Algorithmics and complexity

TD 2/7 – Paths and trees

The goal of this tutorial is to practice solving graph-based problems and to better understand the problems seen in the lecture. In particular, the problem of the shortest path and that of the minimum spanning tree.

Exercise 1 : Some scenes from country life

A hamlet of eight houses A, B, C, D, E, F, G and H is connected by a road network whose simplified structure is defined by the following matrix:

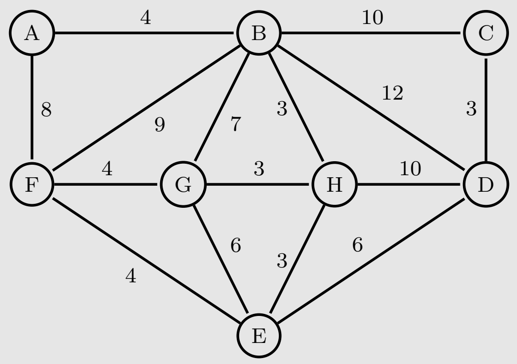
	A	B	C	D	E	F	G	H
A	0	4	∞	∞	∞	8	∞	∞
B	4	0	10	12	∞	9	7	3
C	∞	10	0	3	∞	∞	∞	∞
D	∞	12	3	0	6	∞	∞	10
E	∞	∞	∞	6	0	4	6	3
F	8	9	∞	∞	4	0	4	∞
G	∞	7	∞	∞	6	4	0	3
H	∞	3	∞	10	3	∞	3	0

In this matrix, $M(i, j)$ represents the direct travel time from i to j in minutes. Of course, we have $M(i, i) = 0$. Furthermore, $M(i, j) = \infty$ means that there is no road directly linking i to j.

Question 1

Make a graphical representation with the corresponding undirected graph. A planar representation (without intersection) is possible.

Solution elements :



Question 2

A country doctor living in **G** wants to determine, in urgent cases, the fastest routes linking him to each of the other places in the hamlet. To calculate these itineraries, we will define:

- the nature of this problem,
- the used method,
- the details of each step.

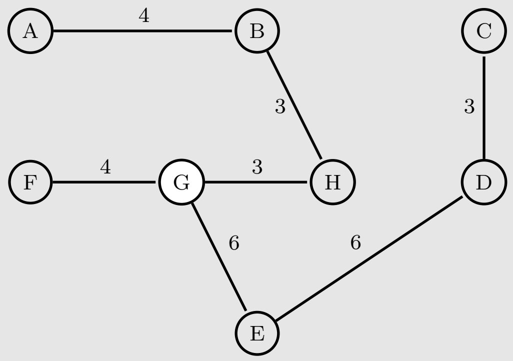
Make a representation of the sub-network (partial graph) corresponding to the obtained routes.

Solution elements :

The frontier corresponds to the shaded boxes and the current visited node is in bold. The first column contains the evolution of the visited vertices.

	A	B	C	D	E	F	H
$V = \{G\}$	∞	7,G	∞	∞	6,G	4,G	**3,G**
$V = \{G, \mathbf{H}\}$	∞	6,H	∞	13,H	6,G	**4,G**	-
$V = \{G, H, \mathbf{F}\}$	12,F	**6,H**	∞	13,H	6,G	-	-
$V = \{G, H, F, \mathbf{B}\}$	10,B	-	16,B	13,H	**6,G**	-	-
$V = \{G, H, F, B, \mathbf{E}\}$	**10,B**	-	16,B	12,E	-	-	-
$V = \{G, H, F, B, E, \mathbf{A}\}$	-	-	16,B	**12,E**	-	-	-
$V = \{G, H, F, B, E, A, \mathbf{D}\}$	-	-	15,D	-	-	-	-
$V = \{G, H, F, B, E, A, D, \mathbf{C}\}$	**10,B**	**6,H**	**15,D**	**12,E**	**6,G**	**4,G**	**3,G**

The tree corresponding to the shortest paths is the following:



Suppose now that the residents want to renew the water supply of the eight houses from a water tower located in **H** by a buried network that must follow some of the existing roads. Each new pipeline has a cost proportional to the length of the corresponding road, therefore to the duration of its travel. For this, we want to build a minimum cost water supply network.

Question 3

What is the structure of this partial graph? How to build such a network?

