- Graph
  - a set of nodes (vertices) connected by edges
  - $(V, E)$ \( V = \{1, 2, \ldots, n\} \)
  - edge \((i, j)\)
  - \(E = \{(1, 2), (2, 3), (3, 4), (4, 1), (2, 4)\}\)

- abstraction
- Examples
  1. road network
  2. supply network
  3. social network (edges = friends)
  4. dependency network (COMPSCI 201 \(\rightarrow\) 320)
  5. internet (edges = hyperlinks)

- directed and undirected graph

- graph problems
  1. shortest path
  2. minimum spanning tree
  3. community detection
  4. scheduling
  5. pagerank

- represent (store) a graph

- adjacency array
  \[ A[i, j] = \begin{cases} 
  1 & \text{if } (i, j) \text{ is an edge} \\
  0 & \text{if } (i, j) \text{ is not an edge}
\end{cases} \]
benefit: simple, quickly know whether \((i,j) \in E\)
but: take \(\Theta(n^2)\) space

- adjacency list
  
  Store a list for every vertex
  list \(i\) contains set of edges from vertex \(i\)

  1: \([2, 4]\)  
  2: \([1, 3, 4]\)  
  3: \([2, 4]\)  
  4: \([1, 2, 3]\)  

  benefit: take \(\Theta(m)\) space
  but: hard to know whether \((i,j)\) is an edge

- Basic Graph algorithms
  
  - Graph traversal: want to visit all nodes of graph \(G\) by following edges.

  - DFS (Depth First Search)

  \[
  \text{DFS} \\
  \text{for } i = 1 \text{ to } n \\
  \quad \text{if } i \text{ is not visited} \\
  \quad \quad \text{DFS}_\text{visit}(i) \\
  \text{DFS}_\text{visit}(i) \\
  \]

  - why do we need this loop?

  - Connectivity

  - directed

  \[
  \begin{array}{c}
  0 \quad 0 \\
  1 \quad 2 \quad 3 \\
  \end{array}
  \]

  (UndirectedGraph is connected if \((i,j)\) there is a path from \(i\) to \(j\))

  - Depth First Search Tree

  - \(j\) is a child of \(i\), if \(\text{DFS}_\text{visit}(i)\) called \(\text{DFS}_\text{visit}(j)\)
DFS: \[0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 2 \rightarrow 4\]

- Pre-order and post-order
  - Pre-order: order of visits \(1, 2, 3, 4\)
  - Post-order: order of DFS\_visit(i) returns \(3, 4, 2, 1\)

- DFS and stack
  - Preorder: order of entering the stack
  - Post-order: order of exiting the stack

- Edge types
  - Tree edge
  - Forward edge
  - Backward edge
  - Cross edge

- BFS (Breadth First Search)

\[
\text{BFS\_visit(i)}
\]

- Put \(i\) into a queue (mark \(i\) as visited)
- While queue is not empty
  - \(u \leftarrow \text{dequeue}\)
  - For each edge \((u, v)\)
    - If \(v\) is not visited
      - Put \(v\) into the queue
      - Mark \(v\) as visited.
BFS tree

BFS order: order of entering the queue

Can be used to compute shortest path.

u is a child of v, if u is added to the queue when processing v

(Shortest path tree)