LINFO 1121

DATA STRUCTURES AND ALGORITHMS

UCLouvain

TP6: Graphes

Question 6.1.1: STOCKER UN GRAPHE

Soit un graphe avec V noeuds et E edges:

- adj = new bool[V][V] tel que adj[i][j] = true si il existe une edge entre les noeuds i et j
- inc = new bool[V][E] tel que inc[i][j] = true si l'arête j viens/pointe vers le noeud i
- edges = new int[E][2] tel que edges[i][0] contient le premier noeud de l'arête i, [1] le second

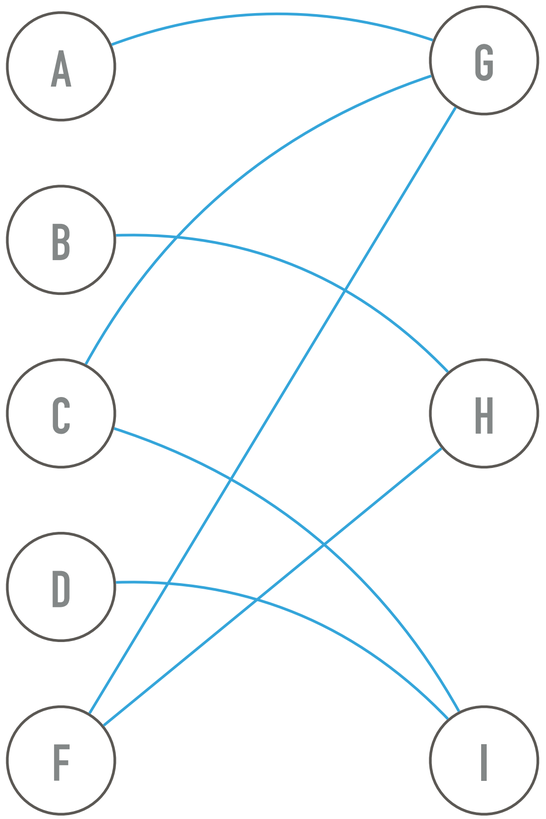
Peut-on faire mieux?

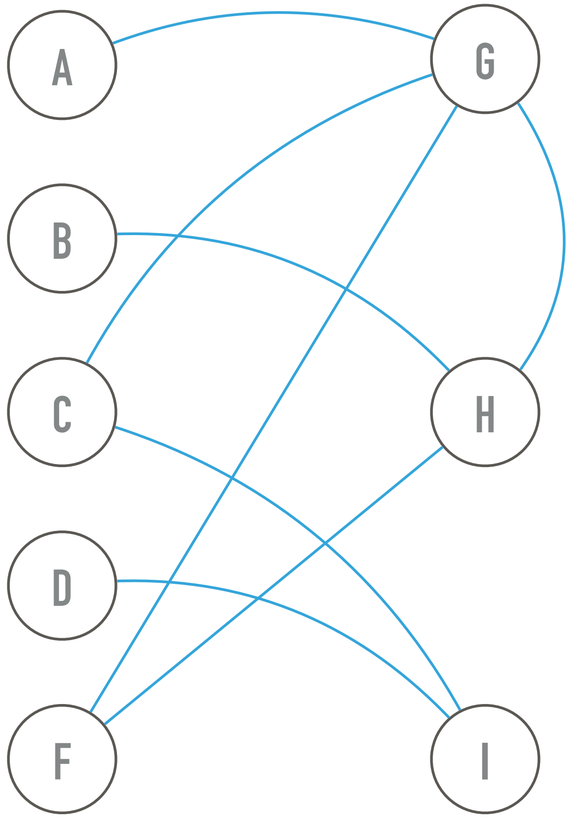
```
LinkedList[] nei = new LinkedList[V];
for(int i = 0; i < V; i++) {
    nei[i] = new LinkedList<Integer>();
    //ajouter les voisins de i
}
```

Question 6.1.1: STOCKER UN GRAPHE avec des poids

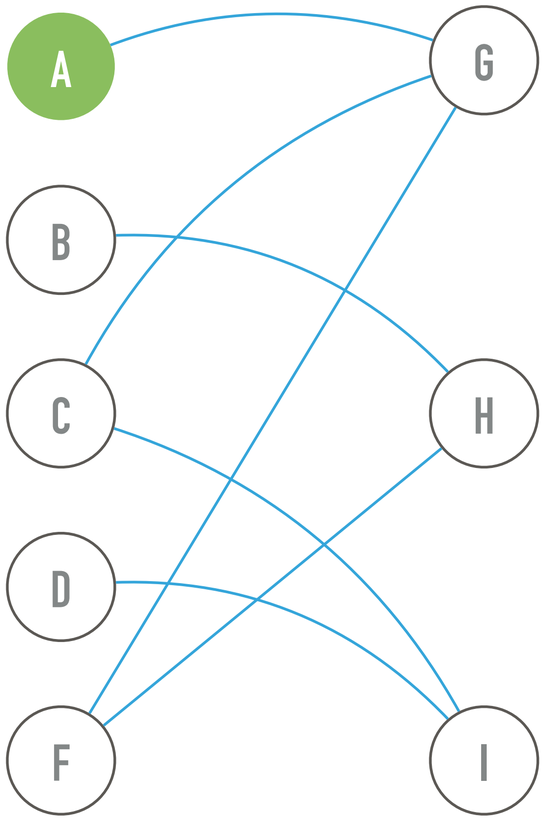
```
HashMap[] nei = new HashMap[V];
for(int i = 0; i < V; i++) {
    nei[i] = new HashMap<Integer, Integer>();
    //ajouter les voisins de i
}
```

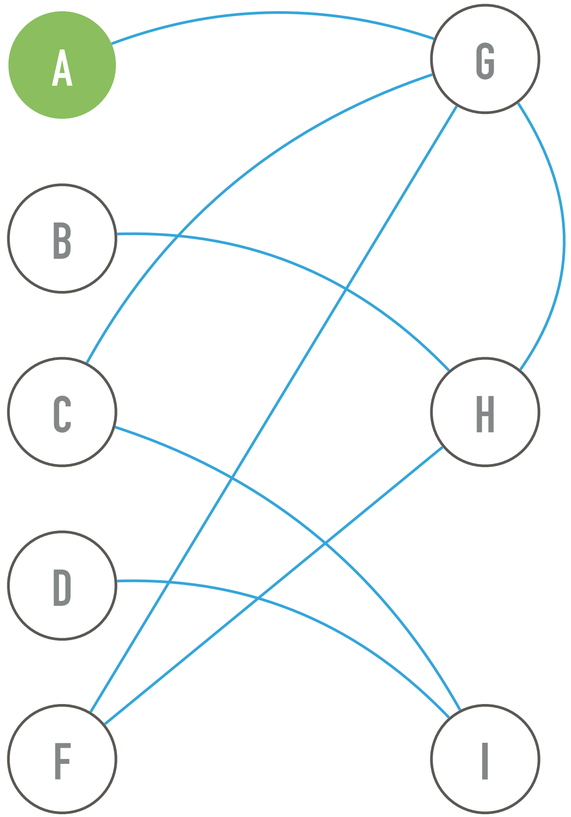
Question 6.1.2: Détecter si un graphe est biparti



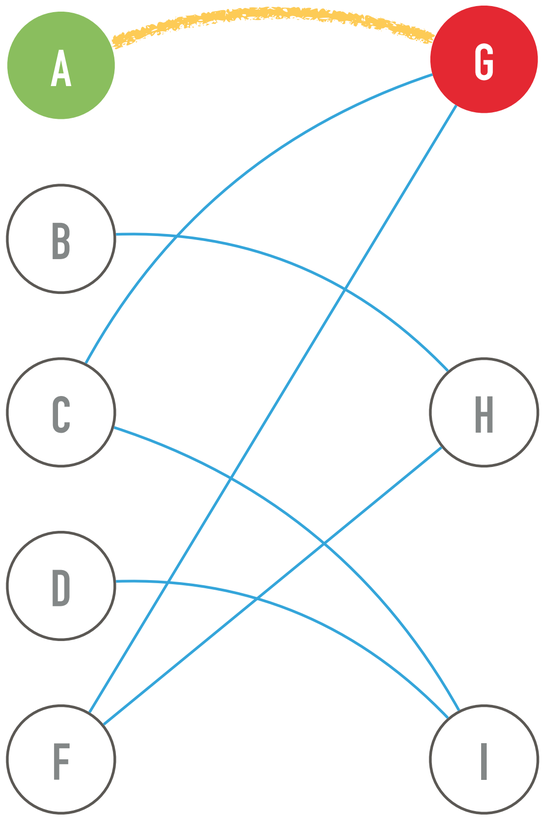


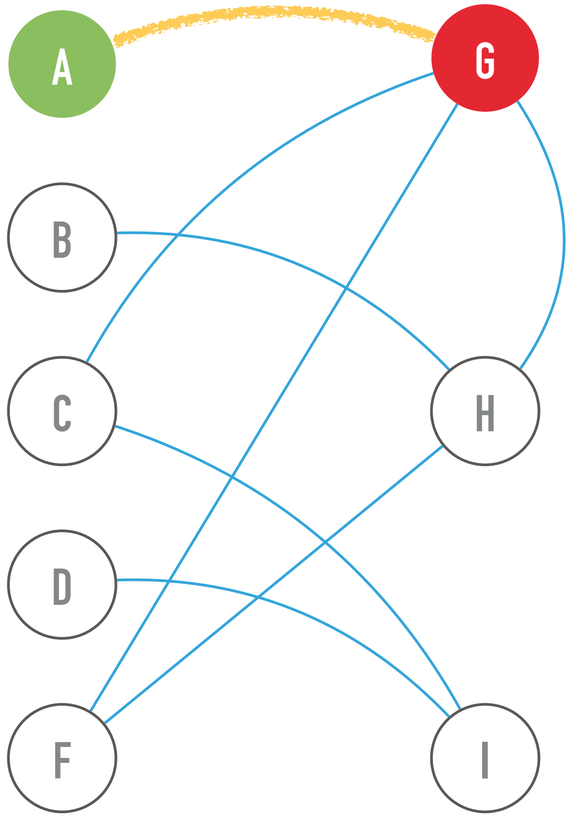
Question 6.1.2: Détecter si un graphe est biparti



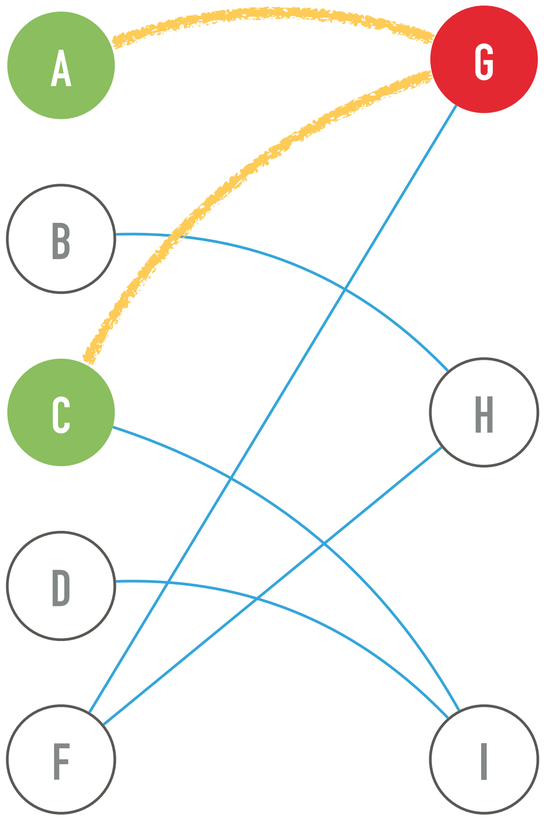


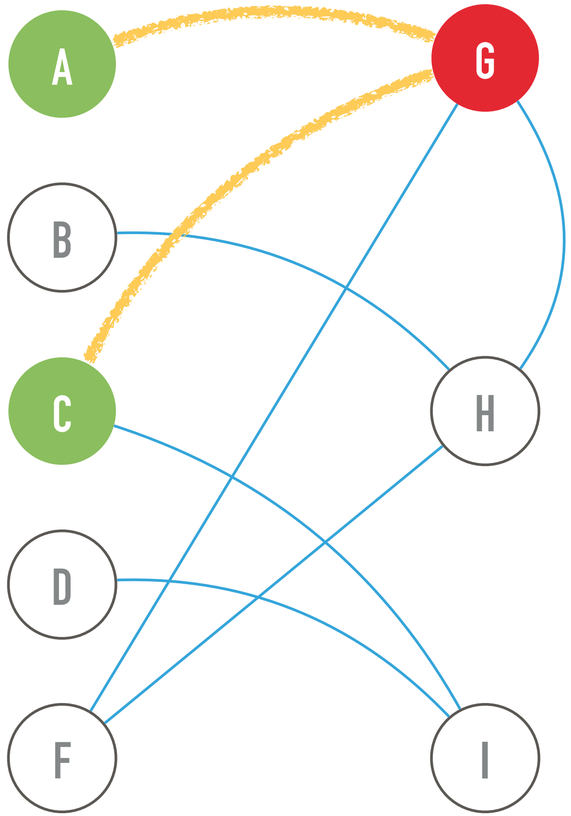
Question 6.1.2: Détecter si un graphe est biparti



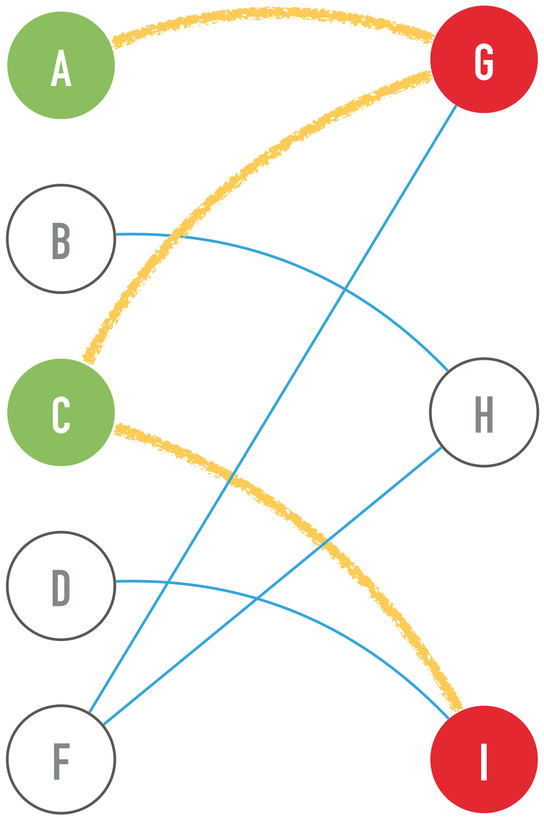


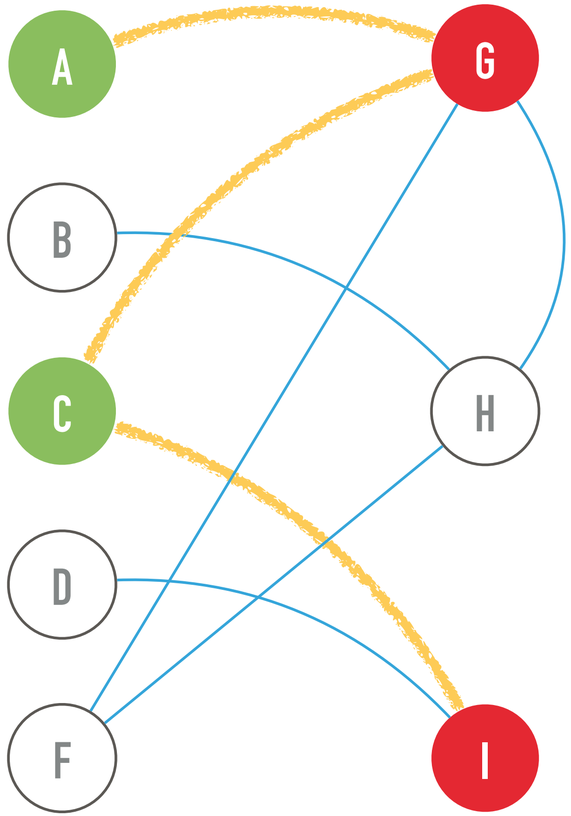
Question 6.1.2: Détecter si un graphe est biparti



