1.7 Road Trip

Andre Monteiro

Term 3, 2025

Problem

This task involves creating a Road Trip R within Austria.

R is a path through Austria where you arrive at a town by plane, visit one or more towns via roads and then leave by plane.

For example:

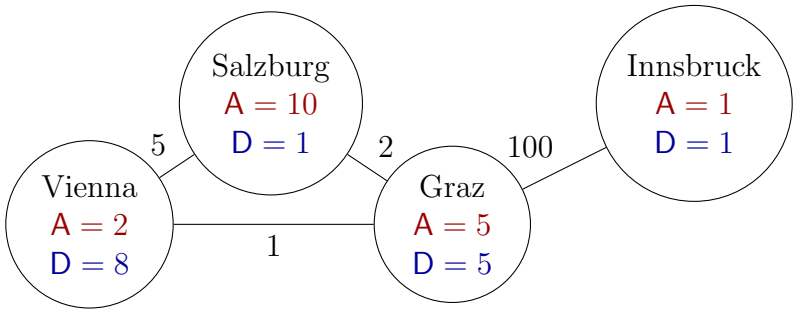

Figure 1

In Figure 1, the cheapest road trip is to enter at Vienna, drive to Graz then Salzburg, and leave via Salzburg. The total cost is

$$\mathsf{Arrive}(\text{Vienna}) + \mathsf{Toll}(\text{Vienna}, \text{Graz}) + \mathsf{Toll}(\text{Graz}, \text{Salzburg}) + \mathsf{Depart}(\text{Salzburg}) = 2 + 1 + 2 + 1 = 6.$$

While it would be cheaper to fly into Innsbruck then immediately fly out, this is not a road trip as no roads are taken. Finally, a road trip does not have to include all towns.

Solution

The intention of this problem is to solve it via reduction, achieving a resultant time complexity of:

$$O(n \log m)$$

We model the problem as a weighted graph with non-negative edges and apply Dijkstra's algorithm to find shortest paths efficiently. To avoid the cost of computing all-pairs distances, we introduce a single Arrival node and a single Departure node. The Arrival node connects to every city with an edge weighted by its arrival cost, and the Departure node connects from every city with an edge weighted by its departure cost.

These nodes ensure there is exactly one source for Dijkstra to start from and that all arrival and departure costs are correctly represented.

At this stage, every path from Arrival to Departure includes exactly one arrival cost and one departure cost, so any path considered by Dijkstra accounts for these endpoints. This reduction allows a single run of Dijkstra's to solve the problem efficiently, avoiding the overhead of all-pairs computations.

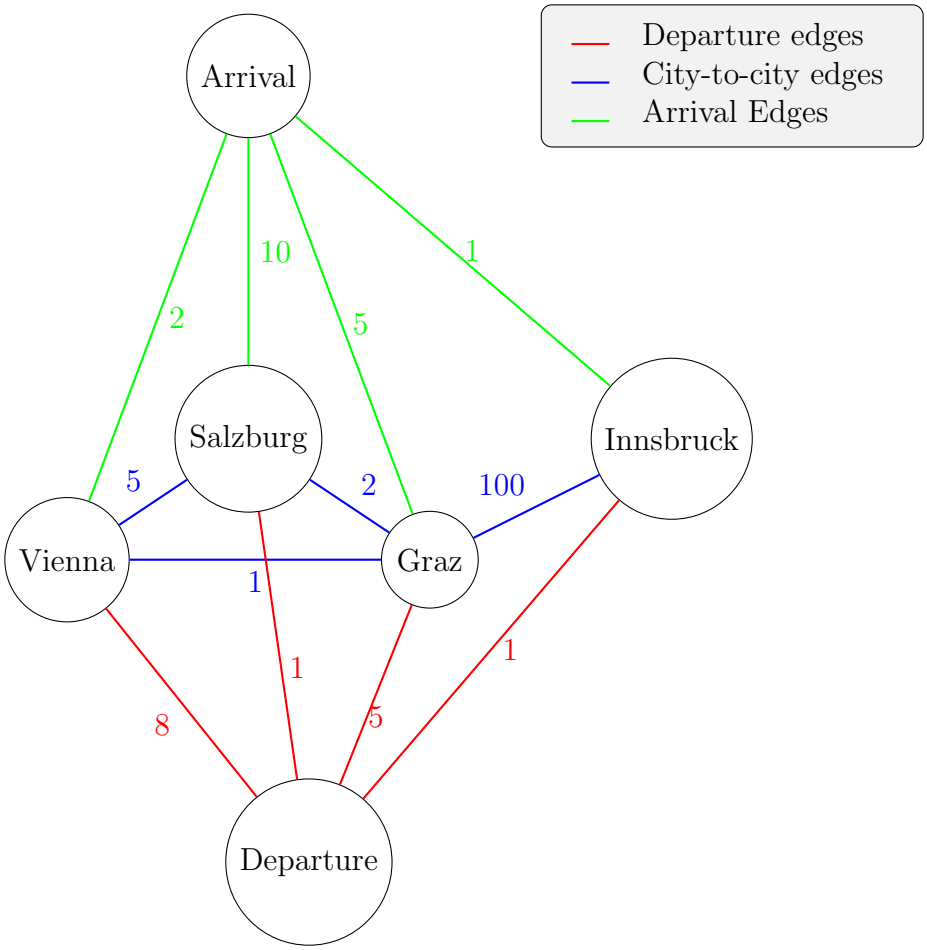

Figure 2

Time complexity:

