Graph Basics

Lecturer: Shi Li

Department of Computer Science and Engineering
University at Buffalo
Outline

Graphs

Connectivity and Graph Traversal
Testing Bipartiteness

Topological Ordering
Examples of Graphs

Figure: Road Networks

Figure: Social Networks

Figure: Internet

Figure: Transition Graphs
(Undirected) Graph $G = (V, E)$

- $V$: set of vertices (nodes);
  - $V = \{1, 2, 3, 4, 5, 6, 7, 8\}$
- $E$: pairwise relationships among $V$;
  - (undirected) graphs: relationship is symmetric, $E$ contains subsets of size 2
  - $E = \{\{1, 2\}, \{1, 3\}, \{2, 3\}, \{2, 4\}, \{2, 5\}, \{3, 5\}, \{3, 7\}, \{3, 8\}, \{4, 5\}, \{5, 6\}, \{7, 8\}\}$
For (undirected) graphs, we often use \((i, j)\) to denote the set \(\{i, j\}\).

We call \((i, j)\) an unordered pair; in this case \((i, j) = (j, i)\).

\[
E = \{(1, 2), (1, 3), (2, 3), (2, 4), (2, 5), (3, 5), (3, 7), (3, 8), (4, 5), (5, 6), (7, 8)\}
\]
- Social Network: Undirected
- Transition Graph: Directed
- Road Network: Directed or Undirected
- Internet: Directed or Undirected
Representation of Graphs

► Adjacency matrix
  ► $n \times n$ matrix, $A[u, v] = 1$ if $(u, v) \in E$ and $A[u, v] = 0$ otherwise
  ► $A$ is symmetric if graph is undirected

► Linked lists
  ► For every vertex $v$, there is a linked list containing all neighbours of $v.$
Comparison of Two Representations

- Assuming we are dealing with undirected graphs
- $n$: number of vertices
- $m$: number of edges, assuming $n - 1 \leq m \leq n(n - 1)/2$
- $d_v$: number of neighbors of $v$

<table>
<thead>
<tr>
<th></th>
<th>Matrix</th>
<th>Linked Lists</th>
</tr>
</thead>
<tbody>
<tr>
<td>memory usage</td>
<td>$O(n^2)$</td>
<td>$O(m)$</td>
</tr>
<tr>
<td>time to check $(u, v) \in E$</td>
<td>$O(1)$</td>
<td>$O(d_u)$</td>
</tr>
<tr>
<td>time to list all neighbours of $v$</td>
<td>$O(n)$</td>
<td>$O(d_v)$</td>
</tr>
</tbody>
</table>
Outline

Graphs

Connectivity and Graph Traversal
  Testing Bipartiteness

Topological Ordering
Connectivity Problem

**Input:** graph $G = (V, E)$, (using linked lists)

- two vertices $s, t \in V$

**Output:** whether there is a path connecting $s$ to $t$ in $G$

- Algorithm: starting from $s$, search for all vertices that are reachable from $s$ and check if the set contains $t$
  - Breadth-First Search (BFS)
  - Depth-First Search (DFS)
Breadth-First Search (BFS)

- Build layers $L_0, L_1, L_2, L_3, \cdots$
- $L_0 = \{s\}$
- $L_{j+1}$ contains all nodes that are not in $L_0 \cup L_1 \cup \cdots \cup L_j$ and have an edge to a vertex in $L_j$
Implementing BFS using a Queue

**BFS**

```plaintext
1: head ← 1, tail ← 1, queue[1] ← s
2: mark s as “visited” and all other vertices as “unvisited”
3: while head ≥ tail do
4:   v ← queue[tail], tail ← tail + 1
5:   for all neighbours u of v do
6:     if u is “unvisited” then
7:       head ← head + 1, queue[head] = u
8:     mark u as “visited”
```

► Running time: $O(n + m)$. 
Example of BFS via Queue
Depth-First Search (DFS)

- Starting from $s$
- Travel through the first edge leading out of the current vertex
- When reach an already-visited vertex ("dead-end"), go back
- Travel through the next edge
- If tried all edges leading out of the current vertex, go back
Implementing DFS using Recursion

DFS(s)

1: mark all vertices as “unvisited”
2: recursive-DFS(s)

recursive-DFS(v)

1: mark v as “visited”
2: for all neighbours u of v do
3: if u is unvisited then recursive-DFS(u)
Outline

