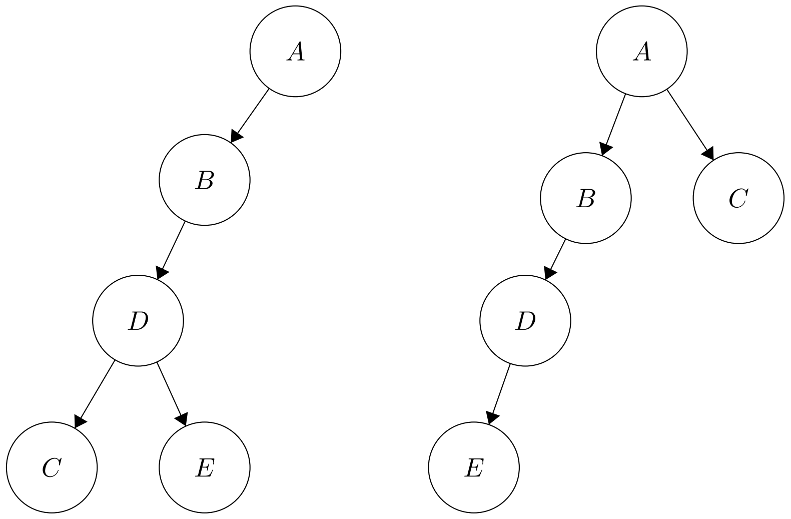
Exercises

1. Left: DFS tree. Right: BFS tree.



2. A is bipartite. B is not.

   English description of code: First find all the BFS trees. For each tree, paint nodes in different levels in alternating colors. For each edge, check if the end nodes are of the same color. If one edge is of the same colors, return false (not bipartite). Return true otherwise.

   Correctness: An edge in the graph can only connect nodes that either are in the same level of a BFS tree or are in neighboring levels. The latter cause even-length cycles so they don't affect bipartiteness. The former cause odd-length cycles so they break bipartiteness.

```
def checkBipartite(A, printdiag=False):
  def paint(tree):
    for i, level in enumerate(tree):
      for node in level:
        node.color = i%2

  def check(A):
    edges = A.getDirEdges()
    for edge in edges:
      if (edge[0].color == edge[1].color):
        return False
    return True

  s = A.vertices[0]
  nn = len(A.vertices)
  cnt = 0
  while (cnt < nn):
    treeA = BFS(s, A)
    paint(treeA)

    flatA = [item for sublist in treeA for item in sublist]
    if printdiag:
      #print(len(treeA))
      print("start", s, "len", len(flatA))
    cnt += len(flatA)
    dictA = {key:0 for key in flatA}
    s = None
    for u in A.vertices:
      if u not in dictA:
        s = u
        break
  return check(A)

print("Is A bipartite?", checkBipartite(A))
print("Is B bipartite?", checkBipartite(B))
```

Problems

---

1. Midterm review.
2. (a) First part: All source sheep are in the same strongly connected component of G.

   Proof by contradiction: Suppose x and y are both source sheep, and they belong to different strongly connected components. By the definition of source sheep, path $x \rightsquigarrow y$ exists, so does $y \rightsquigarrow x$. Therefore x and y are in one strongly connected component, which is a contradiction.

   Second part: Every sheep in that component is a source sheep.

   By the first part, all known source sheep belong to one strongly connected component. Suppose sheep x is one of them. Now suppose some other sheep y, not known to be a source sheep, is in the component. By the definition of strongly connected component, path $y \rightsquigarrow x$ exists, so all y posts will propagate throughout G by way of x. I.e. y is also a source sheep.

   (b) Apply DFS to G, starting with a random node, while recording finish time. Afterwards, if there are undiscovered nodes, apply DFS again. Repeat until each node is covered by one DFS. The node with the highest finish time is a source sheep.

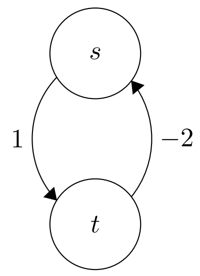
   Correctness: Because there is a path from a source sheep to any other sheep in G, if a DFS starts in the one strongly connected component of G, all nodes will be discovered after that DFS. Hence one of the source sheep will have the highest finish time.

   Running time: All DFSs combined is O(n+m).

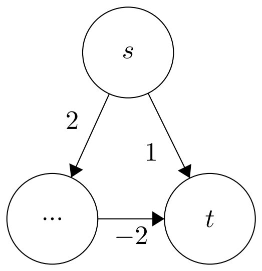
   (c) Apply part (b). Then start one last DFS from the node that (previously) records the highest finish time. If the last DFS discovers all nodes, return true; return false otherwise.

   Correctness: If a source sheep exists, one of its peers, say sheep x, in the strongly connected component must register the highest finish time in (b). Running a DFS from x will discover all nodes. If a source sheep doesn't exist, running a DFS from the sheep with the highest finsih time in (b) will not discover all nodes.

   The running time is that of (b) plus one DFS. O(n+m).
3. (a) A negative-weight cycle can give a -inf path length.



(b) There is no negative-weight cycle, but dijkstra removes t from unsureVertices while d[t]==1, before dijkstra has a chance to see the edge of weight -2.



(c) No. The example in (b) becomes the following, but dijkstra still gives the wrong path for t.