Insertion for Arrival vertices:	$O(m)$
Insertion for Departure vertices:	$O(m)$
Total insertion:	$O(m) + O(m) = O(m)$

A small improvement we can make when considering aymptomatic complexity, is to create the vertices for Arrival and Departure as directed edges, and therefore slightly reduce time:

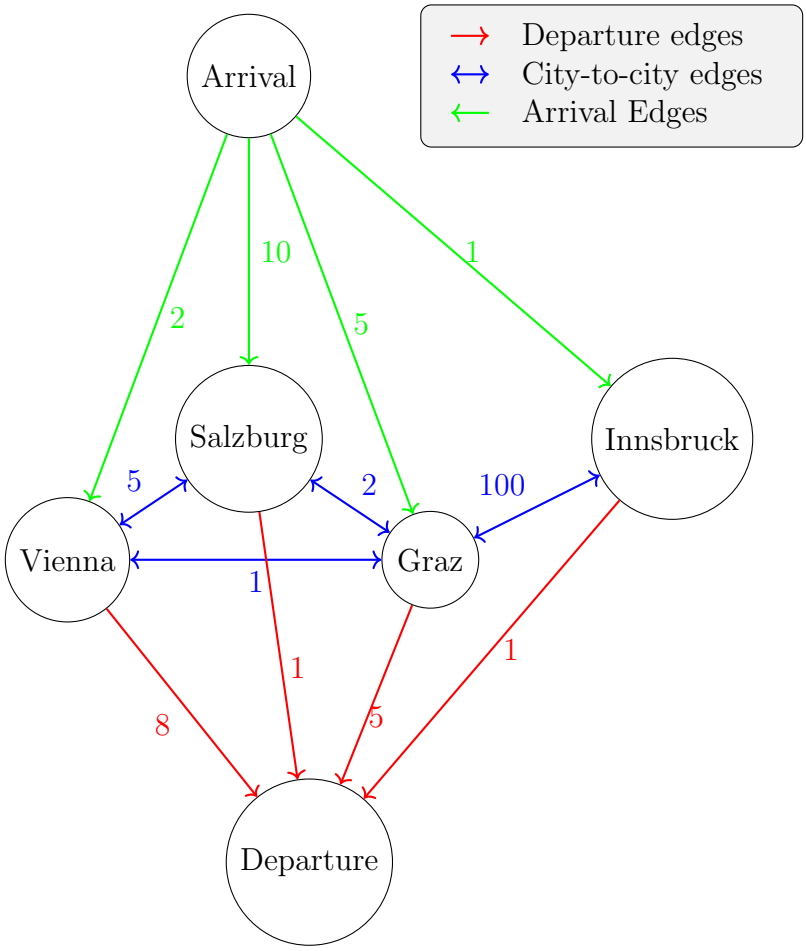

Figure 3

After adding the Arrival and Departure reference vertices, we can run Dijkstra's. However, one illegal path remains:

$$\mathsf{Arrive}(\text{Innsbruck}) + \mathsf{Depart}(\text{Innsbruck}) = 1 + 1 = 2,$$

which violates the "at least one road taken" rule.

To enforce this, we insert a gateway layer of vertices between Arrival and Departure. Each gateway node represents the entry point to a real road. Edges from the Arrival node to these gateways carry the arrival cost for that city, edges from gateways to the corresponding city nodes carry the road tolls, and edges from city nodes to the Departure node carry the departure cost.

This construction guarantees that every valid path must: 1. start at the Arrival node (paying exactly one arrival cost), 2. pass through at least one gateway edge (paying at least one road toll), and 3. end at the Departure node (paying exactly one departure cost).

