Valid Arrangement Of Pairs

solved by	Senan
Platform	LeetCode
difficulty	Hard
Serial	2097
tags	Hash Map
language	C++
solved on	@30/11/2024
link	https://leetcode.com/problems/valid-arrangement-of-pairs/description/
Completion	☑

Intuition

The problem at hand requires finding an Eulerian path in a directed graph. An Eulerian path is a path that visits every edge exactly once. To achieve this:

1. Each directed edge in the graph must be considered as an individual connection between two nodes (start and end).
2. For an Eulerian path to exist, the graph must satisfy the condition that either:
    - Exactly one node has `in-degree = out-degree + 1`, and exactly one node has `out-degree = in-degree + 1` (starting and ending nodes).
    - All other nodes should have equal in-degrees and out-degrees (nodes in the middle of the path).
3. Once the graph is constructed, we will use an iterative depth-first search (DFS) approach (using a stack) to simulate the traversal of the Eulerian path.

Approach

1. **Graph Construction**:
    - First, build the directed graph using an adjacency list. Each directed edge from a pair `(start, end)` will be represented as an edge from `start` to `end`.
    - Track the in-degree and out-degree for each node in the graph. This helps us identify the starting node (if a node has out-degree greater than in-degree by 1) and the ending node (if in-degree is greater by 1).
2. **Find the Starting Node**:
    - The starting node of an Eulerian path is typically a node where the out-degree is one more than the in-degree (or any node if all degrees are balanced).
3. **Eulerian Path Traversal (DFS Simulation)**:
    - Start from the identified node and use a stack to simulate the traversal of the graph.
    - For each node, if there are adjacent nodes (edges), continue traversing. If there are no adjacent nodes, add the node to the result and backtrack.
4. **Result Construction**:
    - Once the Eulerian path is found, construct the solution by reversing the order of traversal.

Complexity

Time Complexity:

- **Graph construction**: O(E), where `E` is the number of edges (pairs) in the graph. We are iterating over each edge to build the adjacency list and degree map.
- **Eulerian path traversal**: O(E), as each edge is visited exactly once during the DFS traversal.
- **Result construction**: O(E), as we process the path to build the result.

Thus, the overall time complexity is **O(E)**, where `E` is the number of edges in the input.

Space Complexity:

- **Adjacency list and degree map**: O(V + E), where `V` is the number of nodes and `E` is the number of edges.
- **Eulerian path storage**: O(E), to store the resulting Eulerian path.

Thus, the overall space complexity is **O(V + E)**.

Code

```
class Solution {
public:
    unordered_map<int, vector<int>> adj;
    unordered_map<int, int> deg;

    inline void build_graph(vector<vector<int>>& pairs) {
        for (auto& edge : pairs) {
            int start = edge[0], end = edge[1];
            adj[start].push_back(end);
            deg[start]++;
            deg[end]--;
        }
    }

    vector<int> rpath;

    inline void euler(int i) {
        vector<int> stk = {i};
        while (!stk.empty()) {
            i = stk.back();
            if (adj[i].empty()) {
                rpath.push_back(i);
                stk.pop_back();
            } else {
                int j = adj[i].back();
                adj[i].pop_back();
                stk.push_back(j);
            }
        }
    }

    vector<vector<int>> validArrangement(vector<vector<int>>& pairs) {
        const int e = pairs.size();
        adj.reserve(e);
        deg.reserve(e);
```

```
        build_graph(pairs);

        int i0 = deg.begin()->first;

        for (auto& [v, d] : deg) {
            if (d == 1) {
                i0 = v;
                break;
            }
        }

        euler(i0);

        vector<vector<int>> ans;
        ans.reserve(e);

        for (int i = rpath.size() - 2; i >= 0; i--) {
            ans.push_back({rpath[i + 1], rpath[i]});
        }

        return ans;
    }
};
```