LeetCode Technical Writing

Hoon Oh

November 13, 2020

1557. Minimum Number of Vertices to Reach All Nodes [Medium]

Given a **directed acyclic graph**, with n vertices numbered from 0 to $n-1$, and an array `edges` where $\texttt{edges}[i] = [\texttt{from}_i, \texttt{to}_i]$ represents a directed edge from node $\texttt{from}_i$ to node $\texttt{to}_i$.

Find *the smallest set of vertices from which all nodes in the graph are reachable*. It's guaranteed that a unique solution exists.

Notice that you can return the vertices in any order.

1 An Example

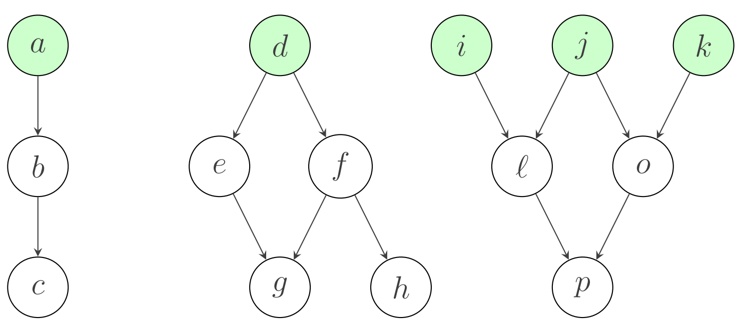

Figure 1: An instance of directed acyclic graph.The smallest set of vertices from which all nodes in the graph are reachable is the set $\{a, d, i, j, k\}$.

2 A variant of Union-find Method

Intuition We can view the problem as a variant of union-find problem. If a node u is reachable from a node v then we want to group u and v together. However in Figure **??**, we do not want to group nodes i and j in the same group. We resolve this problem by combining u and v to the same group only if there is an edge from u to v and v does not belong to another group.

Algorithm We initialize each node's group head as itself. In other words, a node u's group head is u. We go through each directed edge (u, v) (edge from u to v). If v's group head is itself, then label v's group with u's group head. Then at the end of the algorithm, output the set of distinct group heads as our solution.

At any time of the algorithm, a node v can be reached by the node correspond to v's group head. This is true because either the group head is v's parent, or a it is node that can reach the parent. Furthermore, at the end of the algorithm there is no group head u that can reach another group head v. This is true because if such path exists, then there must exists an edge to v and since v's group head is v, the v's group must have changed when we considered such edge.

```
class Solution:
    # Approach 1: A variant of union-find
    def findSmallestSetOfVertices(self, n: int, edges: List[List[int]])
    -> List[int]:

        # Initialize group heads
        groupHead = [i for i in range(n)]

        # asymUnion(a,b) combines group heads of a and b
        def asymUnion(a: int, b: int) -> int:
            i,j = find(a), find(b)
            if i == j: return
            groupHead[j] = i
            return

        # find(a) returns group head of a
        def find(a: int) -> int:
            while (a != groupHead[a]):
                a = groupHead[a]
            return a

        # go through each edge and union each edge.
        for i,j in edges:
            if j != groupHead[j]: continue
```

```
            asymUnion(i,j)

        # output the set of distinct group heads.
        ans = set()
        for i in range(n):
            ans.add(find(i))
        return ans
```

Complexity Analysis

- Time complexity: $O(mn)$ where m is the number of edges, and n is the number of vertices. The algorithm performs an union for each edge, and each union takes at most $O(n)$ times to find its parent.
- Space complexity: $O(n)$ where n is the number of vertices. We only maintain an array of group heads, thus the only takes $O(n)$ space.

Follow up There are more advanced algorithms for union-find data structure that takes $O(\log^*(n))$ to perform union and find. Therefore, the time complexity can be reduced to $O(m \log^*(n))$.

3 Removing Vertices with no in-degree

Intuition There is another way to view this problem. The problem is equivalent to count the number of vertices with no in-degree. Note that a node can only be reach by some other node if it has an incoming edge. Therefore, the number of vertices with no in-degree is an lower bound for our problem. Further, each node has a path from a node with no in-degree, because the group is acyclic. Therefore, it is also an upperbound of our problem.

Algorithm Initialize a set that contains all nodes. If there exists an edge (u, v), then remove v from the set. After going through all edges, return the remaining set as our solution. The proof of correctness is mentioned in the intuition.

Code

```
class Solution:
    # Approach 2: remove vertices with in-degree > 0
```

```python
def findSmallestSetOfVertices(self, n: int, edges: List[List[int]])
-> List[int]:

    ans = set([i for i in range(n)])

    for i,j in edges:
        if j not in ans: continue
        ans.remove(j)
    return ans
```

Complexity Analysis

- Time complexity: $O(m)$ assuming set removal takes only constant time using Hash table, the algorithm only takes $O(m)$ time, where m is the number of edges.
- Space complexity: $O(n)$ we only maintain an array of nodes, thus the algorithm only takes $O(n)$ space.