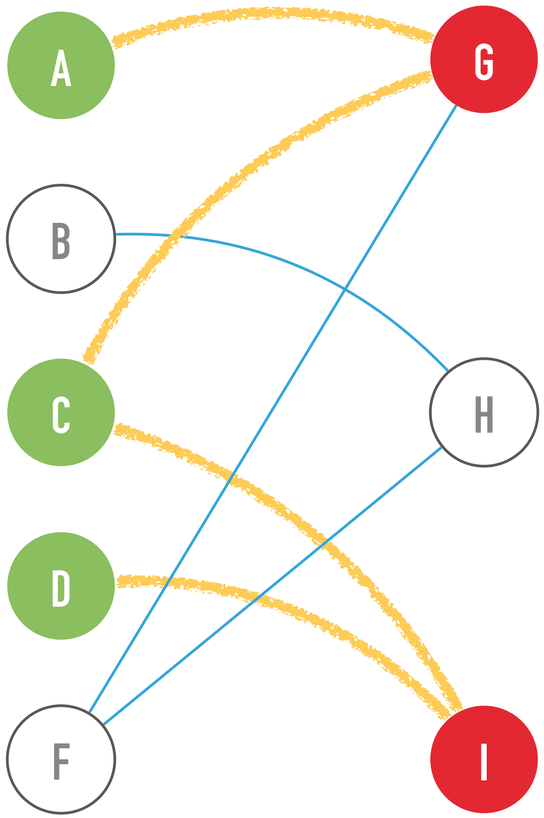


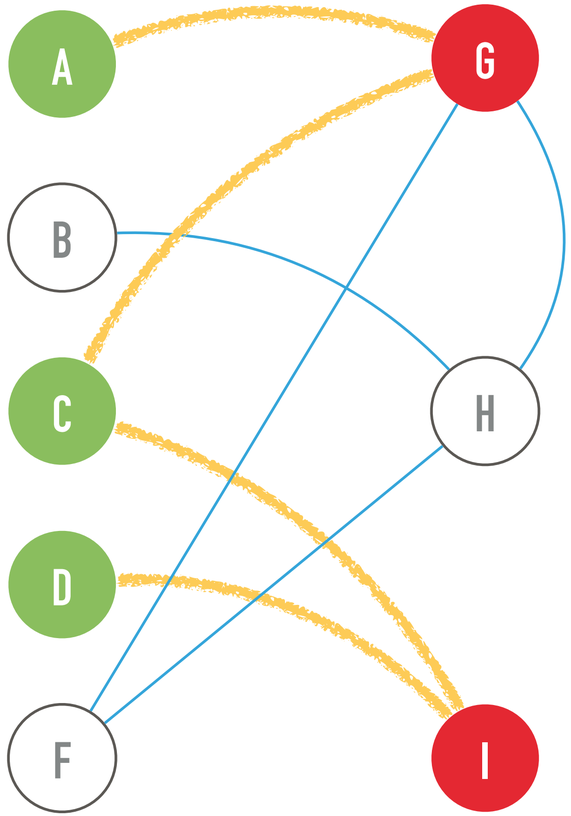
Question 6.1.2: Détecter si un graphe est biparti



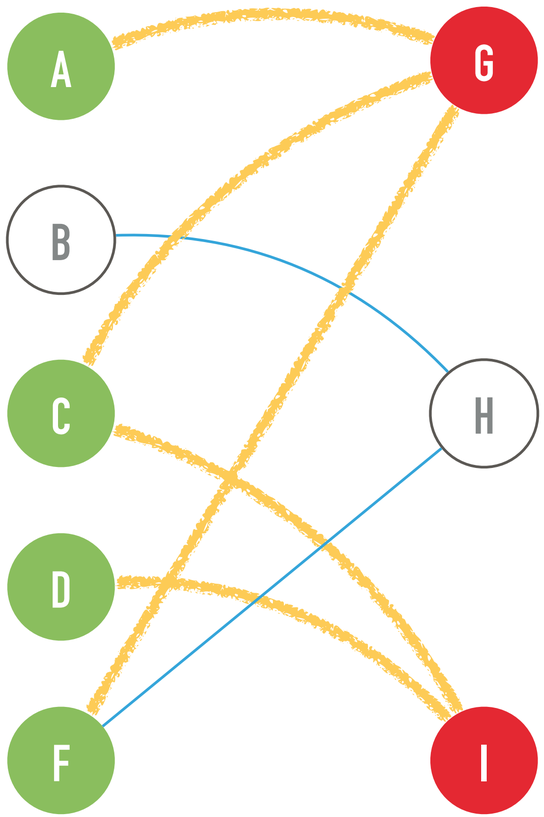


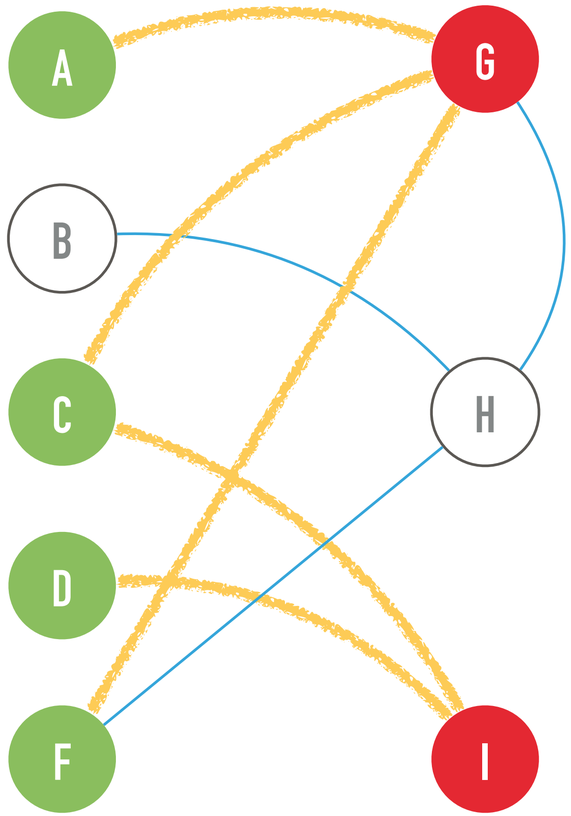
Question 6.1.2: Détecter si un graphe est biparti



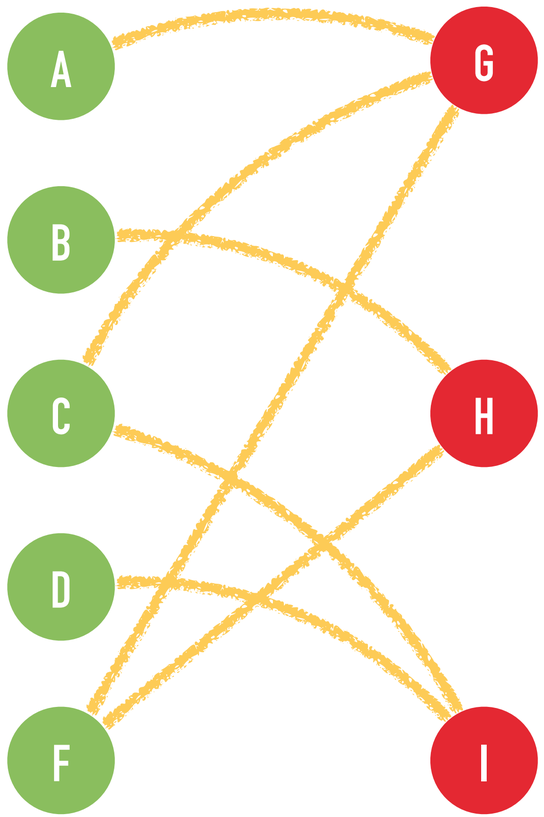


Question 6.1.2: Détecter si un graphe est biparti

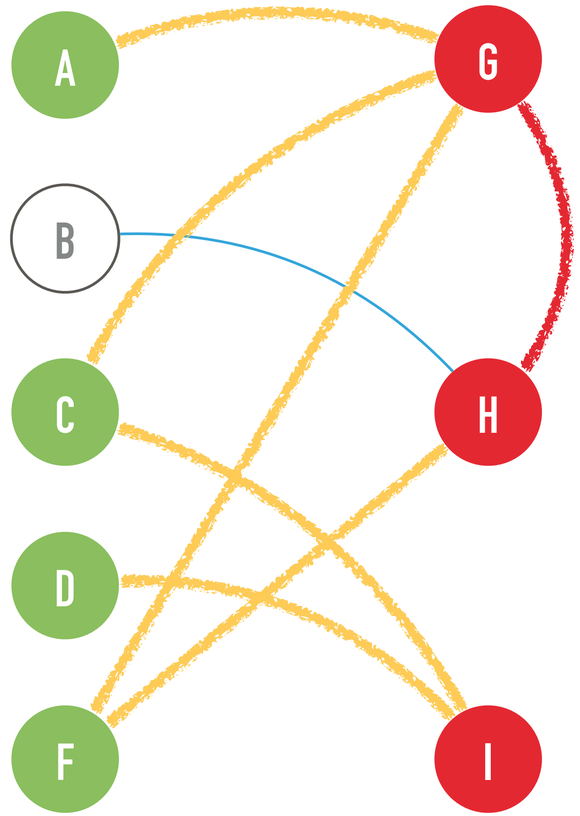




Question 6.1.2: Détecter si un graphe est biparti



OK!



Erreur!

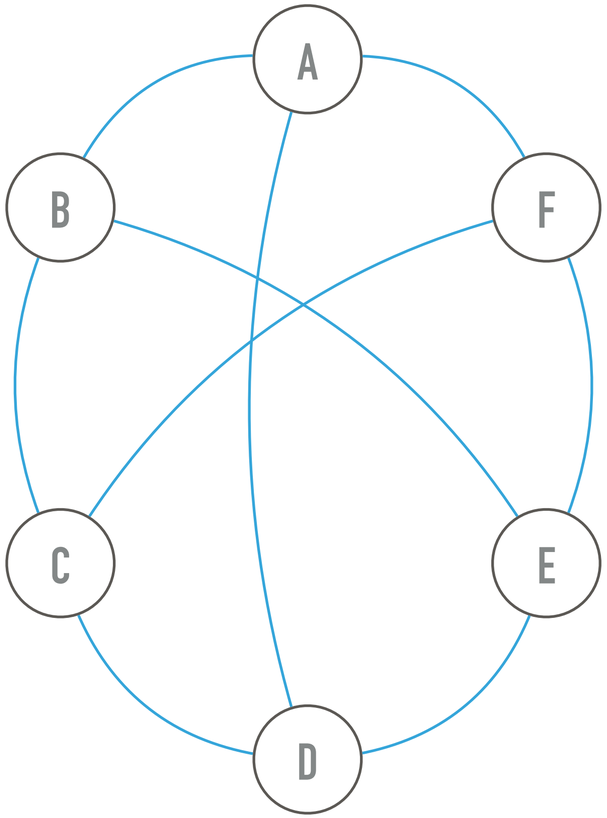
Question 6.1.3: Retirer un noeud et rester connexe

On peux le prouver en construisant une solution. Deux remarques:

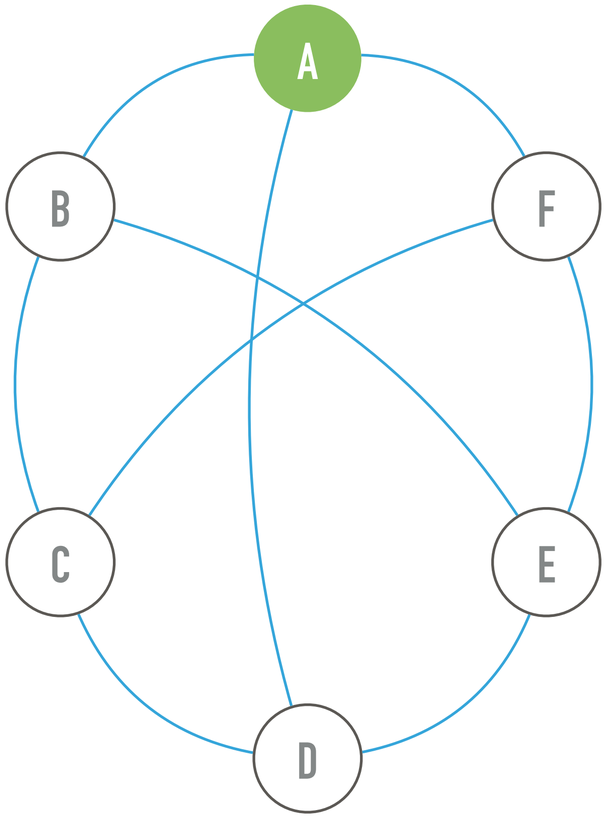
- Le graphe est originellement connexe, donc on peux atteindre n'importe quel noeud depuis n'importe quel noeud.
- Un DFS « remonte » que quand il n'il y a plus de nouveaux noeuds pas encore visités.

=> Utiliser un DFS. Le premier noeud sans voisins non visités peut être retiré sans briser la connexité.

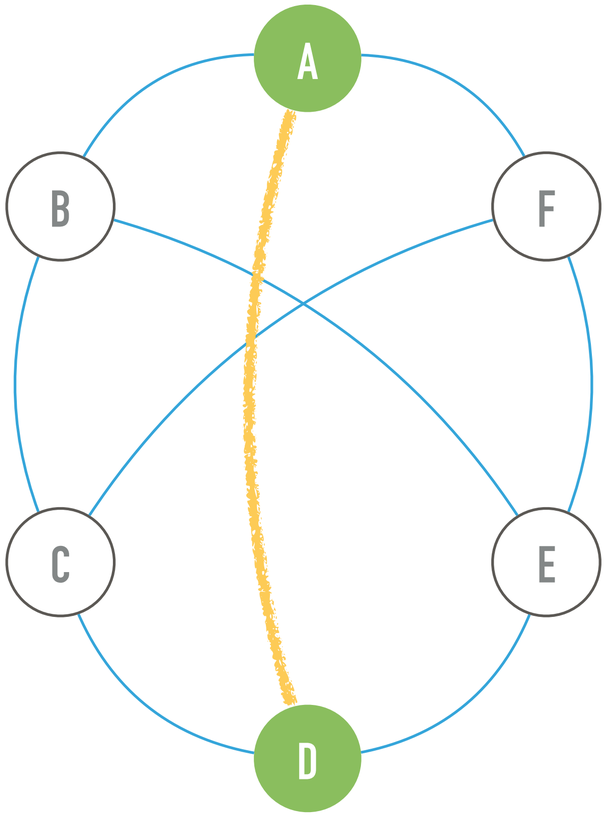
Question 6.1.3: Retirer un noeud et rester connexe



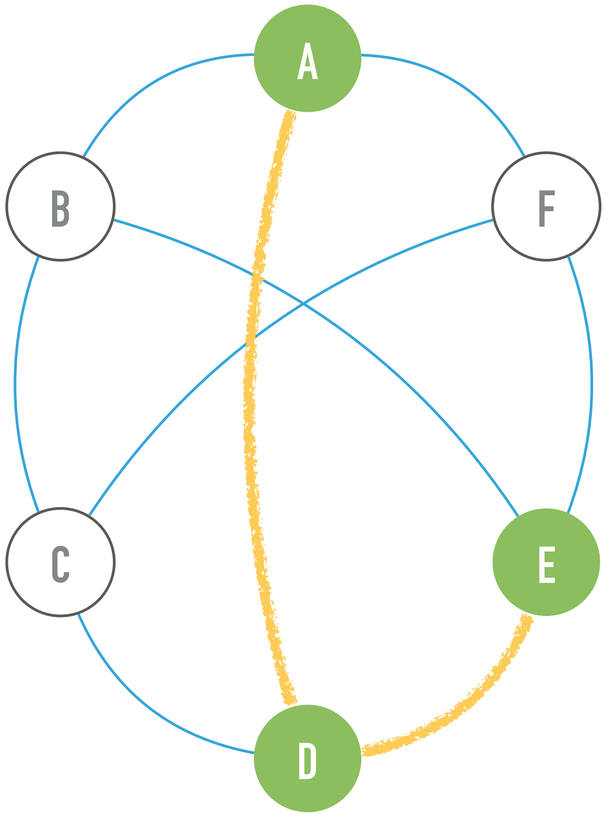
Question 6.1.3: Retirer un noeud et rester connexe



