815. Bus Routes

Description (/problems/bus-routes/description/) Hints (/problems/bus-routes/hints/) Submissions (/problems/bus-routes/submis

Quick Navigation View in Article (/articles/bus-routes/)

Approach #1: Breadth First Search [Accepted]

Intuition

Instead of thinking of the stops as nodes (of a graph), think of the buses as nodes. We want to take the least number of buses, which is a shortest path problem, conducive to using a breadth-first search.

Algorithm

We perform a breadth first search on bus numbers. When we start at `S`, originally we might be able to board many buses, and if we end at `T` we may have many `targets` for our goal state.

One difficulty is to efficiently decide whether two buses are connected by an edge. They are connected if they share at least one bus stop. Whether two lists share a common value can be done by set intersection (HashSet), or by sorting each list and using a two pointer approach.

To make our search easy, we will annotate the depth of each node: `info[0] = node, info[1] = depth`.

Java | Python

Copy

```java
import java.awt.Point;

class Solution {
    public int numBusesToDestination(int[][] routes, int S, int T) {
        if (S==T) return 0;
        int N = routes.length;

        List<List<Integer>> graph = new ArrayList();
        for (int i = 0; i < N; ++i) {
            Arrays.sort(routes[i]);
            graph.add(new ArrayList());
        }
        Set<Integer> seen = new HashSet();
        Set<Integer> targets = new HashSet();
        Queue<Point> queue = new ArrayDeque();

        // Build the graph.  Two buses are connected if
        // they share at least one bus stop.
        for (int i = 0; i < N; ++i)
            for (int j = i+1; j < N; ++j)
                if (intersect(routes[i], routes[j])) {
                    graph.get(i).add(j);
                    graph.get(j).add(i);
                }

        // Initialize seen, queue, targets.
        // seen represents whether a node has ever been enqueued to queue.
```

Notes

Complexity Analysis

- Time Complexity: Let N denote the number of buses, and b_i be the number of stops on the ith bus.
    - To create the graph, in Python we do $O(\sum(N-i)b_i)$ work (we can improve this by checking for which of `r1, r2` is smaller), while in Java we did a $O(\sum b_i \log b_i)$ sorting step, plus our searches are $O(N \sum b_i)$ work.
    - Our (breadth-first) search is on N nodes, and each node could have N edges, so it is $O(N^2)$.
- Space Complexity: $O(N^2 + \sum b_i)$ additional space complexity, the size of `graph` and `routes`. In Java, our space complexity is $O(N^2)$ because we do not have an equivalent of `routes`. Dual-pivot quicksort (as used in `Arrays.sort(int[])`) is an in-place algorithm, so in Java we did not increase our space complexity by sorting.

Analysis written by: @awice (https://leetcode.com/awice).

Join the conversation

Signed in as **tan7**.

Post a Reply

Notes

EatSleepLeetCode commented 3 days ago

@pedro (https://discuss.leetcode.com/uid/133065) Thank you so much!
iscuss.leetcode.com/user/eatsleepleetcode)

pedro commented 3 days ago

@EatSleepLeetCode (https://discuss.leetcode.com/uid/249579) btw, in python the
iscuss.leetcode.com/user/pedro) work to create the graph is O(N^3) - that's what the summation boils down to in the worst case - which dominates the time spent in the BFS actually.

If you pre-process to get a map from stop -> bus route in O(N^2), we can still do a BFS on the stops that is O(N^2), making it significantly faster. Keep track of visited bus stops *and* visited bus routes. Whenever you visit a stop, for each bus you have not used that goes to that stop, add all the unvisited stops from that bus route to the queue.

On each stop you'll have to consider all the bus routes which is O(N) per stop or O(N^2) total and since you only have to look at the stops on a given bus route once, the total work is O(N^2).

pedro commented 3 days ago

@EatSleepLeetCode (https://discuss.leetcode.com/uid/249579) There are N nodes,
iscuss.leetcode.com/user/pedro) each node could have N edges. Max total edges = N^2.

EatSleepLeetCode commented 3 days ago

Thanks for posting the solution. I have confusion regarding time complexity for BFS.
iscuss.leetcode.com/user/eatsleepleetcode) Shouldn't it be O(N + N) i.e. N nodes and N edges i.e. O(N). I am not sure why is it O(N^2).

View original thread (https://discuss.leetcode.com/topic/121458)

Notes