BACKGROUND: A Brief Introduction to Graph Theory

- General definitions; Representations;
- Graph Traversals;
- Topological sort;
Graph theory is a fundamental tool in sparse matrix techniques.

**DEFINITION.** A graph $G$ is defined as a pair of sets $G = (V, E)$ with $E \subset V \times V$. So $G$ represents a binary relation. The graph is undirected if the binary relation is symmetric. It is directed otherwise. $V$ is the vertex set and $E$ is the edge set.

If $R$ is a binary relation between elements in $V$ then, we can represent it by a graph $G = (V, E)$ as follows:

$$(u, v) \in E \iff u \ R \ v$$

Undirected graph $\iff$ symmetric relation
Given the numbers 5, 3, 9, 15, 16, show the two graphs representing the relations

R1: Either \( x < y \) or \( y \) divides \( x \).

R2: \( x \) and \( y \) are congruent modulo 3. \([\mod(x,3) = \mod(y,3)]\)

\[ |E| \leq |V|^2. \] For undirected graphs: \(|E| \leq |V|(|V| + 1)/2. \]

A sparse graph is one for which \(|E| \ll |V|^2. \)
Graphs – Examples and applications

1. Airport connection system: (a) R (b) if there is a non-stop flight from (a) to (b).
2. Highway system;
3. Computer Networks;
4. Electrical circuits;
5. Traffic Flow;
6. Social Networks;
7. Sparse matrices;
...

Basic Terminology & notation:

- If \((u, v) \in E\), then \(v\) is adjacent to \(u\). The edge \((u, v)\) is incident to \(u\) and \(v\).

- If the graph is directed, then \((u, v)\) is an outgoing edge from \(u\) and incoming edge to \(v\).

- \(\text{Adj}(i) = \{j | j \text{ adjacent to } i\}\)

- The degree of a vertex \(v\) is the number of edges incident to \(v\). Can also define the indegree and outdegree. (Sometimes self-edge \(i \rightarrow i\) omitted)

- \(|S|\) is the cardinality of set \(S\) [so \(|\text{Adj}(i)| = \deg(i)\)]

- A subgraph \(G' = (V', E')\) of \(G\) is a graph with \(V' \subset V\) and \(E' \subset E\). 
A graph is nothing but a collection of vertices (indices from 1 to \( n \)), each with a set of its adjacent vertices [in effect a 'sparse matrix without values']

Therefore, can use any of the sparse matrix storage formats - omit the real values arrays.

**Adjacency matrix**

Assume \( V = \{1, 2, \ldots, n\} \). Then the adjacency matrix of \( G = (V, E) \) is the \( n \times n \) matrix, with entries:

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

\[
\begin{bmatrix}
1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1
\end{bmatrix}
\]
An array of linked lists. A linked list associated with vertex $i$, contains all the vertices adjacent to vertex $i$.

General and concise for ‘sparse graphs’ (the most practical situations) - but not economical for use in sparse matrix methods.
More terminology & notation

- For a given $Y \subset X$, the section graph of $Y$ is the subgraph $G_Y = (Y, E(Y))$ where $E(Y) = \{(x, y) \in E | x \in Y, y \in Y\}$.

- A section graph is a clique if all the nodes in the subgraph are pairwise adjacent ($\rightarrow$ dense block in matrix).

- A path is a sequence of vertices $w_0, w_1, \ldots, w_k$ such that $(w_i, w_{i+1}) \in E$ for $i = 0, \ldots, k - 1$.

- The length of the path $w_0, w_1, \ldots, w_k$ is $k$ (# of edges in the path).

- A cycle is a closed path, i.e., a path with $w_k = w_0$.

- A graph is acyclic if it has no cycles.
Find cycles in this graph:

A path in an indirected graph

A path \( w_0, \ldots, w_k \) is simple if the vertices \( w_0, \ldots, w_k \) are distinct (except that we may have \( w_0 = w_k \) for cycles).

An undirected graph is connected if there is a path from every vertex to every other vertex.

A digraph with the same property is said to be strongly connected.
The undirected form of a directed graph is the undirected graph obtained by removing the directions of all the edges.

Another term used "symmetrized" form -

A directed graph whose undirected form is connected is said to be weakly connected or connected.