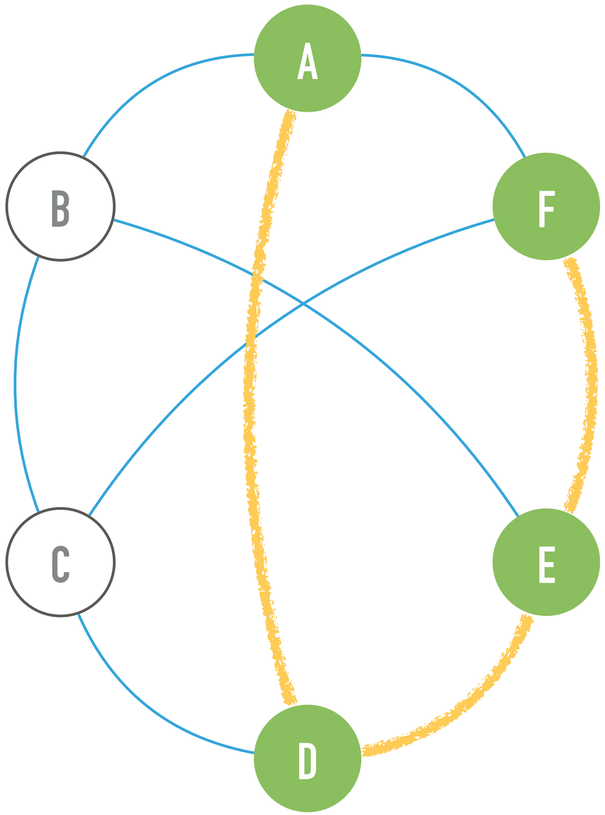
Question 6.1.3: Retirer un noeud et rester connexe



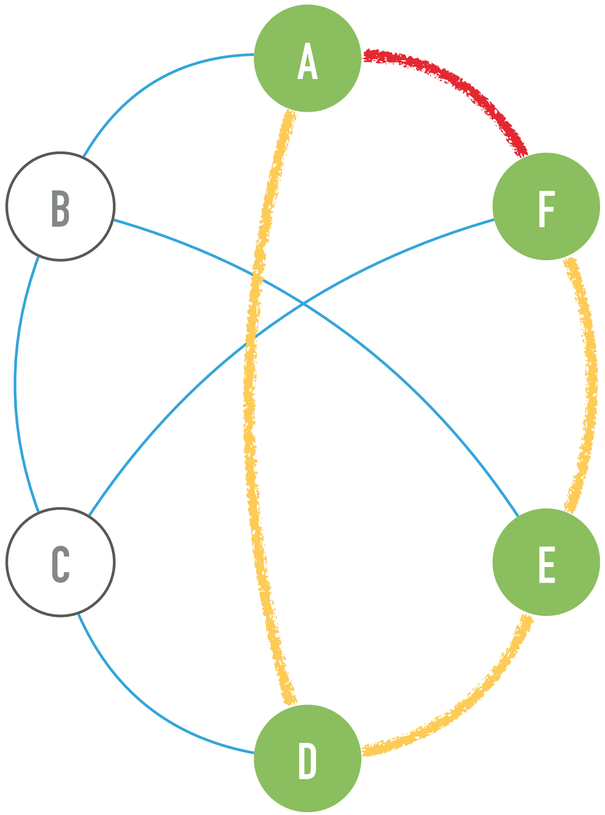
Question 6.1.3: Retirer un noeud et rester connexe



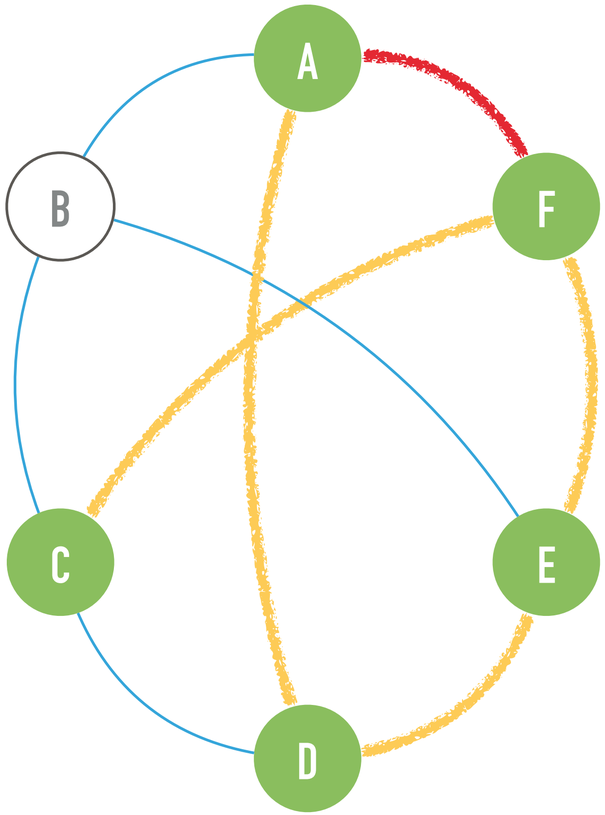
Question 6.1.3: Retirer un noeud et rester connexe



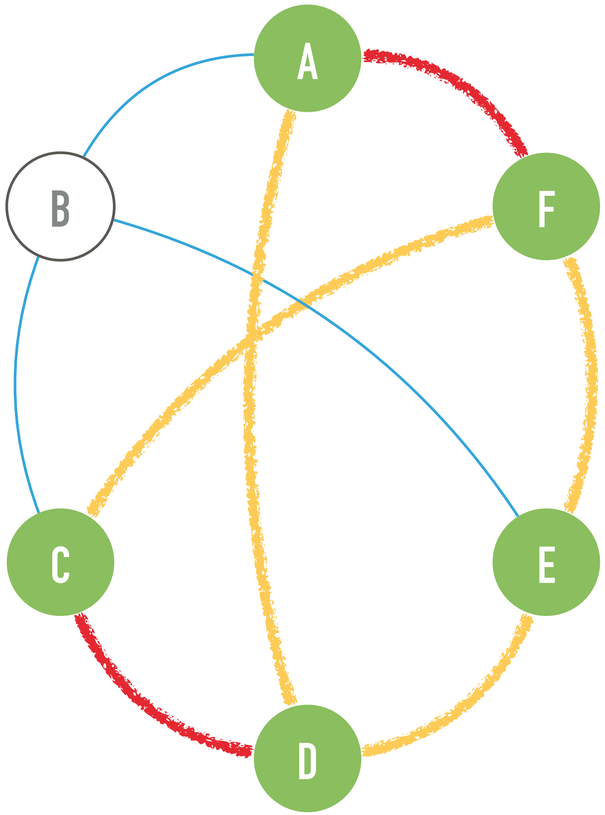
Question 6.1.3: Retirer un noeud et rester connexe



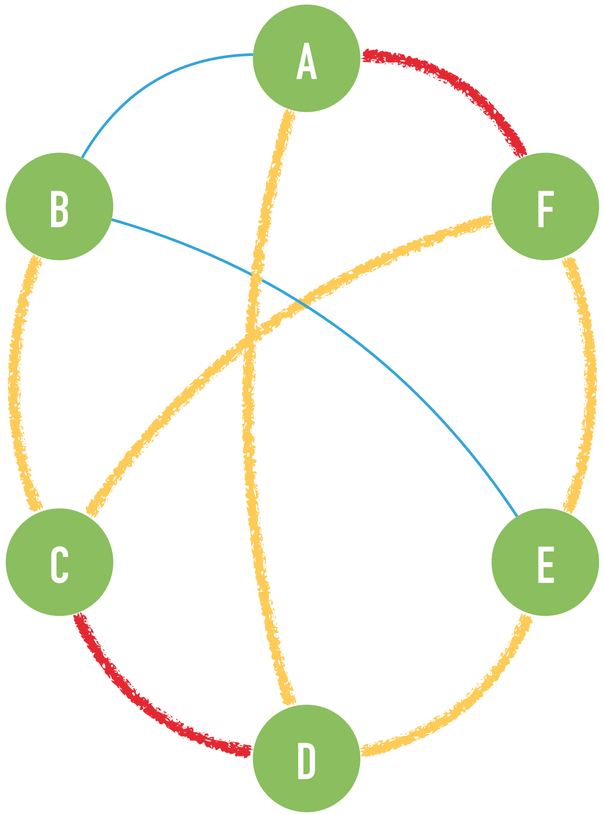
Question 6.1.3: Retirer un noeud et rester connexe



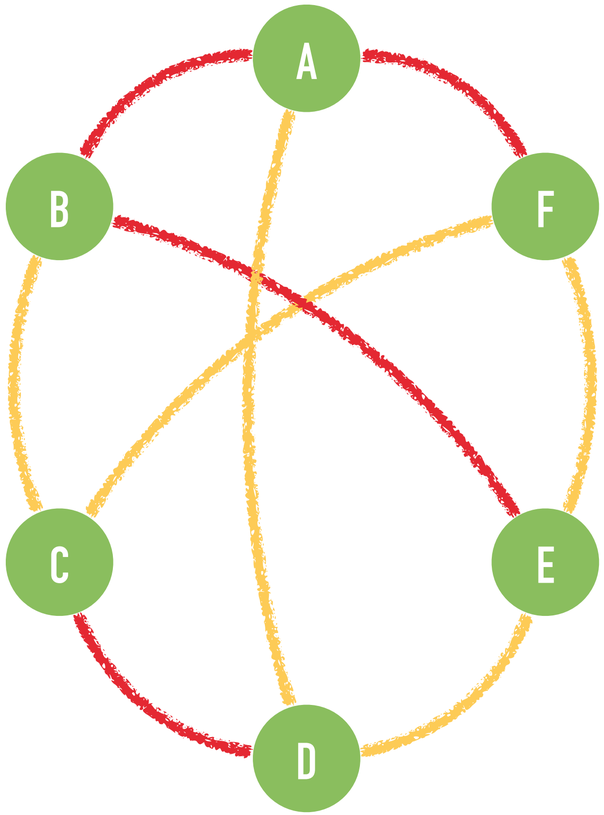
Question 6.1.3: Retirer un noeud et rester connexe



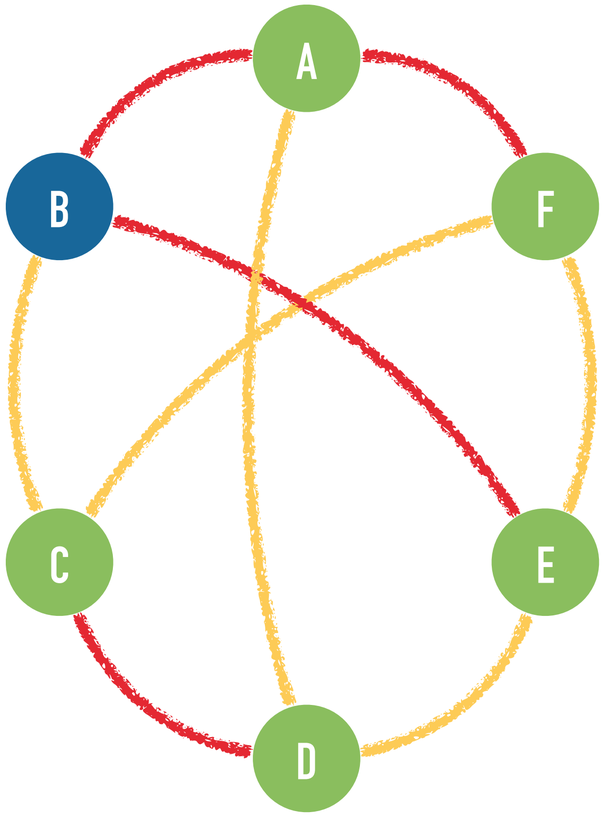
Question 6.1.3: Retirer un noeud et rester connexe



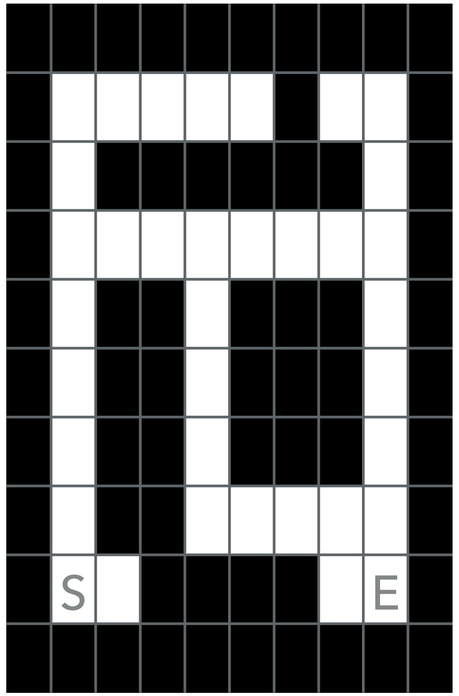
Question 6.1.3: Retirer un noeud et rester connexe