Solution elements :

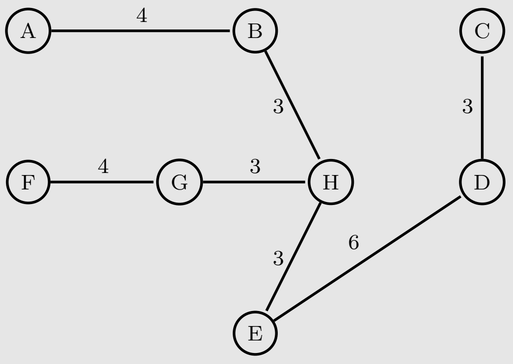
Applying the Kruskal algorithm, the edges are considered in the following order: BH, CD, EH, GH, AB, EF (or FG but not both because it creates a circuit) and finally DE.

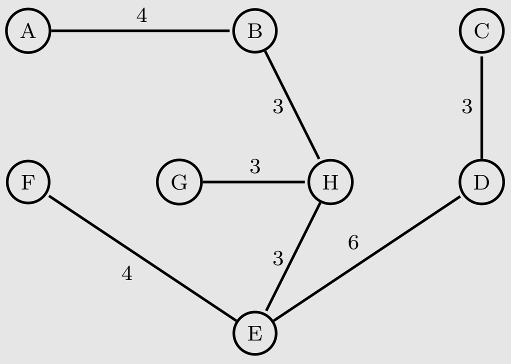
Applying Prim's algorithm, if we start from H, the edges are considered in the following order: BH(dist(B)=3), HG(dist(G)=3), HE(dist(E)=3), AB(dist(A)=4), FG(dist(F)=4) (or EF depending on whether parent F is G or E), ED(dist(D)=6), CD(dist(D)=3).

Question 4

Draw the resulting network. Does the location of the water tower have an impact (structure, cost) on the resulting network?

Solution elements :





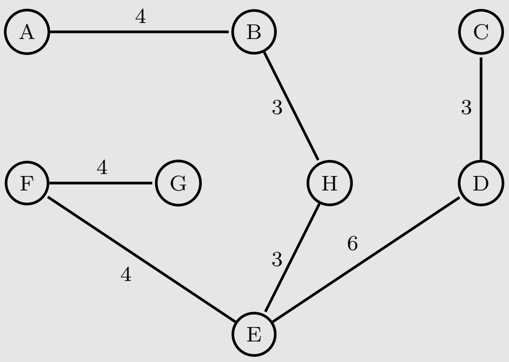
Two minimum spanning trees are possible depending on the used algorithm. Their costs are identical of 26. The obtained trees are independent of the location of the water tower. Note that they are different from the shortest path tree which has a total cost of 29.

Question 5

Bad weather makes the GH linkage unusable. In this case, what are the results of the previous questions (question 2 and question 4)?

Solution elements :

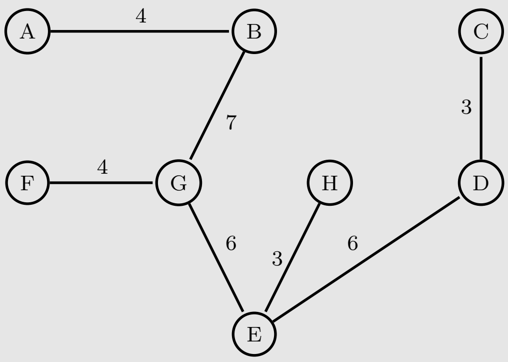
1. For example, if we consider the first MST (on the left) of question 4, it splits into 2 sub-trees containing respectively F, G and A, B, C, D, E, H as vertices. Thus, to obtain the solution, it is enough to choose, among the edges connecting these two sub-trees, i.e. in the set {AF, BF, BG, EF, EG}, an edge of minimal cost, we thus take EF of cost 4, from where the new spanning tree has a weight of 27:



Arbre couvrant minimum

A complementary exercise dedicated to spanning tree updating allows to study this problem in detail.

2. For the shortest paths, it would be necessary to recompute everything **as the edge belongs to the tree of shortest paths**. Be careful, it is not enough to run the algo again on the subgraph containing only the disconnected nodes, even if the shortest paths of the nodes which remain on the right side (with the starting node) of the cut will not see their distance change. We would then obtain:



Plus court chemins

Exercise 2 : DAG

The shortest paths algorithm finds the shortest paths in graphs of any topology. For some families of graphs, it is possible to write more efficient algorithms (in computation time). In this exercise, we will see how to ‘improve” the shortest paths algorithm in the case of *Directed Acyclic Graphs* (DAG).

