Graph Basics

Xiangyu Guo.
(I) What is a graph?

**Def.** $G = (V, E)$: a collection of nodes (vertices) and edges connecting vertices.

**Exmp.**

- Transportation network.
  - e.g. $V = \{ \text{cities} \}$, $E = \{ \text{highways} \}$
  - weight $w_{ij} = \text{distance/length of roads between city } i \text{ and } j$
  - direction: one-way road.
(II) Representation

1. Adjacency matrix

\[ G = (V, E) \]

\[ A = \begin{bmatrix} \vdots & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \]

\[ \text{degree } d(v) : \text{ # of edges touching } v \]

\[ A[i, j] = 1 \iff (i, j) \in E \]

\[ \nu(i, j) = w(i, j) \iff (i, j) \in E \& \nu(i, j) \]

Exmp:

Diagram of a graph with nodes 1, 2, 3, and 4, and an edge between nodes 1 and 2.
2. Adjacency List.
\[ G = (V, E) \], \( V \) is set of vertices

\[ V: \begin{align*}
&1 \rightarrow 2, 3, 4, 5, 6, 7, 8, 9 \\
&2 \rightarrow 1, 4, 1, 3 \\
&3 \rightarrow 1, 3, 2 \\
&4 \\
\end{align*} \rightarrow O(n) \]

Length of list of vertex \( v = d_v \)

\[ \text{Total size/length of lists} = \sum_v d_v \leq 2|E| = 2m \]

E.g. Assume every edge is undirected.

(i,j) appears exactly twice in all lists
**Complexity**

- $G = (V, E)$, $|V| = n$, $|E| = m$ (assume $0 \leq m \leq \frac{n(n-1)}{2}$)
- $d_v$: number of neighbors of $v$

<table>
<thead>
<tr>
<th></th>
<th>Adj Matrix</th>
<th>Adj List</th>
</tr>
</thead>
<tbody>
<tr>
<td>space usage</td>
<td>$O(n^2)$</td>
<td>$O(m+n)$</td>
</tr>
<tr>
<td>time to check</td>
<td>$O(1)$</td>
<td>$O(d_v) = O(m)$</td>
</tr>
<tr>
<td>if $(u, v) \in E$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>time to list</td>
<td>$O(n)$</td>
<td>$O(d_v)$</td>
</tr>
<tr>
<td>all neighbors of $v$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Terminologies

1. Path: $(s, t)$-path: sequence of edges.
   
   $(s, v_0), (v_0, v_1), \ldots, (v_n, t)$

   a) simple path: every vertex appears in the path at most once.

   Counter example:

2. Cycle: seq of edges with same beginning & ending.

   a) simple cycle: every vertex (except for the beginning one) appears \( \leq 1 \) time in the seq.
3. Tree: a type of graph
   - connected: $\exists$ path between every pair of vertices.
   - no cycles.
(II) Connectivity Problem

Problem:
Input: graph $G = (V, E)$
Two vertices $s, t \in V$
Output: "Yes" if $\exists (s, t)$-path
"No" if otherwise.

- Variants:
  - Shortest path problem: when edges have weight (distance)
  - Connected components:
Breadth-First Search (BFS)

Input: $G = (V, E), s \in V$
Output: All $t \in V$ s.t. $s$ & $t$ are connected.

Strategy:
Explore the graph level by level.

$L_0 = \{ s \}$

$L_1 = \{ all \ vertices \ that's \ not \ in \ L_0 \ but \ have \ an \ edge \ to \ L_0 \}$

$L_n = \{ all \ vertices \ that's \ not \ in \ L_0 \cup L_1 \cup \ldots \cup L_{n-1} \ but \ have \ an \ edge \ to \ L_0 \cup L_1 \cup \ldots \cup L_{n-1} \}$
Implementing BFS

- Remember vertices already visited:
  - \( \text{visited}[v] = 1 \)
- Use queue to traverse visited vertices
  - Adding new element
  - Deleting

**BFS(s)**

1. \( \text{head} \leftarrow 1, \text{tail} \leftarrow 1, \text{queue}[1] \leftarrow s \)
2. mark \( s \) as “visited” and all other vertices as “unvisited”
3. while \( \text{head} \geq \text{tail} \)
4. \( \text{v} \leftarrow \text{queue}[\text{tail}], \text{tail} \leftarrow \text{tail} + 1 \)
5. for all neighbours \( u \) of \( v \)
6. if \( u \) is “unvisited” then
7. \( \text{head} \leftarrow \text{head} + 1, \text{queue}[\text{head}] = u \)
8. mark \( u \) as “visited”
Running time:

$$O(n+2m) = O(n+m)$$

$$m = |E|, \ n = |V|$$
Depth-First Search (DFS)

Strategy:
Try to go as deep as you can. If reach a dead end, the backtrack.

Let's say we're visiting vertex $V$

- "can go deeper": if $\exists$ nbr $U$ of $V$ is unvisited
- "dead end": if all its nbrs are visited
- "back track": find the vertex explored before $V$. 
Implementing DFS

$s$ : starting vertex

$\text{DFS}(s)$
- mark all vertices as unvisited.
- recursive-$\text{DFS}(s)$

recursive-$\text{DFS}(u)$
- mark $u$ as visited.
- for each neighbor $v$ of $u$:
  - if $v$ is unvisited
    - recursive-$\text{DFS}(v)$

Running time: $O(n + m)$
**DFS - iterative (G, v)**

S ← empty stack
S. push(v)

While S is not empty:

\[ u \leftarrow S. \text{pop}() \quad \Rightarrow \quad 2m \text{ times} \]

if \( u \) is not visited:

mark \( u \) as visited \( \Rightarrow \quad n \text{ times} \)

for all nbs \( w \) of \( u \):

S. push(w) \( \Rightarrow \quad d_u \text{ times for each vertex } u \)

\[ \sum_u d_u = 2|E| = 2m \]

\[ \Rightarrow \quad O(n + 4m) = O(n + m) \]