Shortest Path in DAG

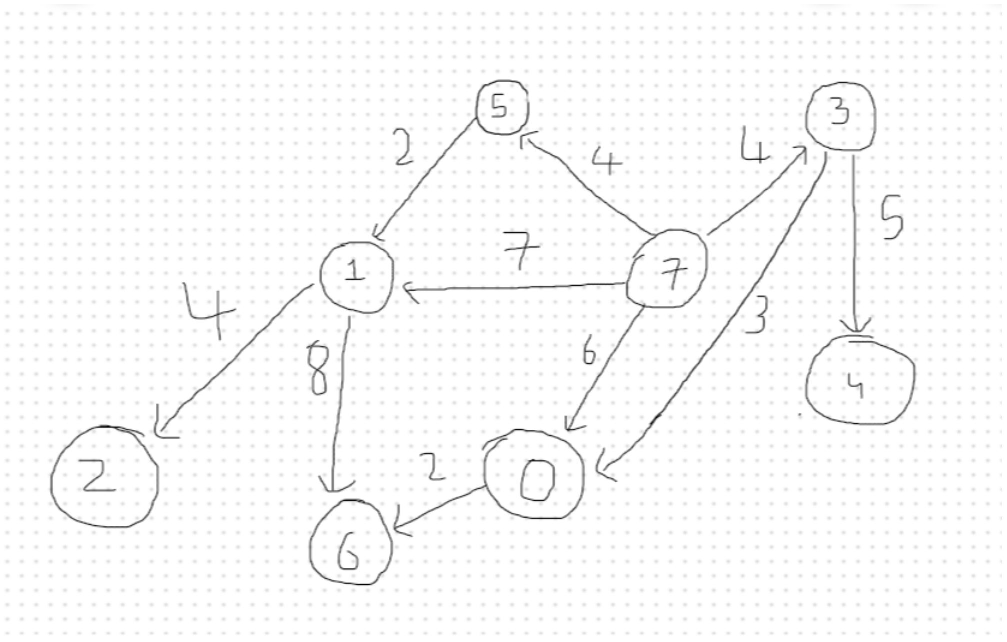
Problem

Given a DAG (Directed Acyclic Graph) and some source as input . Find the shortest path from Source to all vertexes.

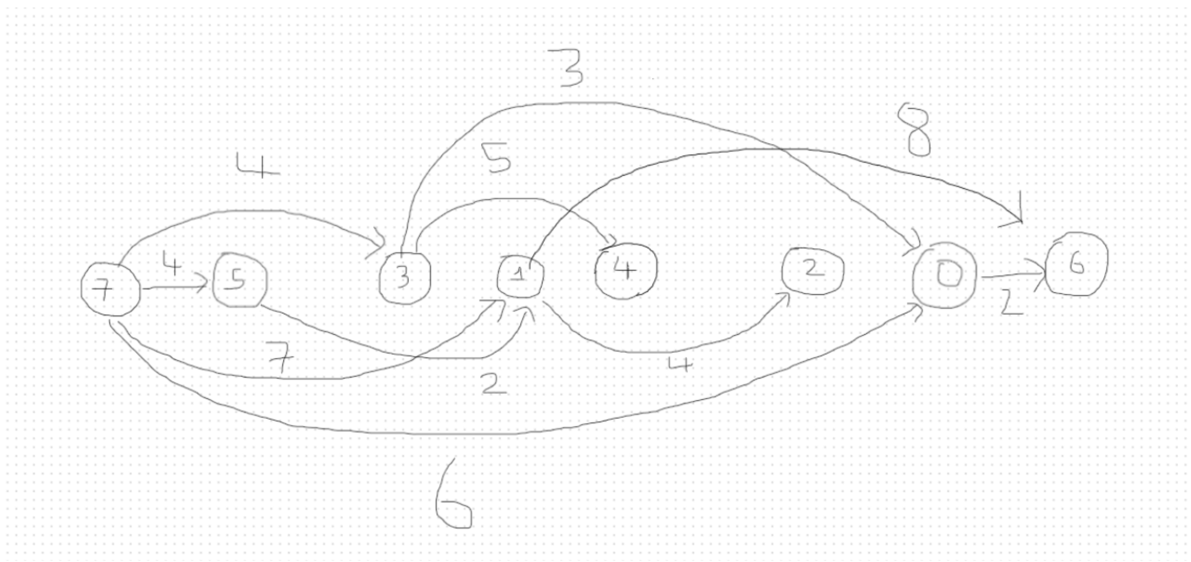
A dag is distinguished by the fact that its nodes may be linearized, that is, It can be arranged on a line such that all edges only going from left to right. This arranging is known as topological sorting

Example

Unsorted



Sorted



For each node, a similar relation may be written. We can always be guaranteed that by the time we arrive to node v, we have all the information we need to compute the distance of node v from source. if we compute these dist values in the left-to-right order of topologically sorted order

Solution

If G is a DAG, a topological sorting of the vertices is always possible. This isn't always a distinct.

Assuming we have vertices in topologically sorted order in an array such that $topo[i] = v_i$. Meaning v_0 is the source and there is no edge from v_i to v_j if $j < i$.

We can give a recursive relation for finding shortest path from vertex v_s to any other vertex v_k as follows

$$SP(v_s, v_k) = \min_{v_i \in V, i<k} \{SP(v_s, v_i) + l(v_i, v_k)\}$$

where SP means shortest path and l means the length

Now if we try solve this recursively(Top to bottom approach). For each vertex v_k that we visit, we will potentially need to make a recursive call for each v_i where i < k (this is the case if every vertex preceding $v_k$has an edge to vk). As a result, our recurrence relationship is

$$T(n) = \sum_{i<n} T(i) + O(n)$$

Resulting time complexity = $O(n^n)$

To rectify this, we shall solve the subproblems from the bottom up rather than from the top down. This will prevent us from resolving the same subproblems repeatedly, which is what is generating the long runtime.

- We maintain 1-D table to store the previously solved problems name DP[]. Initialized with all its value as infinity. where $DP[v_k]$ means $SP(v_0, v_k)$
- Relation can be written as $DP[v_k] = \min_{i<k}\{\ DP[v_i] + l(v_i, v_k)\}$
- So we start from bottommost(smallest) $SP(v_0, v_0) = 0$. So we fill $DP[v_0] = 0$
- Now we have everything that we need to fill in v_1. If we a assume there is an edge (v_0, v_1) and $l[v_0, v_1]$ as l_0. From above relation we will fill $DP[v_1] = l_0$
- Now, we have everything we need to fill in the value of v_3. Similarly we go on and on unltil we have filled $DP[v_n]$.

Algorithm

```
1. Initialize a dist array will all it's elements initialized to INF
2. Initialize dist(source) = 0
3. For each vertex v in linearized order
    1. For each vertex u in graph
        1. dist(v) = min(dist(v), G(v,u) + dist(u))
```

This algorithm involves tackling a set of subproblems, dist(u): u,v in E.
We'll start with the smallest, dist(s), because we already know the solution is = 0. Then continuously solve the larger and larger problem in topological order very fast since we already have all the information required

Time complexity

Here time complexity is clearly visible from the code as

final time complexity = O(V+E)

Space Complexity

We here need to feel only 1-D table so
Final space complexity = O(V)