Directed Graphs

A digraph is a graph whose edges are all directed.
- Short for "directed graph"

Applications
- One-way streets
- Flights
- Task scheduling

Digraph Properties

A graph \( G=(V,E) \) such that
- Each edge goes in one direction:
  - Edge \((a,b)\) goes from \(a\) to \(b\), but not \(b\) to \(a\).
- If \( G \) is simple, \( m \leq n^{*}(n-1) \).
- If we keep in-edges and out-edges in separate adjacency lists, we can perform listing of in-edges and out-edges in time proportional to their size.

Digraph Application

Scheduling: edge \((a,b)\) means task \(a\) must be completed before \(b\) can be started.

The good life
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 vertex \( s \) determines the vertices reachable from \( s \).

Reachability

- DFS tree rooted at \( v \): vertices reachable from \( v \) via directed paths

Strong Connectivity

- Each vertex can reach all other vertices.

Strong Connectivity Algorithm

- Pick a vertex \( v \) in \( G \).
- Perform a DFS from \( v \) in \( G \):
  - If there’s a \( w \) not visited, print “no”.
- Let \( G' \) be \( G \) with edges reversed.
- Perform a DFS from \( v \) in \( G' \):
  - If there’s a \( w \) not visited, print “no”.
  - Else, print “yes”.

Running time: \( O(n+m) \).
**Strongly Connected Components**

- Maximal subgraphs such that each vertex can reach all other vertices in the subgraph.
- Can also be done in $O(n+m)$ time using DFS, but is more complicated (similar to biconnectivity).

\[ \{a, c, g\} \]

\[ \{f, d, e, b\} \]

---

**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.

---

**Computing the Transitive Closure**

- We can perform DFS starting at each vertex:
  - $O(n(n+m))$

Alternatively ...

Use dynamic programming:

The Floyd-Warshall Algorithm

If there's a way to get from $A$ to $B$ and from $B$ to $C$, then there's a way to get from $A$ to $C$.

---

**Floyd-Warshall Transitive Closure**

- Idea #1: Number the vertices 1, 2, ..., $n$.
- Idea #2: Consider paths that use only vertices numbered 1, 2, ..., $k$, as intermediate vertices:

Uses only vertices numbered 1, ..., $k$

(add this edge if it's not already in)

Uses only vertices numbered 1, ..., $k$ 1

Uses only vertices numbered 1, ..., $k$ 1
Floyd-Warshall’s Algorithm

Floyd-Warshall’s algorithm numbers the vertices of $G$ 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 $G_{k-1}$

Running time: $O(n^3)$, assuming areAdjacent is $O(1)$ (e.g., adjacency matrix)

Algorithm FloydWarshall($G$)
Input digraph $G$
Output transitive closure $G^*$ of $G$

$i \leftarrow 1$
for all $v \in G$.vertices() do
    denote $v$ as $v_i$
    $i \leftarrow i + 1$
for $k \leftarrow 1$ to $n$ do
    $G_k \leftarrow G$
    for $i \leftarrow 1$ to $n$ (i.e., $i \neq k$) do
        for $j \leftarrow 1$ to $n$ (j.e., $j \neq i, k$) do
            if $G_{k-1}$.areAdjacent($v_i, v_j$) \&\& $G_{k-1}$.areAdjacent($v_j, v_k$)
                if $\neg G_k$.areAdjacent($v_i, v_j$)
                    $G_k$.insertDirectedEdge($v_i, v_j, k$)
    return $G_n$
Directed Graphs

Floyd-Warshall, Conclusion

DAGs and Topological Ordering

A directed acyclic graph (DAG) is a digraph that has no directed cycles.

A topological ordering of a digraph is a numbering 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 is a task sequence that satisfies the precedence constraints.

Theorem

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

Topological Sorting

Number vertices, so that \((u,v)\) in \(E\) implies \(u < v\) for a typical student day.

Algorithm for Topological Sorting

Note: This algorithm is different than the one in Goodrich-Tamassia.

Method: 

\[
\begin{align*}
H & \leftarrow G \quad // \text{Temporary copy of } G \\
& \text{for } n \leftarrow G.numVertices() \\
\text{while } H \text{ is not empty do} \\
& \text{Let } v \text{ be a vertex with no outgoing edges} \\
& \text{Label } v \leftarrow n \\
& \text{Remove } v \text{ from } H \\
\end{align*}
\]

Running time: \(O(n + m)\). How...?
Topological Sorting
Algorithm using DFS

Simulate the algorithm by using depth-first search.

Algorithm \text{topologicalDFS}(G, v)
\begin{itemize}
  \item Input graph \( G \) and a start vertex \( v \) of \( G \)
  \item Output \( G \) and a start vertex \( v \) of \( G \)
  \item \( n \leftarrow G.numVertices() \)
  \item for all \( u \in G\)\text{.vertices()} \( setLabel(u, \text{UNEXPLORED}) \)
  \item for all \( e \in G\)\text{.incidentEdges}(v) \( setLabel(e, \text{UNEXPLORED}) \)
  \item for all \( v \in G\)\text{.vertices()} \( if getLabel(v) = \text{UNEXPLORED} \)
    \( topologicalDFS(G, v) \)
\end{itemize}

\( O(n+m) \) time.

Topological Sorting Example

Algorithm \text{topologicalDFS}(G, v)
\begin{itemize}
  \item Input graph \( G \) and a start vertex \( v \) of \( G \)
  \item Output \( G \) and a start vertex \( v \) of \( G \)
  \item \( n \leftarrow G.numVertices() \)
  \item for all \( u \in G\)\text{.vertices()} \( setLabel(u, \text{UNEXPLORED}) \)
  \item for all \( e \in G\)\text{.edges()} \( setLabel(e, \text{UNEXPLORED}) \)
  \item for all \( v \in G\)\text{.vertices()} \( if getLabel(v) = \text{UNEXPLORED} \)
    \( topologicalDFS(G, v) \)
\end{itemize}

Label \( v \) with topological number \( n \)
\( n \leftarrow n - 1 \)

Topological Sorting Example

Topological Sorting Example

Topological Sorting Example
Topological Sorting Example