INF421 PI 2022: Most Pleasant Itineraries

Patrick Loiseau

patrick.loiseau@inria.fr

Preliminary version – December 5, 2022

Paris is a beautiful city with great charm. Unfortunately, local Parisians complain that many paths in the city are very noisy because of tourists or vehicles. As a result, when they travel inside the city, they do not necessarily take shortest itineraries, but instead, more pleasant ones, i.e., itineraries with less noise. Could you please help them to find out the most pleasant itineraries?

You are given a connected and undirected graph $G = (V, E)$ representing the map of the city. For every edge $e \in E$, there is a non-negative integer value c_e indicating the noise level at the edge e.

For a pair (u, v) of vertices from V, *a most pleasant itinerary between u and v* is a u-to-v path in the graph G such that the maximum noise level c_e of all edges e along this path is minimized.[1]

You are given a set of queries, each containing a pair of vertices (u, v) from V. For each query, you need to compute the maximum noise level on a most pleasant itinerary between u and v.[2]

Programs should be written in either C++ or Java. A description of the test data is provided at the end of this document.

Task 1

Consider any minimum spanning tree T of G. Show that for any query (u, v), the answer to this query (i.e., the maximum noise level on a most pleasant itinerary between u and v) in the graph G is the same as the answer to that query in the tree T.

Task 2

Based on the above observation, write a program `itineraries_v1` for the problem of the most pleasant itineraries. The program first computes a minimum spanning tree T, and then answers the queries naively in the tree T.

Task 3

In the program `itineraries_v1`, the time to answer each query might be $O(n)$, which is not affordable when there is a large number of queries. The goal in this task is to reduces the query time to $O(\log n)$.

[1] Such a path might not be unique.

[2] By default, the maximum noise level on an empty path is 0.

We observe that a u-to-v path in a tree is unique, and this path can be decomposed into two parts with respect to the *Lowest Common Ancestor* of u and v.

Definition 1. The *Lowest Common Ancestor (LCA)* of two vertices u and v in a tree T is the lowest (i.e., deepest) vertex that has both u and v as descendants.[3]

Therefore, you need to solve the following subproblems, each within $O(\log n)$ time:

1. Find the LCA of two vertices u, v in the tree T;
2. Compute the maximum noise level on the path between a vertex u and some ancestor of u in the tree T.

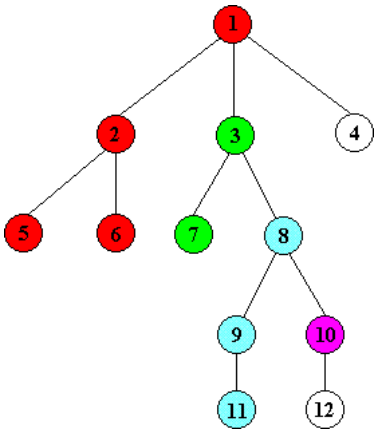
Hint: You may want to use a preprocessing step, which computes, for each vertex u and for *every value* $h \in \mathbb{N}$ *that is a power of* 2, the ancestor of u that is at distance h from u, as well as the maximum noise level on the path between u and this ancestor. Also, if u and v are not at the same level in the tree T (suppose the level of u is larger), you can first find an ancestor u' of u that is at the same level of v, and then compute the LCA of u' and v.

Write a program `itineraries_v2` for the problem of the most pleasant itineraries, based on the above approach. Conclude that the preprocessing time is $O(n \log n)$ and afterwards the time to answer each query is $O(\log n)$.

Task 4

The goal in this task is to improve upon the time complexity of `itineraries_v2`, by using Tarjan's algorithm. The main novelty in Tarjan's algorithm is to perform a Depth-First-Search (DFS) and to compute the answer to each query (u, v) when the DFS procedure is visiting one of the vertices u and v.

Let us look at an example:



When the DFS is visiting the purple vertex, the algorithm computes the answers to all queries (u, v) such that u is the purple vertex, and v has been visited previously in the DFS. There are three cases:

- If v is among the blue vertices, then LCA(u, v)=Node 8;

[3] We define each vertex to be a descendant of itself. Thus if u has a direct connection from v, then v is the lowest common ancestor.

- If v is among the green vertices, then LCA(u, v)=Node 3;
- If v is among the red vertices, then LCA(u, v)=Node 1.

Therefore, for the vertices already visited in the DFS previously, the key is to decide which of the three cases they belong to.[4] This can be achieved by the disjoint-set data structure.

The pseudocode below determines the lowest common ancestor of each pair of vertices from the set of queries P, given the root r of a tree T. It uses the MakeSet, Find, and Union functions of the disjoint-set data structure, in which each element has a parent pointer. MakeSet(u) creates a singleton set containing u, Find(u) returns the standard representative of the set containing u, and Union(u, v) merges the set containing u with the set containing v. The function TarjanLCA is first called on the root r.

Algorithm 1 Tarjan's lowest common ancestors algorithm

```
1:  function TarjanLCA(u)
2:      MakeSet(u)
3:      u.parent ← u
4:      for each child v of u in the tree T do
5:          TarjanLCA(v)
6:          Union(u, v)
7:          Find(u).parent ← u
8:      u.visited ← true
9:      for each v such that (u, v) ∈ P do
10:         if v.visited then
11:             the answer to query (u, v) ← Find(v).parent
```

Each vertex is marked as *visited* after it and all its children have been visited. The lowest common ancestor of the pair (u, v) is available immediately after u is marked as visited, provided that v has already been visited before. Otherwise, it will be available later, immediately after v is marked as visited.

Write a program `itineraries_v3` for the problem of the most pleasant itineraries, by implementing Tarjan's algorithm. Show that the preprocessing time is linear in n and afterwards the time to answer each query is constant in average. (We remark that Tarjan's algorithm can be applied only if the set of queries P is given in advance.)

Tests

There are 10 tests, and the input data can be found by clicking here.

Input Format. Your program reads from the input file `itineraries.in`, which contains the following:

- On the first line, there are two integers n and m, indicating the number of vertices and the number of edges in the graph G;
- On the next m lines, each line contains three integers u, v, c with $u, v \in \{1, \ldots, n\}$ and $0 \leq c < 2^{31}$, indicating that there is an edge (u, v) in the graph G with noise level c;

[4] In general, there could be an arbitrary number of cases.

- On the $(m+2)^{\text{th}}$ line, there is an integer ℓ, indicating the number of queries;
- On the next ℓ lines, each line contains two integers $u, v \in \{1, \ldots, n\}$, indicating a query (u, v).

We ensure that $n \leq 200000$, $m \leq 300000$, and $\ell \leq 500000$.

Output Format. Your program outputs the solution to the file `itineraries.out`. This file should contain ℓ lines, where the i^{th} line is an integer indicating the answer to the i^{th} query.

Report on the Running Time. In your report, please include a summary of the running time by each of the three algorithms `itineraries_v1`, `itineraries_v2`, and `itineraries_v3` and on each of the 10 tests. (Indicate "N/A" if some algorithm does not provide an output on a test after 30 seconds.)