1 Basic Graph Definitions

- A graph $G = (V, E)$ consists of a finite set of vertices $V$ and a finite set of edges $E$.
  - Directed graphs: $E$ is a set of ordered pairs of vertices $(u, v)$ where $u, v \in V$
    
    - Undirected graph: $E$ is a set of unordered pairs of vertices $\{u, v\}$ where $u, v \in V$

  
  - Edge $(u, v)$ is incident to $u$ and $v$
  - Degree of vertex in undirected graph is the number of edges incident to it.
  - In (out) degree of a vertex in directed graph is the number of edges entering (leaving) it.
  - A path from $u_1$ to $u_2$ is a sequence of vertices $< u_1 = v_0, v_1, v_2, \ldots, v_k = u_2 >$ such that $(v_i, v_{i+1}) \in E$ (or $\{v_i, v_{i+1}\} \in E$)
    - We say that $u_2$ is reachable from $u_1$
    - The length of the path is $k$
    - It is a cycle if $v_0 = v_k$
  - An undirected graph is connected if every pair of vertices are connected by a path
    - The connected components are the equivalence classes of the vertices under the “reachability” relation. (All connected pair of vertices are in the same connected component).
• A directed graph is *strongly connected* if every pair of vertices are reachable from each other
  
  – The *strongly connected components* are the equivalence classes of the vertices under the “mutual reachability” relation.

• Graphs appear all over the place in all kinds of applications, e.g:
  
  – Trees ($|E| = |V| - 1$)
  – Connectivity/dependencies (house building plans, WWW-page connections = internet graph)

• Often the edges $(u, v)$ in a graph have weights $w(u, v)$, e.g.
  
  – Road networks (distances)
  – Cable networks (capacity)

1.1 **Representation**

• *Adjacency-list* representation:
  
  – Array of $|V|$ list of edges incident to each vertex.

Examples:

- Note: For undirected graphs, every edge is stored twice.
- If graph is weighted, a weight is stored with each edge.

• *Adjacency-matrix* representation:
- \( |V| \times |V| \) matrix \( A \) where

\[
a_{ij} = \begin{cases} 
1 & \text{if } (i, j) \in E \\
0 & \text{otherwise}
\end{cases}
\]

Examples:

- Note: For undirected graphs, the adjacency matrix is symmetric along the main diagonal \((A^T = A)\).
- If graph is weighted, weights are stored instead of one's.

- Comparison of matrix and list representation:

<table>
<thead>
<tr>
<th>Adjacency list</th>
<th>Adjacency matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>( O(</td>
<td>V</td>
</tr>
<tr>
<td>Good if graph sparse ((</td>
<td>E</td>
</tr>
<tr>
<td>No quick access to ((u, v))</td>
<td>( O(1) ) access to ((u, v))</td>
</tr>
</tbody>
</table>

- We will use adjacency list representation unless stated otherwise \(O(|V| + |E|)\) space).

2 Graph traversal

- There are two standard (and simple) ways of traversing all vertices/edges in a graph in a systematic way
  - Breadth-first
  - Depth-first

- We can use them in many fundamental algorithms, e.g finding cycles, connected components, ...
2.1 Breadth-first search (BFS)

- Main idea:
  - Start at some source vertex \( s \) and visit,
  - All vertices at distance 1,
  - Followed by all vertices at distance 2,
  - Followed by all vertices at distance 3,
  :

- BFS corresponds to computing shortest path distance (number of edges) from \( s \) to all other vertices.

- To control progress of our BFS algorithm, we think about coloring each vertex
  - \textit{White} before we start,
  - \textit{Gray} after we visit the vertex but before we have visited all its adjacent vertices,
  - \textit{Black} after we have visited the vertex and all its adjacent vertices (all adjacent vertices are gray).

- We use a queue \( Q \) to hold all gray vertices—vertices we have seen but are still not done with.

- We remember from which vertex a given vertex \( v \) is colored gray—i.e. the node that discovered \( v \) first; this is called parent\([v]\).

