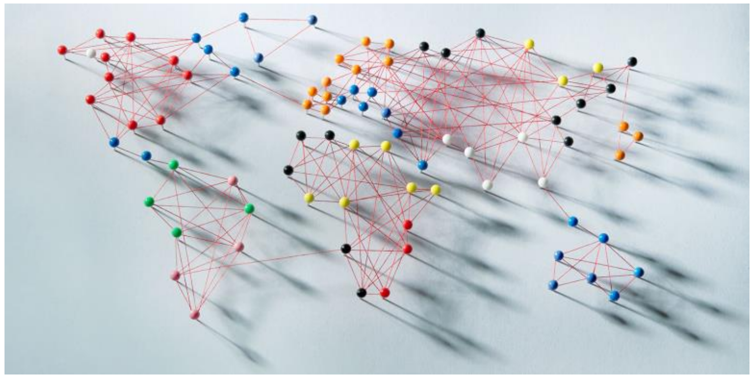
Implementation of Graphs and Digraphs



Textbook Reading:

Chapter 5, pp. 209-214.

Read pp. 197-208 to review graph theory

Standard Implementations

Let G = (V,E) be a graph, where $V = \{0,1, . . . , n – 1\}$

Two standard implementations of G are

- adjacency matrix implementation
- adjacency lists implementation

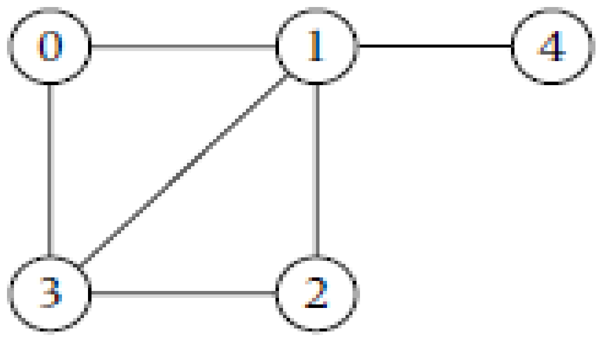
Adjacency Matrix Implementation

The *adjacency matrix* of a graph G is the $n \times n$ symmetric matrix $A = (a_{ij})$ given by

$$a_{ij} = \begin{cases} 1 & \text{vertices } i \text{ and } j \text{ are adjacent in } G. \\ 0 & \text{otherwise,} \end{cases} \qquad i, j \in \{0, \ldots, n-1\}.$$

Sample Graph and its Adjacency Matrix

$G=$



$$A=\begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

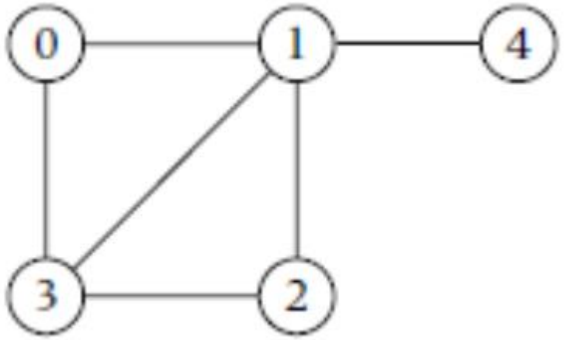
Pros and Cons

- Implementing *G* using its adjacency matrix makes it easy to perform many standard operations on *G*, such as adding a new edge or deleting an existing edge.
- The adjacency matrix of *G* allocates n^2 memory locations no matter how many edges are in the graph.
- Implementing *G* using its adjacency matrix is inefficient if the graph is **sparse**, i.e., the number of edges of *G* is small relative to the number of vertices.
- For example, if *G* is a tree with *n* vertices, then *G* has only $n - 1$ edges, which means only n – 1 out of n^2 entries of the matrix are being used, i.e., have the value 1.
- If graph is **dense**, e.g., $m \in \Theta(n^2)$, then the adjacency matrix representation is an efficient way to implement *G*.

Adjacency Lists Implement

In the adjacency lists implementation we represent the graph with a list of edges the are adjacent to each node.

