3243. Shortest Distance After Road Addition Queries I

You are given an integer `n` and a 2D integer array `queries`.

There are `n` cities numbered from `0` to `n - 1`. Initially, there is a **unidirectional** road from city `i` to city `i + 1` for all `0 <= i < n - 1`.

`queries[i] = [ui, vi]` represents the addition of a new **unidirectional** road from city `ui` to city `vi`. After each query, you need to find the **length** of the **shortest path** from city `0` to city `n - 1`.

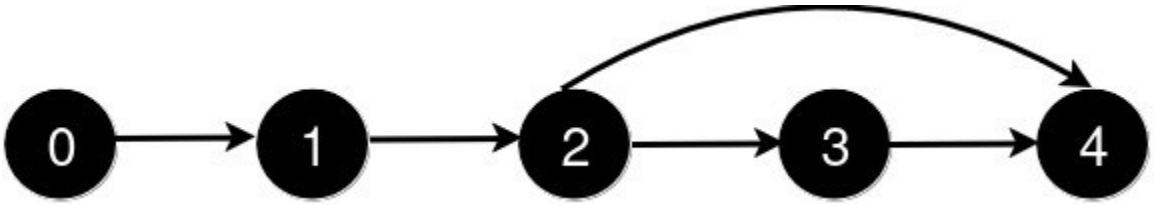
Return an array `answer` where for each `i` in the range `[0, queries.length - 1]`, `answer[i]` is the *length of the shortest path* from city `0` to city `n - 1` after processing the **first** `i + 1` queries.

Example 1:

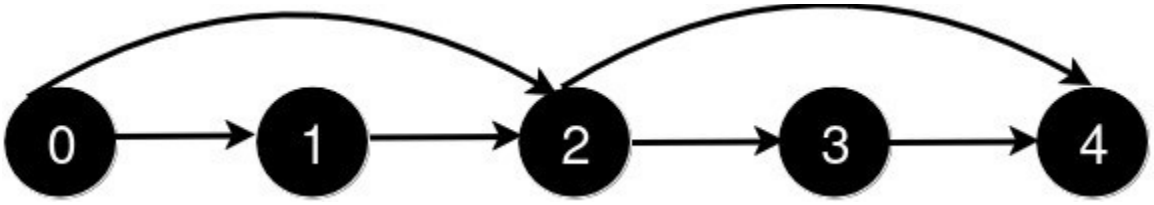
Input: n = 5, queries = [[2,4],[0,2],[0,4]]

Output: [3,2,1]

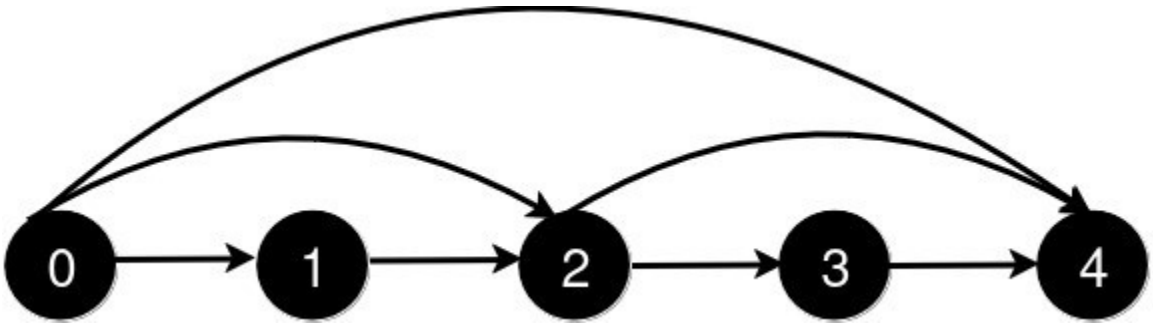
Explanation:



After the addition of the road from 2 to 4, the length of the shortest path from 0 to 4 is 3.



After the addition of the road from 0 to 2, the length of the shortest path from 0 to 4 is 2.



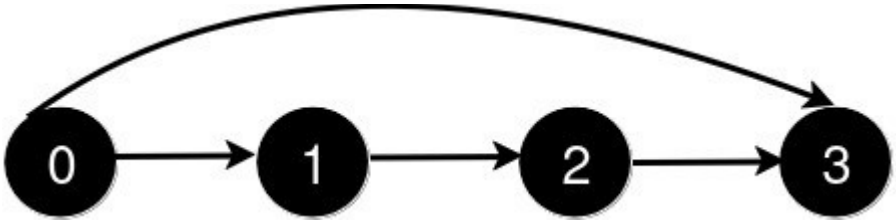
After the addition of the road from 0 to 4, the length of the shortest path from 0 to 4 is 1.

