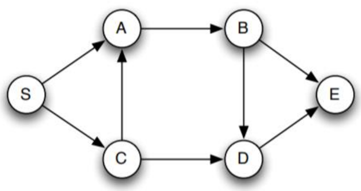
Q: *Longest path in a DAG*

a. Design an efficient algorithm for finding the length of the longest path in a DAG. (This problem is important both as a prototype of many other dynamic programming applications and in its own right because it determines the minimal time needed for completing a project comprising precedence constrained tasks.)

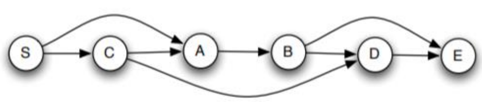
b. Show how to reduce the coin-row problem discussed in this section to the problem of finding a longest path in a DAG.

A:

a. Say the given unweighted DAG is:



Recall that DAG can be topologically sorted, which can traverse all vertices to a linear order. In this case, the given DAG is taken in linearized order {*S, C, A, B, D, E*}:



Before finding the longest path in the graph, it is best to first understand how to find the longest path from vertex *S* to any other vertex. Say the path from vertex *S* to vertex *B*. The only possible ways to get to vertex *B* from vertex *S* is either through vertex *A* or through vertex *C*. To compute the longest path from *S* to *B* is to compute the longest path from *S* to *A* and compute the longest path from *S* to *C*, then get the longer path from the two and add 1.

In the general case, to get the longest path from vertex *u* to *v* is to get the longest paths from vertex *u* to all the predecessors of vertex *v*, choose the largest among the longest paths, and add 1:

$\text{dist}\ (v) = \max_{(u,v)\in E} \{ \text{dist}\ (u) + 1 \}$

Now, starting from vertex *S*, find the longest paths to all the vertices:

for each vertex v in undirected DAG (in a linearized order)
do $\text{dist}\ (v) = \max_{(u,v)\in E} \{ \text{dist}\ (u) + 1 \}$

With this, one can find the longest path in an unweighted DAG by getting the largest of all { dist (v), $v \in V$ }:

longest path in a graph = $\max_{v\in V} \{ \text{dist}(v) \}$

ALGORITHM: //LongestPathInUnweightedDAG
//Input: An Unweighted DAG $G = (V,E)$
//Output: Longest path cost in G

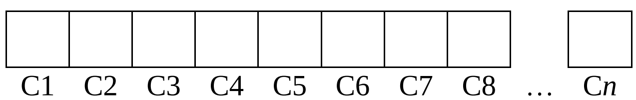
TopologicalSort(G)
for each vertex $v \in V$ in linearized order
  do dist (v) = max $_{(u,v)\in E}$ { dist (u) + 1 }
return max$_{v\in V}$ { dist(v) }

In case if given DAG is weighted, one finds the longest path of DAG in terms of edge weights, instead of number of edges:

ALGORITHM: //LongestPathInWeightedDAG
//Input: A Weighted DAG $G = (V,E)$ with edge weights $w\ (u, v)$
//Output: Longest path cost in G

TopologicalSort(G)
for each vertex $v \in V$ in linearized order
  do dist (v) = max $_{(u,v)\in E}$ { dist (u) + w (u, v) }
return max$_{v\in V}$ { dist(v) }

b. Say there are n coins in a row represented as an array:



This coin-row problem can be transformed to a weighted DAG with n + 1 vertices (one vertex to start $C0$ and the rest to represent the coins C1, C2, …, Cn). This DAG has the following edges:

n - 1 edges from any other vertex to Cn [(C0, Cn) (C1, Cn) (C3, Cn) … (Cn - 2, Cn)] with weight equal to the value of Cn, n – 2 edges (C0, Cn - 1) (C1, Cn - 1) … (Cn - 3, Cn - 1) with weight equal to the value of Cn – 1, and so one until two edges (C0, C3) (C1, C3) with weight equal to the value of C3, one edge (C0, C2) with weight equal to the value of C2, and one edge (C0, C1) with weight equal to the value of C1.

The longest path of this DAG is the maximum amount of money that can be picked up in the coin-row problem.