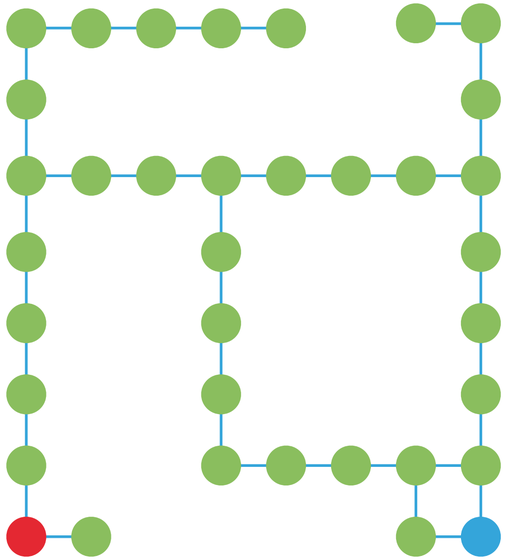


Question 6.1.3: Retirer un noeud et rester connexe

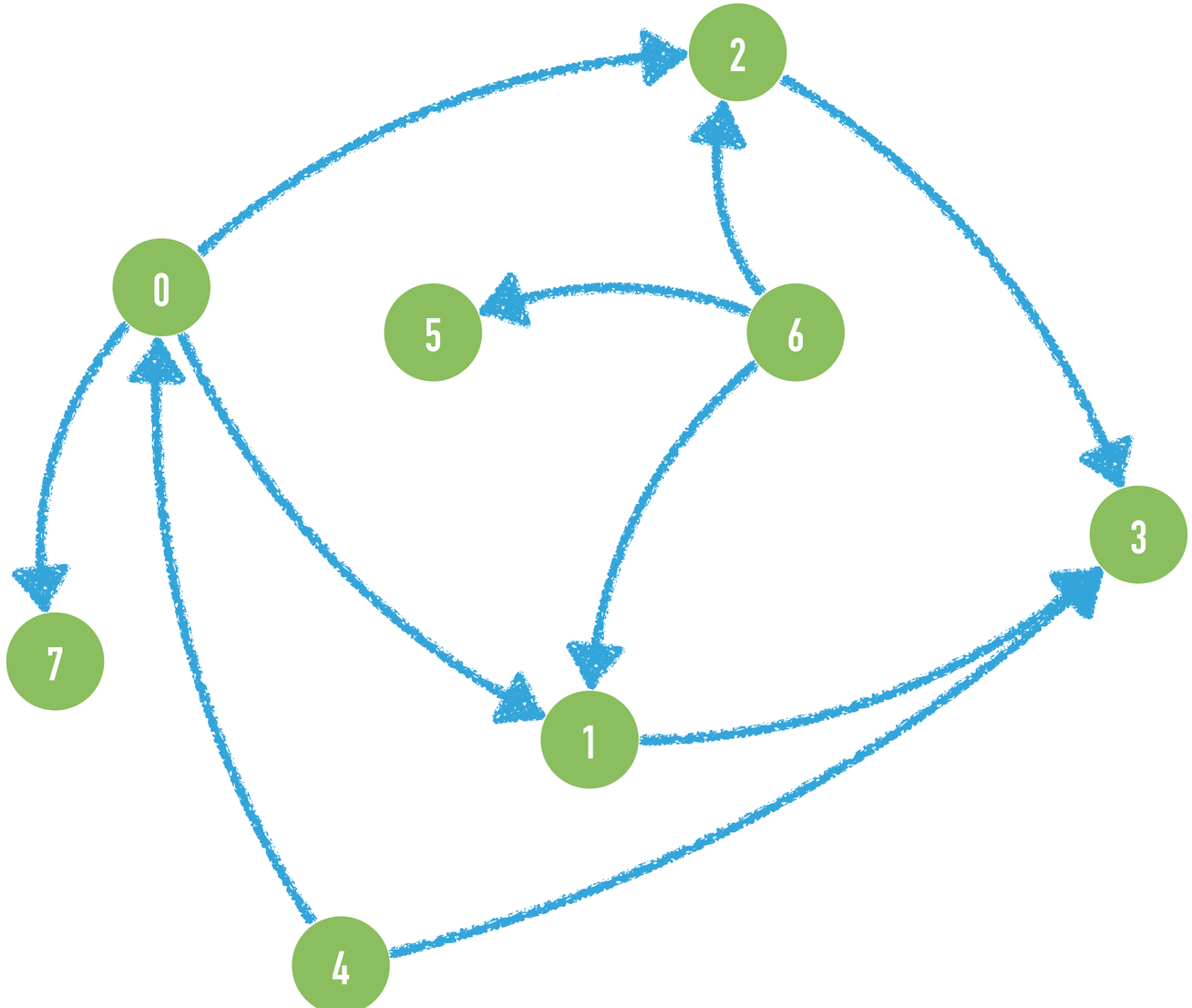


Question 6.1.4: ROBOT

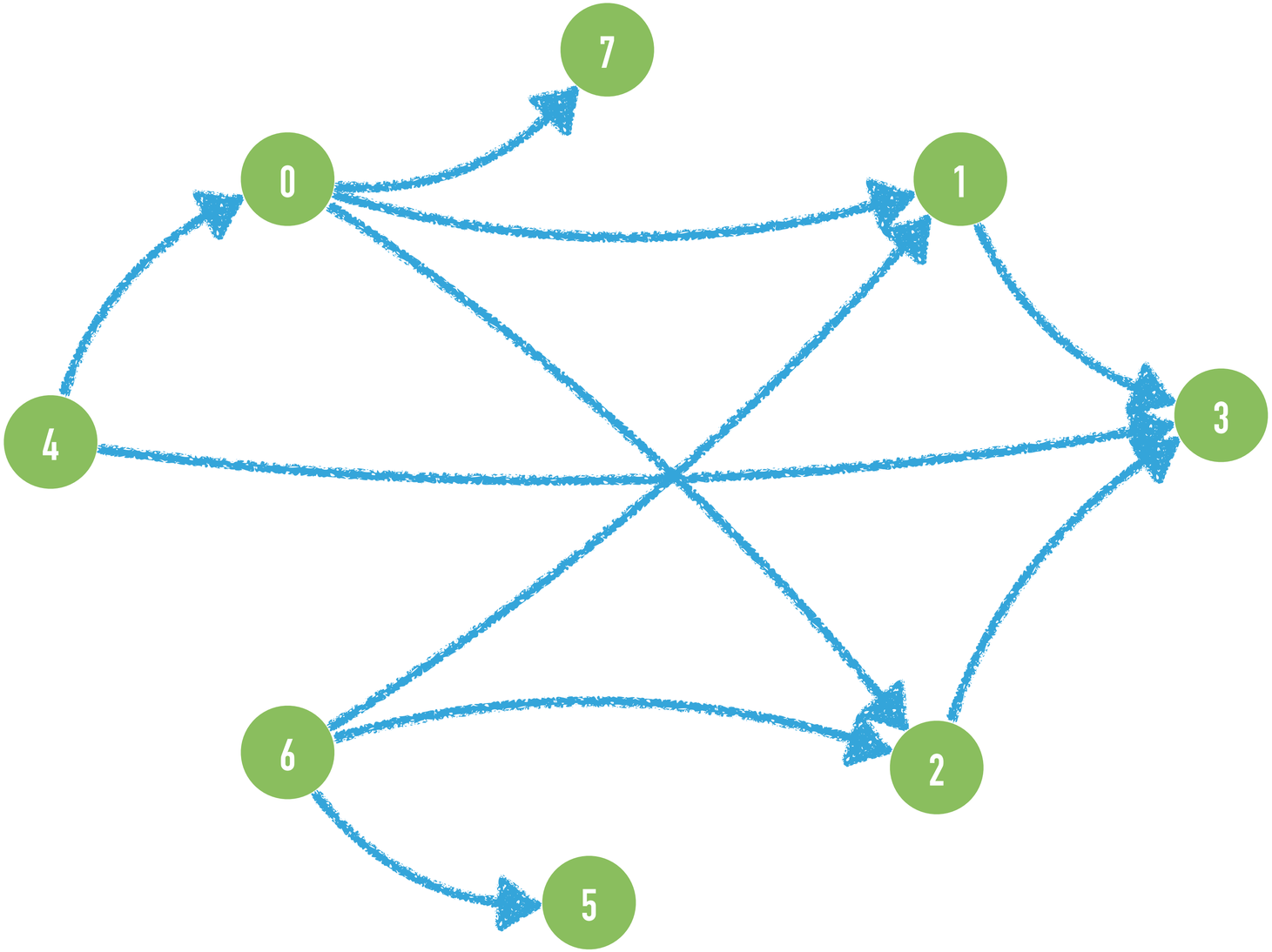




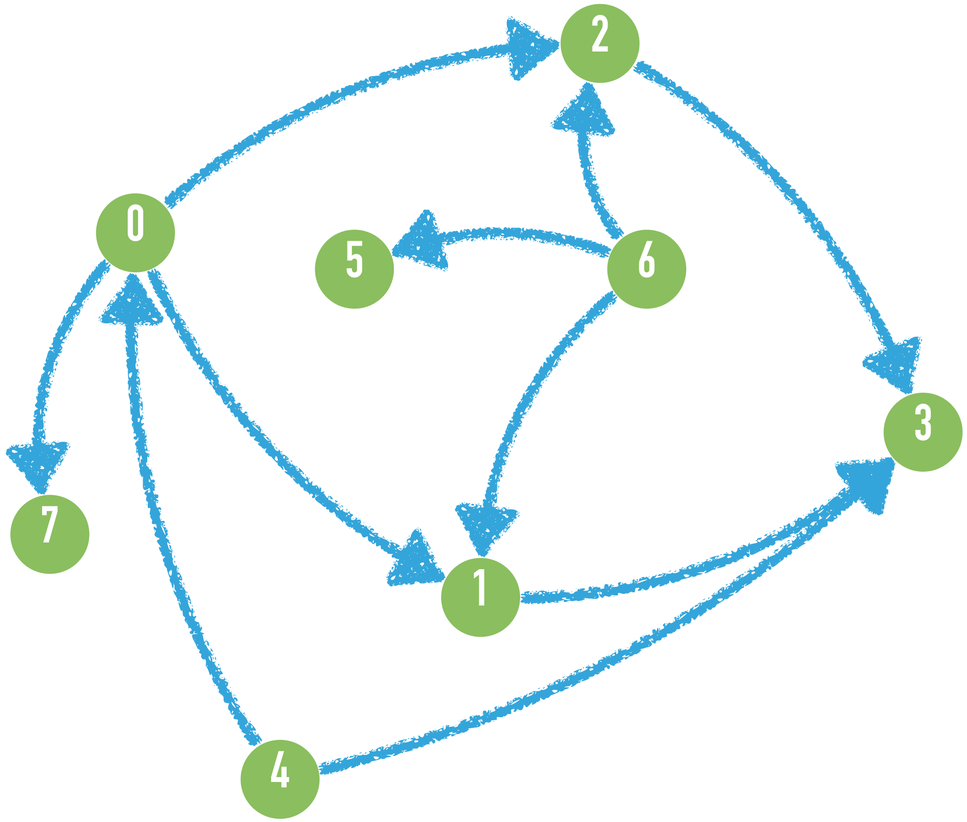
Question 6.1.6: TOPOSORT



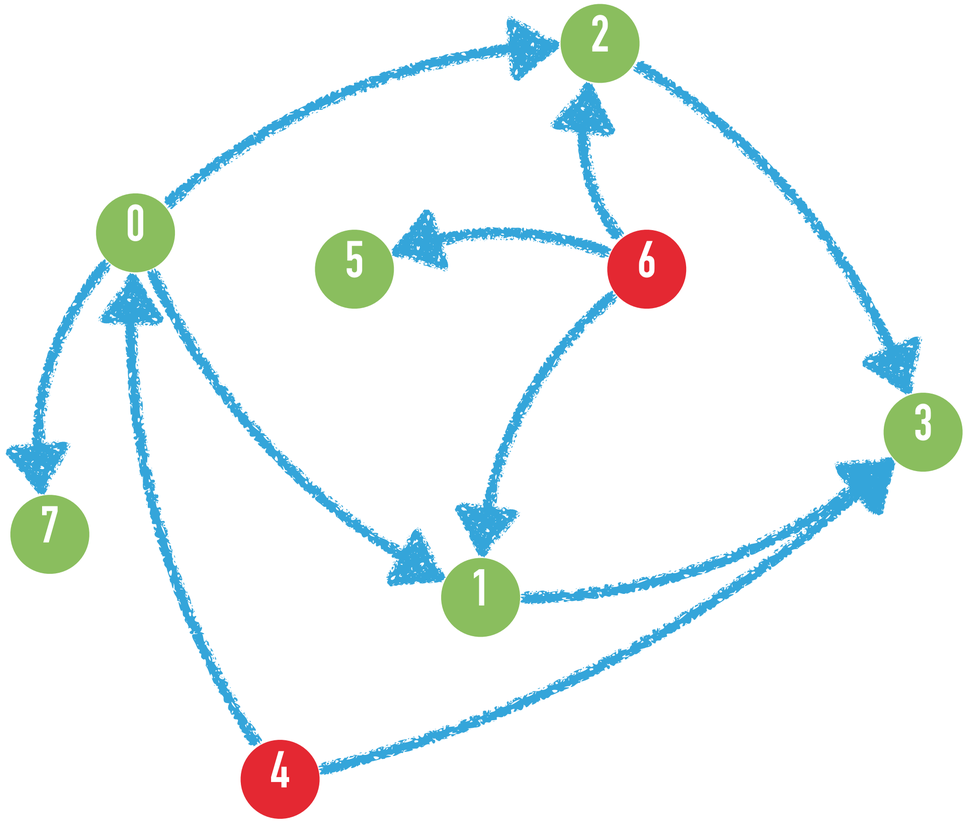
Question 6.1.5: TOPOSORT



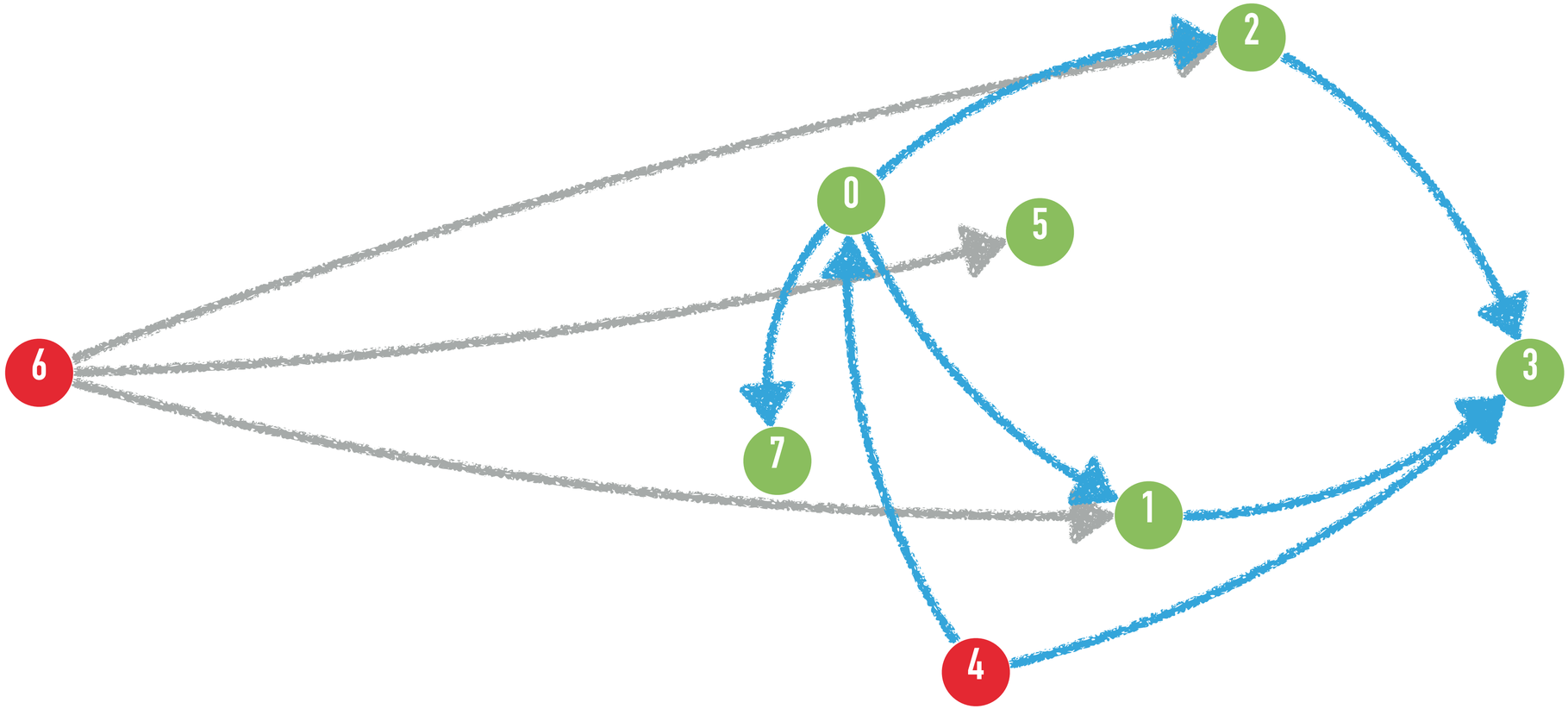
Question 6.1.4: TOPOSORT



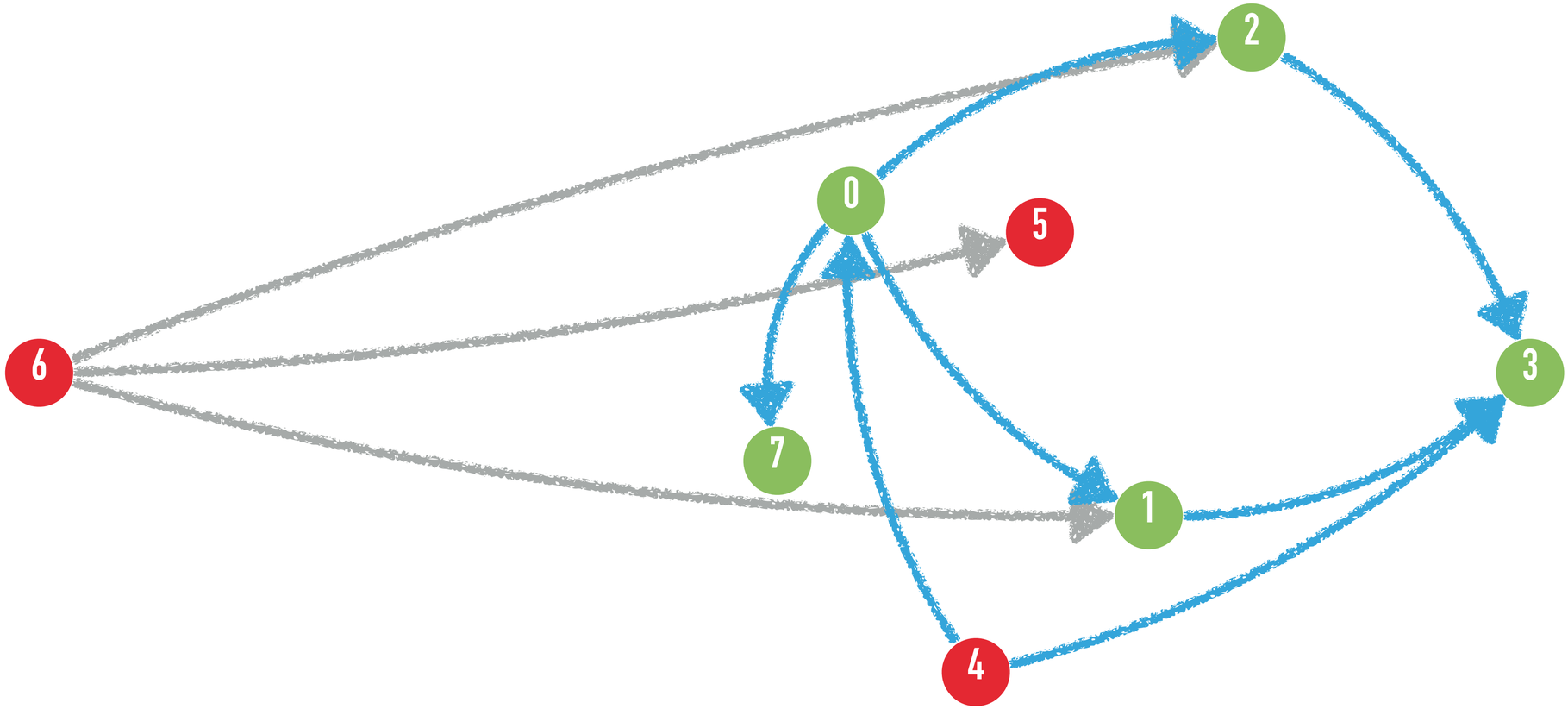