Example 2:

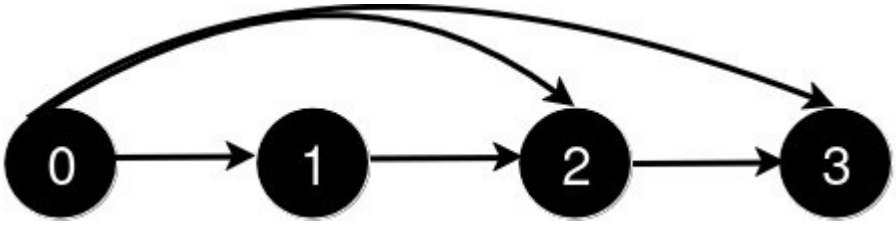
Input: n = 4, queries = [[0,3],[0,2]]

Output: [1,1]

Explanation:



After the addition of the road from 0 to 3, the length of the shortest path from 0 to 3 is 1.

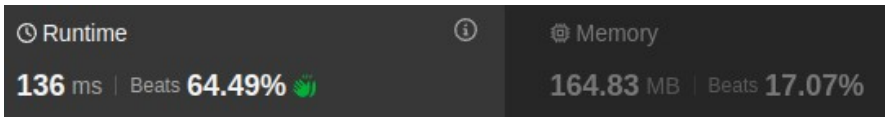


After the addition of the road from 0 to 2, the length of the shortest path remains 1.

Constraints:

- `3 <= n <= 500`
- `1 <= queries.length <= 500`
- `queries[i].length == 2`
- `0 <= queries[i][0] < queries[i][1] < n`
- `1 < queries[i][1] - queries[i][0]`
- There are no repeated roads among the queries.

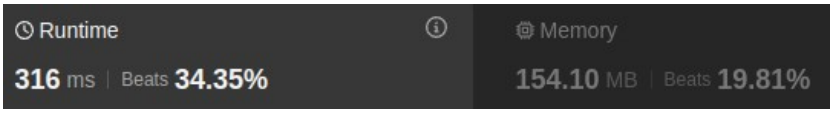
3243. Shortest Distance After Road Addition Queries I




```
/*
   BFS
   Time complexity: O(V+Q.(E+V))
   Space compelxity: O(V)
*/
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;
typedef std::pair<int,int> ii;
typedef std::vector<ii> vii;

class Solution{
   private:
      vvi graph;
   public:
      // Build the adjacency matrix of graph G(V,E)
      // Time compelxity: O(V)
      // Space compelxity: O(V)
      void build_graph(int n,vvi& edges){
         graph.resize(n);
         for(int i=0;i<n-1;++i){
            graph[i].push_back(i+1);
         }
      }
```

```
    // Time complexity: O(V+E)
    // Space compelxity: O(V)
    int bfs(int start, int target){
        vi visited(target+1,false);
        vi distances(target+1,INT_MAX);
        distances[0]=0;
        std::queue<int> q;
        q.push(0);
        int dist=0;
        while(!q.empty()){
            int cur_node=q.front();
            q.pop();

            if(cur_node==target) return distances[target];

            if(visited[cur_node]) continue;
            visited[cur_node]=true;

            for(auto& neighbor: graph[cur_node]){
                if(distances[neighbor]!=INT_MAX) continue;
                distances[neighbor]=distances[cur_node]+1;
                q.push(neighbor);
            }
        }
        return distances[target];
    }

    // Overall time complexity: O(V+Q.(V+E))
    // Overall space compelxity: O(V)
    vi shortestDistanceAfterQueries(int n, vvi& queries) {
        build_graph(n,queries);

        vi ans;

        for(auto& query: queries){
            int u=query[0];
            int v=query[1];
            graph[u].push_back(v);

            int shortest_path_lenght=bfs(0,n-1);
            ans.push_back(shortest_path_lenght);

        }
        return ans;
    }
};
```

