Algorithmics
Correction Final Exam #4 (P4)

Undergraduate 2^{nd} year S4 – Epita

15 May 2018 - 10 : 00

Solution 1 **(Exhausting deposit... –** ***3 points*****)**

1. A possibility can ben for instance, the graph on the figure 1.

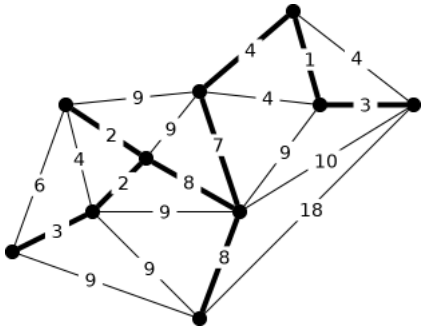


Figure 1: Minimum Spanning Tree of the graphe on figure 2.

2. No, this solution is not the only one.
   The cost of the edges are not all distinct, the MST obtained is not the only one.

***Solution 2* (Asterix and the Soothsayer – *13 points*)**

1. **The algorithm:**

   (a) *What is the name of this algorithm?* A* ("Astar")

   (b) *Open vertices:* A heap.
   closed: a boolean vector.

   (c) *Algorithm complexity:* With the heap $O(p \log(n))$ (as Dijkstra).

   (d)

```python
def __buildPath(src, dst, p):
    path = [dst]
    while dst != src:
        dst = p[dst]
        path.insert(0, dst)
    return path

def asterix(G, src, dst):
    dist = [inf] * G.order
    p = [-1] * G.order
    closed = [False] * G.order
    dist[src] = 0
    openHeap = heap.Heap(G.order)
    x = src
    while x != dst:
        closed[x] = True
        for y in G.adjlists[x]:
            if not closed[y] and dist[x] + G.costs[(x, y)] < dist[y]:
                dist[y] = dist[x] + G.costs[(x, y)]
                p[y] = x
                openHeap.update(y, dist[y] + heuristix(G, y, dst))
        if openHeap.isEmpty():
            raise Exception("dst not reachable")
        (_, x) = openHeap.pop()
    return __buildPath(src, dst, p)

def astar2(G, src, dst, heur):
    dist = [inf] * G.order
    p = [-1] * G.order
    closed = [False] * G.order
    dist[src] = 0
    openHeap = heap.Heap(G.order)
    openHeap.push(src, 0)
    while not openHeap.isEmpty():
        (_, x) = openHeap.pop()
        if x == dst:
            return (dist, p)
        closed[x] = True
        for y in G.adjlists[x]:
            if not closed[y] and dist[x] + G.costs[(x, y)] < dist[y]:
                dist[y] = dist[x] + G.costs[(x, y)]
                p[y] = x
                openHeap.update(y, dist[y] + heuristix(G, y, dst))
    if dist[dst] == inf:
        raise Exception("dst not reachable")
    return __buildPath(src, dst, p)
```

2. **Deviners:**

(a) ⋆ **Heuristix the Dutchman** (*HeuristixD*)
Processed vertices (in order): 1 3 6 7 2 8 5 9 4 10

Result:

	1	2	3	4	5	6	7	8	9	10
dist	0	4	1	6	5	2	3	4	5	6
pere	-1	1	1	2	2	3	6	7	8	5

⋆ **Heuristix of the New World** (*HeuristixM*)
Processed vertices (in order): 1 3 2 5 10

Result:

	1	2	3	4	5	6	7	8	9	10
dist	0	4	1	6	5	2	∞	∞	∞	6
pere	-1	1	1	2	2	3	-1	-1	-1	5

(b) The solution with HeuristixD is the Dijkstra algorithm.

(c) *The estimation of HeuristixM is better than HeuristixB's:*

- It gives the shortest pass ;
- using a minimum of vertices.

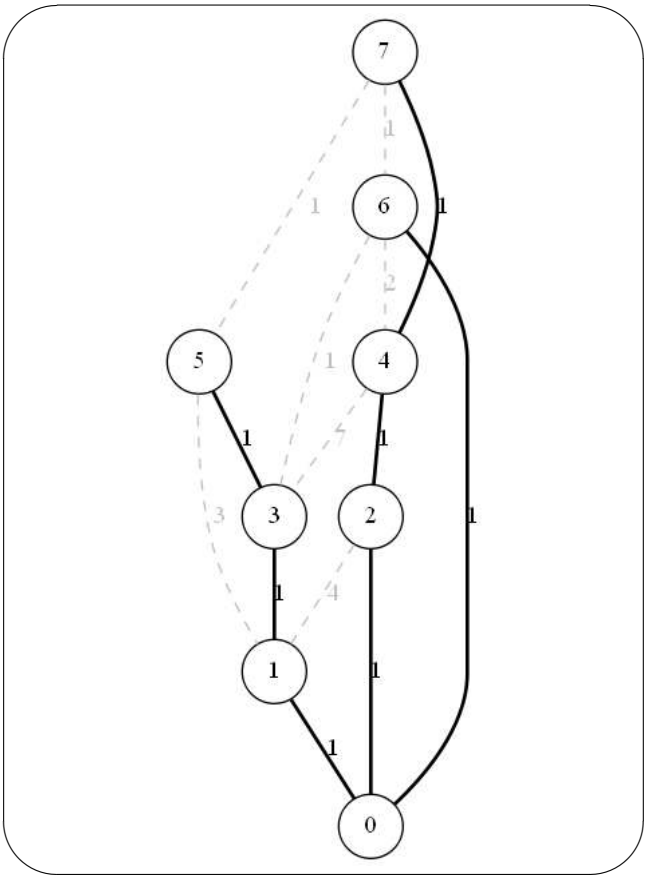
***Solution 3* (What is this? – *4 points*)**

1. *What does the function* `dfs(G, x, y)` *test?*

   It tests whether a path between x and y exists in G, without the edge $x - -y$.

2. The function `what`

   (a) The graph:



   (b) *What property has the graph after application of the function?*

   It is a minimum spanning tree of the initial graph.

   (c) How can this function be optimized?

   No need to "empty" the list L, we can stop before:

   - Count the number of edges during the construction of L;
   - stop the second loop as soon as there is $n - 1$ remaining edges.