Algorithms
Shortest Paths and MST
Undergraduate 2nd year S4 EPITA

Correction Exam B8
May, the 14th 2025

Solution 1 **(Longest-shortest – *9 points*)**

1. 3 shortest paths (all with minimum cost = 11) between vertices 0 and 7 in the graph `G2`:

   - Path 1 = $0 \to 3 \to 5 \to 7$
   - Path 2 = $0 \to 1 \to 6 \to 4 \to 7$
   - Path 3 (longest-shortest) = $0 \to 3 \to 5 \to 1 \to 6 \to 4 \to 7$

2. **Specifications:** The function `longest_shortest(G, src, dst)` returns a *longest-shortest path* between the two distinct vertices `src` and `dst` in the directed graph `G` with strictly positive costs, its cost and its length. If the path between `src` and `dst` does not exist in `G`, the function returns `None`.

   Graphe à coûts strictement positifs ⇒ Dijkstra.

```python
def __build_path(P, dst):
    # build the path to dst using P: predecessor/parent vector
    path = []
    while dst != -1:
        path.append(dst)
        dst = P[dst]
    path.reverse()
    return path

def longest_shortest(G, src, dst):
    dist = [infty] * G.order
    dist[src] = 0
    P = [None] * G.order
    P[src] = -1
    length = [0] * G.order
    H = heap.Heap(G.order)
    H.push(src, 0)
    while not H.is_empty():
        (_, x) = H.pop()
        if x == dst:
            return (__build_path(P, dst), dist[dst], length[dst])
        for y in G.adjlists[x]:
            c = dist[x] + G.costs[(x, y)]
            if dist[y] == c:
                if length[x] + 1 > length[y]:
                    length[y] = length[x] + 1
                    P[y] = x
            elif c < dist[y]:
                dist[y] = c
                P[y] = x
                length[y] = length[x] + 1
                H.update(y, c)
    return None
```

***Solution 2* (What is this? – *3 points*)**

When calling `mystery(`G_3`, 0, 7)` with G_3 the graph in figure 3:

1. List `L` built by the call `aux(G, src)`, line 30:

   `[7, 4, 6, 1, 5, 3, 2, 0]`

2. Vectors `dist1` and `P1` results of the first call to `shortestpath` line 32:

	0	1	2	3	4	5	6	7	8	9	10
dist1	0	2	8	1	7	3	3	8	∞	∞	∞
P1	-1	3	0	0	6	3	1	6	None	None	None

3. Vectors `dist2` and `P2` results of the second call to `shortestpath` line 36:

	0	1	2	3	4	5	6	7	8	9	10
dist2	0	4	∞	∞	10	∞	∞	12	∞	∞	∞
P2	-1	0	None	None	1	None	None	4	None	None	None

4. Result returned by `mystery(`G_3`, 0, 7)`: `(8, 12)`

***Solution 3* (MST or not? – *3 points*)**

The presented algorithm is similar to Prim's algorithm, but **the partial graph T is not a MST of G!**

Justifications

- The algorithm has to continue from the extremity of the last added edge:
  - if a cut-edge is used, there is no going back: the algorithm cannot finish, the solution will not contain $n-1$ edges ; (Pb 1)
  - a high cost edge can be chosen. (Pb 2)
- The algorithm does not use two complementary sets of vertices: the set of vertices already connected to the source s and the set of vertices that are not yet connected to s.
  The problem in this case is not being able to detect cycles. (Pb 3)

Example

In the graph bellow, if the algorithm starts from the vertex 3:

- After the first selected edge (3-4), the process must be repeated from 4: there is no turning back, and vertices 0, 1 and 2 can never be reached (Pb 1).
- The algorithm continues to vertices 5 and 7. After edge 7-6, edge 6-4 is chosen, even though it is not part of the solution (Pb 2) and creates a cycle (Pb 3).

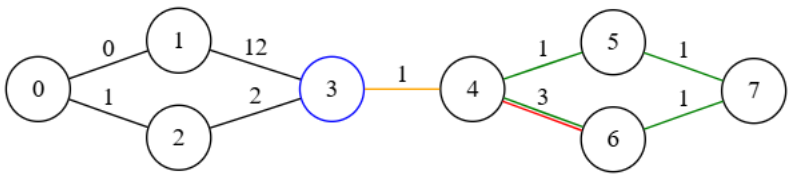

Figure 1: Start from 3

***Solution 4* (MST via Kruskal – *2 points*)**

1. The chosen edges are :

   `{B,F,1},{B,G,2},{B,C,4},{G,H,5},{A,E,6},{A,D,7},{A,B,10}`

2. The rejected edges are :

   `{F,G,3},{C,F,8},{D,E,9}`