Question 6.1.4: TOPOSORT



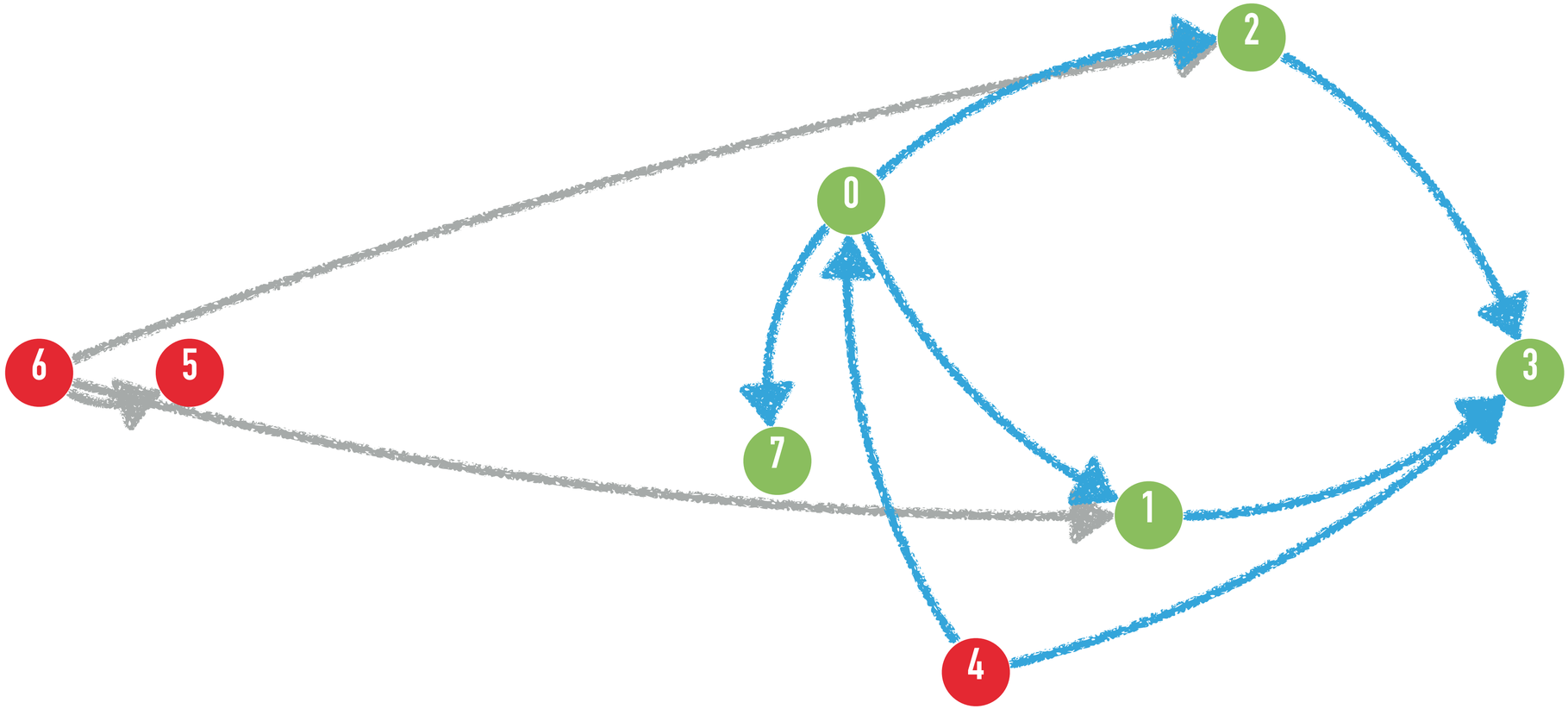
Question 6.1.4: TOPOSORT



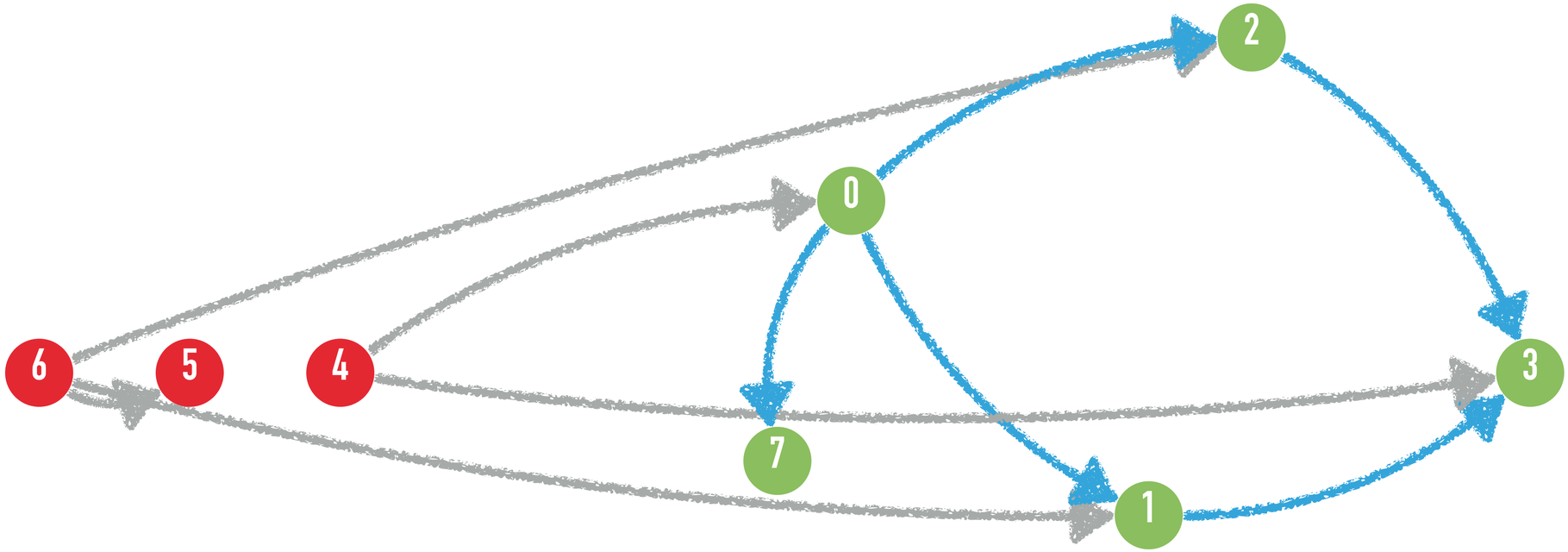
Question 6.1.4: TOPOSORT



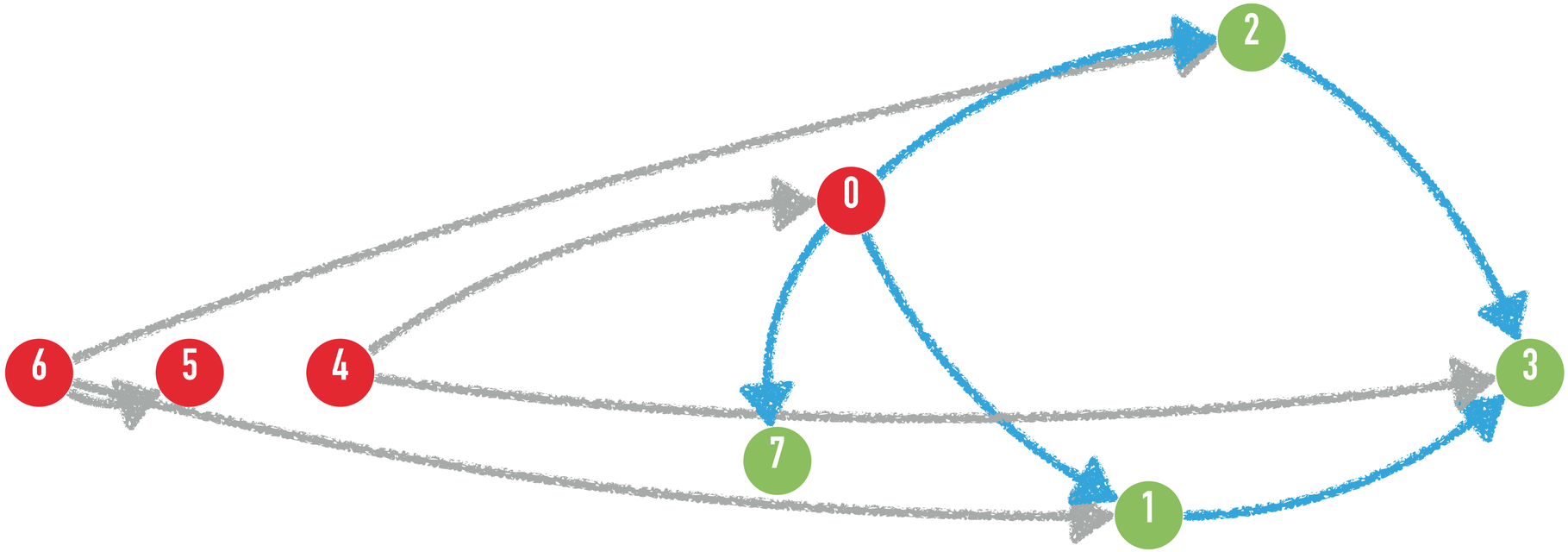
Question 6.1.4: TOPOSORT



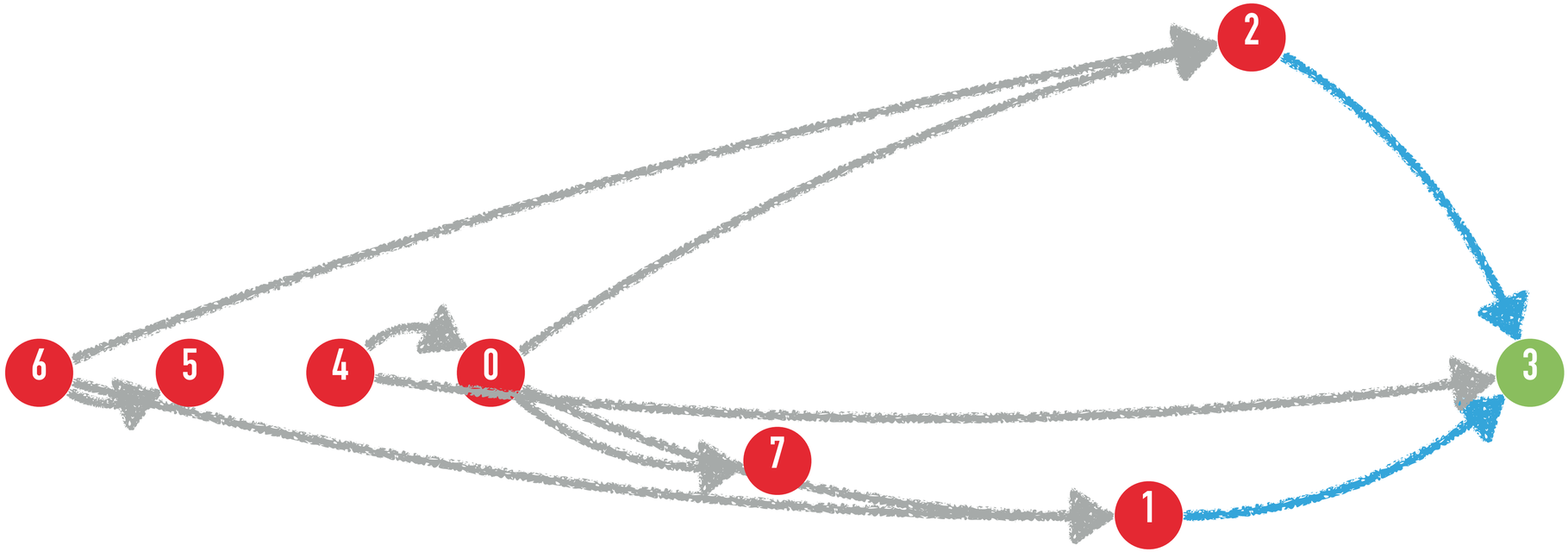
Question 6.1.4: TOPOSORT



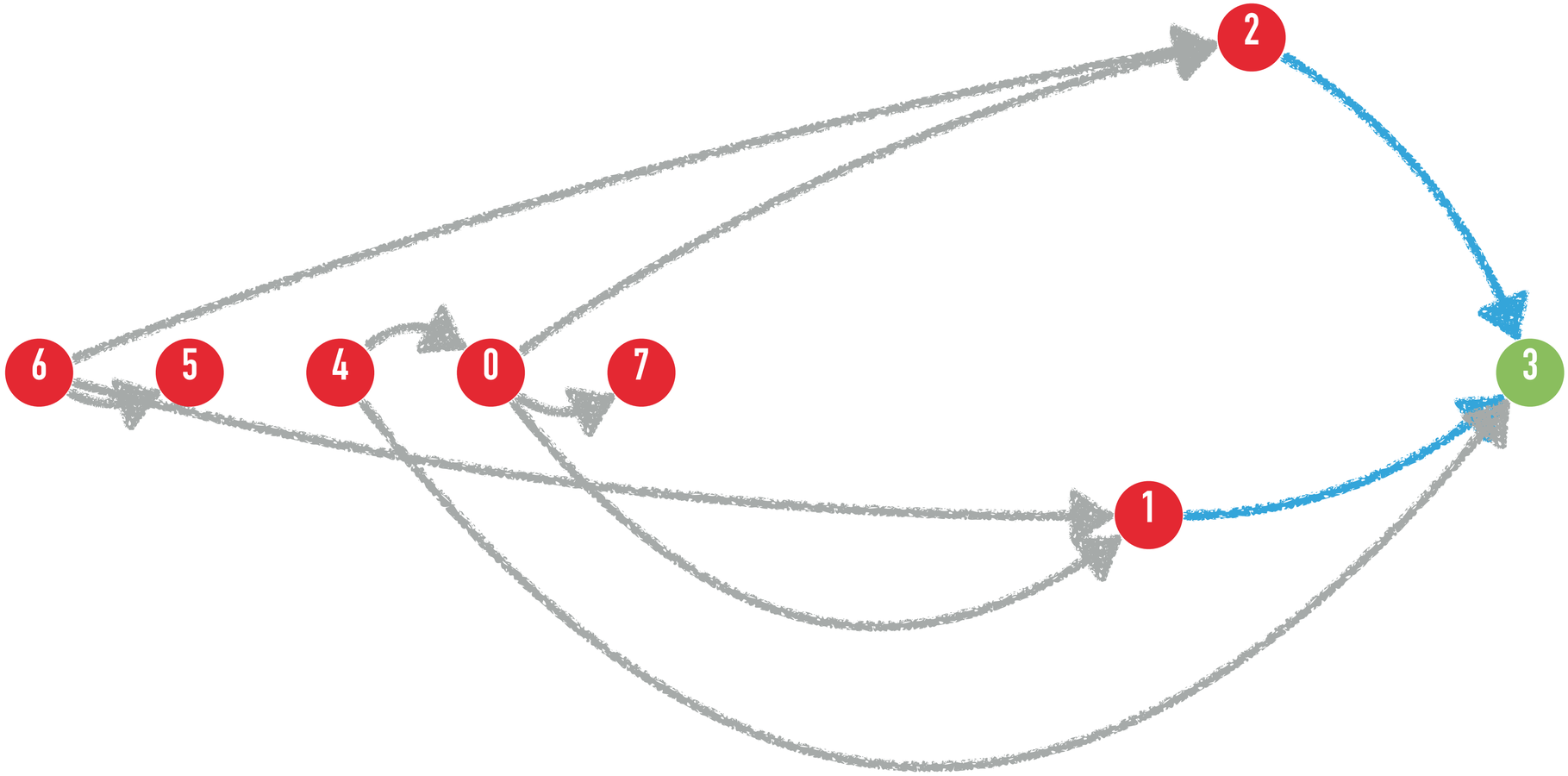
Question 6.1.4: TOPOSORT



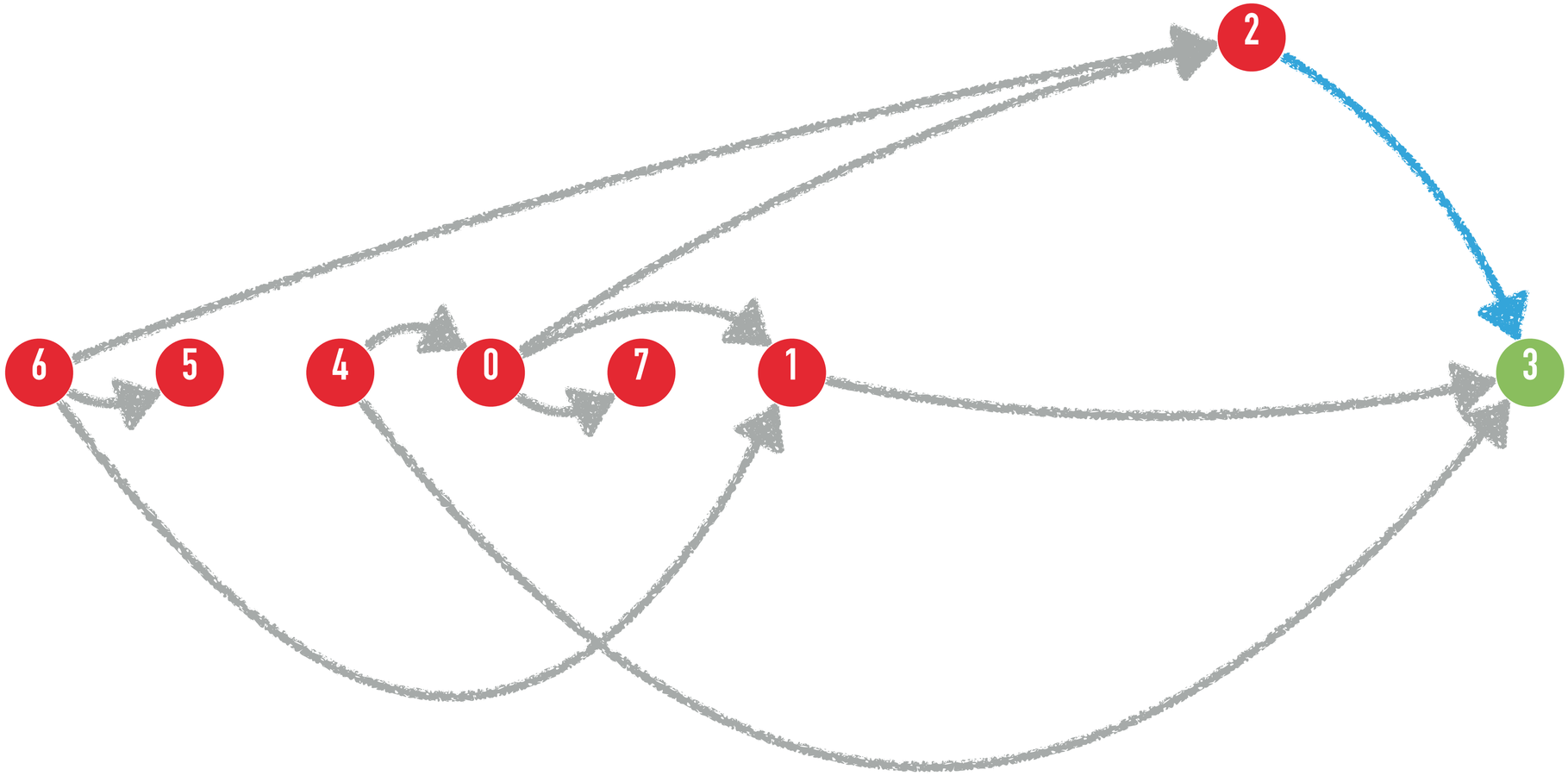
Question 6.1.4: TOPOSORT



