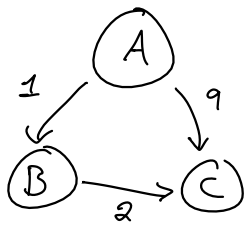
O(|E| · log |V|)

(There exist $O(|V|^2)$)

Either

If cost.get(current) = priority of current when it was removed

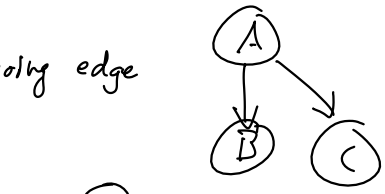
9, C



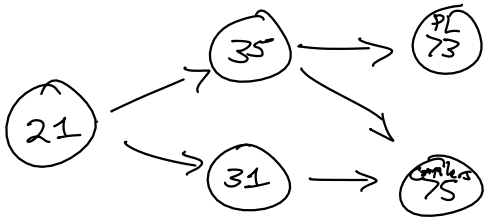
Topological Sort

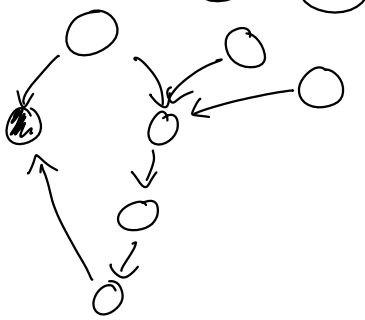
Given a directed, acyclic graph (DAG), give an ordering for vertices s.t. I visit every vertex before any of its neighbors.

outgoing edge



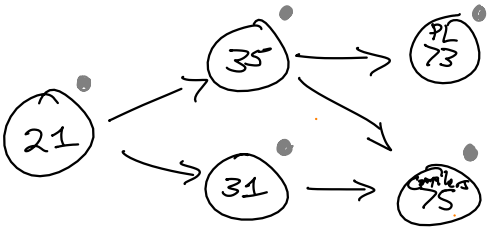
Given a partial order as a graph, produce a total order that respects it.





21, 35, 73, 31, 75

21, 31, 35, 73, 75



- Never Seen
- CurrentlyExploring
- Finished

21, 35, 73, 31, 75


```
Function TopoSort(graph):
    result ← new LinkedList
    For each vertex in graph:
        Explore(vertex, graph, result)
    Return result
EndFunction
```

```
Function Explore(vertex, graph, result):
    If vertex is ● : Return
    If vertex is ● : !!
    Mark vertex as ●
    For each neighbor of vertex in graph:
        Explore(neighbor, graph, result)
    EndFor
    result.insertFirst(vertex)
    Mark vertex as ●
EndFunction
```

Minimum Spanning Tree

Prim's Algorithm