How to detect cycles in a directed graph using the iterative version of DFS?

Asked 3 years ago Active 1 month ago Viewed 3k times

5

4

In the recursive DFS, we can detect a cycle by coloring the nodes as `WHITE`, `GRAY` and `BLACK` as explained here.

A cycle exists if a `GRAY` node is encountered during the DFS search.

My question is: When do I mark the nodes as `GRAY` and `BLACK` in this iterative version of DFS? (from Wikipedia)

```
1  procedure DFS-iterative(G,v):
2      let S be a stack
3      S.push(v)
4      while S is not empty
5          v = S.pop()
6          if v is not labeled as discovered:
7              label v as discovered
8              for all edges from v to w in G.adjacentEdges(v) do
9                  S.push(w)
```

algorithm graph-algorithm depth-first-search Edit tags

edited Sep 30 '17 at 19:11

asked Sep 30 '17 at 19:00

shubham tibra
61 1 6

See my answer here: stackoverflow.com/a/60196714/1763149 – Marcin Raczyński Feb 12 at 21:35

4 Answers

Active Oldest Votes

7

One option is to push each node twice to the stack along the information if you're entering or exiting it. When you pop a node from stack you check if you're entering or exiting. In case of enter color it gray, push it to stack again and advance to neighbors. In case of exit just color it black.

Here's a short Python demo which detects a cycle in a simple graph:

```
WHITE = 0
GRAY = 1
BLACK = 2

EDGES = [(0, 1), (1, 2), (0, 2), (2, 3), (3, 0)]

ENTER = 0
EXIT = 1

def create_graph(edges):
    graph = defaultdict(list)
    for x, y in edges:
        graph[x].append(y)

    return graph

def dfs_iter(graph, start):
    state = {v: WHITE for v in graph}
    stack = [(ENTER, start)]
    while stack:
        act, v = stack.pop()

        if act == EXIT:
            print('Exit', v)
            state[v] = BLACK
        else:
            print('Enter', v)
            state[v] = GRAY
            stack.append((EXIT, v))
            for n in graph[v]:
                if state[n] == GRAY:
                    print('Found cycle at', n)
                elif state[n] == WHITE:
                    stack.append((ENTER, n))

graph = create_graph(EDGES)
dfs_iter(graph, 0)
```

Output:

```
Enter 0
Enter 2
Enter 3
Found cycle at 0
Exit 3
Exit 2
Enter 1
Exit 1
Exit 0
```

answered Sep 30 '17 at 19:32

niemmi
15.7k 7 26 32

Here you assumed that there is only one connected component. – Varun Narayanan Oct 5 at 8:35

@VarunNarayanan: The question was asking on how/when to mark nodes as gray/black and quoted iterative DFS algorithm instead of whole algorithm for finding connected components. As a result the answer is only about iterative DFS. – niemmi Oct 5 at 8:46

that makes sense :) – Varun Narayanan Oct 5 at 11:32

0

I have solved this problem as a solution for this Leetcode problem - https://leetcode.com/problems/course-schedule/

I have implemented it in Java - using recursive DFS using colors, recursive DFS using visited array, iterative DFS and BFS using indegree and calculating topological sort.

```
class Solution {
    //prereq is the edges and numCourses is number of vertices
    public boolean canFinish(int numCourses, int[][] prereq) {

        //0 -> White, -1 -> Gray, 1 -> Black
        int [] colors = new int[numCourses];
        boolean [] v = new boolean[numCourses];
        int [] inDegree = new int[numCourses];
        Map<Integer, List<Integer>> alMap = new HashMap<>();

        for(int i = 0; i < prereq.length; i++){
            int s = prereq[i][0];
            int d = prereq[i][1];
            alMap.putIfAbsent(s, new ArrayList<>());
```

```
        alMap.get(s).add(d);
        inDegree[d]++;
    }
    // if(hasCycleBFS(alMap, numCourses, inDegree)){
    //     return false;
    // }
    for(int i = 0; i < numCourses; i++){
        if(hasCycleDFS1(i, alMap, colors)){
        // if(hasCycleDFS2(i, alMap, v)){
        //if(hasCycleDFSIterative(i, alMap, colors)){
            return false;
        }
   }
    return true;
}
//12.48
boolean hasCycleBFS(Map<Integer, List<Integer>> alMap, int numCourses, int [] inDegree){
    //short [] v = new short[numCourses];
    Deque<Integer> q = new ArrayDeque<>();
    for(int i = 0; i < numCourses; i++){
        if(inDegree[i] == 0){
            q.offer(i);
        }
    }
```

answered Aug 26 at 2:26

coding_pleasures
797 1 9 19

2

You could do that simply by not popping the stack element right away. For every iteration, do `v = stack.peek()` and if `v` is `White`, mark it as `Grey` and go ahead exploring its neighbours.

However, if `v` is Grey, it means that you have encountered `v` for the second time in the stack and you have completed exploring it. Mark it `Black` and continue the loop.

Here's how your modified code should look like:

```
procedure DFS-iterative(G,v):
    let S be a stack
    S.push(v)
    while S is not empty
        v = S.peek()
        if v is not labeled as Grey:
            label v as Grey
            for all edges from v to w in G.adjacentEdges(v) do
                if w is labeled White do
                    S.push(w)
                elif w is labeled Grey do
                    return False    # Cycle detected
                                    # if w is black, it's already explored so ignore
        elif v is labeled as Grey:
            S.pop()                 # Remove the stack element as it has been explored
            label v as Black
```

If you're using a `visited` list to mark all visited nodes and another `recStack` i.e a list which keeps track of nodes currently being explored, then what you can do is, instead of popping the element from stack, just do `stack.peek()`. If the element is not visited (it means you're encountering that element for the first time in the stack), just mark it `True` in `visited` and `recStack` and explore its children.

However, if the `peek()` value is already visited, it means you're ending the exploration of that node so just pop it and make it's `recStack` as False again.

edited Jul 18 at 21:11

Nick Shebanov
601 7 21

answered Jul 18 at 17:06

Pranav Gor
21 3

0

In `DFS`, end of a branch is nodes that has no children these nodes is **Black**. Then checked parents of these nodes. If a parent do not has `Gray` child then it is **Black**. Likewise, if you continue to set black color to nodes, color of all nodes becomes black.

For example, I want to perform `DFS` in graph below.

ref. Introduction to Algorithms by Thomas Cormen

`DFS` starts from `u` and visited `u -> v -> y -> x`. `x` has no children and you should change color of this node to **Black**.

Then return to parent of `x` in visited path according to `discovery time`. So parent of `x` is `y`. `y` has no children with `Gray` color so you should change color of this node to **Black**.

edited Sep 30 '17 at 19:37

answered Sep 30 '17 at 19:16

Ali Soltani
7,764 3 19 38