Directed Graphs

- Directed Graphs
- Representation
- Search
- Topological Sorting
- Directed Acyclic Graphs
- Strongly Connected Components
- Implicit Graphs

Directed Graph

A set of vertices pairwise joined by directed edges.

Vertex = intersection, edge = (one-way) road.
Garbage Collection
- Vertex = object, edge = pointer/reference.
- Which objects are reachable from a root?

WWW
- Vertex = homepage, edge = hyperlink.
- Web Crawling
- PageRank

Automata and Regular Expressions
- Vertex = state, edge = state transition.
- Does the automaton accept “aab” = is there a path from 1 to 10 that matches “aab”? Regular expressions can be represented as automata.

 Dependencies
- Vertices = topics, edge = dependency.
- Are there any cyclic dependencies? Can we find an ordering of vertices that avoids cyclic dependencies?

R = a·(a·)·(b(c)
Dependencies

- Lemma. $\sum_{v \in V} \deg(v) = \sum_{v \in V} \deg^-(v) = m$.
- Bevis. Every edge has exactly one start and end vertex.

Directed Graphs

Applications

<table>
<thead>
<tr>
<th>Graph</th>
<th>Vertices</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>internet</td>
<td>homepage</td>
<td>hyperlink</td>
</tr>
<tr>
<td>transport</td>
<td>intersection</td>
<td>one-way road</td>
</tr>
<tr>
<td>scheduling</td>
<td>job</td>
<td>precedence relation</td>
</tr>
<tr>
<td>disease outbreak</td>
<td>person</td>
<td>infects relation</td>
</tr>
<tr>
<td>citation</td>
<td>paper</td>
<td>citation</td>
</tr>
<tr>
<td>object graph</td>
<td>objects</td>
<td>pointers/references</td>
</tr>
<tr>
<td>object hierarchy</td>
<td>class</td>
<td>inheritance</td>
</tr>
<tr>
<td>control-flow</td>
<td>code</td>
<td>jump</td>
</tr>
</tbody>
</table>

Algorithmic Problems on Directed Graphs

- Path. Is there a path from s to t?
- Shortest path. What is the shortest path from s to t.
- Directed acyclic graph. Is there a cycle in the graph?
- Topological sorting. Can we order the vertices such that all edges are directed in same direction?
- Strongly connected component. Is there a path between all pairs of vertices?
- Transitive closure. For which vertices is there a path from v to w?
Directed Graphs

- Directed Graphs
- Representation
- Search
- Topological Sorting
- Directed Acyclic Graphs
- Strongly Connected Components
- Implicit Graphs

Representation

- Directed graph G with n vertices and m edges.
- **Representation.** We need the following operations on directed graphs.
  - **POINTS**\(v, u\): determine if \(v\) points to \(u\).
  - **NEIGHBORS**\(v\): return all vertices that \(v\) points to.
  - **INSERT**\(v, u\): add edge \((v, u)\) to \(G\) (unless it is already there).

Adjacency Matrix

- Directed graph G with n vertices and m edges.
- **Adjacency matrix.**
  - 2D \(n \times n\) array \(A\).
  - \(A[i][j] = 1\) if \(i\) points to \(j\), \(0\) otherwise.
- **Space.** \(O(n^2)\)
- **Time.**
  - **POINTS** in \(O(1)\) time.
  - **NEIGHBORS** in \(O(n)\) time.
  - **INSERT**\(v, u\) in \(O(1)\) time.

Adjacency List

- Directed graph G with n vertices and m edges.
- **Adjacency list.**
  - Array \(A[0..n-1]\).
  - \(A[i]\) is a linked of all nodes that \(i\) points to.
- **Space.** \(O(n + \sum_{v \in V} \deg(v)) = O(n + m)\)
- **Time.**
  - **POINTS**\(v\), **NEIGHBORS** and **INSERT** in \(O(\deg(v))\) time.
### Representation

<table>
<thead>
<tr>
<th>Data structure</th>
<th>PointsTo</th>
<th>Neighbors</th>
<th>Insert</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>adjacency matrix</td>
<td>O(1)</td>
<td>O(n)</td>
<td>O(1)</td>
<td>O(n²)</td>
</tr>
<tr>
<td>adjacency list</td>
<td>O(deg(v))</td>
<td>O(deg(v))</td>
<td>O(deg(v))</td>
<td>O(n+m)</td>
</tr>
</tbody>
</table>

### Directed Graphs

- Directed Graphs
- Representation
- Search
  - Topological Sorting
  - Directed Acyclic Graphs
  - Strongly Connected Components
- Implicit Graphs

### Søgning

- Depth first search from s.
  - Unmark all vertices and visit s.
  - Visit vertex s:
    - Mark v.
    - Visit all unmarked neighbors that v points to recursively.
- Breadth first search from s.
  - Unmark all vertices and initialize queue Q.
  - Mark s and Q.QUEUE(s).
  - While Q is not empty:
    - v = Q.DEQUEUE().
    - For each unmarked neighbor u that v points to:
      - Mark u.
      - Q.QUEUE(u).
- Time. O(n + m)
Topological Sorting

- **Topological sorting.** Ordering of vertices $v_0, v_1, \ldots, v_{n-1}$ from left to right such that all edges are directed to the right.

