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, $m \leq n \cdot (n - 1)$
- If we keep in-edges and out-edges in separate adjacency lists, we can perform listing of incoming edges and outgoing edges in time proportional to their size

Digraph Application

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

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).

Strongly Connected Components

{ 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.

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,...,$k$

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

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

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

(add this edge if it’s not already in)
Floyd-Warshall’s Algorithm

- Number vertices \( v_1, \ldots, v_n \)
- Compute digraphs \( G_0, \ldots, G_n \)
  - \( G_i = G \)
  - \( G_k \) has directed edge \( (v_i, v_j) \) if \( G \) has a directed path from \( v_i \) to \( v_j \) with intermediate vertices in \( \{v_1, \ldots, v_k\} \)
- 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 \( \text{FloydWarshall}(G) \)

Input: digraph \( G \)
Output: transitive closure \( G^* \) of \( G \)

\[
\begin{align*}
\text{for all } v \in G.\text{vertices}() & \quad \text{denote } v \text{ as } v_i \\
i & \leftarrow 1 \\
\text{for } i \leftarrow 1 \text{ to } n \text{ do} & \\
\quad G_k & \leftarrow G_{k-1} \\
\quad \text{for } i \leftarrow 1 \text{ to } n \text{ (} i \neq k \text{) do} & \\
\quad \quad \text{for } j \leftarrow 1 \text{ to } n \text{ (} j \neq i, k \text{) do} & \\
\quad \quad \quad \text{if } G_{k-1}.\text{areAdjacent}(v_i, v_k) \land G_{k-1}.\text{areAdjacent}(v_k, v_j) & \\
\quad \quad \quad \quad \quad \text{if } \neg G_k.\text{areAdjacent}(v_i, v_j) & \\
\quad \quad \quad \quad \quad \quad \quad G_k.\text{insertDirectedEdge}(v_i, v_j, k) & \\
\quad \quad \quad \quad \quad \text{return } G_n & \\
\end{align*}
\]

Floyd-Warshall Example

Floyd-Warshall, Iteration 1

Floyd-Warshall, Iteration 2
### 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 \( v_1, \ldots, v_n \) 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.

### Algorithm for Topological Sorting

- Number vertices, so that \((u,v)\) in \(E\) implies \(u < v\).

#### A typical student day

1. Wake up
2. Study computer sci.
3. Eat
4. Nap
5. More c.s.
6. Work out
7. Play
8. Write c.s. program
9. Bake cookies
10. Sleep
11. Dream about graphs

#### Algorithm

```plaintext
Algorithm TopologicalSort(G)
1. H ← G // Temporary copy of G
2. n ← G.numVertices()
3. while H is not empty do
   4. Let v be a vertex with no outgoing edges
   5. Label v ← n
   6. n ← n − 1
   7. Remove v from H
```

- Running time: \(O(n + m)\)
Implementation with DFS

- Simulate the algorithm by using depth-first search
- \(O(n+m)\) time.

Algorithm \textit{topologicalDFS}(G, v)
\begin{algorithm}
\textbf{Input} graph \(G\) and a start vertex \(v\) of \(G\)
\textbf{Output} labeling of the vertices of \(G\) in the connected component of \(v\)
\begin{itemize}
  \item setLabel\((v, \text{VISITED})\)
  \item for all \(e \in G\).outEdges\((v)\)
    \begin{itemize}
      \item outgoing edges
      \item \(w \leftarrow \text{opposite}(v, e)\)
      \item if getLabel\((w) = \text{UNEXPLORED}\)
        \begin{itemize}
          \item \(e\) is a discovery edge
          \item topologicalDFS\((G, w)\)
        \end{itemize}
      \item else
        \begin{itemize}
          \item \(e\) is a forward or cross edge
        \end{itemize}
    \end{itemize}
\item Label \(v\) with topological number \(n\)
\item \(n \leftarrow n - 1\)
\end{itemize}
\end{algorithm}

Topological Sorting Example

9

Topological Sorting Example

8
9
Topological Sorting Example

Diagram showing a directed graph with vertices labeled 7, 8, 9, 6, 5, 4, 8, 7, 9, and 4, illustrating the process of topological sorting.