Chapter 20. Elementary Graph Algorithms

Joon Soo Yoo

May 27, 2025

Assignment

- Read §20.1
- Problems
  - §20.1 - 2

Chapter 20: Elementary Graph Algorithms

- **Chapter 20.1: Representations of Graphs**
- Chapter 20.2: Breadth-First Search
- Chapter 20.3: Depth-First Search
- Chapter 20.4: Topological Sort
- Chapter 20.5: Strongly Connected Components

What is a Graph? $G = (V, E)$

- A **graph** is a data structure used to model relationships between objects.
- It is defined as:

$$G = (V, E)$$

  where:
  - V is the set of **vertices** (or nodes)
  - E is the set of **edges** (connections between vertices)

Graph Example

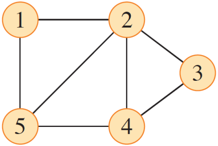
$V = \{1, 2, 3, 4, 5\}$

$E = \{(1,2), (2,3)$
$(3,4),$ " $\}$

G

directed

undirected



(a)

$G = (V, E)$

- Example:

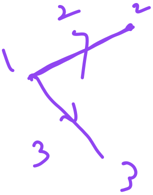
$$V = \{1, 2, 3, 4, 5\}$$

$$E = \{(1, 2), (1, 5), (2, 3), (2, 4), (2, 5), (3, 4), (4, 5)\}$$

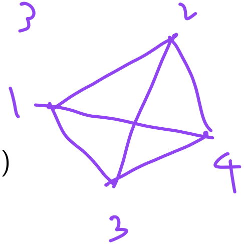
- Each edge connects a pair of vertices, and can be **undirected** or **directed**.

Two Ways to Represent a Graph $G = (V, E)$

- A graph can be represented in two common ways:
  - **Adjacency List** V
  - **Adjacency Matrix** E
- Both methods can represent:
  - Directed or undirected graphs
  - Weighted or unweighted graphs
- The right choice depends on:
  - Graph size (sparse vs. dense)
  - Algorithm requirements (speed vs. memory)



G



Adjacency List: Structure and Examples

$\mathrm{Adj}[1] = \{2, 5\}$

$\mathrm{Adj}[2] = \{1, 5, 4, 3\}$

- An **adjacency list** represents a graph $G = (V, E)$ using an array of lists:

$$\texttt{Adj}[1, \ldots, |V|]$$

- For each vertex $u \in V$, the list $\texttt{Adj}[u]$ contains all vertices v such that $(u, v) \in E$

$G.\mathrm{Adj}[u] = \{v \mid (u,v) \in E\}$

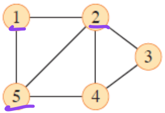
- In pseudocode:

$$G.\texttt{Adj}[u] = \{v \mid (u, v) \in E\}$$

Adjacency List Example

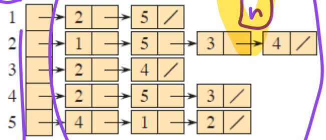
E

$|V|^2$



(a)

V



(b)

1 → 2 ⋮ n

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

(c)

sparse

- Undirected graph $G = (V, E)$ with $V = \{1, 2, 3, 4, 5\}$
- Adjacency list:

$$
\begin{array}{c|ccccc}
 & 1 & 2 & 3 & 4 & 5 \\
\hline
1 & 0 & 1 & 0 & 0 & 1 \\
2 & 1 & 0 & 1 & 1 & 1 \\
3 & & & & & \\
4 & & & & & \\
5 & & & & &
\end{array}
$$

`Adj[1]` = [2, 5]

`Adj[2]` = [1, 5, 3, 4]

`Adj[3]` = [2, 4]

`Adj[4]` = [2, 5, 3]

`Adj[5]` = [4, 1, 2]

1

n

n

1

n

- Each edge (u, v) appears in both `Adj[u]` and `Adj[v]`

Adjacency Matrix Representation

G |V| 1 2 3 4 5

- The graph $G = (V, E)$ can also be represented by a $|V| \times |V|$ matrix A

Θ |V|

- Each entry $A[i][j]$ indicates whether there is an edge from vertex i to vertex j

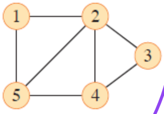
$A[i][j] = 1$

- For unweighted graphs:

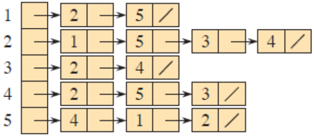
$$A[i][j] = \begin{cases} 1 & \text{if } (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$

i j

Adjacency Matrix Example



(a)



(b)

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

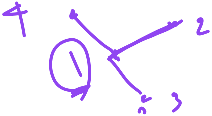
(c)

$A^T = A$

symmetric

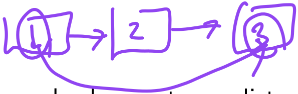
- The matrix represents an undirected graph with $V = \{1, 2, 3, 4, 5\}$
- Example interpretations:
  - Since there is an edge between vertices 1 and 2, we have $A[1][2] = A[2][1] = 1$
  - No edge between 1 and 3 $\Rightarrow A[1][3] = 0$
  - Edge between 4 and 5 $\Rightarrow A[4][5] = A[5][4] = 1$

Adjacency List: Pros and Cons



$\deg(1) = 3$

- **Advantages**:
  - Space-efficient for sparse graphs: $\Theta(V + E)$
  - Fast iteration over neighbors: $O(\deg(u))$
    - *Degree* of vertex u, denoted $\deg(u)$, is the number of edges connected to u



- **Disadvantages**:
  - Slower edge existence check: must scan list $\Rightarrow O(\deg(u))$
  - Less suitable for dense graphs

Adjacency Matrix: Pros and Cons

- **Advantages**:
  - Constant-time edge lookup: $O(1)$
  - Simple and easy to implement
  - Works well for dense graphs
  - For unweighted graphs: only 1 bit per entry
- **Disadvantages**:
  - Always uses $\Theta(V^2)$ space, even if few edges
  - Iterating over all neighbors or edges takes $\Theta(V^2)$ time
  - Wastes memory for sparse graphs

Problem: Adjacency List and Matrix

Problem:

- Give an adjacency-list representation for a complete binary tree on 7 vertices. Give an equivalent adjacency-matrix representation. Assume that the edges are undirected and that the vertices are numbered from 1 to 7 as in a binary heap.

Adjacency List Representation

- Complete binary tree with 7 nodes (numbered as in a heap):

$$\begin{aligned}
\texttt{Adj[1]} &= [2, 3] \\
\texttt{Adj[2]} &= [1, 4, 5] \\
\texttt{Adj[3]} &= [1, 6, 7] \\
\texttt{Adj[4]} &= [2] \\
\texttt{Adj[5]} &= [2] \\
\texttt{Adj[6]} &= [3] \\
\texttt{Adj[7]} &= [3]
\end{aligned}$$

- Each edge appears twice since the graph is **undirected**

Adjacency Matrix Representation

- $A[i][j] = 1$ if there is an edge between vertices i and j
- Symmetric matrix since the graph is **undirected**

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Question?