DS 320 Algorithms for Data Science **Lecture #5 Worksheet**
Fall 2024 Profs. Krzysztof Onak and Aldo Pacchiano

DFS Application: Topological Sort

We now focus on directed graphs. Hence every edge goes from some vertex v to some vertex u, but there is no guarantee that there is an edge going in the opposite direction. For simplicity we assume that there are no self–loops in our graphs (i.e., directed edges going from a vertex to itself). They may make sense in some applications but we avoid them to not complicate our algorithms.

Let us start with two crucial definitions:

Definition 1 (Topological ordering)**.** Let $G = (V, E)$ be a directed graph. We say that a permutation $\sigma = (v_1, v_2, \ldots, v_n)$ of vertices in V is a *topological ordering* of G if for every directed edge $(u, v) \in E$ (i.e., an edge from u to v), u appears before v in σ.

Definition 2 (DAG)**.** We say that a graph is a *directed acyclic graph (DAG)* if it is a directed graph with no cycle.[1]

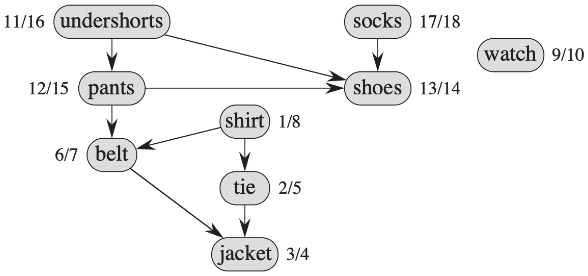


Figure 1: A sample topological sorting graph from CLRS.

[1] You can find a definition of a cycle in a directed graph in the worksheet with solutions for Lecture 4.

Theorem 1. *G has a topological ordering $\iff$ G is a DAG.*

Topological Sort Algorithm:

Theorem 2. *If the graph has a topological ordering and tasks are ordered by decreasing postorder number, then all precedence constraints are satisfied.*

Many Cool Applications of DFS

We are just listing some examples here, but among many other things, DFS can be used to design efficient algorithms for:

- Finding "bridges"/"cut edges" (a bridge is an edge such that when removed, one of connected components breaks into two)
- Finding "articulation points"/"cut vertices" (an articulation point is a vertex such that when removed, one of connected components breaks into two or more)
- Turning a connected undirected graph with no bridges into a directed one by assigning a direction to every edge such that any vertex is still reachable from any other one

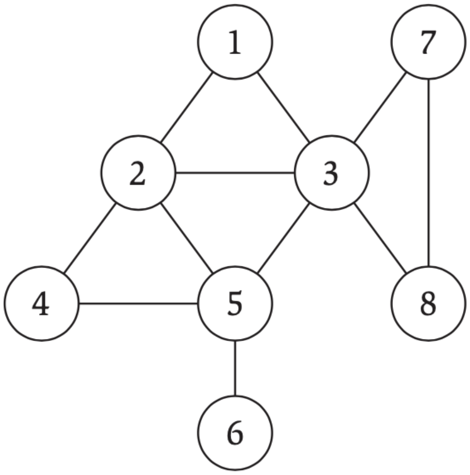
- Determining strongly connected components in a directed graph (i.e., partitioning the set of vertices into groups so that any two vertices reachable from each other are in the same group and all vertices in the same group are reachable from each other)

Breadth-First Search (BFS)

Breadth-First Search: Search in all directions simultaneously.

Algorithm 1 BFS(G, v)

```
  Input: graph $G = (V, E)$ and vertex $v$
  for each $v \in V$ do
      set $v$'s parent to Null
      $distance[v] = \infty$
  end for
  $Q \leftarrow$ [empty queue]
  $Q$.enqueue($v$)
  $distance[v] = 0$
  while $Q$ is not empty do
      $w = Q$.dequeue()
      for each $(w, z) \in V$ do
          if $distance[z] = \infty$ then
              $distance[z] = distance[w] + 1$
              set $z$'s parent to $w$
              $Q$.enqueue($z$)
          end if
      end for
  end while
  return array $distance$ and forest $F$ formed by
  parents
```



Exercise: Simulate BFS($G, 1$), where G is the above graph. Show how the content of Q evolves.

Theorem 3. BFS(G, v) *computes distances from v to every other vertex and the corresponding shortest paths.*

Question: What is BFS's running time if G is represented via adjacency lists?

Based on material developed by Prof. Kira Goldner