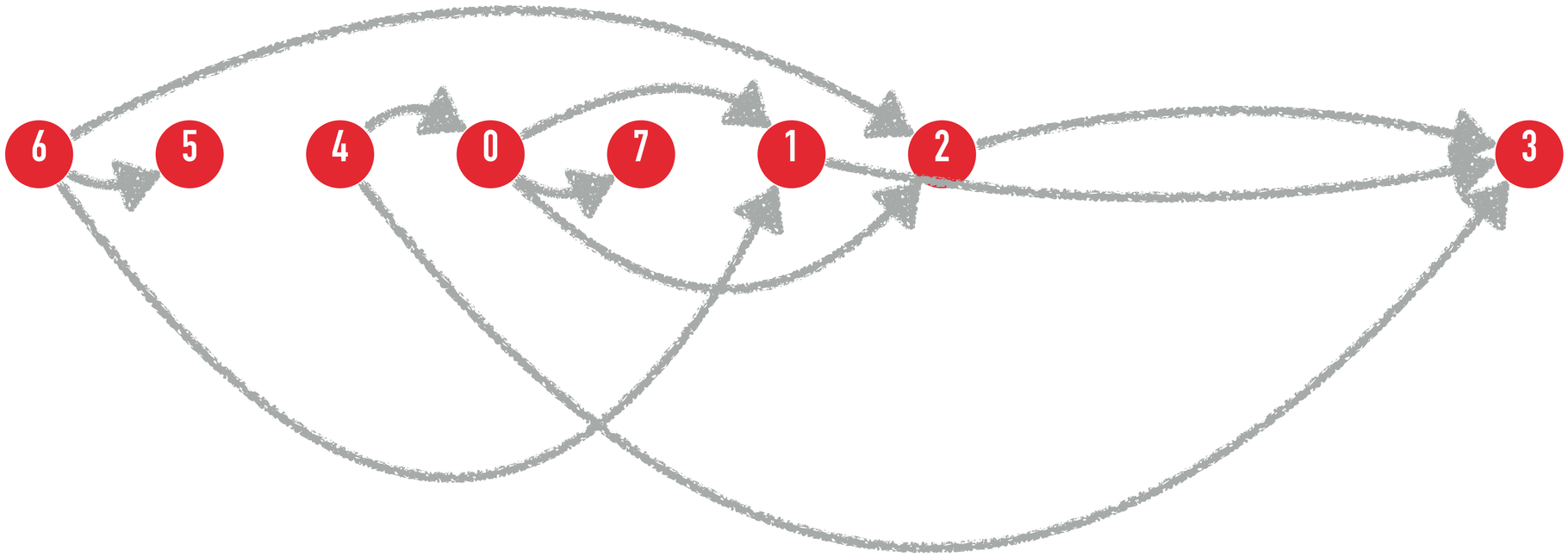
Question 6.1.4: TOPOSORT



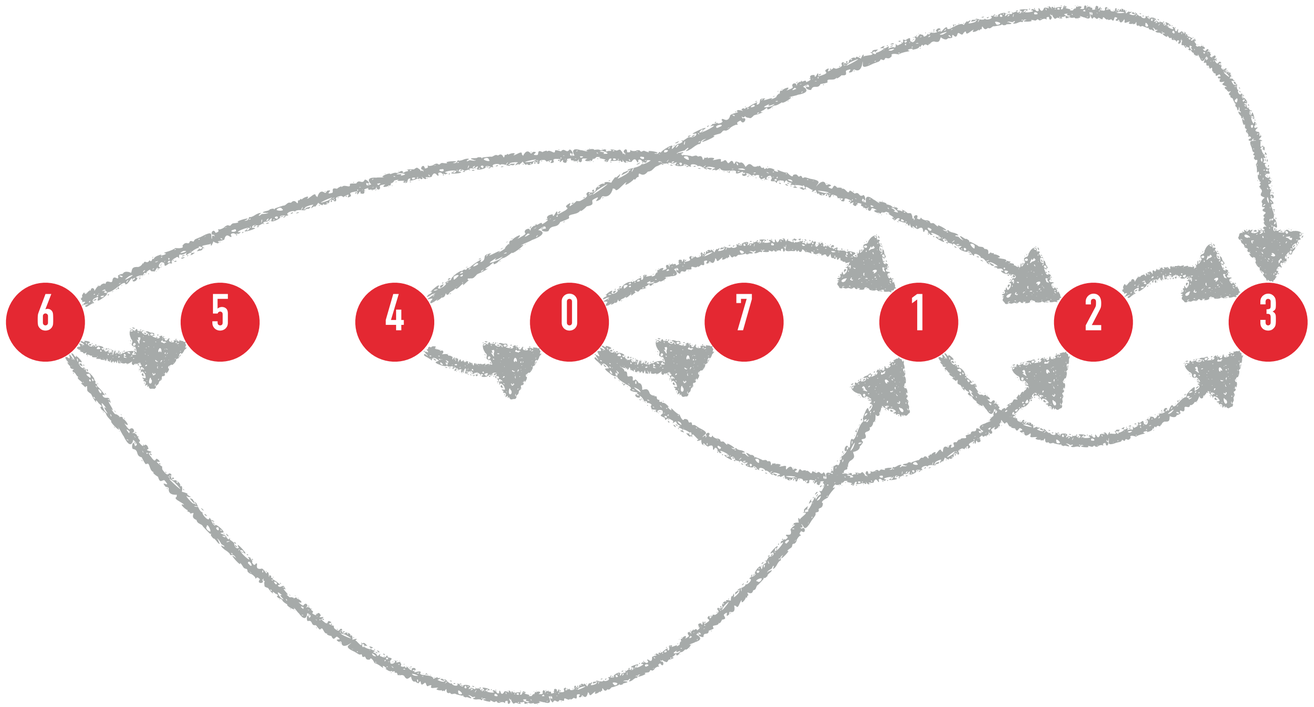
Question 6.1.4: TOPOSORT



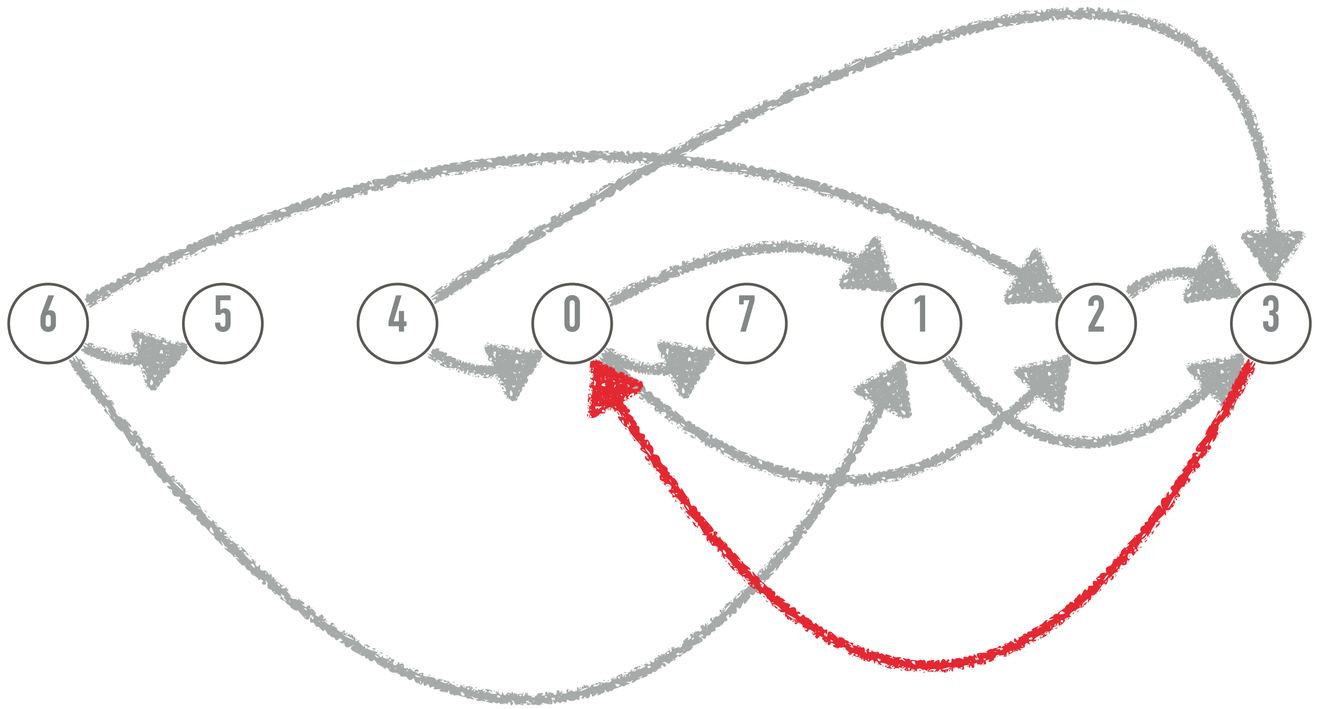
Question 6.1.4: TOPOSORT



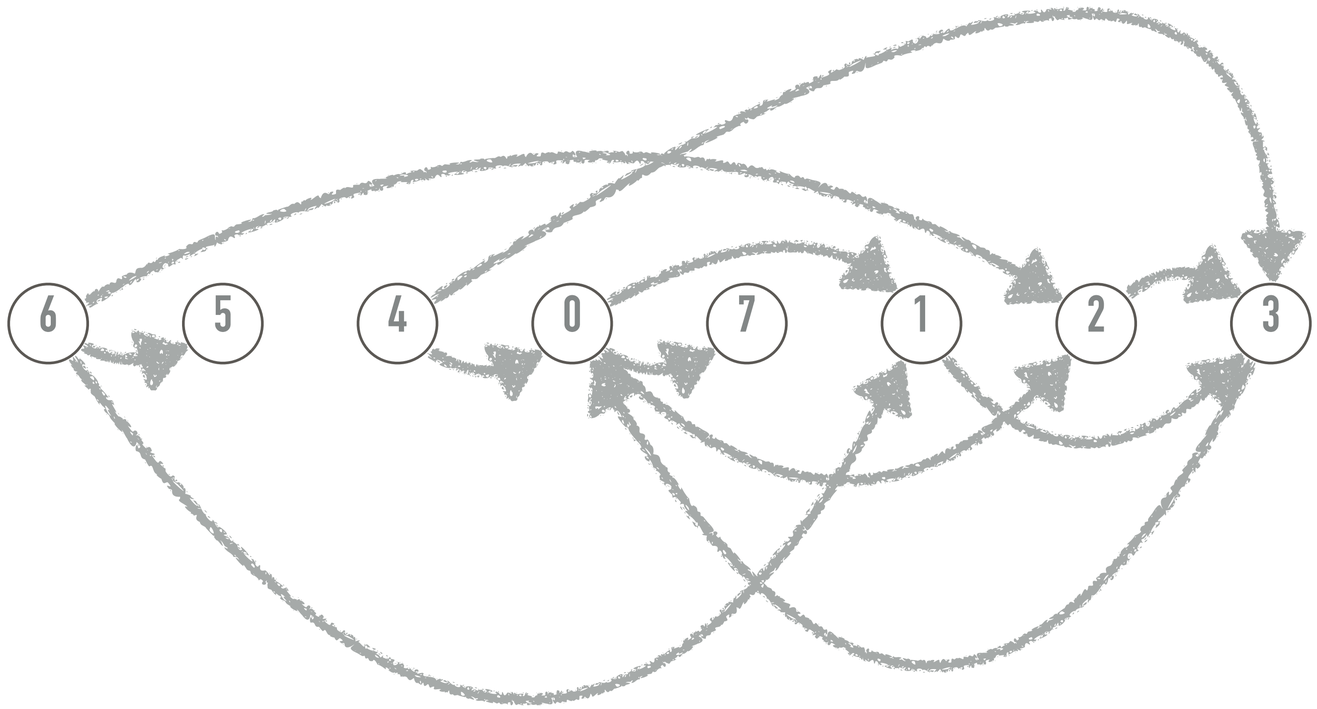
Question 6.1.4: TOPOSORT



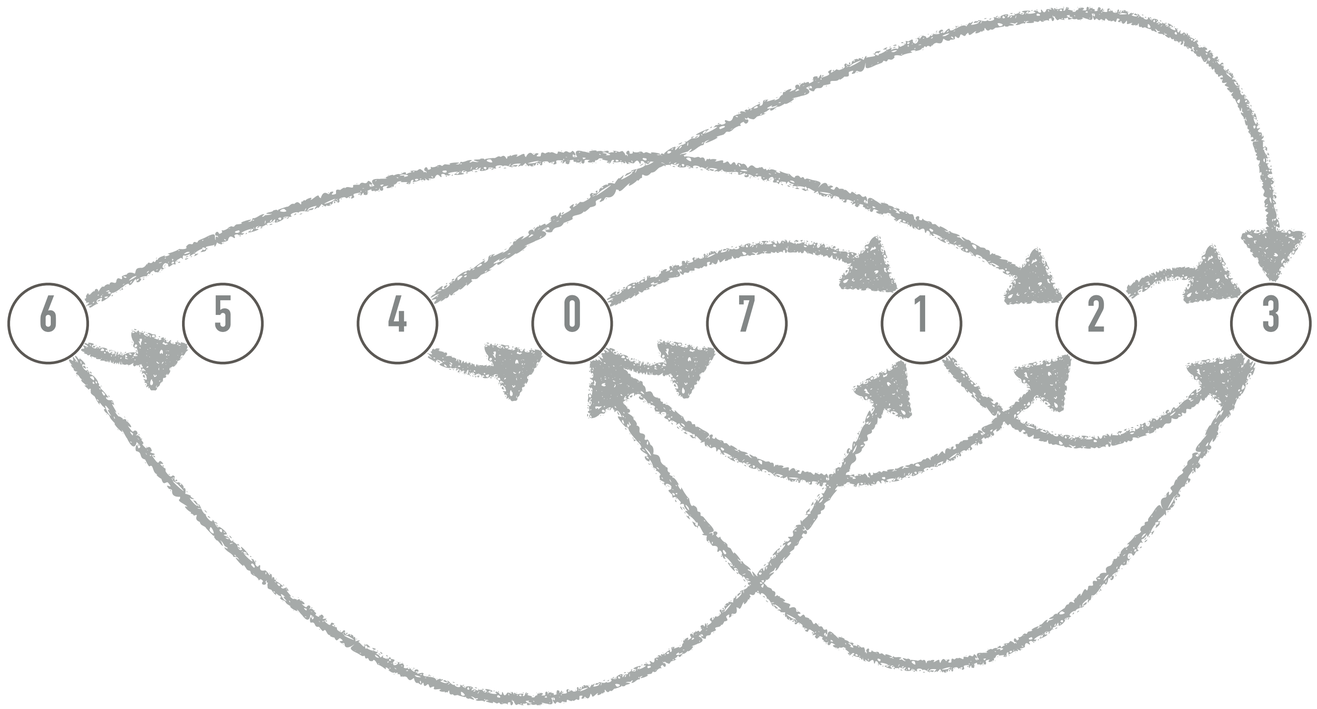
Question 6.1.5: detection de cycle



Question 6.1.5: detection de cycle

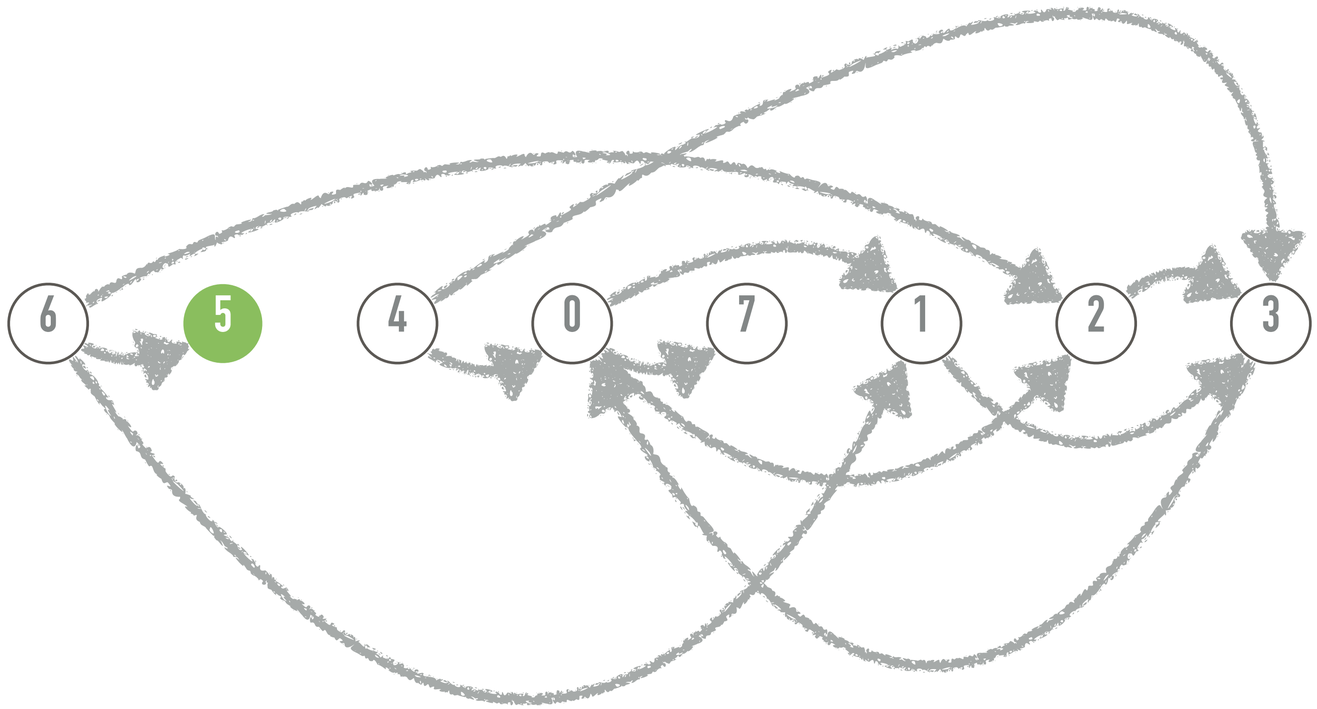


Question 6.1.5: detection de cycle