Therefore every path in the graph exactly matches a legal road trip: it always factors in the arrival cost + at least one road toll + the departure cost, so Dijkstra's algorithm finds the true minimal-cost trip that satisfies the "one road taken" constraint.

$$\mathsf{Arrive}(\text{Vienna}) + \mathsf{Toll}(\text{Vienna}, \text{Graz}) + \mathsf{Toll}(\text{Graz}, \text{Salzburg}) + \mathsf{Depart}(\text{Salzburg}) = 2 + 1 + 2 + 1 = 6.$$

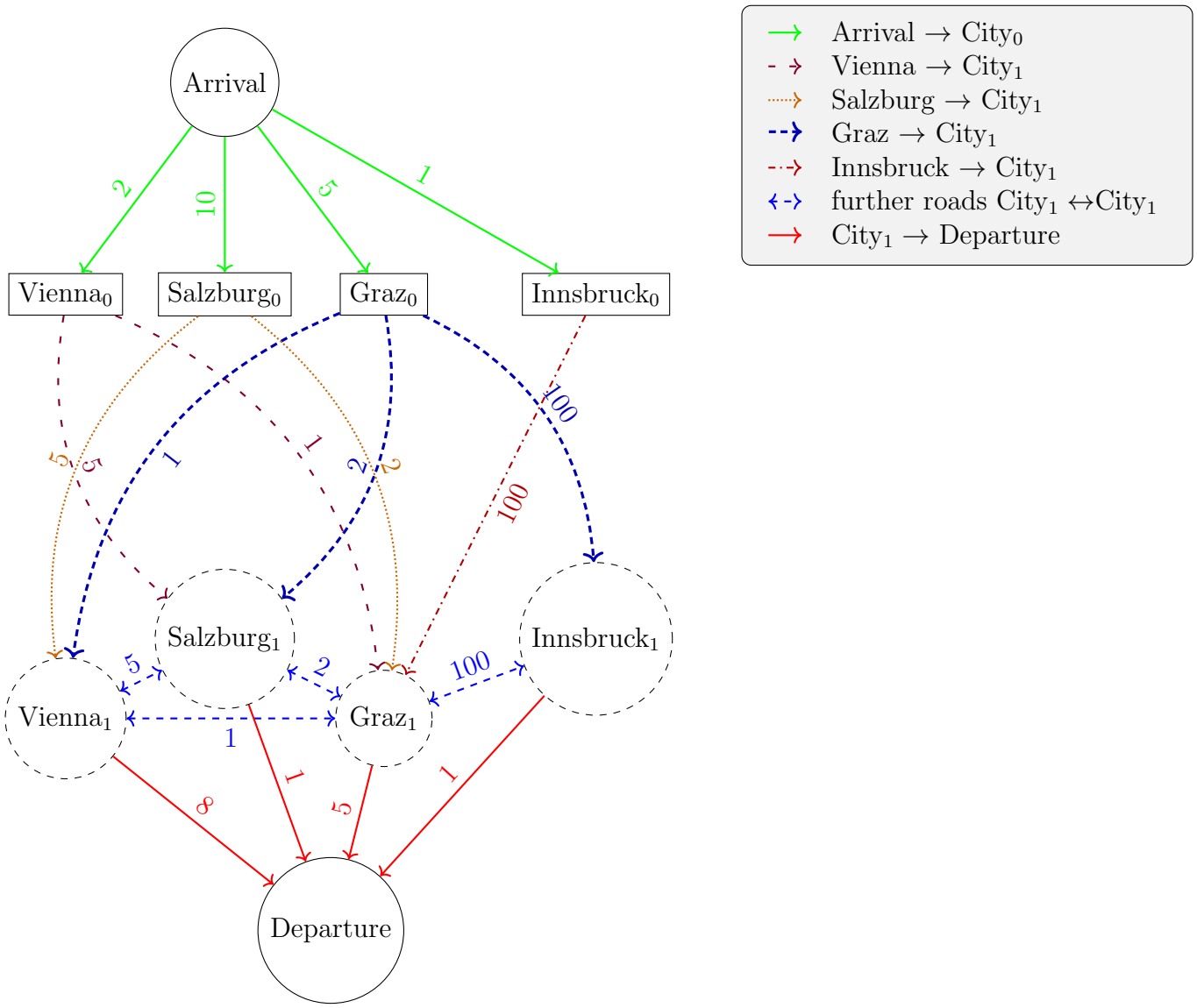

Figure 4: Layered graph with new "before road" (_0) and "after road" (_1) vertices.

Time Complexity

The time complexity of adding this new layer of vertices is

Vertices:	
Vertice vertices:	$2n$ (one City_0 and one City_1 for each original city)
Dep and Arr:	2
Total vertices:	$2n + 2 = O(n)$
Edges:	
Edges for Arrival/Departure:	$2n$ (Arrival $\rightarrow$ City_0, City_1 $\rightarrow$ Departure)
Edges per original road:	$4m$ (bidirectional original edge + $\text{City}_0 \rightarrow \text{City}_1$ edge endpoints)
Total edges:	$2n + 4m = O(m + n)$

Final Time Complexity

Total vertices:	$2n + 2 = O(n)$
Total edges:	$2n + 4m = O(m + n)$
Shortest path using Dijkstra:	$O((m + n) \log n)$
Simplification for connected graph:	$m \geq n - 1 \implies O((m + n) \log n) = O(m \log n)$

As we know that the amount of edges will always be at least double than the amount of vertices (worst case, every vertice requires at worst an edge to Arr and Dep), we can conclude that the total time is:

$$\boxed{O(m \log n)}$$