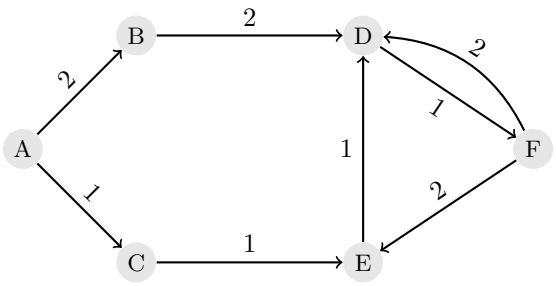
# 159.271 Computational Thinking
# Tutorial - Shortest Path

Design and implement a one-to-all shortest path algorithm for directed graphs with edge weights, where each edge weight is either 1 or 2. E.g. the graph below would be a valid input graph:




The catch here is that **your algorithm must run in linear time**, i.e., $O(|V| + |E|)$. In particular this means that Dijkstra's algorithm with a binary heap for organizing candidate nodes won't do.

*Tip 1:* Consider how much the shortest path lengths to candidate nodes can differ.

*Tip 2:* You can use `float('inf')` to represent infinite distance.

Some skeleton code is given below. Here the graph shown above is encoded as a list of lists of `(successor, edge_weight)` pairs.

```
graph = [
    [(1,2), (2,1)], # A
    [(3,2)],        # B
    [(4,1)],        # C
    [(5,1)],        # D
    [(3,1)],        # E
    [(3,2), (4,2)], # F
];

def shortest_paths(g, root):
    """ Compute distance from root node to all other nodes
    """
    TODO

# expected: [0, 2, 1, 3, 2, 4]
print(shortest_paths(graph, 0))
```