CS2351
Data Structures

Lecture 12:
Graph and Tree Traversals III
About this lecture

• We introduce the Topological Sort problem on directed acyclic graph (DAG)

• We give two linear-time algorithms:
  (1) Using Queue
  (2) Using Stack
Topological Sort

- Directed graph can be used to indicate precedence among a set of events
- E.g., a possible precedence is dressing

Diagram:
- under-shorts
- pants
- belt
- shirt
- tie
- shoes
- socks
- watch
- jacket
Topological Sort

• The previous directed graph is also called a precedence graph

Question: Given a precedence graph \( G \), can we order the events such that if \((u,v)\) is in \( G \) (i.e. \( u \) should complete before \( v \)) then \( u \) appears before \( v \) in the ordering?

We call this problem topological sorting of \( G \)
Topological Sort

Fact: If $G$ contains a cycle, then it is impossible to find a desired ordering (Prove by contradiction)

• However, if $G$ is acyclic (contains no cycles) we shall give two algorithms that always find the desired ordering
Topological Sort (with Queue)

Topological-Sort($G$) // given $G$ is acyclic
{
    while ($G$ contains a vertex)
    {
        1. Pick a vertex $v$ with in-degree = 0;
        2. Remove all its outgoing edges;
        3. Output $v$;
    }
}

Why is the algorithm correct?
Topological Sort (with Queue)

Theorem:
If $G$ is acyclic, the previous algorithm produces a topological sort of $G$

Proof:
Two cases may happen when we run the previous algorithm.
Case 1: All vertices are output
Case 2: Some vertex may not be output
Proof

• In Case 1, vertices are sorted correctly

• In Case 2, the remaining vertices must each have in-degree $\geq 1$. Now, we pick a vertex $v$ in this group, repeatedly visit another vertex by tracing an incoming edge, some vertex will be visited twice (why?) $\Rightarrow$ a cycle is found!!
Performance

• Let $G = (V,E)$ be the input directed graph
• Running time for Topological-Sort:
  1. Each vertex keeps # incoming edges
  2. Finding vertices with in-degree = 0:
     Naïve method: $O(|V|^2)$ total time
     Clever method: (use a queue $Q$)
     Enqueue vertex once its in-degree = 0
• Total time: $O(|V|+|E|)$
Topological Sort (Example)

When a vertex is output, its indegree is 0.
Topological Sort (with Stack)

Topological-Sort(G) // given G is acyclic
{
    1. Call DFS on G
    2. Output vertices in decreasing order of their finishing times;
}

Why is the algorithm correct?
Theorem:
If $G$ is acyclic, the previous algorithm produces a topological sort of $G$

Proof: Let $(u, v)$ be an edge. We shall show $f(v) < f(u)$ so that the ordering is correct.
Firstly, during DFS, there are two cases
• Case 1: $u$ is visited before $v$
• Case 2: $v$ is visited before $u$
Proof

- In Case 1, u cannot finish before DFS is performed on all its neighbors. Since v is a neighbor of u, we must have
  \[ d(u) < d(v) < f(v) < f(u) \]
- In Case 2, v must have finished before u starts (else, there will be a path from v to u and the graph contains a cycle.) Thus,
  \[ f(v) < d(u) \implies f(v) < f(u) \]
- Both cases show \( f(v) < f(u) \) \( \implies \) Done!
Topological Sort (Example)

Discovery and Finishing Times after a possible DFS
Ordering Finishing Times (in descending order)

If we order the events from left to right, anything special about the edge directions?
Performance

• Let $G = (V,E)$ be the input directed graph
• Running time for Topological-Sort:
  1. Perform **DFS**: $O(|V|+|E|)$ time
  2. Sort finishing times
     Naïve method: $O(|V| \log |V|)$ time
     Clever method: (use an extra stack $S$)
        During DFS, push a node into stack $S$ once finished $\Rightarrow$ no need to sort !!
• Total time: $O(|V|+|E|)$