G =



0 : 1 3

1 : 0 2 4

2: 1 3

3: 0 1 2

4: 1

Order Doesn’t Matter

The order that the vertices are listed in the adjacency lists does not matter. Another representation could be

0 : 1 3

1 : 4 2 0

2: 1 3

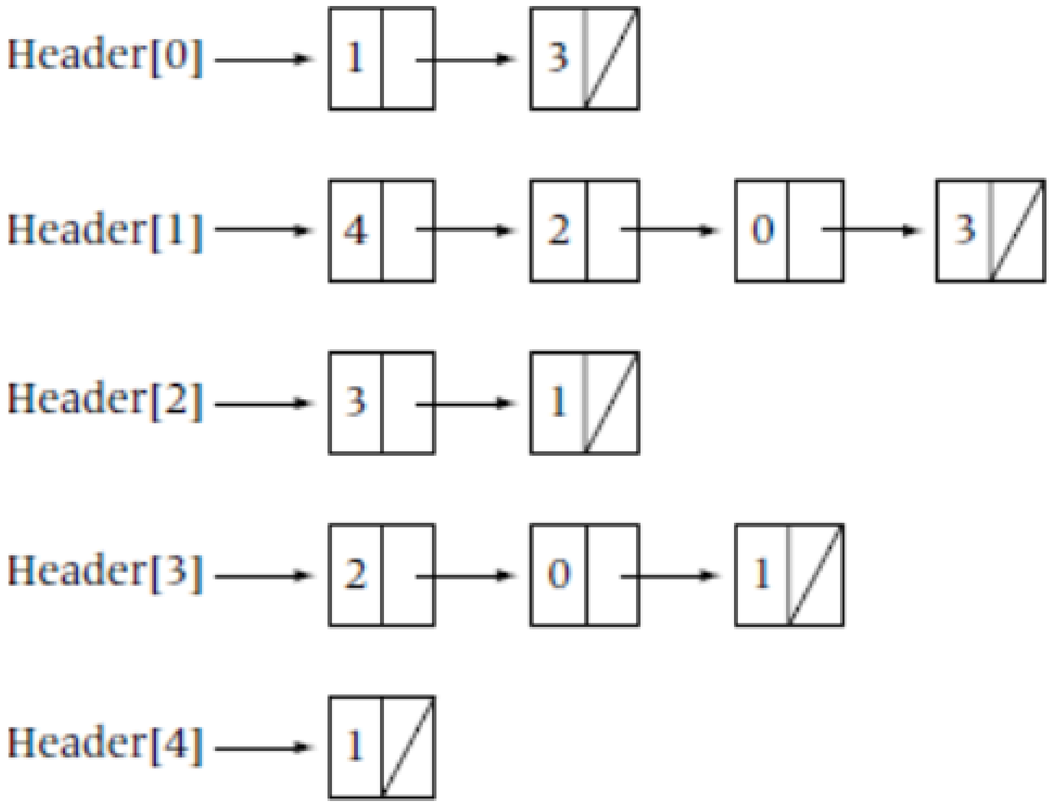
3: 2 0 1

4: 1

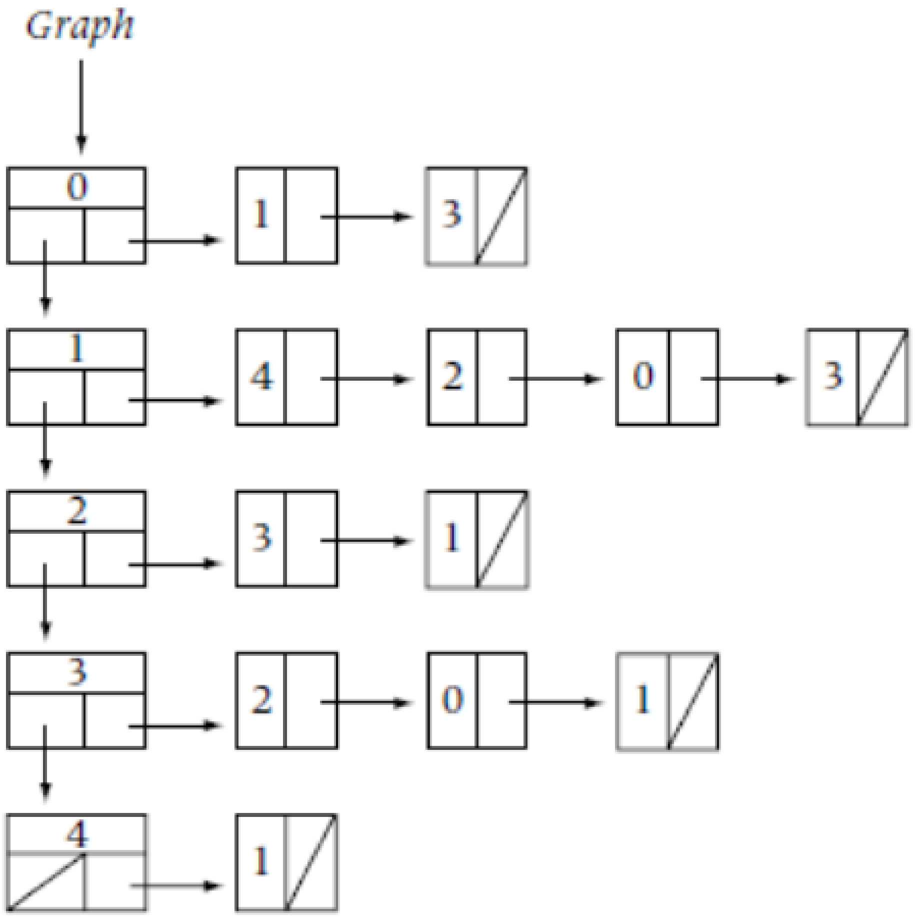
The implementation of a graph for I/O (input-output) simply stores the collection of vertices and the collection of pairs representing the edges of the graph. For example, there are XML format for graphs such as Graph Modelling Language (GML) and GraphML and that stores the graph this way.

When a graph is input the edges, i.e., pair of vertices, are inserted into the adjacency list in the order they are read.

Adjacency Lists – Array of Header Nodes



Adjacency Lists – Link List of Header Nodes



Adjacency Matrix vs. Adjacency Lists

Adjacency Matrix allows for direct access so is more efficient for adding and deleting edges.

Adjacency Lists is more efficient in terms of storage

If a graph is dense, i.e., close to the complete graph it is often better to use adjacency matrix

