REMINDERS

- Assignment 09 due Monday Dec 2\textsuperscript{nd} at 10:00 am
- Final Exam is on Thursday, Dec 19\textsuperscript{th}, 7:00 p.m.
- Q&A sessions are available.
- Piazza is still open for questions and announcements
REVIEW: UNDIRECTED GRAPHS

• An undirected **graph** \( G \) is a set \( V \), of **vertices**, and a set \( E \), of unordered pairs from \( V \), called **edges**. We write \( G = (V, E) \).

• \((4, 6)\) in Graph 1 is an **edge**
  – 4, 6 are **neighbours**
  – 4, 6 are **adjacent**

• Degree is number of neighbours of a vertex
  – **Degree** of 5 in Graph 1 is 3

• \( G_1 \) is **connected** and \( G_2 \) is **disconnected**
QUESTION 1: REPRESENTING GRAPHS IN DIFFERENT WAYS

Question:
Show how to represent the graph using:
• **Vertex and Edge list** representation
• **Adjacency list** representation
• **Adjacency matrix** representation (call 'A' vertex 0, 'B' vertex 1, etc.)
QUESTION 1 ANS:
VERTEX AND EDGE LIST REPRESENTATION

$V = ['A','B','C','D','E','F','G']$

$E = [['A','B'], ['A','C'], ['B','C'], ['B','D'], ['C','E'], ['E','F'], ['E','G']]$

- Each pair only appears once.
QUESTION 1 ANS:
ADJACENCY LIST

{ 'A': ['B', 'C'],
  'B': ['A', 'C', 'D'],
  'C': ['A', 'B', 'E'],
  'D': ['B'],
  'E': ['C', 'F', 'G'],
  'F': ['E'],
  'G': ['E'] }

• Key: label of the vertex
• Value: List of adjacent vertices (neighbours)
QUESTION 1 ANS:
ADJACENCY MATRIX REPRESENTATION

```
  A B C D E F G
[[0,1,1,0,0,0,0], #A
 [1,0,1,1,0,0,0], #B
 [1,1,0,0,1,0,0], #C
 [0,1,0,0,0,0,0], #D
 [0,1,0,0,0,0,0], #E
 [0,0,1,0,0,1,1], #F
 [0,0,0,1,0,0,1]] #G
```

Tutorial I I: Graph Theory
QUESTION 2A: DRAW THE GRAPH

Draw the graph corresponding to the following adjacency list.

```json
{'A': ['B', 'C'],
'B': ['A', 'D', 'E', 'F'],
'C': ['A'],
'D': ['B', 'E'],
'E': ['B', 'D', 'F'],
'F': ['B', 'E']}
```
QUESTION 2A ANS: ADJACENCY LIST
QUESTION 2B: DRAW THE GRAPH

Draw the graph corresponding to the following adjacency matrix.

\[
\begin{bmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]
QUESTION 2B ANS: ADJACENCY MATRIX
REVIEW: TRAVERSALS

• Traversals:
  – Finding all connected vertices, starting from A

• Two approaches
  – **Breadth-first search** (from starting vertex, then neighbours, then neighbours of neighbours, etc.)
    Note that neighbours need not be processed in a particular order
  – **Depth-first search** (from starting vertex follow path as far as possible, back up to closest unvisited neighbour, repeatedly)
QUESTION 3A

- Perform **bfs** and **dfs** traversals for the following graphs.
- Starting from A and E

Ex:
Bfs: D, B, A, C, E, G, F
Dfs: D, B, C, E, G, F, A

Graph 1

Graph 2
Recursive implementation of dfs traversal

```python
def dfs(graph, v):
    visited = []
    return visit(graph, v, visited)

def visit(g, v, all):
    all.append(v)
    for neighbour in g[v]:
        if neighbour not in all:
            visit(g, neighbour, all)
    return all
```

Fall 2018 Tutorial 11: Graph Theory
Write the function `vertices_count` that consumes a nonempty graph \( G \) (stored as an adjacency list) and returns the number of vertices in \( G \).

Note that for adjacency list:
- The vertices labels are keys in dictionary
- Paired values are lists with neighbours (its adjacent vertices)

Example:
\[
G1 = \{1:[2,5], 2:[1,3], 3:[2], 4:[5], 5:[1,4]\}
\]
`vertices_count(G1) => 5`
**QUESTION 4B**

**EDGES_COUNT**

Write the function `edges_count` that consumes a nonempty graph `G` (stored as an **adjacency list**) and returns the number of edges in `G`.

Examples:

\[ G_1 = \{1 : [2, 5], 2 : [1, 3], 3 : [2], 4 : [5], 5 : [1, 4] \} \]

\[ G_2 = \{1 : [2, 4], 2 : [1, 3, 4, 5], 3 : [2, 5], 4 : [1, 2], 5 : [2, 3], 6 : [] \} \]

`edges_count(G1) => 4`

`edges_count(G2) => 6`
Write the function `degree_adj_mat` that consumes a nonempty graph G (stored as an adjacency matrix) and a vertex number \( v \), and returns the degree of vertex \( v \) in G.

Note that for adjacency matrix:
- the vertices are numbered \( 0, 1, \ldots, n-1 \).
- G has \( n \) lists, and each list has a length \( n \) as well.

**Challenge Question:**
On your own, implement the functions `degree_adj_list` and `degree_edges`, to determine the degree of a vertex using the other representations.