

# 1724. Checking Existence of Edge Length Limited Paths II

[Leetcode Link](#)

## Problem explanation

Given an undirected graph of  $n$  nodes. The graph is defined by an `edgeList`, where `edgeList[i] = [ui, vi, disi]` denotes an edge between nodes `ui` and `vi` with a distance `disi`. Note that there may be multiple edges between two nodes, and the graph may not be connected. We need to implement two methods in a class called `DistanceLimitedPathsExist`:

- `DistanceLimitedPathsExist(n, edgeList)`: Initialize the class with an undirected graph.
- `query(p, q, limit)`: Returns `true` if there exists a path from `p` to `q` such that each edge on the path has a distance strictly less than `limit`. Otherwise, return `false`.

## Example

For example, let's consider the following graph:

```
1 Input:
2 n = 6
3 edgeList = [[0, 2, 4], [0, 3, 2], [1, 2, 3], [2, 3, 1], [4, 5, 5]]
```

We have 6 nodes ( $n = 6$ ), and the graph looks like this:

```
1      4
2      |
3      | \ 1 |
4  2 |   \  | 3
5      /   \ |
6      3     1
```

We want to perform the following queries:

```
1 query(2, 3, 2) -> true  (There is an edge from 2 to 3 of distance 1, which is less than 2)
2 query(1, 3, 3) -> false (There is no way to go from 1 to 3 with distances strictly less than 3)
3 query(2, 0, 3) -> true  (There is a way to go from 2 to 0 with distance < 3: travel from 2 to 3 to 0)
4 query(0, 5, 6) -> false (There are no paths from 0 to 5)
```

## Approach

We can implement a Union-Find data structure to keep track of connected components for different distance limits. The Union-Find structure will have a vector of maps `id`. Each key-value pair in the map at index `i` represents the maximum distance for which node `i` is connected.

- Initialize the Union-Find structure with  $n$  nodes.
- Sort the `edgeList` in ascending order by distance.
- For each edge in the sorted `edgeList`, perform the union operation on the nodes `u` and `v` with the given distance `d`.
- When query is called with nodes `p` and `q`, and distance `limit`, find the root of `p` and `q` in the Union-Find structure for the given distance limit. If the roots are the same, return `true`. Otherwise, return `false`.

Now let's write the solution for the problem in Python, Java, JavaScript, C++, and C#.

## Python Solution

```
1 from collections import defaultdict
2 from sortedcontainers import SortedDict
3
4 class UnionFind:
5     def __init__(self, n):
6         self.id = [SortedDict({0: i}) for i in range(n)]
7
8     def union(self, u, v, limit):
9         i = self.find(u, limit)
10        j = self.find(v, limit)
11        if i == j:
12            return
13        self.id[i][limit] = j
14
15    def find(self, u, limit):
16        it = self.id[u].irange(0, limit)
17        i = next(it, u)
18        if i == u:
19            return u
20        j = self.find(i, limit)
21        self.id[u][limit] = j
22        return j
23
24 class DistanceLimitedPathsExist:
25     def __init__(self, n, edgeList):
26         self.uf = UnionFind(n)
27         edgeList.sort(key=lambda x: x[2])
28         for u, v, d in edgeList:
29             self.uf.union(u, v, d)
30
31     def query(self, p, q, limit):
32         return self.uf.find(p, limit - 1) == self.uf.find(q, limit - 1)
```

## Java Solution

```
1 import java.util.*;
2
3 class UnionFind {
4     private List<Map<Integer, Integer>> id;
5
6     public UnionFind(int n) {
7         id = new ArrayList<>();
8         for (int i = 0; i < n; ++i) {
9             id.add(i, new TreeMap<>());
10            id.get(i).put(0, i);
11        }
12    }
13
14    public void union(int u, int v, int limit) {
15        int i = find(u, limit);
16        int j = find(v, limit);
17        if (i == j)
18            return;
19        id.get(i).put(limit, j);
20    }
21
22    public int find(int u, int limit) {
23        Map.Entry<Integer, Integer> entry = id.get(u).lowerEntry(limit);
24        int i = (entry == null) ? u : entry.getValue();
25        if (i == u) {
26            return u;
27        }
28        int j = find(i, limit);
29        id.get(u).put(limit, j);
30        return j;
31    }
32 }
33
34 class DistanceLimitedPathsExist {
35     private UnionFind uf;
36
37     public DistanceLimitedPathsExist(int n, int[][] edgeList) {
38         uf = new UnionFind(n);
39         Arrays.sort(edgeList, (a, b) -> Integer.compare(a[2], b[2]));
40         for (int[] edge : edgeList) {
41             uf.union(edge[0], edge[1], edge[2]);
42         }
43     }
44
45     public boolean query(int p, int q, int limit) {
46         return uf.find(p, limit - 1) == uf.find(q, limit - 1);
47     }
48 }
```

