Bus Routes

We have a list of bus routes. Each `routes[i]` is a bus route that the i-th bus repeats forever. For example if `routes[0] = [1, 5, 7]`, this means that the first bus (0-th indexed) travels in the sequence 1->5->7->1->5->7->1->... forever.

We start at bus stop `S` (initially not on a bus), and we want to go to bus stop `T`. Travelling by buses only, what is the least number of buses we must take to reach our destination? Return -1 if it is not possible.

Example:
Input:
```
routes = [[1, 2, 7], [3, 6, 7]]
S = 1
T = 6
```
Output: 2
Explanation:
The best strategy is take the first bus to the bus stop 7, then take the second bus to the bus stop 6.

Note:

- `1 <= routes.length <= 500`.
- `1 <= routes[i].length <= 500`.
- `0 <= routes[i][j] < 10 ^ 6`.

Solution 1

For each of the bus stop, we maintain all the buses (bus routes) that go through it. To do that, we use a HashMap, where bus stop number is the key and all the buses (bus routes) that go through it are added to an ArrayList.

We use BFS, where we process elements in a level-wise manner. We add the Start bus stop in the queue. Next, when we enter the while loop, we add all the bus stops that are reachable by all the bus routes that go via the Start. Thus, if we have the input as [[1, 2, 7], [3, 6, 7]] and Start as 6, then upon processing bus stop 6, we would add bus stops 3 and 7.
With this approach, all the bus stops at a given level, are “equal distance” from the start node, in terms of number of buses that need to be changed.

To avoid loops, we also maintain a HashSet that stores the buses that we have already visited.

```
class Solution {
    public int numBusesToDestination(int[][] routes, int S, int T) {
       HashSet<Integer> visited = new HashSet<>();
       Queue<Integer> q = new LinkedList<>();
       HashMap<Integer, ArrayList<Integer>> map = new HashMap<>();
       int ret = 0;

       if (S==T) return 0;

       for(int i = 0; i < routes.length; i++){
            for(int j = 0; j < routes[i].length; j++){
                ArrayList<Integer> buses = map.getOrDefault(routes[i][j], new Array
List<>());
                buses.add(i);
                map.put(routes[i][j], buses);
            }
        }

       q.offer(S);
       while (!q.isEmpty()) {
           int len = q.size();
           ret++;
           for (int i = 0; i < len; i++) {
               int cur = q.poll();
               ArrayList<Integer> buses = map.get(cur);
               for (int bus: buses) {
                    if (visited.contains(bus)) continue;
                    visited.add(bus);
                    for (int j = 0; j < routes[bus].length; j++) {
                        if (routes[bus][j] == T) return ret;
                        q.offer(routes[bus][j]);
                    }
               }
           }
        }
        return -1;
    }
}
```

written by trotro original link here

Solution 2

The first part loop on routes and record stop to routes mapping in `to_route`.
The second part is general bfs. Take a stop from queue and find all connected route.
The set `seen` record all stops visted and we won't check a stop for twice.
We can also use a set to record all routes visted , or just clear a route after visit.

C++:

```
    int numBusesToDestination(vector<vector<int>>& routes, int S, int T) {
        unordered_map<int, unordered_set<int>> to_routes;
        for (int i = 0; i < routes.size(); ++i) for (auto& j : routes[i]) to_routes
[j].insert(i);
        queue<pair<int, int>> bfs; bfs.push(make_pair(S, 0));
        unordered_set<int> seen = {S};
        while (!bfs.empty()) {
            int stop = bfs.front().first, bus = bfs.front().second;
            bfs.pop();
            if (stop == T) return bus;
            for (auto& route_i : to_routes[stop])
                for (auto& next_stop : routes[route_i])
                    if (seen.find(next_stop) == seen.end()) {
                        seen.insert(next_stop);
                        bfs.push(make_pair(next_stop, bus + 1));
                    }
        }
        return -1;
    }
```

Java:

```
    public int numBusesToDestination(int[][] routes, int S, int T) {
        HashMap<Integer, HashSet<Integer>> to_routes = new HashMap<>();
        for (int i = 0; i < routes.length; ++i)
            for (int j : routes[i]) {
                if (!to_routes.containsKey(j)) to_routes.put(j, new HashSet<Integer
>());
                to_routes.get(j).add(i);
            }
        Queue<Point> bfs = new ArrayDeque();
        bfs.offer(new Point(S, 0));
        HashSet<Integer> seen = new HashSet<>();
        seen.add(S);
        while (!bfs.isEmpty()) {
            int stop = bfs.peek().x, bus = bfs.peek().y;
            bfs.poll();
            if (stop == T) return bus;
            for (int route_i : to_routes.get(stop))
                for (int next_stop : routes[route_i])
                    if (!seen.contains(next_stop)) {
                        seen.add(next_stop);
                        bfs.offer(new Point(next_stop, bus + 1));
                    }
        }
        return -1;
    }
```

Python:

```python
    def numBusesToDestination(self, routes, S, T):
        to_routes = collections.defaultdict(set)
        for i,route in enumerate(routes):
            for j in route: to_routes[j].add(i)
        bfs = [(S,0)]
        seen = set([S])
        for stop, bus in bfs:
            if stop == T: return bus
            for route_i in to_routes[stop]:
                for next_stop in routes[route_i]:
                    if next_stop not in seen:
                        bfs.append((next_stop, bus+1))
                        seen.add(next_stop)
                routes[route_i] = []
        return -1
```

written by lee215 original link here

Solution 3

Using BFS here is straightforawrd.

Maintain a queue for routes.

Every time, you pop out a route from the queue and go through every stop in the route.

- If the stop is the target, then you are done.
- if the stop is not visited before, add its “related” and “unvisited” routes into queue.
- if the stop is visited before, ignore it.

```
from collections import defaultdict
from collections import deque
class Solution:
    def numBusesToDestination(self, routes, S, T):
        """
        :type routes: List[List[int]]
        :type S: int
        :type T: int
        :rtype: int
        """
        if S==T:
            return 0
        visitedRoute = set()
        visitedStop = set()
        stopToRoute = defaultdict(list)
        for i,route in enumerate(routes):
            for stop in route:
                stopToRoute[stop].append(i)
        Queue = deque([])
        visitedStop.add(S)
        for routeNumber in stopToRoute[S]:
            visitedRoute.add(routeNumber)
            Queue.append([routes[routeNumber], 1])
        while len(Queue):
            route, take = Queue.popleft()
            for stop in route:
                if stop == T:
                    return take
                elif stop not in visitedStop:
                    visitedStop.add(stop)
                    for routeNumber in stopToRoute[stop]:
                        if routeNumber not in visitedRoute:
                            Queue.append([routes[routeNumber],take+1])
        return -1
```

written by Peterzy original link here