ADA Theory Homework-4

Aditya Kumar (2022033) Nishant Kumar (2022326)

March 29, 2024

1 Algorithm Description

1.1 Initialization

- Initialize an empty list `cut_vertices` to store cut vertices.
- Implement a DFS-based algorithm to find articulation points (cut vertices) in the graph.
- Initialize arrays/lists: `visited` to track visited vertices, `disc` for discovery times, `low` for lowest reachable vertex, `parent` for parent vertices, and `is_articulation` to mark articulation points.

1.2 Depth First Search (DFS)

- Begin DFS traversal from every vertex in the graph.
- For each vertex u:
  - Set `disc[u]` and `low[u]` to the current time.
  - Explore each adjacent vertex v of u:
    - If v is not visited, mark u as its parent and recursively visit v.
    - Update `low[u]` by taking the minimum of `low[u]` and `low[v]`.
    - If u is the root of DFS and has more than one child, mark u as an articulation point.
    - If u is not the root and `low[v]` is greater than or equal to `disc[u]`, mark u as an articulation point.
    - Update `low[u]` to consider back edges.

1.3 Finding (s, t)-Cut Vertices

- After DFS traversal, iterate through all vertices.
- Any vertex marked as an articulation point is added to the `cut_vertices` list.

1.4 Output

- Return the list `cut_vertices`.

2 Time Complexity Analysis

2.1 Depth First Search (DFS)

The DFS traversal visits each vertex and edge once, resulting in a time complexity of $O(n+m)$, where n is the number of vertices and m is the number of edges in the graph. During DFS, for each vertex, we explore its adjacent vertices, which takes $O(1)$ time per edge. Since we visit each edge at most twice (once for each endpoint), the total time complexity for exploring adjacent vertices is $O(m)$. Therefore, the total time complexity for DFS is $O(n+m)$.

2.2 Finding (s, t)-Cut Vertices

After DFS traversal, iterating through all vertices to identify cut vertices takes $O(n)$ time since we examine each vertex once.

2.3 Overall Time Complexity

The overall time complexity of the algorithm is dominated by the time complexity of DFS traversal, which is $O(n + m)$. Therefore, the algorithm runs in $O(n + m)$ time.

3 Space Complexity Analysis

3.1 DFS Traversal

During DFS traversal, we maintain several arrays/lists: `visited`, `disc`, `low`, `parent`, and `is_articulation`, each requiring $O(n)$ space to store information for each vertex. Additionally, the recursive call stack for DFS may require $O(n)$ space in the worst case if the graph is a linear chain.

3.2 Finding (s, t)-Cut Vertices

The space required for storing the list of cut vertices is $O(n)$ since it can contain at most all vertices.

3.3 Overall Space Complexity

The overall space complexity is $O(n)$ since the dominant factor is the space required for DFS traversal and the storage of cut vertices.

4 Explanation of correctness of algorithm

The algorithm correctly identifies articulation points (cut vertices) using Depth First Search (DFS) traversal.

- An articulation point is a vertex whose removal disconnects the graph or increases its number of connected components. This property holds true for any graph, including directed acyclic graphs (DAGs).
- During DFS traversal, the algorithm meticulously explores the graph, maintaining necessary information such as discovery times (`disc`) and lowest reachable vertices (`low`) for each vertex.
- By identifying articulation points, the algorithm effectively finds vertices whose removal separates the path from s to t. This property is crucial in identifying (s, t)-cut vertices in a DAG.
- The algorithm ensures that all relevant (s, t)-cut vertices are correctly identified by examining the connectivity of the graph and the impact of vertex removal on the specified path from s to t.

In summary, the algorithm efficiently computes all (s, t)-cut vertices in a directed acyclic graph using DFS-based articulation point identification. It operates within polynomial time and space complexities and ensures correctness by correctly identifying vertices whose removal disconnects the specified path.

5 Pseudocode

Algorithm 1 Finding (s, t)-Cut Vertices

```
function FIND_CUT_VERTICES(G, s, t)
    cut_vertices ← []
    visited ← array of size |V| initialized to False
    disc ← array of size |V| initialized to 0
    low ← array of size |V| initialized to 0
    parent ← array of size |V| initialized to -1
    is_articulation ← array of size |V| initialized to False
    time ← 0
    for each vertex u in V do
        if not visited[u] then
            DFS(u, G, s, t, visited, disc, low, parent, is_articulation)
        end if
    end for
    for each vertex u in V do
        if is_articulation[u] then
            cut_vertices.append(u)
        end if
    end for
    return cut_vertices
end function

function DFS(u, G, s, t, visited, disc, low, parent, is_articulation)
    visited[u] ← True
    disc[u] ← low[u] ← time + 1
    children ← 0
    for each adjacent vertex v of u do
        if not visited[v] then
            children ← children + 1
            parent[v] ← u
            DFS(v, G, s, t, visited, disc, low, parent, is_articulation)
            low[u] ← min(low[u], low[v])
            if (u == s and children > 1) or (u ≠ s and low[v] ≥ disc[u]) then
                is_articulation[u] ← True
            end if
        else if v ≠ parent[u] then
            low[u] ← min(low[u], disc[v])
        end if
    end for
end function
```