Tree = a graph whose undirected form, i.e., symmetrized form, is acyclic & connected

Forest = a collection of trees

In a rooted tree one specific vertex is designated as a root.

Root determines orientation of the tree edges in parent-child relation

In example: $v_3$ is parent of $v_6$, $v_8$ and $v_6$, $v_8$ are children of $v_3$.

Nodes that have no children are leaves. In example: $v_{10}$, $v_7$, $v_8$, $v_4$.

Descendent, ancestors, ...
Tree traversals

- Tree traversal is a process of visiting all vertices in a tree. Typically traversal starts at root.
- Want: systematic traversals of all nodes of tree – moving from a node to a child or parent
- Preorder traversal: Visit parent before children [recursively]
  In example: $v_1, v_2, v_9, v_{10}, v_3, v_8, v_6, v_7, v_5, v_4$
- Postorder traversal: Visit children before parent [recursively]
  In example: $v_{10}, v_9, v_2, v_8, v_7, v_6, v_3, v_4, v_5, v_1$
Graph Traversals – Depth First Search

➢ Issue: systematic way of visiting all nodes of a general graph

➢ Two basic methods: Breadth First Search (wll’s see later) & ...

➢ Depth-First Search

Algorithm \( List = DFS(G, v) \) (DFS from \( v \))

• Visit and Mark \( v \);

• for all edges \((v, w)\) do
  – if \( w \) is not marked then \( List = DFS(G, w) \)
  – Add \( v \) to top of list produced above

➢ If \( G \) is undirected and connected, all nodes will be visited

➢ If \( G \) is directed and strongly connected, all nodes will be visited
Depth First Search – undirected graph example

Assume adjacent nodes are listed in alphabetical order.

DFS traversal from A?
**Depth First Search – directed graph example**

Assume adjacent nodes are listed in alphabetical order.

DFS traversal from A?

NOTE: We will now use a column-oriented graph representation:

\[ j \rightarrow i \text{ if } a_{ij} \neq 0 \]
function [Lst, Mark] = dfs(u, A, Lst, Mark)
%% function [Lst, Mark] = dfs(u, A, Lst, Mark)
%% dfs from node u -- Recursive
%%-----------------------------------
[ii, jj, rr] = find(A(:,u));
Mark(u) = 1;
for k=1:length(ii)
    v = ii(k);
    if (~Mark(v))
        [Lst, Mark] = dfs(v, A, Lst, Mark);
    end
end
Lst = [u,Lst]
Depth-First-Search Tree: Consider the parent-child relation: $v$ is a parent of $u$ if $u$ was visited from $v$ in the depth first search algorithm. The (directed) graph resulting from this binary relation is a tree called the Depth-First-Search Tree. To describe tree: only need the parents list.

➢ To traverse all the graph we need a DFS($v,G$) from each node $v$ that has not been visited yet – so add another loop. Refer to this as

\[ \text{DFS}(G) \]

➢ When a new vertex is visited in DFS, some work is done. Example: we can build a stack of nodes visited to show order (reverse order: easier) in which the node is visited.
EXAMPLE

We assume adjacency list is in increasing order. [e.g. Adj(4) = (1, 5, 6, 7)]

DFS traversal: 1 --> 2 --> 3 --> 4 --> 5 --> 6 --> 7

Parents list: 1 1 1 4 4 6

Depth First Search Tree
Back edges, forward edges, and cross edges

- Thick red lines: DFS traversal tree from A
- \( A \rightarrow F \) is a Forward edge
- \( F \rightarrow B \) is a Back edge
- \( C \rightarrow B \) and \( G \rightarrow F \) are Cross-edges.
Consider the ‘List’ produced by DFS.

Lst=[A, C, G, B, D, F, E]

Order in list is important for some algorithms.

Notice: Label nodes in List from 1 to n. Then:

- Tree-edges / Forward edges: labels increase in →
- Cross edges: labels in/decrease in → [depends on labeling]
- Back-edges: labels decrease in →
Properties of Depth First Search

If $G$ is a connected undirected (or strongly connected) graph, then each vertex will be visited once and each edge will be inspected at least once.

Therefore, for a connected undirected graph, The cost of DFS is $O(|V| + |E|)$

If the graph is undirected, then there are no cross-edges. (all non-tree edges are called ‘back-edges’)

**Theorem:** A directed graph is acyclic iff a DFS search of $G$ yields no back-edges.

**Terminology:** Directed Acyclic Graph or $DAG$
Problem: Given a Directed Acyclic Graph (DAG), order the vertices from 1 to $n$ such that, if $(u, v)$ is an edge, then $u$ appears before $v$ in the ordering.

