Digraphs
Outline and Reading

- Digraphs (§13.4)
- Traversals of digraphs (§13.4.1)
- Transitive closure (§13.4.2)
- Floyd-Warshall’s algorithm (§13.4.2)
- Directed acyclic graphs (§13.4.3)
- Topological ordering (§13.4.3)
Digraphs

A digraph is a directed graph whose edges are all directed

Applications

- one-way streets
- flights
- task scheduling
Directed DFS

- We can specialize the traversal algorithms (DFS and BFS) to digraphs by traversing edges only along their direction.
- In the directed DFS algorithm, we have four types of edges:
  - discovery edges
  - back edges
  - forward edges
  - cross edges
- A directed DFS starting at a vertex $s$ determines the vertices reachable from $s$.
Transitive Closure

Given a digraph $G$, the transitive closure of $G$ is the digraph $G^*$ such that:

- $G^*$ has the same vertices as $G$
- if $G$ has a directed path from $u$ to $v$ ($u \neq v$), $G^*$ has a directed edge from $u$ to $v$

The transitive closure provides reachability information about a digraph.

We can compute the transitive closure in time $O(n(n + m))$ by repeated applications of directed DFS.
Floyd-Warshall’s Algorithm

- Floyd-Warshall’s algorithm numbers the vertices of a digraph $G$ as $v_1, \ldots, v_n$ and computes a series of digraphs $G_0, \ldots, G_n$
  - $G_0 = G$
  - $G_k$ has a directed edge $(v_i, v_j)$ if $G$ has a directed path from $v_i$ to $v_j$ with intermediate vertices in the set $\{v_1, \ldots, v_k\}$

We have that $G_n = G^*$

- In phase $k$, digraph $G_k$ is computed from $G_{k-1}$

Algorithm $FloydWarshall(G)$

Input digraph $G$
Output transitive closure $G^*$ of $G$

$i \leftarrow 1$

for all $v \in G.\text{vertices}()$
  - denote $v$ as $v_i$
  - $i \leftarrow i + 1$
  - $G_0 \leftarrow G$
for $k \leftarrow 1$ to $n$ do
  $G_k \leftarrow G_{k-1}$
  for $i \leftarrow 1$ to $n$ ($i \neq k$) do
    for $j \leftarrow 1$ to $n$ ($j \neq i, k$) do
      if $G_{k-1}.\text{areDirAdjacent}(v_p, v_k) \land G_{k-1}.\text{areDirAdjacent}(v_k, v_j)$
        if $\neg G_k.\text{areDirAdjacent}(v_p, v_j)$
          $G_k.\text{insertDirectedEdge}(v_p, v_j, k)$
return $G_n$
Example

$G = G_0 = G_1 = G_2$

$G_3$

$G_4 = G_5 = G^*$
A directed acyclic graph (DAG) is a digraph that has no directed cycles.

A topological ordering of a digraph is a numbering

\[ v_1, \ldots, v_n \]

of the vertices such that for every edge \((v_i, v_j)\), we have \(i < j\).

Example: in a task scheduling digraph, a topological ordering a task sequence that satisfies the precedence constraints.

Theorem

A digraph admits a topological ordering if and only if it is a DAG.
Topological Ordering

- A stack stores the vertices whose predecessors have all been numbered
- We store two labels with each vertex:
  - Counter of predecessors not yet numbered
  - Rank in the topological ordering
- The algorithm runs in time $O(n + m)$

Algorithm \textit{TopologicalOrdering}(G)

\begin{algorithm}
  \begin{itemize}
    \item $S \leftarrow \text{new stack}$
    \item \textbf{for all} $v \in G.\text{vertices}()$
      \begin{itemize}
        \item $\text{setCount}(v, \text{inDegree}(v))$
        \item \textbf{if} $\text{getCount}(v) = 0$
          \begin{itemize}
            \item $S.\text{push}(v)$
          \end{itemize}
      \end{itemize}
    \item $i \leftarrow 1$
    \item \textbf{while} $\neg S.\text{isEmpty}()$
      \begin{itemize}
        \item $u \leftarrow S.\text{pop}()$
        \item $\text{setRank}(u, i)$
        \item $i \leftarrow i + 1$
        \item \textbf{for all} $e \in G.\text{outgoingEdges}(u)$
          \begin{itemize}
            \item $z \leftarrow G.\text{opposite}(u, e)$
            \item $\text{setCount}(z, \text{getCount}(z) - 1)$
            \item \textbf{if} $\text{getCount}(z) = 0$
              \begin{itemize}
                \item $S.\text{push}(z)$
              \end{itemize}
          \end{itemize}
      \end{itemize}
  \end{itemize}
\end{algorithm}
Example
Example