REMINDERS

• Final Exam is on Monday, April 16th
  – Time: 12:30 PM – 3:00 PM
  – Location: PAC

• Office hours and review sessions are available.

• Piazza is still open for questions and announcements

Check PIAZZA!!!
REVIEW: UNDIRECTED GRAPHS

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

• $(V_4, V_6)$ in Graph 1 is an edge
  - $V_4, V_6$ are neighbours
  - $V_4, V_6$ are adjacent

• Degree is number of neighbours of a vertex
  - Degree of $V_5$ in Graph 1 is 3

• $G_1$ is connected and $G_2$ is unconnected
QUESTION 1: REPRESENTING GRAPHS IN DIFFERENT WAYS

Question:

Show how to represent the graph using:

• **Edge list** representation
• **Adjacency list** representation
• **Adjacency matrix** representation
  (call 'A' vertex 0, 'B' vertex 1, etc)
Draw the graph corresponding to the following adjacency list:

```python
{'A': ['B', 'C'],
 'B': ['A', 'D', 'E', 'F'],
 'C': ['A'],
 'D': ['B', 'E'],
 'E': ['B', 'D', 'F'],
 'F': ['B', 'E']}
```
QUESTION 2(B): 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 3: degree_adj_mat

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 a length \( n \), and each list in \( G \) 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.
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.)
  – **Depth-first search** (from starting vertex follow path as far as possible, back up to closest unvisited neighbour, repeatedly)
QUESTION 4 (A)

• Perform bfs and dfs traversals for the following graphs.
• Starting from A and E
 Recursive implementation of dfs traversal

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

QUESTION 4 (B)