- Equivalently, label vertices from 1 to $n$ so that in any (directed) path from a node labelled $k$, all vertices in the path have labels $> k$.

- Many Applications
- Prerequisite requirements in a program
- Scheduling of tasks for any project
- Parallel algorithms;
- ...

Topological Sort
Property exploited: An acyclic Digraph must have at least one vertex with indegree = 0.

Prove this

First label these vertices as 1, 2, ... , k;
Remove these vertices and all edges incident from them
Resulting graph is again acyclic ... ∃ nodes with indegree = 0. label these nodes as k + 1, k + 2, ... , k + 1, k + 2, ...
Repeat..

Explore implementation aspects.
Alternative method: Topological sort from DFS

- Depth first search traversal of graph.
- Do a ‘post-order traversal’ of the DFS tree.

Algorithm \( \text{Lst} = \text{Tsort}(\text{G}) \)

(post-order DFS from \( v \))

\[
\begin{align*}
\text{Mark} &= \text{zeros}(n,1); \quad \text{Lst} = \emptyset \\
\text{for} \quad v=1:n \quad \text{do:}
\quad \text{if} \quad (\text{Mark}(v) == 0) \\
\quad \quad \text{[Lst, Mark]} &= \text{dfs}(v, \text{G}, \text{Lst}, \text{Mark}); \\
\quad \quad \text{end}
\quad \text{end}
\end{align*}
\]

\( \text{dfs}(v, \text{G}, \text{Lst}, \text{Mark}) \) is the \( \text{DFS}(\text{G},v) \) which adds \( v \) to the top of \( \text{Lst} \) after finishing the traversal from \( v \)
\[ Lst = DFS(G, v) \]

- Visit and Mark \( v \);
- for all edges \((v, w)\) do
  - if \( w \) is not marked then \( Lst = DFS(G, w) \)
- \( Lst = [v, Lst] \)

➢ Topological order given by the final \( Lst \) array of \( \text{Tsort} \)

- Explore implementation issue
- Implement in matlab
- Show correctness [i.e.: is this indeed a topol. order? hint: no back-edges in a DAG]
• See Chap. 3 of text
• Sparse matrices and graphs.
• Bipartite model, hypergraphs
• Application: back propagation
Graph Representations of Sparse Matrices. Recall:

Adjacency Graph $G = (V, E)$ of an $n \times n$ matrix $A$:

$V = \{1, 2, \ldots, N\} \quad E = \{(i, j)|a_{ij} \neq 0\}$

- $G$ is undirected if $A$ has a symmetric pattern

Example:
Show the matrix pattern for the graph on the right and give an interpretation of the path \( v_4, v_2, v_3, v_5, v_1 \) on the matrix

A separator is a set \( Y \) of vertices such that the graph \( G_{X-Y} \) is disconnected.

**Example:** \( Y = \{v_3, v_4, v_5\} \) is a separator in the above figure.
Example: Adjacency graph of:

\[
A = \begin{bmatrix}
\star & \star \\
\star & \star & \star \\
\star & \star \\
\end{bmatrix}.
\]

Example: For any adjacency matrix \( A \), what is the graph of \( A^2 \)? [interpret in terms of paths in the graph of \( A \)]
Two graphs are **isomorphic** if there is a mapping between the vertices of the two graphs that preserves adjacency.

Are the following 3 graphs isomorphic? If yes find the mappings between them.

Graphs are identical – labels are different

Determining graph isomorphism is a **hard** problem
Bipartite graph representation

- Rows and columns are (both) represented by vertices;
- Relations only between rows and columns: Row $i$ is connected to column $j$ if $a_{ij} \neq 0$

Example:

\[
\begin{bmatrix}
\ast & \ast \\
\ast & \ast & \ast \\
\ast & \ast \\
\end{bmatrix}
\]

Bipartite models used only for specific cases [e.g. rectangular matrices, ...] - By default we use the standard definition of graphs.
Interpretation of graphs of matrices