If the graph is sparse it is generally more efficient to use adjacency lists

Must use Adjacency Lists for Big Graphs

- For big graph such as the friendship graph for Facebook, i.e., the vertices are users and two users are joined with an edge if there friends, then it is necessary to use the adjacency list implementation.
- Number of users on Facebook is over one billion.
- Therefore size of adjacency matrix would be the square of a billion would be a billion billion or 1,000,000,000,000,000,000
- Much too large to store!
- On the other hand the average number of friends of a user on Facebook is estimated to be about 338.
- It follows from Euler's degree formula that the average degree equals $2m/n$. Therefore we have

$$338 = \frac{2m}{n} \Rightarrow 2m = 338n = 338{,}000{,}000{,}000$$

- The size of the adjacency lists would be twice the number of edges plus the number of vertices or

$$2m + n = 339{,}000{,}000{,}000$$

- 339 billion is not too large to store.

Implementation of digraphs

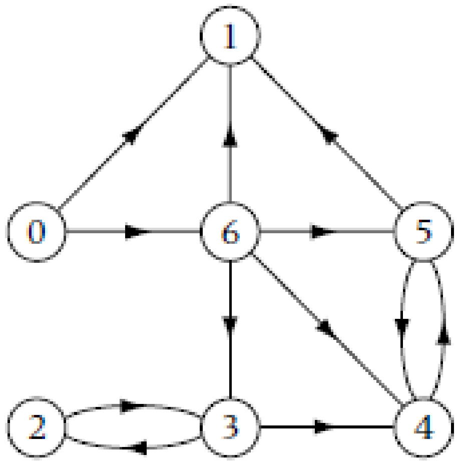
Digraphs are implemented in a similar way to graphs except the edges are ordered.