- **Challenge.** Compute a topological sorting or determine that none exists.

```
0  1  6  4  3  2  5
```

- **Algorithm.**
  - Find $v$ with in-degree 0.
  - Output $v$ and recurse on $G - \{v\}$. 

```
0  1  6  4  3  2  5
```

```
Correctness?
- Lemma. \( G \) has topological sorting \( \iff \) \( G \) has vertex \( v \) with in-degree 0 and \( G - \{v\} \) has topological sorting.

Challenge. How do we implement algorithm efficiently on adjacency list representation?

Solution 1. Construct reverse graph \( G^R \).
- Search in adjacency list representation of \( G^R \) to find vertex \( v \) with in-degree 0.
- Remove \( v \) and edges out of \( v \).
- Put \( v \) leftmost.
- Repeat.

Time per vertex.
- Find vertex \( v \) with in-degree 0: \( O(n) \).
- Remove edges out of \( v \): \( O(\text{deg}(v)) \).
- Total time, \( O(n^2 + \sum_{v \in V} \text{deg}(v)) = O(n^2 + m) = O(n^2). \)

Solution 2. Maintain in-degree of every vertex + linked list of all vertices with in-degree 0.
- Remove \( v \) and edges out of \( v \).
- Put \( v \) leftmost.
- Repeat.

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

Time per vertex.
- Remove vertex \( v \) with in-degree 0: \( O(1) \).
- Remove edges out of \( v \): \( O(\text{deg}(v)) \).
- Total time, \( O(n + \sum_{v \in V} \text{deg}(v)) = O(n + m) = O(n + m). \)
Directed Graphs

- Directed Graphs
- Representation
- Search
- Topological Sorting
- Directed Acyclic Graphs
- Strongly Connected Components
- Implicit Graphs

Directed Acyclic Graphs

- Directed acyclic graph (DAG). A graph is a DAG if it contains no directed cycles.

- Challenge. Determine whether or not $G$ is a DAG.
- Equivalence of DAGs and topological sorting. A graph $G$ is a DAG if and only if it has a topological sorting (see exercises).

- Algorithm.
  - Compute a topological sorting.
  - If success output yes, otherwise no.
- Time. $O(n + m)$

Strongly Connected Components

- Definition. $v$ and $u$ are strongly connected if there is a path from $v$ to $u$ and $u$ to $v$.
- Definition. A strongly connected component is a maximal subset of strongly connected vertices.

- Theorem. We can compute the strongly connected components in a graph in $O(n + m)$ time.
- See CLRS 22.5.
Implicit Graphs
- Implicit graph. Undirected/directed graph with implicit representation.
- Implicit representation.
  - Start vertex s + algorithm to generate neighbors of a vertex.
- Applications. Games, AI, etc.

Directed Graphs
- Directed Graphs
- Representation
- Search
- Topological Sorting
- Directed Acyclic Graphs
- Strongly Connected Components
- Implicit Graphs

Implicit Graphs
- Rubik's cube
  - \( n+m = 43,252,003,274,856,000 \approx 43 \) trillions.
  - What is the smallest number of moves needed to solve a cube from any starting configuration?

<table>
<thead>
<tr>
<th>year</th>
<th>lower bound</th>
<th>upper bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>1981</td>
<td>18</td>
<td>52</td>
</tr>
<tr>
<td>1990</td>
<td>18</td>
<td>42</td>
</tr>
<tr>
<td>1992</td>
<td>18</td>
<td>39</td>
</tr>
<tr>
<td>1992</td>
<td>18</td>
<td>37</td>
</tr>
<tr>
<td>1995</td>
<td>18</td>
<td>29</td>
</tr>
<tr>
<td>1995</td>
<td>20</td>
<td>29</td>
</tr>
<tr>
<td>2005</td>
<td>20</td>
<td>28</td>
</tr>
<tr>
<td>2006</td>
<td>20</td>
<td>27</td>
</tr>
<tr>
<td>2007</td>
<td>20</td>
<td>26</td>
</tr>
<tr>
<td>2008</td>
<td>20</td>
<td>25</td>
</tr>
<tr>
<td>2008</td>
<td>20</td>
<td>23</td>
</tr>
<tr>
<td>2008</td>
<td>20</td>
<td>22</td>
</tr>
<tr>
<td>2010</td>
<td>20</td>
<td>20</td>
</tr>
</tbody>
</table>