1 Graphs

• $G(V, E)$ — $V$: vertex set; $E$: edge set.

• Directed graphs, undirected graphs, weighted graphs.

• No self-loops.

• Degree, in-degree, out-degree of a vertex.

• A path from a vertex $v$ to a vertex $v'$ is a sequence of vertices, $(v_0, v_1, \ldots, v_k)$, such that $v = v_0$, $v' = v_k$ and $(v_{i-1}, v_i) \in E$ for $i = 1, 2, \ldots, k$.

• Length of path: either the number of edges in the path or the total weight.

• Simple path: all vertices in the path are distinct.

• Cycle, simple cycle

• Acyclic graph

• Graph representations
  
  – Adjacency matrix
  
  – Adjacency lists
2 Basic Depth-First Search

procedure Search($G = (V, E)$)
  // Assume $V = \{1, 2, \ldots, n\}$
  // global array $visited[1..n]$
  $visited[1..n] \leftarrow 0$;
  for $i \leftarrow 1$ to $n$
    if $visited[i] = 0$ then call $dfs(i)$

procedure $dfs(v)$
  $visited[v] \leftarrow 1$;
  for each node $w$ such that $(v, w) \in E$
    if $visited[w] = 0$ then call $dfs(w)$

• How to implement the for-loop if an adjacency matrix $A$ is used to represent the graph?

• In the entire depth first search, how many times in total is $dfs()$ called?

• In the entire depth first search, how many times in total is the “if $visited[w] = 0$” part of the “if $visited[w] = 0$ then call $dfs(w)$” statement executed?

• Time complexity
  - Using adjacency matrix: $O(n^2)$
  - Using adjacency lists: $O(|V| + |E|)$
3 Connectivity

- An undirected graph is *connected* if every pair of vertices are connected by a path.

- A *connected component* is a subgraph which is connected and is not contained in any bigger connected subgraph.

- A connected component is usually identified by the vertices in that component.

- **Problem**: Given an undirected graph, identify all its connected components.

```plaintext
procedure Connected_Components(G = (V, E))
    // Assume V = {1, 2, ..., n} //
    // global array component[1..n] //
    component[1..n] ← 0
    cn ← 0
    for i ← 1 to n
        if component[i] = 0 then
            cn ← cn + 1
            call dfs(i, cn)

procedure dfs(v, cn)
    component[v] ← cn;
    for each node w such that (v, w) ∈ E do
        if component[w] = 0 then call dfs(w, cn)
```
4 Bipartite Graph

• Definition: An undirected graph \( G(V, E) \) is said to be bipartite if \( V \) can be divided into two sets \( V_1 \) and \( V_2 \) such that all edges in \( G \) go between \( V_1 \) and \( V_2 \).

• Theorem: An undirected graph is bipartite if and only if it contains no cycle of odd length.

• Problem: Given a graph, determine if it is bipartite.

procedure \( \text{Bipartite}(G = (V, E)) \)

// Assume \( V = \{1, 2, \ldots, n\} \) //
// global array \( \text{visited}[1..n] \), \( \text{flag} \) //
\( \text{visited}[1..n] \leftarrow 0; \)
\( \text{flag} \leftarrow \text{true}; \)
for \( i \leftarrow 1 \) to \( n \)
    if \( \text{visited}[i] = 0 \) then call \( \text{dfs}(i, 1) \)
return(\( \text{flag} \))

procedure \( \text{dfs}(v, c) \)
\( \text{visited}[v] \leftarrow c; \)
for each node \( w \) such that \( (v, w) \in E \) do
    if \( \text{visited}[w] = 0 \) then call \( \text{dfs}(w, -c) \)
    elseif \( \text{visited}[w] = c \) then \( \text{flag} \leftarrow \text{false}; \)
5 Advanced Depth-First Search

procedure Search\( (G = (V, E)) \)
\[
\text{// Assume } V = \{1, 2, \ldots, n\} \text{//}
\]
\[
time \leftarrow 0;
\]
\[
vn[1..n] \leftarrow 0; \quad /* vn stands for visit number */
\]
\[
\text{for } i \leftarrow 1 \text{ to } n
\]
\[
\quad \text{if } vn[i] = 0 \text{ then call } dfs(i)
\]

procedure dfs\( (v) \)
\[
vn[v] \leftarrow time \leftarrow time + 1;
\]
\[
\text{for each node } w \text{ such that } (v, w) \in E \text{ do}
\]
\[
\quad \text{if } vn[w] = 0 \text{ then call } dfs(w);
\]
\[
fn[v] \leftarrow time \leftarrow time + 1 \quad /* fn stands for finish number */
\]

- Depth first tree/forest, denoted as \( G_\pi \)
- Tree edges: those edges in \( G_\pi \)
- Forward edges: those non-tree edges \((u, v)\) connecting a vertex \( u \) to a descendant \( v \).
- Back edges: those edges \((u, v)\) connecting a vertex \( u \) to an ancestor \( v \).
- Cross edges: all other edges.
- If \( G \) is undirected, then there is no distinction between forward edges and back edges. Just call them back edges.
6 Topological Sort

- Problem: given a directed graph $G = (V, E)$, sort the vertices into a linear list such that for every edge $(u, v) \in E$, $u$ is ahead of $v$ in the list.

- Observation: the finish numbers in descending order gives such a list.

- Algorithm:
  - Use depth-first search, with an initially empty list $L$.
  - At the end of procedure $dfs(v)$, insert $v$ to the front of $L$.
  - $L$ gives a topological sort of the vertices.
7 Strongly Connected Components

- A directed graph is *strongly connected* if for every two nodes $u$ and $v$ there is a path from $u$ to $v$ and one from $v$ to $u$.

- Decide if a graph $G$ is strongly connected:
  
  - $G$ is strongly connected iff (i) there is a path from node 1 to every other node and (ii) there is a path from every other node to node 1.
  
  - Condition (1) can be checked by calling $dfs(1)$ on $G$ and then checking if all nodes have been visited.
  
  - Condition (2) can be checked by calling $dfs(1)$ on $G^T$ and then checking if all nodes have been visited, where $G^T$ is the graph obtained from $G$ by reversing the edges.

- A *strongly connected component* of a directed graph is a subgraph which is strongly connected and is not contained in any bigger strongly connected subgraph.

- An interesting problem is to find all strongly connected components of a directed graph.

- Each node belongs in exactly one component. So, we identify each component by its vertices.

- The component containing $v$ equals
  \[
  \{dfs(v) \text{ on } G\} \cap \{dfs(v) \text{ on } G^T\},
  \]
  where $\{dfs(v) \text{ on } G\}$ denotes the set of all vertices visited during $dfs(v)$ on $G$. 


• Algorithm:

1. Apply depth-first search to $G$ and compute $fn[u]$ for each node.
2. Compute $G^T$.
3. Apply depth-first search to $G^T$:

   $\text{visited}[1..n] \leftarrow 0$

   for each vertex $u$ in decreasing order of $fn[u]$ do

   if $\text{visited}[u] = 0$ then call $dfs(u)$

4. The vertices on each tree in the depth-first forest of the preceding step form a strongly connected component.