CS/ECE 374 ✧ Spring 2017

☙ Homework 7 Question 2 ❧

Due Wednesday, March 29, 2017 at 10am

Ho Yin Au (hoyinau2), Lanxiao Bai(lbai5), Renheng Ruan(rruan2)

- Question 2

  The problem is equivalence to checking the input graph satisfying the following condition: For a directed graph $G(V, E)$, for any $u, v \in V$, $u \neq v$, there exist a directed path from u to v and a directed path from v to u.

  As we need all vertices satisfying this condition, we can start searching from any vertex s and check if all other vertices have directed paths to and from s.

  Notice that for any vertices u, v, for any directed path from u to v, if u has a path from s, and v has a path to s, all vertices along the path satisfy the condition. (notice this is true even u is descendent of v).

  We can search the graph using DFS from an initial vertex s. We first need to mark vertices visited in each visit to prevent infinite loop. We know that any vertices reachable by DFS has a path from s. If we reach s again, we notice that there is a directed cycle of s, which contains directed paths to and from s and any vertices in that cycle. We stop searching until all vertices are checked. Then, we get all vertices that is in directed cycles with s. However, there may also exist repeated vertices in the paths to and from s, which unluckily reach ascendant and stop in the first run of DFS (as we mark vertices visited to prevent infinite loop). Luckily, if we keep the information of vertices meet the condition, we can do second DFS search to identify whether those vertices meet the condition.

  We make 2 arrays storing whether a vertex is visited (VISITED)and whether it can reach s (TOINITVERTEX, this is also identify the good intersections for part 2), we also define s as the first element of $V(G)$ array (we can actually pick it by random). We do 2 DFS to get all the information stored in the TOINITVERTEX array, and iterate through it to get whether all vertices are qualified.

  For the running time, even we do not care about checking any vertex is not visited in DFS, we still get $O(m + n)$ time, where m is number of edges and n is number of vertices. The for loop of initialization and checking is done by $O(n)$. DFS use $O(m + n)$ (nearly $O(n)$ as we prevent revisit vertices). As we run them sequentially, we get $O(m + n)$ time.

```
CyclesBetweenAllVertices(Graph G):

  boolean ToInitVertex[1..|V(G)|]
  boolean visited[1..|V(G)|]

  for v in V(G)
      ToInitVertex[v] = false
      visited[v] = false
  s ← V(G)[1]

  DFS(s)

  for v in V(G)
      visited[v] = false
  DFS(s)

  for v in V(G)
      if ToInitVertex[v] = false
          return false
  return true
```

```
DFS(Vertex u):

  visited[u] = true
  for each uv in Out(u):
      if v = s
          ToInitVertex[u] = true
      if visited[v] = false
          DFS(v)
      if ToInitVertex[v] = true
          ToInitVertex[u] = true
```

As we record the information of whether a vertex u has directed paths to and from initial vertex s, we can know the percentage of good intersection easily using this information. We just count the number of $true$ in the ToInitVertex array, divide it by total number of vertices in the graph to get the percentage of good intersection. If the percentage is not less than 0.95, we return true; else return false.
As the algorithm is similar to the CyclesBetweenAllVertices above excpet from the last ToInitVertex array checking part, it shares the same running time of CyclesBetweenAllVertices, which is $O(m+n)$.

```
GoodIntersections(Graph G):

    boolean ToInitVertex[1..|V(G)|]
    boolean visited[1..|V(G)|]

    for v in V(G)
        ToInitVertex[v] = false
        visited[v] = false
    s ← V(G)[1]

    DFS(s)

    for v in V(G)
        visited[v] = false
    DFS(s)

    count ← 0
    for v in V(G)
        if ToInitVertex[v] = true
            count ← count + 1
    if count/|V(G)| >= 0.95
        return true
    else
        return false
```