- Algorithm:

\[
\begin{align*}
\text{BFS}(s) & \\
\text{color}[s] &= \text{gray} \\
\text{d}[s] &= 0 \\
\text{ENQUEUE}(Q, s) \\
\text{WHILE} & \ Q \text{ not empty} \ DO \\
\qquad & \text{DEQUEUE}(Q, u) \\
\qquad & \text{FOR} \ (u, v) \in E \ DO \\
\qquad & \quad \text{IF} \ \text{color}[v] = \text{white} \ \text{THEN} \\
\qquad & \quad \quad \text{color}[v] = \text{gray} \\
\qquad & \quad \quad \text{d}[v] = \text{d}[u] + 1 \\
\qquad & \quad \quad \text{parent}[v] = u \\
\qquad & \quad \quad \text{ENQUEUE}(Q, v) \\
\qquad & \quad \FI \\
\qquad & \text{color}[u] = \text{black} \\
\text{OD}
\end{align*}
\]
- Algorithm runs in $O(|V| + |E|)$ time
- Example (for directed graph):

  a) ![Diagram](image1)
  b) ![Diagram](image2)
  c) ![Diagram](image3)
  d) ![Diagram](image4)
  e) ![Diagram](image5)
  f) ![Diagram](image6)
  g) ![Diagram](image7)
  h) ![Diagram](image8)
  i) ![Diagram](image9)

- Note:
  - parent$[v]$ forms a tree; **BFS-tree**.
  - $d[v]$ contains length of shortest path from $s$ to $v$. (Prove by induction)
  - We can use parent$[v]$ to find the shortest path from $s$ to a given vertex.
- If graph is not connected we have to try to start the traversal at all nodes.

```plaintext
FOR each vertex $u \in V$ DO
    IF color$[u] = \text{white}$ THEN BFS($u$)
OD
```
– Note: We can use algorithm to compute connected components in $O(|V| + |E|)$ time.

2.2 Depth-first search (DFS)

• If we use stack instead of queue $Q$ we get another traversal order; depth-first
  – We go “as deep as possible”,
  – Go back until we find unexplored adjacent vertex,
  – Go as deep as possible,
  ;
• Often we are interested in “start time” and “finish time” of vertex $u$
  – Start time ($d[u]$): indicates at what “time” vertex is first visited.
  – Finish time ($f[u]$): indicates at what “time” all adjacent vertices have been visited.
• We can write DFS iteratively using the same algorithm as for BFS but with a STACK instead of a QUEUE, or, we can write a recursive DFS procedure
  – We will color a vertex gray when we first meet it and black when we finish processing all adjacent vertices.

• Algorithm:

```
DFS(u)
  color[u] = gray
  d[u] = time
  time = time + 1
  FOR (u, v) $\in$ E DO
    IF color[v] = white THEN
      parent[v] = u
      DFS(v)
    FI
  OD
  color[u] = black
  f[u] = time
  time = time + 1
```

• Algorithm runs in $O(|V| + |E|)$ time
  – As before we can extend algorithm to unconnected graphs and we can use it to detect cycles in $O(|V| + |E|)$ time.
• Example:

a) 

b) 

c) 

d) 

e) 

f) 

g) 

h) 

i) 

j) 

k) 

l)
• As previously parent\[v\] forms a tree; DFS-tree
  
  – Note: If \(u\) is descendent of \(v\) in DFS-tree then \(d[v] < d[u] < f[u] < f[v]\)

3 Topological sorting

• Definition: Topological sorting of directed acyclic graph \(G = (V, E)\) is a linear ordering of vertices \(V\) such that \((u, v) \in E \Rightarrow u\) appear before \(v\) in ordering.

• Topological ordering can be used in scheduling:
  
  – Example: Dressing (arrow implies “must come before”)

We want to compute order in which to get dressed. One possibility:
The given order is one possible topological order.

- **Algorithm**: Topological order just reverse DFS finish time ($\Rightarrow O(|V| + |E|)$ running time).

- **Correctness**: $(u, v) \in E \iff f(v) < f(u)$
  
  - Proof: When $(u, v)$ is explored by DFS algorithm, $v$ must be white or black (gray $\Rightarrow$ cycle).
    - $v$ white: $v$ visited and finished before $u$ is finished $\Rightarrow f(v) < f(u)$
    - $v$ black: $v$ already finished $\Rightarrow f(v) < f(u)$

- **Alternative algorithm**: Count in-degree of each vertex and repeatedly number and remove in-degree 0 vertex and its outgoing edges: Homework.