The *adjacency matrix* of a digraph *D*, whose vertices are labeled 0,1, . . . , $n - 1$, is the $n \times n$ matrix $A = (a_{ij})$ defined by

$$a_{ij} = \begin{cases} 1 & \text{if there is an edge from } i \text{ to } j \\ 0 & \text{otherwise} \end{cases}$$

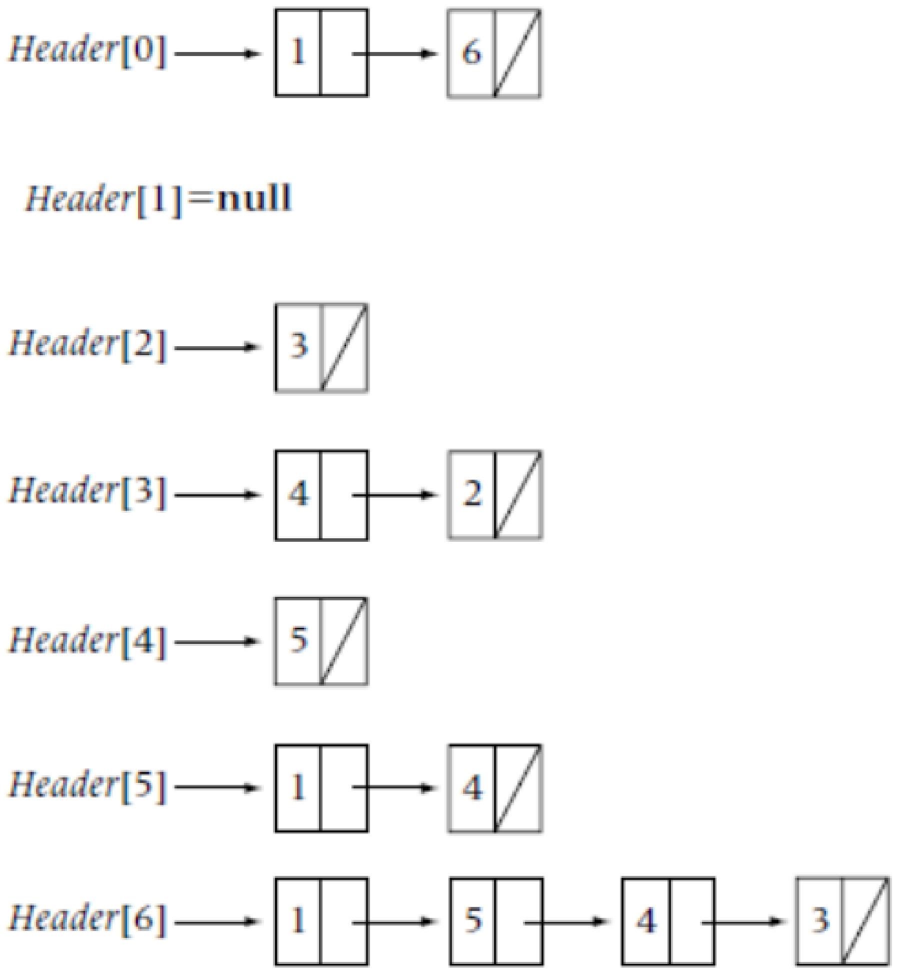
Adjacency Matrix for Sample Digraph

$D=$

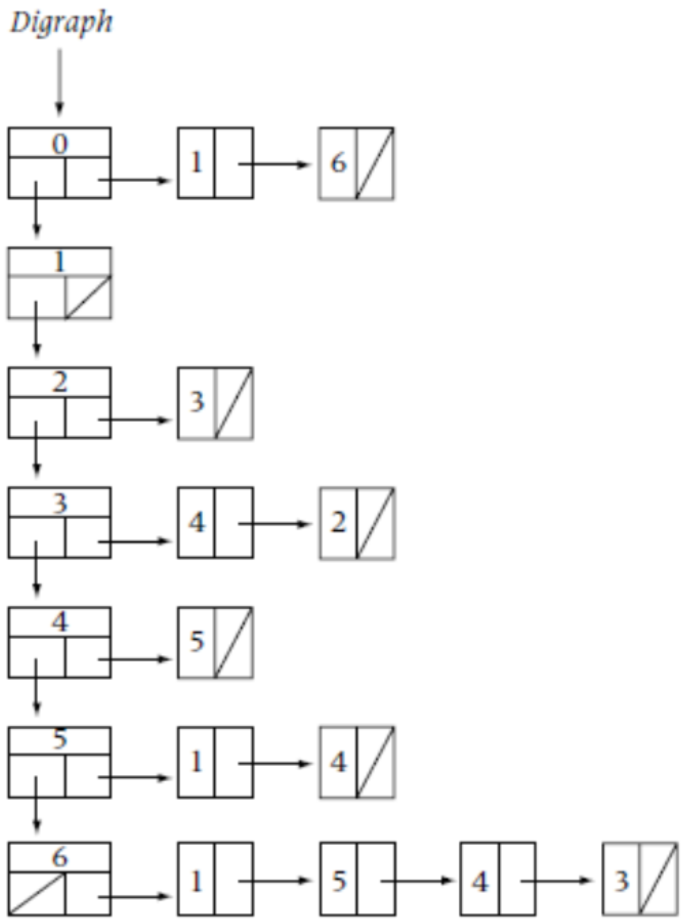


$$A=\begin{pmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 1\\
0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 1 & 0 & 0 & 0\\
0 & 0 & 1 & 0 & 1 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 1 & 0\\
0 & 1 & 0 & 0 & 1 & 0 & 0\\
0 & 1 & 0 & 1 & 1 & 1 & 0
\end{pmatrix}$$

Adjacency Lists – Array of Header Nodes

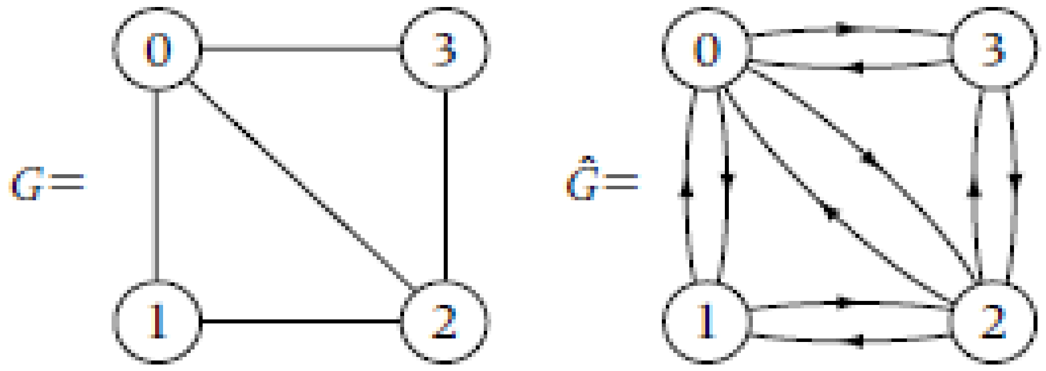