3243. Shortest Distance After Road Addition Queries I




```
/*
   Dijkstra
   Time complexity: O(V+Q.(E log V))
   Space compelxity: O(V)
*/

typedef std::vector<int> vi;
typedef std::vector<vi> vvi;
typedef std::pair<int,int> ii;
typedef std::vector<ii> vii;

class Solution{
   private:
      vvi graph;
   public:
      // Build the adjacency matrix of graph G(V,E)
      // Time compelxity: O(V)
      // Space compelxity: O(V)
      void build_graph(int n,vvi& edges){
         graph.resize(n);
         for(int i=0;i<n-1;++i){
            graph[i].push_back(i+1);
         }
      }
```

```
    // Time complexity: O(E log V)
    // Space compelxity: O(V)
    int dijkstra(int start, int target){
        vi visited(target+1,false);
        vi distances(target+1,INT_MAX);
        distances[start]=0;
        std::priority_queue<ii,vii,std::greater<ii>> min_heap;
        min_heap.push({0,start});
        while(!min_heap.empty()){
            auto [dist,cur_node]=min_heap.top();
            min_heap.pop();
            if(visited[cur_node]) continue;
            visited[cur_node]=true;
            if(cur_node==target) return distances[target];
            for(auto& neighbor: graph[cur_node]){
                if(distances[neighbor]>distances[cur_node]+1){
                    distances[neighbor]=distances[cur_node]+1;
                    min_heap.push({distances[neighbor],neighbor});
                }
            }
        }
        return distances[target];
    }

    // Overall time complexity: O(V+Q.(E log V))
    // Overall space compelxity: O(V)
    vi shortestDistanceAfterQueries(int n, vvi& queries) {
        build_graph(n,queries);

        vi ans;
        for(auto& query: queries){
            int u=query[0];
            int v=query[1];
            graph[u].push_back(v);
            int shortest_path_lenght=dijkstra(0,n-1);
            ans.push_back(shortest_path_lenght);
        }
        return ans;
    }
};
```

3244. Shortest Distance After Road Addition Queries II

You are given an integer `n` and a 2D integer array `queries`.

There are `n` cities numbered from `0` to `n - 1`. Initially, there is a **unidirectional** road from city `i` to city `i + 1` for all `0 <= i < n - 1`.

`queries[i] = [ui, vi]` represents the addition of a new **unidirectional** road from city `ui` to city `vi`. After each query, you need to find the **length** of the **shortest path** from city `0` to city `n - 1`.

There are no two queries such that `queries[i][0] < queries[j][0] < queries[i][1] < queries[j][1]`.

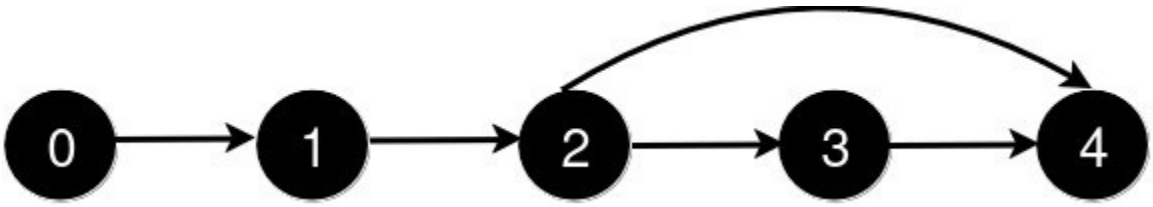
Return an array `answer` where for each `i` in the range `[0, queries.length - 1]`, `answer[i]` is the *length of the shortest path* from city `0` to city `n - 1` after processing the **first** `i + 1` queries.

Example 1:

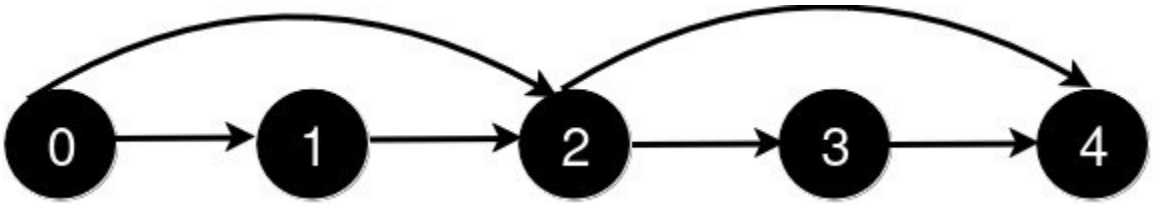
Input: n = 5, queries = [[2,4],[0,2],[0,4]]

Output: [3,2,1]

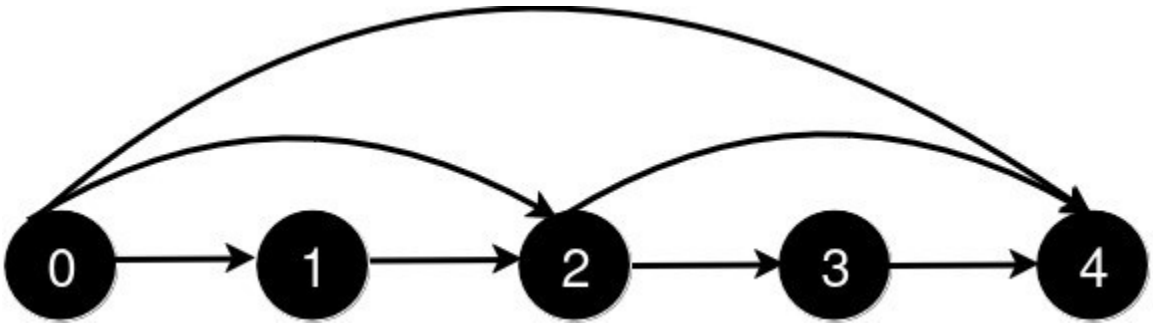
Explanation:



After the addition of the road from 2 to 4, the length of the shortest path from 0 to 4 is 3.