Graphs

Connectivity and Graph Traversal
  Testing Bipartiteness

Topological Ordering
Def. A graph $G = (V, E)$ is a bipartite graph if there is a partition of $V$ into two sets $L$ and $R$ such that for every edge $(u, v) \in E$, we have either $u \in L, v \in R$ or $v \in L, u \in R$. 
Testing Bipartiteness

- Taking an arbitrary vertex \( s \in V \)
- Assuming \( s \in L \) w.l.o.g
- Neighbors of \( s \) must be in \( R \)
- Neighbors of neighbors of \( s \) must be in \( L \)
- \( \ldots \)
- Report “not a bipartite graph” if contradiction was found
- If \( G \) contains multiple connected components, repeat above algorithm for each component
Test Bipartiteness

bad edges!
Testing Bipartiteness using BFS

**BFS** (*s*)

1. \( head \leftarrow 1, \ tail \leftarrow 1, \ queue[1] \leftarrow s \)
2. mark \( s \) as “visited” and all other vertices as “unvisited”
3. \( color[s] \leftarrow 0 \)
4. **while** \( head \geq tail \) **do**
5. \( v \leftarrow queue[tail], \ tail \leftarrow tail + 1 \)
6. **for** all neighbours \( u \) of \( v \) **do**
7. \[ \text{if } u \text{ is “unvisited” then} \]
8. \( head \leftarrow head + 1, \ queue[head] = u \)
9. \( \text{mark } u \text{ as “visited”} \)
10. \( color[u] \leftarrow 1 - color[v] \)
11. **else if** \( color[u] = color[v] \) **then**
12. \( \text{print(“G is not bipartite”) and exit} \)
Testing Bipartiteness using BFS

1: mark all vertices as “unvisited”
2: for each vertex $v \in V$ do
3:   if $v$ is “unvisited” then
4:     test-bipartiteness$(v)$
5:   print(“$G$ is bipartite”)

Obs. Running time of algorithm = $O(n + m)$
Outline

Graphs

Connectivity and Graph Traversal
  Testing Bipartiteness

Topological Ordering
Topological Ordering Problem

**Input:** a directed acyclic graph (DAG) $G = (V, E)$

**Output:** 1-to-1 function $\pi : V \to \{1, 2, 3 \cdots , n\}$, so that

- if $(u, v) \in E$ then $\pi(u) < \pi(v)$
Algorithm: each time take a vertex without incoming edges, then remove the vertex and all its outgoing edges.
Topological Ordering

- Algorithm: each time take a vertex without incoming edges, then remove the vertex and all its outgoing edges.

Q: How to make the algorithm as efficient as possible?

A:
- Use linked-lists of outgoing edges
- Maintain the in-degree $d_v$ of vertices
- Maintain a queue (or stack) of vertices $v$ with $d_v = 0$
topological-sort\((G)\)

1: let \(d_v \leftarrow 0\) for every \(v \in V\)
2: for every \(v \in V\) do
3: for every \(u\) such that \((v, u) \in E\) do
4: \(d_u \leftarrow d_u + 1\)
5: \(S \leftarrow \{v : d_v = 0\}, i \leftarrow 0\)
6: while \(S \neq \emptyset\) do
7: \(v \leftarrow\) arbitrary vertex in \(S\), \(S \leftarrow S \setminus \{v\}\)
8: \(i \leftarrow i + 1, \pi(v) \leftarrow i\)
9: for every \(u\) such that \((v, u) \in E\) do
10: \(d_u \leftarrow d_u - 1\)
11: if \(d_u = 0\) then add \(u\) to \(S\)
12: if \(i < n\) then output “not a DAG”

- \(S\) can be represented using a queue or a stack
- Running time = \(O(n + m)\)
### $S$ as a Queue or a Stack

<table>
<thead>
<tr>
<th>DS</th>
<th>Queue</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>$\text{head} \leftarrow 0, \text{tail} \leftarrow 1$</td>
<td>$\text{top} \leftarrow 0$</td>
</tr>
<tr>
<td>Non-Empty?</td>
<td>$\text{head} \geq \text{tail}$</td>
<td>$\text{top} &gt; 0$</td>
</tr>
</tbody>
</table>
| Add($v$)      | $\text{head} \leftarrow \text{head} + 1$  
$S[\text{head}] \leftarrow v$ | $\text{top} \leftarrow \text{top} + 1$  
$S[\text{top}] \leftarrow v$ |
| Retrieve $v$  | $v \leftarrow S[\text{tail}]$  
$\text{tail} \leftarrow \text{tail} + 1$ | $v \leftarrow S[\text{top}]$  
$\text{top} \leftarrow \text{top} - 1$ |
Example