12. What is the graph of $A + B$ (for two $n \times n$ matrices)?

13. What is the graph of $A^T$?

14. What is the graph of $A \cdot B$?
A few words on hypergraphs

- Hypergraphs are very general... Ideas borrowed from VLSI work
- Main motivation: to better represent communication volumes when partitioning a graph. Standard models face many limitations
- Hypergraphs can better express complex graph partitioning problems and provide better solutions.
- Example: completely nonsymmetric patterns...
- Even rectangular matrices. Best illustration: Hypergraphs are ideal for text data
Example: \( V = \{1, \ldots, 9\} \) and \( E = \{a, \ldots, e\} \) with
\[
a = \{1, 2, 3, 4\}, \quad b = \{3, 5, 6, 7\}, \quad c = \{4, 7, 8, 9\}, \\
d = \{6, 7, 8\}, \quad \text{and} \quad e = \{2, 9\}
\]

Boolean matrix:

\[
\begin{array}{cccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
1 & 1 & 1 & 1 & & & & & \\
1 & 1 & 1 & 1 & & & & & \\
1 & 1 & 1 & 1 & & & & & \\
1 & 1 & 1 & 1 & & & & & \\
1 & & & & & & & & \\
1 & & & & & & & & \\
1 & & & & & & & & \\
1 & & & & & & & & \\
\end{array}
\]

\[
A =
\]

- graph
Computational graphs: graphs where nodes represent computations whose evaluation depend on other (incoming) nodes.

Consider the following expression:

\[ g(x, y) = (x + y - 2) \times (y + 1) \]

We can decompose this as

\[
\begin{align*}
    z &= x + y \\
    v &= y + 1 \\
    g &= (z - 2) \times v
\end{align*}
\]
Computational graph →

Given $x, y$ we want:

(a) Evaluate the nodes and
(b) derivatives w.r.t $x, y$

(a) is trivial - just follow the graph up - starting from the leaves (that contain $x$ and $y$)

(b): Use the chain rule – here shown for $x$ only using previous setting

For the above example compute values and derivatives at all nodes when $x = -1, y = 2$. 

\[
\frac{\partial g}{\partial x} = \frac{\partial g}{\partial a} \frac{\partial a}{\partial x} + \frac{\partial g}{\partial b} \frac{\partial b}{\partial x}
\]
Often we want to compute the gradient of the function at the root, once the nodes have been evaluated.

The derivatives can be calculated by going backward (or down the tree).

Here is a very simple example from Neural Networks:

\[
\begin{align*}
L &= \frac{1}{2} (y - t)^2 \\
y &= \sigma(z) \\
z &= wx + b
\end{align*}
\]

Note that \(t\) (desired output) and \(x\) (input) are constant.
Back-Propagation: General computational graphs

- Last node ($v_n$) is the target function. Let us rename it $f$.
- Nodes $v_i, i = 1, \cdots, e$ with indegree 0 are the variables.
- Want to compute $\frac{\partial f}{\partial v_1}, \frac{\partial f}{\partial v_2}, \cdots, \frac{\partial f}{\partial v_e}$.
- Use the chain rule.

$$\frac{\partial f}{\partial v_k} = \frac{\partial f}{\partial v_j} \frac{\partial v_j}{\partial v_k} + \frac{\partial f}{\partial v_l} \frac{\partial v_l}{\partial v_k} + \frac{\partial f}{\partial v_m} \frac{\partial v_m}{\partial v_k}$$
Let $\delta_k = \frac{\partial f}{\partial v_k}$ (called ‘errors’). Then

$$\delta_k = \delta_j \frac{\partial v_j}{\partial v_k} + \delta_l \frac{\partial v_l}{\partial v_k} + \delta_m \frac{\partial v_m}{\partial v_k}$$

To compute the $\delta_k$’s once the $v_j$’s have been computed (in a ‘forward’ propagation) – proceed backward.

$\delta_j, \delta_l, \delta_m$ available and $\partial v_i/\partial v_k$ computable. Note $\delta_n \equiv 1$.

However: cannot just do this in any order. Must follow a topological order in order to obey dependencies.
Example:

\[ a_i^1 = w_{i,1}^T x \]

\[ z_i^1 = \sigma(a_i^1) \]

\[ a_i^2 = w_{i,2}^T z_i^1 \]

\[ z_i^2 = \sigma(a_i^2) \]

\[ E(z_i^2) \]