Graph Algorithms

What is a graph?

\[ |V| = n \]
\[ |E| = m \]

\( V \) - vertices
\[ E \subseteq V \times V \] - edges

Representation:
- adjacency matrix
- adjacency lists

Why graphs?

Directed / undirected

Vertex degrees:
\[ \text{deg}(v) = \# \text{neighbors} \]
\[ = \text{# outgoing edges} \]
Graph Algorithms

Graph properties:

- connected - can get from any vertex to any vertex by a path (sequence of "connected" edges)
- cyclic - \( \exists \text{ a cycle} \)
- ...

Tree - a connected acyclic (undirected) graph
Graph Traversals

Objective: list all vertices reachable from a given vertex s

 DFS (depth-first search)

 √ → vertex has been visited

 (note: left & right picture does not use same ordering of neighbors)

 BFS (breadth-first search)

 dist → order in which vertex discovered
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]=∞ 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[beg];
5.  for every u s.t. (head,u) is an edge and not seen[u] do
7.  seen[u]=true; end++;
8.  beg++;
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. seen[v] = false for every vertex v
2. **DFS(s)**

**DFS(v)**
1. seen[v] = true, timestamp[v] = time, time++
2. for every neighbor u of v
3. if not seen[u] then DFS(u)

Running time:

\[O(\Sigma \text{deg} + n)\]

\[= O(m)\] assuming G is connected

because \(\Sigma \text{deg} = 2m\)
because every edge is visited twice

Side comment:

\[m \leq \binom{n}{2} = \frac{n^2-n}{2} = O(n^2)\]
Applications of DFS: topological sort

Def: A topological sort of a directed graph is an order of vertices such that every edge goes from “left to right.”
Applications of DFS: topological sort

Blue ordering:
0 1 2 3 4 5 6 7

Blue > timestamp for being done w. the vertex.
Applications of DFS: topological sort

7 6 5 4 3 2 1 0
Applications of DFS: topological sort

If graph cyclic
   - can run the algo
     but will not get a top. ordering
       (none exists)

7 6 5 4 3 2 1 0
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)

Applications of DFS: topological sort
Applications of DFS: topological sort

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

DFS(v)
1. \text{seen}[v]=true
2. for every neighbor u of v
3. if not seen[u] then
4. DFS(u)
5. time++
6. \text{fin}[v]=\text{time} \ (\text{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 \( u \).

A **strongly connected component** is a part of the graph where any vertex can be reached from any other vertex by following a directed path.

- **Strongly connected:** A directed graph is strongly connected if you can get from any vertex to any other vertex.
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.

How to find strongly connected components?

We can solve by running DFS from each order (resetting the "seen" before each run).
STRONGLY-CONNECTED COMPONENTS (G=(V,E))

1. for every vertex v
2. seen[v] = false
3. fin[v] = \_1^{undet.}
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)
STRONGLY-CONNECTED COMPONENTS ( \( G=(V,E) \) )

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

DFS
• find articulation points → vertex, if removed, disconnect
• find bridges → edge, if removed, graph gets disconnected

BFS
• e.g. sokoban
Many other applications of D/BFS

BFS
• e.g. sokoban

Running time: \( O(n+m) \)

where

\( n \leq (ab)^2 \)

\( m \leq \binom{n}{2} \approx n^2 \)

\( \sum \deg(v) = 2m \)

\( \leq 4n \quad \Rightarrow \quad m \leq 2n \)