1 Given a directed graph $G = (V, E)$ with non-negative edge lengths $\ell(e), e \in E$ and a node $s \in V$, describe an algorithm to find the length of a shortest cycle containing the node s.

Solution:

The algorithm. Run Dijkstra's algorithm to find the shortest distances $d(s, v)$ from s to all the vertices $v \in V$. The length of the shortest cycle containing s is

$$\alpha = \min_{v \in V} \bigl(d(s, v) + \ell(v, s)\bigr).$$

To this end, let v^* be the vertex realizing this minimum, let π be the shortest path from s to v^*. The desired cycle is $\pi + (v^*, s)$.

Correctness. The algorithm outputs a cycle π that goes through s, and its length is α. A shortest cycle containing s must be composed of some path from s to a vertex v, followed by an edge (v, s). Namely, the length of the shortest desired cycle is at least α (since it is one of the solutions considered by the min). As such, the computed cycle is a shortest cycle through s.

Running time analysis. Running Dijkstra takes $O(n \log n + m)$ time. Computing α, and thus v^* can be done in $O(n + m)$ time (this can be done in $O(n)$ time if the graph presentation lists all the incoming edges into a vertex). Extracting π from the shortest path tree can be done in $O(n)$ time. As such, the overall running time is $O(n \log n + m)$.

2 Suppose we have a collection of cities and different airlines offer flights between various pairs of cities. Some airlines only fly between some pairs of cities. Some pairs of cities are served by many airlines. Each airline charges perhaps different amounts for their one-way tickets.

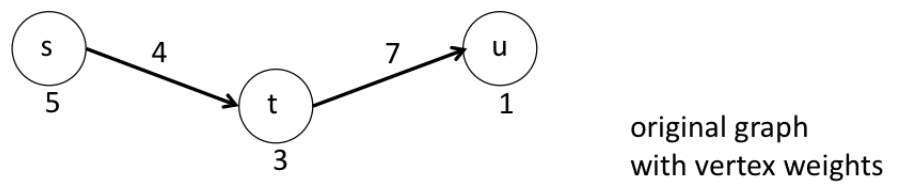
- Suppose you'd like to get from City A to City B at the least total cost. Describe an efficient solution. (Your solution may change planes to a different airline as needed.)
- It turns out that airports charge usage taxes. Different airports may charge different amounts in tax. Your cost of traveling from A to B now includes all of the flight costs, plus all of the taxes of the airports that you stopover along the way from A to B. Model this as a graph problem and give an efficient solution to find the least cost way to get from A to B.

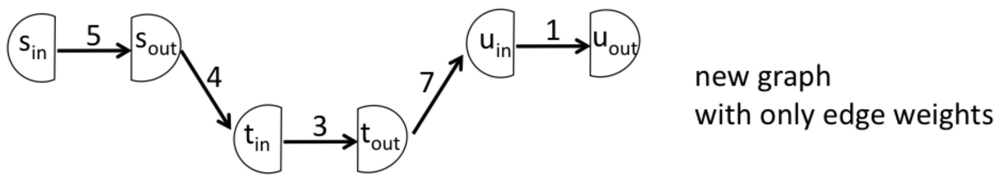
Solution:

Part (A) is immediately seen to be an application of shortest path, where the graph $G = (V, E)$ is given by $V =$ the set of airports, and $E = \{(u, v) \mid$ some airline flies directly from city u to city v. But what is the length $\ell(u, v)$ of edge (u, v)? We could include multi edges in which for each airline flying directly from u to v we include an edge with length the cost of the airfare, or simpler, we use a single edge with length $\ell(u, v)$ defined to be the minimum fare offered by any airline that flies directly from u to v.

Finding a shortest path from city A to city B in G thus gives us the minimum airfare from A to B in the stated problem.

Part (B) is a bit trickier because of the airport taxes. We employ a trick typically used when a graph has costs associated with vertices as well as costs on edges: we split each vertex into two pieces, and insert an edge between them with weight equal to the cost of the vertex. Each vertex will have an "in" version, and an "out" version, with all incoming edges directed to the "in" version, and all outgoing edges emanating from the "out" version. Refer to the picture for an example.





The solution is then to create the graph as in part (a), but then split nodes as described and insert edges with lengths corresponding to the airport taxes. The minimum cost path from A_{in} to B_{out} will give the least cost of traveling from A to B, paying all airport taxes, including those at airports A and B. (What if taxes at airports A and B were not required to be paid?)

Formally, the graph we create is $G = (V, E)$, where V and E and the edge weights are defined as follows:

- $V = V_{in} \cup V_{out}$, where $V_{in} = \{v_{in} \mid v \text{ is a city}\}$, and $V_{out} = \{v_{out} \mid v \text{ is a city}\}$.
- $E = E_1 \cup E_2$, where $E_1 = \{(u_{out}, v_{in}) \mid \text{there is an airline that flies from city } u \text{ to city } v\}$, and $E_2 = \{(v_{in}, v_{out}) \mid v \text{ is a city}\}$.
- If $e = (u_{out}, v_{in}) \in E_1$, then $\ell(e) =$ the least cost airfare for flying directly from city u to city v.
- If $e = (v_{in}, v_{out}) \in E_2$, then $\ell(e) =$ the airport tax for the airport in city v.

The time to create the graph G is linear in the number of cities and number of flights specified, and the running time of the algorithm is thus dominated by running the shortest path algorithm which takes time $O(m + n \log n)$, where m is the number of flights, and n the number of cities.

3 Describe and analyze an algorithm to compute the shortest path from vertex s to vertex t in a directed graph with weighted edges, where exactly *one* edge $u \to v$ has negative weight. First check whether G has a negative length cycle. Then, find the shortest path length from s to t. (**Hint:** Modify the input graph and run Dijkstra's algorithm.)

Solution:

Let G denote the input graph, let $w(x \to y)$ denote the weight of edge $x \to y$, and let $u \to v$ denote the unique edge in G with negative weight.

Remove edge $u \to v$ from G and let G' denote the resulting graph. Note that G' has no negative length edges. For any nodes x and y, let $dist(x, y)$ and $dist'(x, y)$ denote the distances from x to y in G and G', respectively.

If G has a negative length cycle then it must contain the edge $u \to v$: the shortest length cycle containing this arc can be seen to consist of a shortest path P from y to x in G' together with the arc $x \to y$. The length of this cycle is $dist'(v, u) + w(u \to v)$. G has a negative length cycle iff this quantity is negative. Thus, we can check if G has a negative length cycle by computing $\mathit{dist}'(v,u)$ in G' via Dijkstra's algorithm.

Suppose G does not have a negative length cycle. The shortest path in G from s to t either traverses the edge $u \to v$ or it does not; we consider each case separately. Then we have

$$\mathit{dist}(s,t) = \min \begin{Bmatrix} \mathit{dist}'(s,t) \\ \mathit{dist}'(s,u) + w(u \to v) + \mathit{dist}'(v,t) \end{Bmatrix}$$

Thus, we can compute $\mathit{dist}(s,t)$ by running Dijkstra *twice* in G': once starting at s to compute both $\mathit{dist}'(s,t)$ and $\mathit{dist}'(s,u)$, and once starting from v to compute $\mathit{dist}'(v,t)$. The algorithm runs in $\boldsymbol{O(V \log V + E)}$ ***time***.