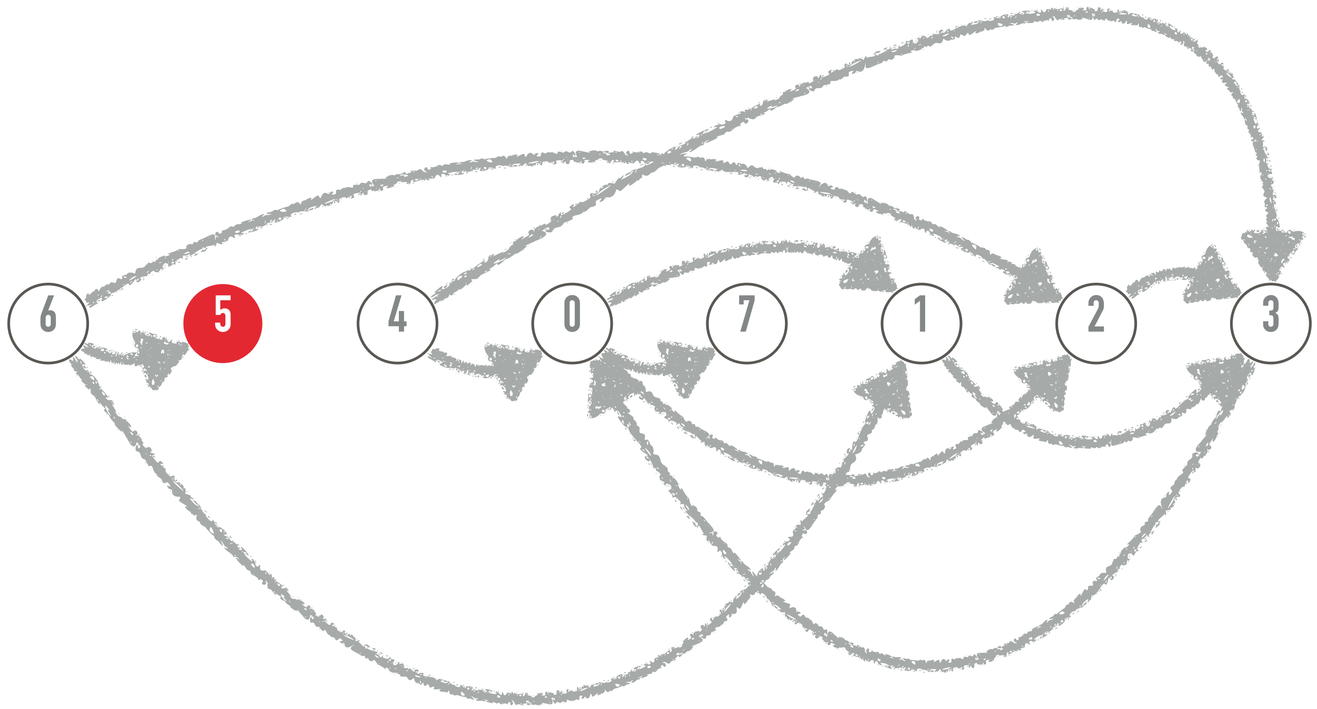
OPEN CLOSED UNVISITED

Question 6.1.5: detection de cycle



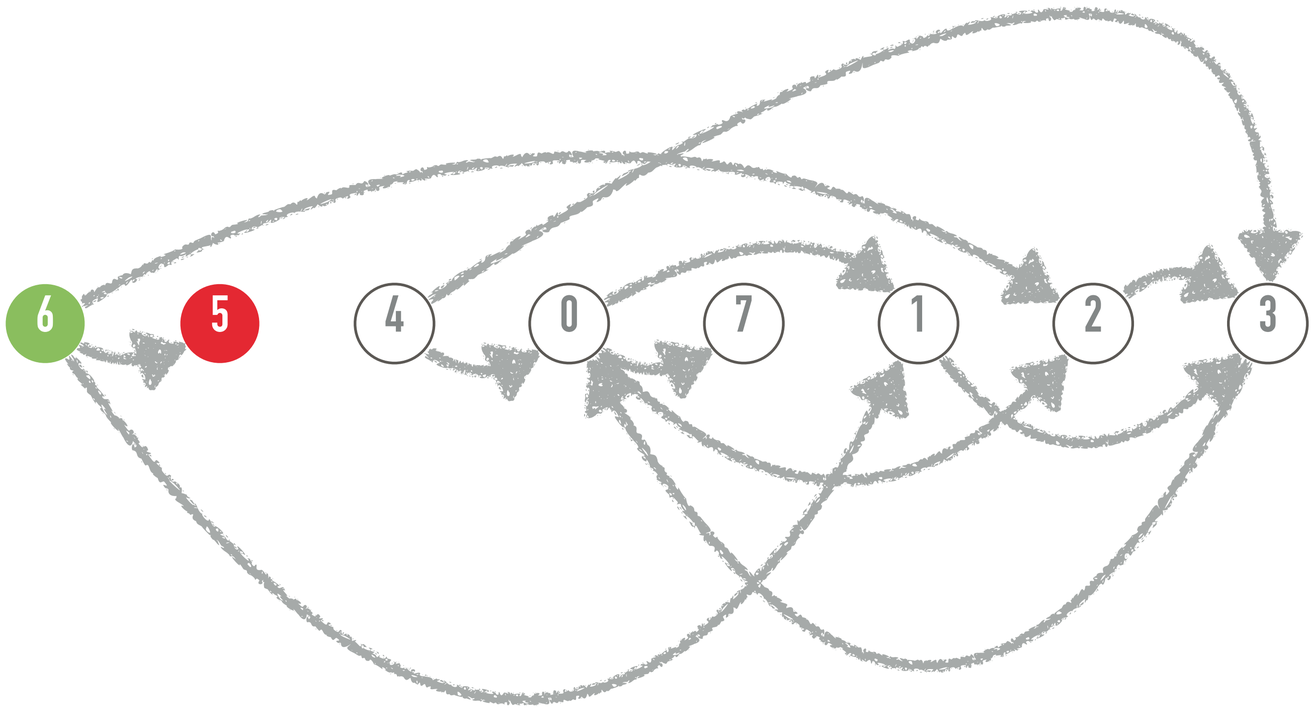
OPEN CLOSED UNVISITED

Question 6.1.5: detection de cycle



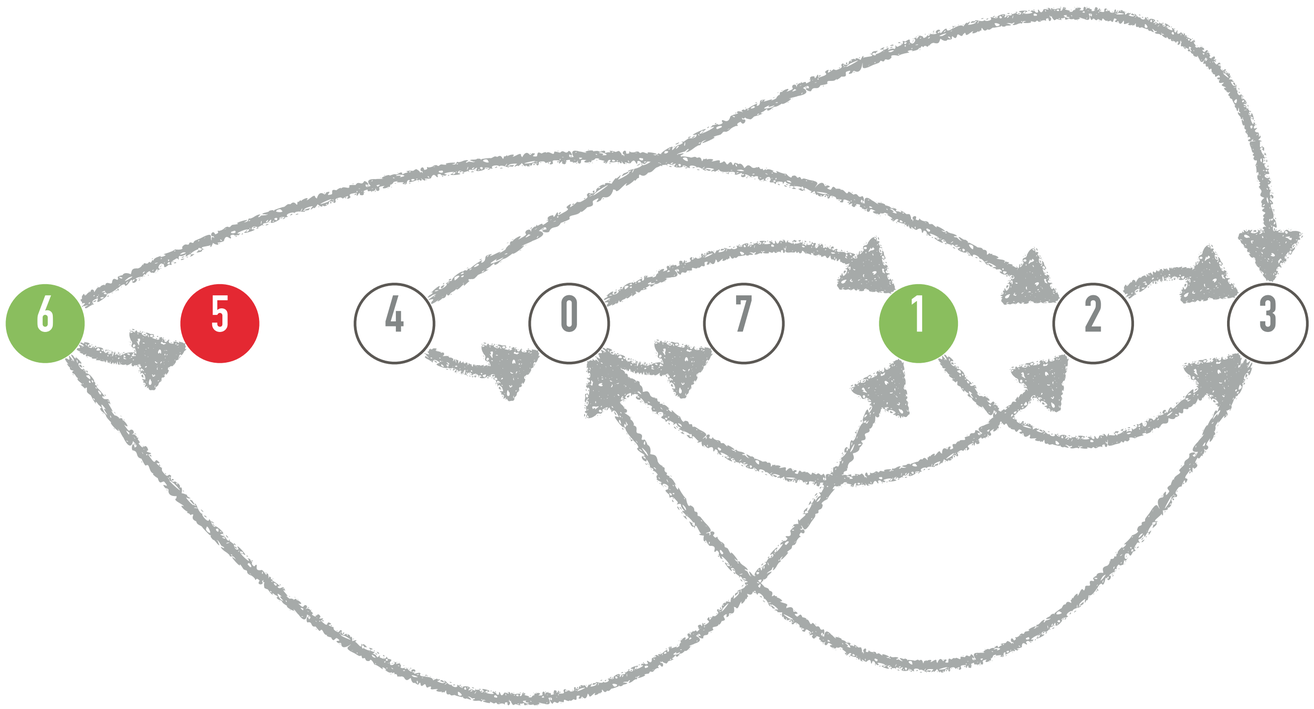
OPEN CLOSED UNVISITED

Question 6.1.5: detection de cycle



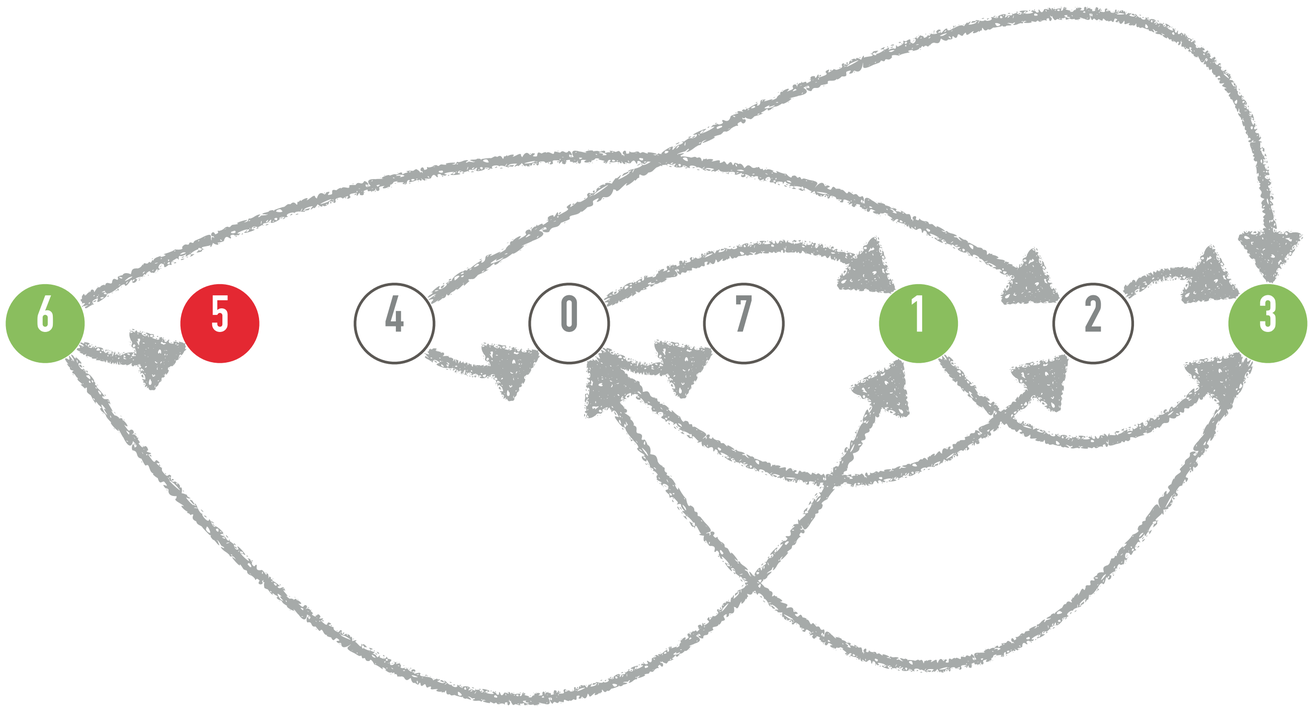
OPEN CLOSED UNVISITED

Question 6.1.5: detection de cycle



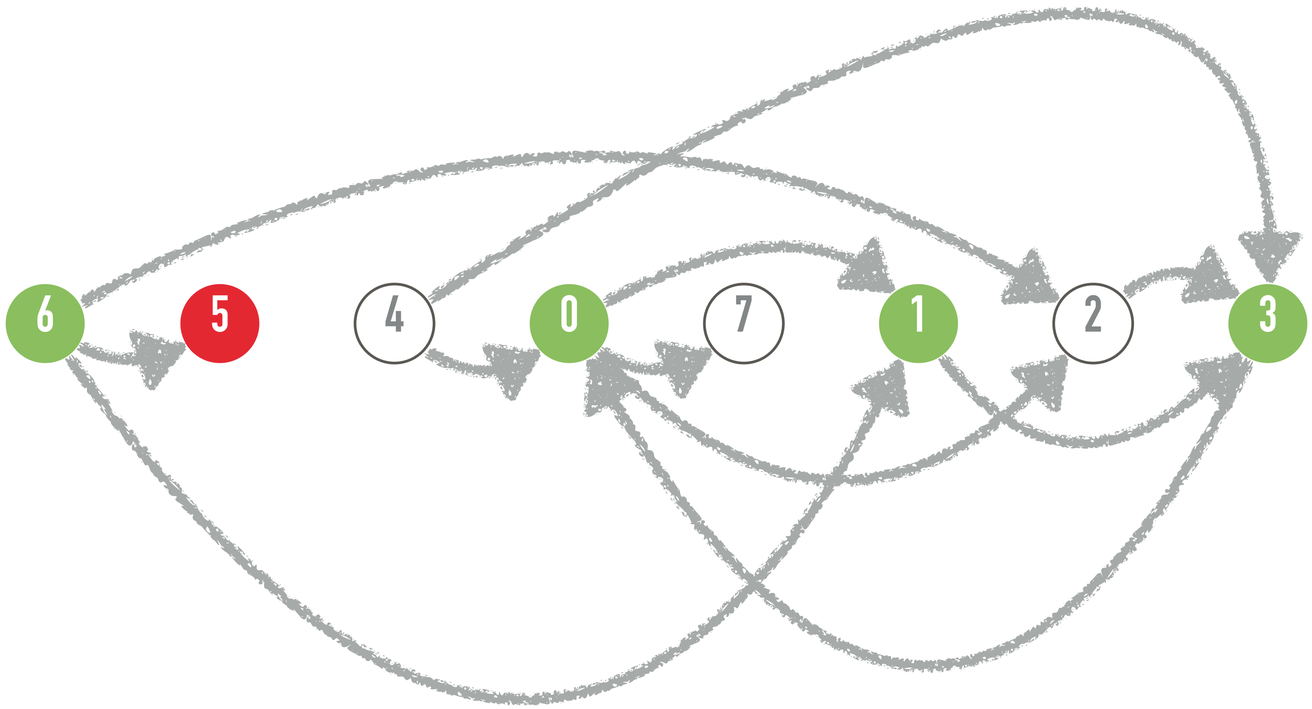