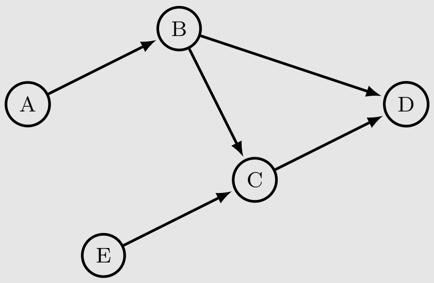
Consider $G = (V, E)$, a weighted DAG.

Question 1

Draw a graph of this type with few vertices.

Solution elements :

Some DAG :

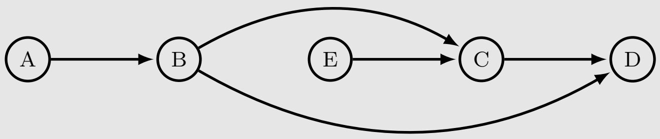


Question 2

Draw **the same graph** by placing the vertices on a line such that all arcs have the same direction. What do you deduce?

Solution elements :

One solution:



It is a **topological order**. A complementary exercise in TD1 fully addresses this problem.

Question 3

By using the property observed in the previous question, please propose a variant of the shortest paths algorithm that calculates the shortest paths in $\mathcal{O}(|V| + |E|)$. Write your algorithm in python or pseudo-code by precisely describing the used data structures.

Solution elements :

The shortest path algorithm is no longer necessary, because the nodes are traversed in topological order:

- one does not need any more to keep the visited nodes!
- one does not need any more to maintain the frontier and to extract the minimum at each step!

The topological order ensures that at the time of computing the shortest distance for a vertex v all the paths leading to it from the starting vertex s pass through the vertices whose shortest distances are already final.

```
def PCCs_topological(graph, s)
    parent = {}
    dist = {v: math.inf for v in nodes(graph)}
    dist[s] = 0

    for v in topological_order(nodes(graph)): # parcours en ordre topologique
        for u in neighbors(graph, v):
            if dist[u] > dist[v] +  distance(graph,v,u):
                dist[u] = dist[v] +  distance(graph,v,u)
                parent[u] = v
```

Exercice 3 : Minimal spanning arborescence (advanced part)

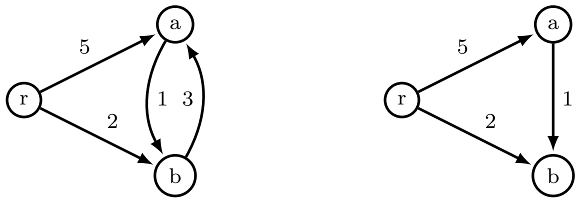
Attention: this exercise is reserved for advanced groups.

Given a directed and weighted graph G with a root r from which all vertices are reachable, a minimal spanning tree is that of G starting from r. So this is the oriented version of a minimal spanning tree. It is also called a minimal spanning arborescence.

Consider $G = (V, E)$ a directed and weighted graph with a root.

Question 1

Can we apply Kruskal's algorithm and Prim's algorithm on G to find a minimal spanning arborescence? If yes, explain why. If not, give a counterexample. You can justify your answer using the following two graphs with the root r.



Élements de correction :

For the graph on the left, we have three spanning arborescences from r.

- $r \to a \to b$, with the weight 6
- $r \to b \to a$, with the weight 5
- $r \to b$ et $r \to a$, with the weight 7

The minimal spanning arborescence for this graph is therefore $r \to b \to a$ with weight 5.

For the graph on the right, we have two spanning arborescences starting from r.

- $r \to a \to b$, with the weight 6
- $r \to b$ et $r \to a$, with the weight 7

The minimal spanning arborescence for this graph is therefore $r \to a \to b$ with weight 6.

We cannot apply Kruskal's algorithm and Prim's algorithm on a directed graph.

- Kruskal's algorithm: with Kruskal, for the graph on the left, $a \to b$ is the first arc to choose, which is not part of the minimal spanning arborescence for this graph.
- Prim's algorithm: with Prim, from r, for the graph on the right, the arc $r \to b$ is first chosen before taking the arc $r \to a$, which is not the minimal spanning arborescence for this graph.

