Notes DAG

A diagram of a network

Description automatically generatedA diagram of a network

Description automatically generated

*Problems*

1. vertices - activities
   * (x, y) - activity y cannot start before activity x is completed
2. vertices - topics
   * (x, y) - y cannot be understood without understanding topic x
3. vertices - computation steps / results
   * (x, y) - y takes as input the result of x

*Topological sorting*

2 problem types:

1. Find if there is any circular dependency (if the graph is a DAG or not)
   * done while attempting to do the topological sorting
2. [topological sorting] Put the vertices in a list such that if x comes before y

*Property: Topological sorting is possible cycles in the graph*

*Algorithms:*

1. *Predecessor counting algorithm:*

Input: G : directed graph

Output: sorted : a list of vertices in topological sorting order, or null if G has cycles

Algorithm:

sorted = emptyList

Queue q

Dictionary count

for x in X do

count[x] = indeg(x)

if count[x] == 0 then

q.enqueue(x)

endif

endfor

while not q.isEmpty() do

x = q.dequeue()

sorted.append(x)

for y in Nout(x) do

count[y] = count[y] - 1

if count[y] == 0 then

q.enqueue(y)

endif

endfor

endwhile

if sorted.size() < X.size() then

sorted = null

endif

Steps:

|  |
| --- |
| * Repeat:   + take a vertex with no predecessors   + put it on the sorted list   + eliminate it from the graph (keep, for each vertex, a counter of predecessors still in the graph) * Finally, we have either   + processed all vertices 🡺 the topologically sorted list   + cannot get a vertex with no predecessors 🡺 a cycle |

1. *DFS-based algorithm*

|  |
| --- |
| Input:  G : directed graph  Output:  sorted : a list of vertices in topological sorting order, or null if G has cycles  Subalgotithm TopoSortDFS(Graph G, Vertex x, List sorted, Set fullyProcessed, Set inProcess)  inProcess.add(x)  for y in Nin(x)  if y in inProcess then  return false  else if y not in fullyProcessed then  ok = TopoSortDFS(G, y, sorted, fullyProcessed, inProcess)  if not ok then  return false  endif  endif  endfor  inProcess.remove(x)  sorted.append(x)  fullyProcessed.add(x)  return true  Algorithm:  sorted = emptyList  fullyProcess = emptySet  inProcess = emptySet  for x in X do  if x not in fullyProcessed then  ok = TopoSortDFS(G, x, sorted, fullyProcessed, inProcess)  if not ok then  sorted = null  return  endif  endif  endfor |

Scheduling problem:

**Input**: you are given a list of activities to be done for a project, and each activity has a list of prerequisite activities and a duration

**Output**: a scheduling of the activities (the starting and the ending time for each activity). If activity B depends on activity A, then B must start when or after A ends; however, two activities that do not depend on each other can be executed in parallel.

* 2 scheduling methods:
  + *Earliest scheduling*, where every activity is scheduled as early as possible;
  + *Latest scheduling*, where every activity is scheduled as late as possible - but while keeping the project finish as early as possible;

A table with numbers and letters

Description automatically generated