Graph Algorithms

What is a graph? \( G = (V, E) \)

- \( V \) - vertices \( |V| = n \)
- \( E \subseteq V \times V \) - edges \( |E| = m \)

Directed / undirected

Representation:
- adjacency matrix
- adjacency lists

Why graphs?

\begin{tabular}{|c|c|c|c|}
\hline
1 & 2 & 3 & 4 \\
\hline
1 & 0 & 1 & 0 & 1 \\
2 & 1 & 0 & 1 & 1 \\
3 & 0 & 1 & 0 & 0 \\
4 & | & 1 & 1 & 0 \\
\hline
\end{tabular}
Graph Algorithms

Graph properties:

- connected: if can get from every vtx to every vtx
- cyclic: if 3 cycle
- ...

Tree - a connected acyclic (undirected) graph

tree with n vertices
n-1 edges
Graph Traversals

Objective: list all vertices reachable from a given vertex $s$
Breadth-first search (BFS)

Finds all vertices “reachable” from a starting vertex.

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

Running time: $O(n+m)$

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

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

Steps 5-8, when accounting for all vertices:

\[ O(\text{deg}(u)+\text{deg}(w)+\ldots+\text{deg}(v_n)) \]
\[ = O(2m) = O(m) \]

Steps 4 & 9, across all vertices:

\[ O(n) \]

TOTAL: \( O(n+m) \)
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. seen[v]=false for every vertex v
2. DFS(s)

DFS(v)
1. seen[v]=true
2. for every neighbor u of v
3. if not seen[u] then DFS(u)
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.”

```plaintext
1) Find a node with no incoming edges, output it.
2) Remove the node with its adjacent edges.
3) Keep doing the same while nodes exist.
```

$O(n)$
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

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

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

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

What if the graph contains a cycle?
then, run TopSort and for the resulting order, check if every edge goes left to right → O(n+m)
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 $u$. 

![Diagram of strongly connected components](image-url)
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?
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