Now we study another algorithm to find a minimal spanning arborescence on a given directed graph.

The main steps are described below:

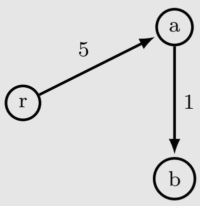
1. delete all arcs in the graph G whose destination is the root r
2. for each vertex v other than the root, take the arc of least weight whose destination is v, denoted (u, v) for which $min_u \omega(u, v)$
3. Consider the set of chosen arcs $S = \{(u, v) \mid v \in V \setminus \{r\} \wedge min_u \omega(u, v)\}$. If S does not contain cycles, then S is a minimal spanning arborescence. Otherwise, for each cycle in S, denoted C, it would have to be treated as follows.
   (a) for each arc $(i, j)(i \in V \setminus C, j \in C)$ (each arc entering the cycle), modify its weight: $\omega'(i, j) = \omega(i, j) - \omega(x(j), j)$, where $\omega(x(j), j)$ is the weight of the arc whose destination is j in C.
   (b) add the arc entering the cycle $(i, j) \in E$ whose modified weight $\omega'(i, j)$ is the lowest among all the arcs entering the cycle
   (c) remove the arc in C whose destination is j

Question 2

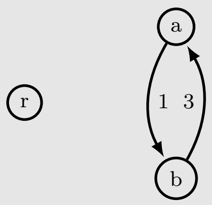
Apply this new algorithm to the two graphs above and give the result for each step.

Élements de correction :

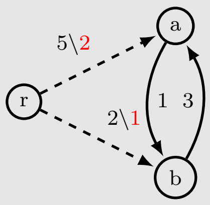
- For the graph on the right, we give the result for each step.
  1. as there is no arc whose destination is r, then we do nothing.
  2. concerning each vertex other than r: for a, we take the arc $r \to a$ because there is only one arc whose destination is a; for b, we take the weakest arc whose destination is b, therefore $a \to b$.



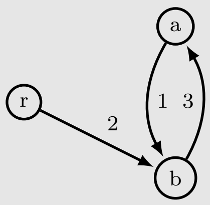
  3. as now, the set of chosen arcs ($r \to a$ and $a \to b$) do not contain a cycle, so it is indeed the minimal spanning arborescence (see the answer for the precedent question).
- For the graph on the left:
  1. There is no arc whose destination is r, so we do nothing.
  2. concerning each vertex other than r: for a, we take the arc $b \to a$ because its weight is the lowest among those whose destination is a; for b, we take the weakest arc among those whose destination is b, therefore $a \to b$. So we get



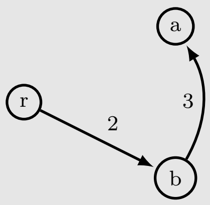
  3. the set of chosen arcs ($a \to b$ and $b \to a$) indeed contain a cycle $C = a \to b \to a$, so we continue to treat this cycle as follows.
     (a) in C, $min_k(\omega(x(k), k))$ is indeed 1, so apply the formula above, we have $\omega'(r, a) = \omega(r, a) - \omega(b, a) = 2$ and $\omega'(r, b) = \omega(r, b) - \omega(a, b) = 1$ Now we have the following graph, where the red weight is the modified weight.



     (b) the arc entering this cycle whose modified weight is the lowest is therefore $r \to b$. We add this original arc



     (c) the arc in C whose destination is b is indeed $a \to b$, thus it should be deleted.



     There is no longer any cycle in the newly constructed graph, thus it is indeed the minimal spanning arborescence (see the answer for the previous question).

Question 3

Since now you understand each step of the algorithm, implement it in python code before testing with the following code. your main function is $acm(root, graph)$ with graph represented as adjacency list (see following code).

```
graphe={"r":{"a":1,"b":10},
            "a":{"b":8,"e":2},
            "b":{"d":6},
            "c":{"b":4,"a":7},
            "d":{"c":3},
            "e":{"c":9,"d":11}
    }
h = acm("r",graphe)
assert h == {'r': {'a': 1}, 'c': {}, 'a': {'e': 2, 'b': 8}, 'b': {'d': 6}, 'd': {'c': 3}, 'e': {}}
```