After the addition of the road from 0 to 2, the length of the shortest path from 0 to 4 is 2.



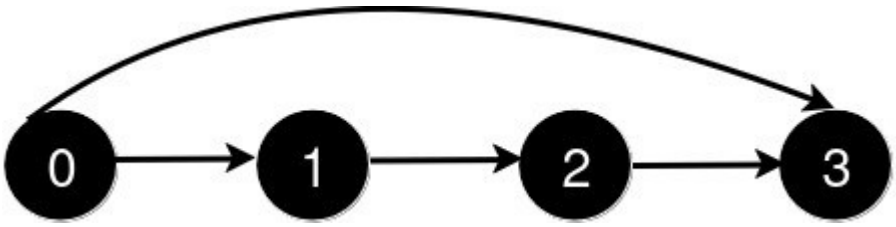
After the addition of the road from 0 to 4, the length of the shortest path from 0 to 4 is 1.

Example 2:

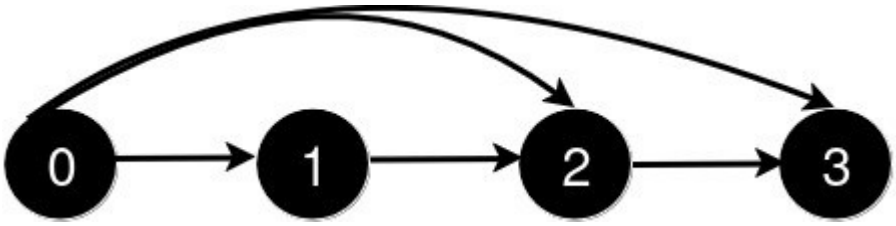
Input: n = 4, queries = [[0,3],[0,2]]

Output: [1,1]

Explanation:



After the addition of the road from 0 to 3, the length of the shortest path from 0 to 3 is 1.



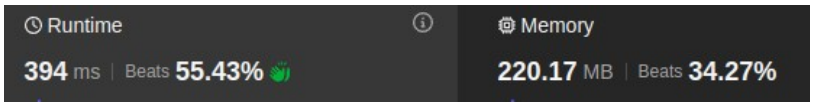
After the addition of the road from 0 to 2, the length of the shortest path remains 1.

Constraints:

- `3 <= n <= 105`
- `1 <= queries.length <= 105`
- `queries[i].length == 2`
- `0 <= queries[i][0] < queries[i][1] < n`
- `1 < queries[i][1] - queries[i][0]`
- There are no repeated roads among the queries.
- There are no two queries such that `i != j` and `queries[i][0] < queries[j][0] < queries[i][1] < queries[j][1]`.

3244. Shortest Distance After Road Addition Queries II

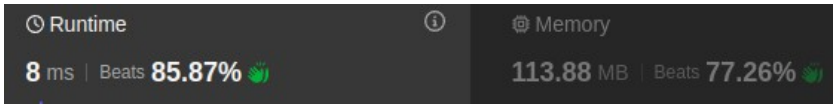
/*
Ordered set
Time complexity: O(nlogn+Q.n.logn)
Space complexity: O(n)
*/




```
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;

class Solution{
   public:
      std::vector<int> shortestDistanceAfterQueries(int n,vvi & queries){
         std::set<int> s;
         for(int i=0;i<n;++i) s.insert(i);
         vi ans;
         for(auto& query: queries){
            int u=query[0];
            int v=query[1];
            auto u_it=s.lower_bound(u+1);
            auto v_it=s.upper_bound(v-1);
            if(*v_it>=*u_it)  s.erase(u_it,v_it);
            ans.push_back(s.size()-1);
         }
         return ans;
      }
};
```

3244. Shortest Distance After Road Addition Queries II

/*
Array
Time complexity: O(Q.n)
Space complexity: O(n)
*/




```
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;


class Solution{
   public:
      std::vector<int> shortestDistanceAfterQueries(int n,vvi & queries){
         vi cities(n, 0), ans;
         for(int i=0;i<n-1;++i) cities[i]=i+1;
         int  dist=n-1;
         for(auto &query: queries){
            int u=query[0];
            int v=query[1];
            if (cities[u]!=INT_MIN){
               for(int i=cities[u];i<v;++i){
                  if (cities[i]!=INT_MIN){
                     cities[i]=INT_MIN;
                     dist--;
                  }
               }
               cities[u]=v;
            }
            ans.push_back(dist);
         }
         return ans;
      }
};
```