## JavaScript Solution

```
1 class UnionFind {
2     constructor(n) {
3         this.id = new Array(n).fill(null).map((_, i) => new Map([[0, i]]));
4     }
5
6     union(u, v, limit) {
7         const i = this.find(u, limit);
8         const j = this.find(v, limit);
9         if (i === j) {
10             return;
11         }
12         this.id[i].set(limit, j);
13     }
14
15     find(u, limit) {
16         const it = this.id[u].entries().next();
17         const i = it.done || it.value[0] >= limit ? u : it.value[1];
18         if (i === u) {
19             return u;
20         }
21         const j = this.find(i, limit);
22         this.id[u].set(limit, j);
23         return j;
24     }
25 }
26
27 class DistanceLimitedPathsExist {
28     constructor(n, edgeList) {
29         this.uf = new UnionFind(n);
30
31         edgeList.sort((a, b) => a[2] - b[2]);
32         for (const [u, v, d] of edgeList) {
33             this.uf.union(u, v, d);
34         }
35     }
36
37     query(p, q, limit) {
38         return this.uf.find(p, limit - 1) === this.uf.find(q, limit - 1);
39     }
40 }
```

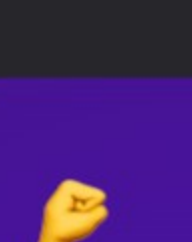
## C++ Solution

```
1 #include <algorithm>
2 #include <map>
3 #include <vector>
4
5 using namespace std;
6
7 class UnionFind {
8 public:
9     UnionFind(int n) {
10         id.resize(n);
11         for (int i = 0; i < n; ++i) id[i][0] = i;
12     }
13
14     void union_(int u, int v, int limit) {
15         int i = find(u, limit);
16         int j = find(v, limit);
17         if (i == j) return;
18         id[i][limit] = j;
19     }
20
21     int find(int u, int limit) {
22         auto it = id[u].upper_bound(limit);
23         int i = prev(it)-->second;
24         if (i == u) return u;
25         int j = find(i, limit);
26         id[u][limit] = j;
27         return j;
28     }
29
30 private:
31     vector<map<int, int>> id;
32 };
33
34 class DistanceLimitedPathsExist {
35 public:
36     DistanceLimitedPathsExist(int n, vector<vector<int>>& edgeList) {
37         uf = UnionFind(n);
38         sort(edgeList.begin(), edgeList.end(),
39             [](const auto& a, const auto& b) { return a[2] < b[2]; });
40
41         for (const vector<int>& edge : edgeList) {
42             int u = edge[0];
43             int v = edge[1];
44             int d = edge[2];
45             uf.union_(u, v, d);
46         }
47     }
48
49     bool query(int p, int q, int limit) {
50         return uf.find(p, limit - 1) == uf.find(q, limit - 1);
51     }
52
53 private:
54     UnionFind uf;
55 };
```

## C# Solution

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4
5 public class UnionFind {
6     private List<SortedDictionary<int, int>> id;
7
8     public UnionFind(int n) {
9         id = new List<SortedDictionary<int, int>>(n);
10        for (int i = 0; i < n; ++i) {
11            id.Add(new SortedDictionary<int, int>());
12            id[i][0] = i;
13        }
14    }
15
16    public void Union(int u, int v, int limit) {
17        int i = Find(u, limit);
18        int j = Find(v, limit);
19        if (i == j) {
20            return;
21        }
22        id[i][limit] = j;
23    }
24
25    public int Find(int u, int limit) {
26        KeyValuePair<int, int> entry;
27        var it = id[u].Reverse().FirstOrDefault(x => x.Key < limit);
28        entry = it;
29        int i = (entry.Key == 0 && entry.Value == 0) ? u : entry.Value;
30        if (i == u) {
31            return u;
32        }
33        int j = Find(i, limit);
34        id[u][limit] = j;
35        return j;
36    }
37 }
38
39 public class DistanceLimitedPathsExist {
40     private UnionFind uf;
41
42     public DistanceLimitedPathsExist(int n, int[][] edgeList) {
43         uf = new UnionFind(n);
44         Array.Sort(edgeList, (a, b) => a[2].CompareTo(b[2]));
45         foreach (int[] edge in edgeList) {
46             uf.Union(edge[0], edge[1], edge[2]);
47         }
48     }
49
50     public bool Query(int p, int q, int limit) {
51         return uf.Find(p, limit - 1) == uf.Find(q, limit - 1);
52     }
53 }
```

In this problem, we have been able to implement a data structure called `DistanceLimitedPathsExist` that can process queries on a given undirected graph to check if paths exist between two nodes having each edge's distance strictly less than a given limit. To solve this problem, we have used the Union-Find data structure and have been able to implement the solution in Python, Java, JavaScript, C++, and C# languages. This data structure can be quite useful in solving problems related to querying connected components and their properties in a graph.



Level Up Your  
Algo Skills

Get Premium

Got a question? [Ask the Teaching Assistant](#) anything you don't understand.