Adjacency Lists – Linked List of Header Nodes



Digraphs are more general than Graphs

A graph $G = (V, E)$ can be identified with its symmetric digraph $G = (V, \hat{E})$ where $\hat{E}$ is obtained by replacing each unordered pair $\{a, b\}$ with the two ordered pairs (a, b) and (b, a) for every edge $\{a, b\} \in E$.



Extracurricular

BEYOND ·THE· CLASSROOM!

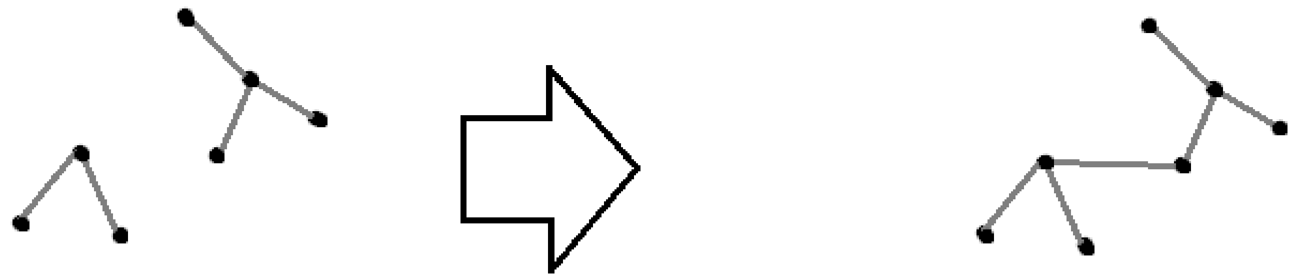
Gephi is an award-winning open-source platform for visualization and exploration for graphs, which you can download free:

https://gephi.org/

Another nice open-source graph software package is Network X:

https://networkx.github.io/

Sparse Graph Joke



Deforestation:
When adding a branch gives you fewer trees.