Graph Algorithms

What is a graph?

V - vertices (or nodes)

E \subseteq V \times V - edges

Representation:

- adjacency matrix
- adjacency lists

Why graphs?

V = \{A, B, C, D, E\}

E = \{(A,B), (B,A), (C,B), (D,C), (E,B), (B,E)\}

directed: \text{sum of outdegrees} = |E|

degree, i.e., \# adjacent edges to this vertex

\text{sum of the degrees:}

3 + 3 + 2 + 4 + 4 + 4 + 3 + 3 = 26

= 2|E| \quad \text{i.e.}

|E| = 13
Graph Algorithms

What is a graph?

\[ V - \text{vertices} \]

\[ E \subseteq V \times V - \text{edges} \]

Representation:

- adjacency matrix
- adjacency lists

Why graphs?

<table>
<thead>
<tr>
<th>Operation</th>
<th>Adjacency Matrix</th>
<th>Adjacency List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checking Adjacency</td>
<td>( O(1) )</td>
<td>( O(\text{deg}) )</td>
</tr>
<tr>
<td>Listing All Neighbors</td>
<td>( O(n) )</td>
<td>( O(\text{deg}) )</td>
</tr>
</tbody>
</table>
Graph Algorithms

Graph properties:

- **connected**: exists a path from every vertex to every other vertex
- **cyclic**: exists a cycle
- ...

Tree - a connected acyclic (undirected) graph

\[ m = n - 1 \quad \text{for trees} \]

\[ \text{i.e. } m = O(n) \]
Graph Traversals

Objective: list all vertices reachable from a given vertex s

assuming: adjacency lists

breadth-first search (BFS)

queue: S 1 2 3

level 0

level 1

level 2

level 3

depth-first search (DFS)
**Breadth-first search (BFS)**

Finds all vertices “reachable” from a starting vertex.

Byproduct: computes distances from the starting vertex to every vertex.

---

**BFS** ( \( G=(V,E), s \) )

1. seen\[v\]=false, dist\[v\]=\( \infty \) for every vertex \( v \)
2. beg=1; end=2; \( Q[1]=s \); seen\[s\]=true; dist\[s\]=0;
3. while (beg<end) do
   4. head=\( Q[\text{beg}] \);
   5. for every \( u \) s.t. (head,\( u \)) is an edge and not seen\[u\] do
      6. \( Q[\text{end}]=u \); dist\[u\]=dist[head]+1;
      7. seen\[u\]=true; end++;
   8. beg++;

*runtime: \( O(n+m) \)*
Depth-first search (DFS)

Finds all vertices “reachable” from a starting vertex, in a different order than BFS.

**DFS-RUN** ( \( G=(V,E), s \) )
1. \( \text{seen}[v]=\text{false} \) for every vertex \( v \)
2. \( \text{DFS}(s) \)

**DFS(v)**
1. \( \text{seen}[v]=\text{true} \)
2. for every neighbor \( u \) of \( v \)
3. if not \( \text{seen}[u] \) then \( \text{DFS}(u) \)

Running time:
\( O(n+m) \)
Applications of DFS: topological sort

Def: A **topological sort** of an acyclic directed graph is an order of vertices such that every edge goes from "left to right."

**Idea 1:**
1. create an empty order
2. while there are still vertices:
3. find a vertex v with no incoming edges
4. place v at the end of the order
5. remove all outgoing edges of v
6. remove v

Running time: about $O(n^2)$
Applications of DFS: topological sort

TopSort ( G=(V,E) )
1. for every vertex v
2. seen[v]=false
3. fin[v]=∞
4. time=0
5. for every vertex s
6. if not seen[s] then
7. DFS(s)

DFS(v)
1. seen[v]=true
2. for every neighbor u of v
3. if not seen[u] then
4. DFS(u)
5. time++
6. fin[v]=time (and output v)
Applications of DFS: topological sort

`TopSort ( G=(V,E) )`
1. for every vertex v
2. seen[v]=false
3. fin[v]=\(\infty\)
4. time=0
5. for every vertex s
6. if not seen[s] then
7. DFS(s)

`DFS(v)`
1. seen[v]=true
2. for every neighbor u of v
3. if not seen[u] then
4. DFS(u)
5. time++
6. fin[v]=time (and output v)

What if the graph contains a cycle?
Appl. DFS: strongly connected components

Vertices $u, v$ are in the same **strongly connected component** if there is a (directed) path from $u$ to $v$ and from $v$ to $u$. 
Apply DFS: strongly connected components

Vertices $u, v$ are in the same strongly connected component if there is a (directed) path from $u$ to $v$ and from $v$ to $u$.

How to find strongly connected components?
Appl. DFS: strongly connected components

STRONGLY-CONNECTED COMPONENTS ( G=(V,E) )

1. for every vertex v
2. seen[v]=false
3. fin[v]=1
4. time=0
5. for every vertex s
6. if not seen[s] then
7. DFS(G,s) (the finished-time version)
8. compute G^T by reversing all arcs of G
9. sort vertices by decreasing finished time
10. seen[v]=false for every vertex v
11. for every vertex v do
12. if not seen[v] then
13. output vertices seen by DFS(v)
Many other applications of D/BFS

DFS
• find articulation points
• find bridges

BFS
• e.g. sokoban