OPEN CLOSED UNVISITED

Question 6.1.5: detection de cycle



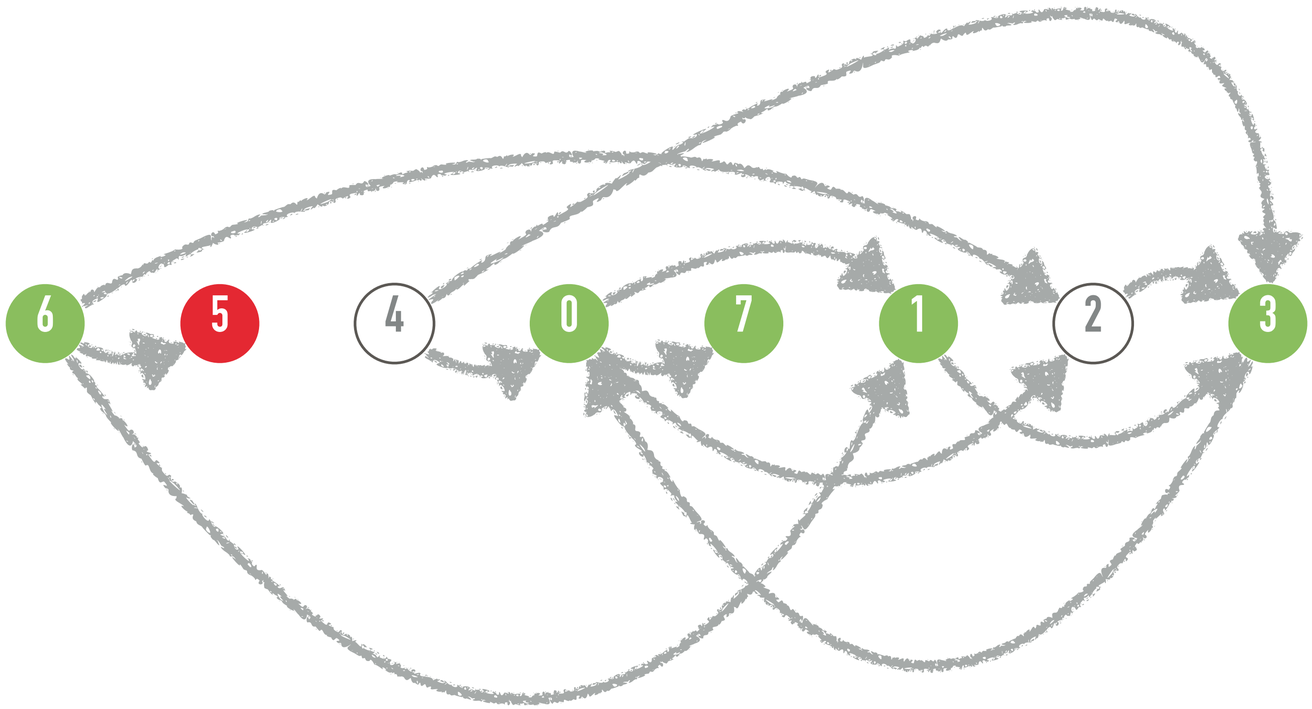
OPEN CLOSED UNVISITED

Question 6.1.5: detection de cycle



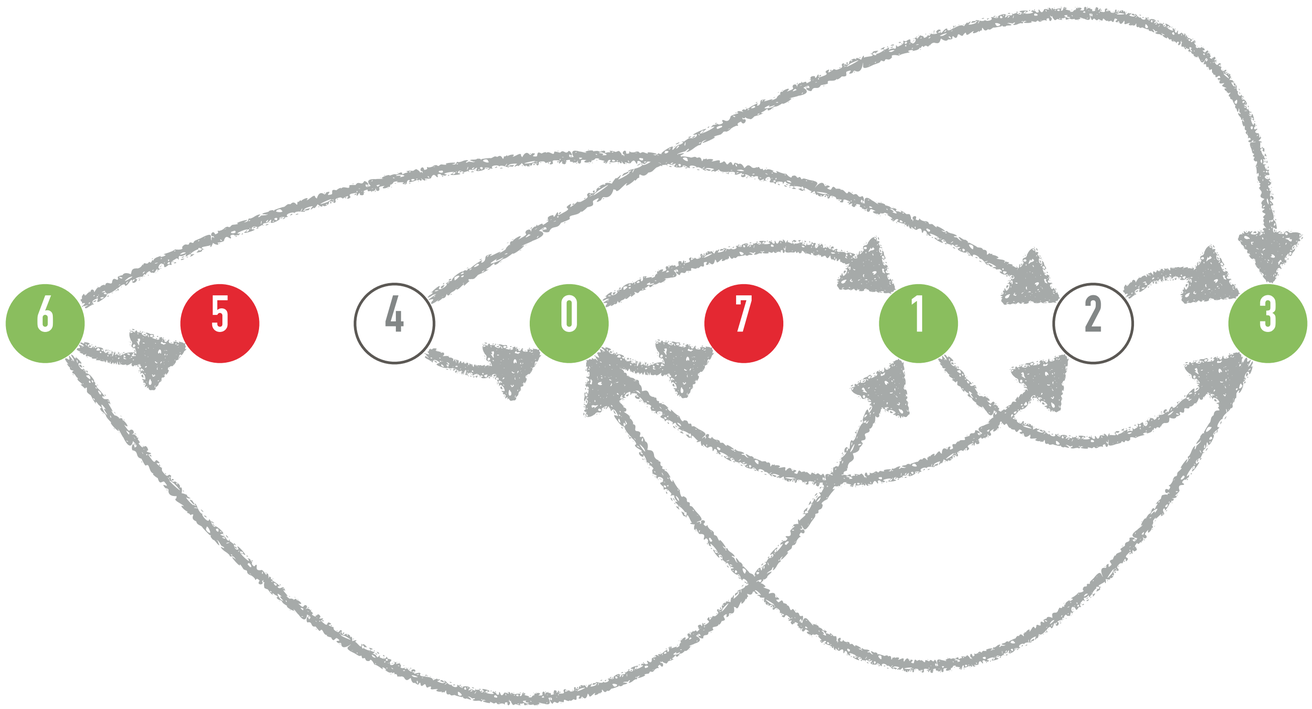
OPEN CLOSED UNVISITED

Question 6.1.5: detection de cycle



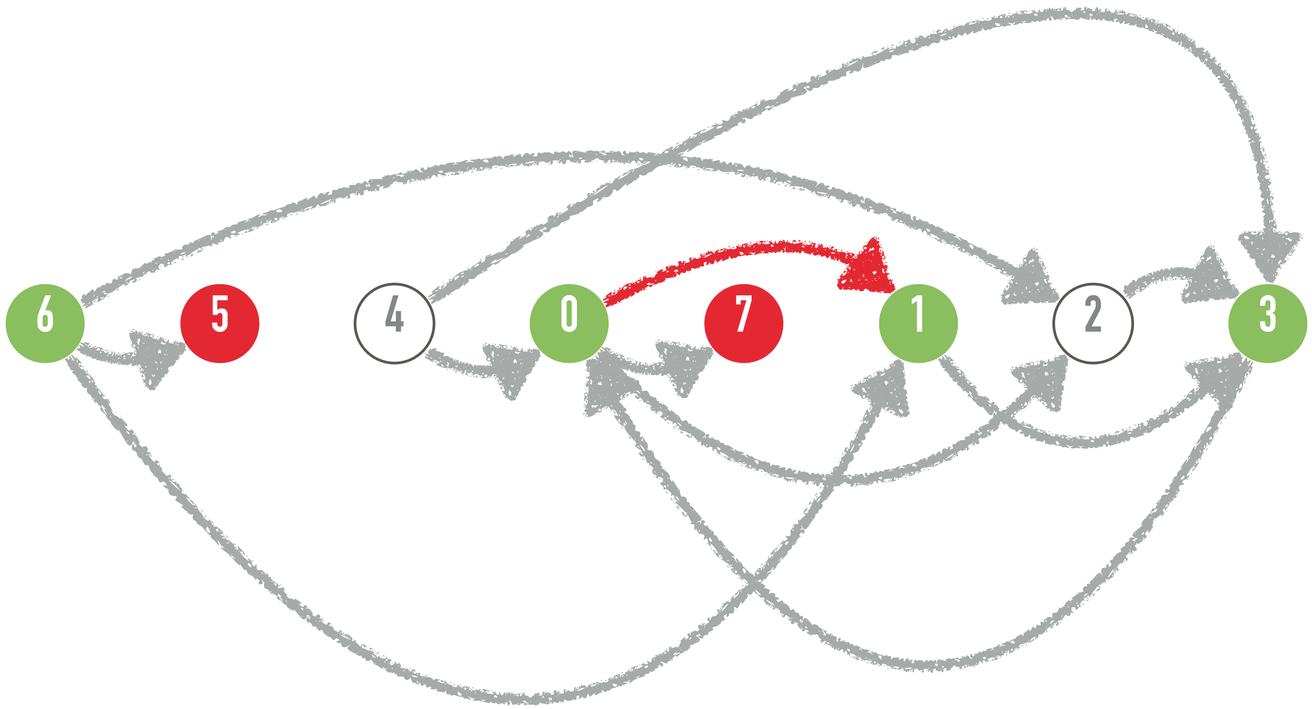
OPEN CLOSED UNVISITED

Question 6.1.5: detection de cycle



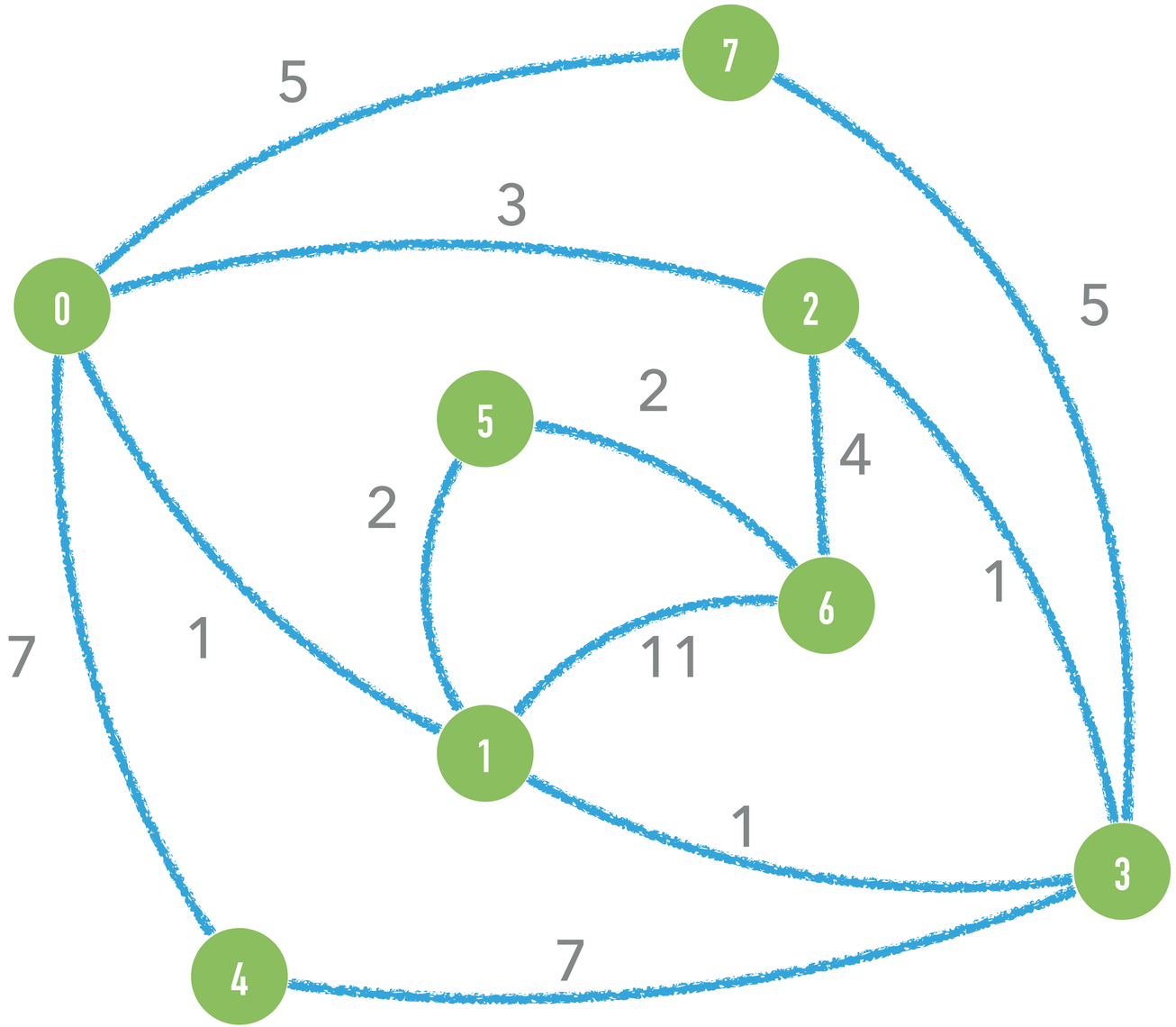
OPEN CLOSED UNVISITED

Question 6.1.5: detection de cycle

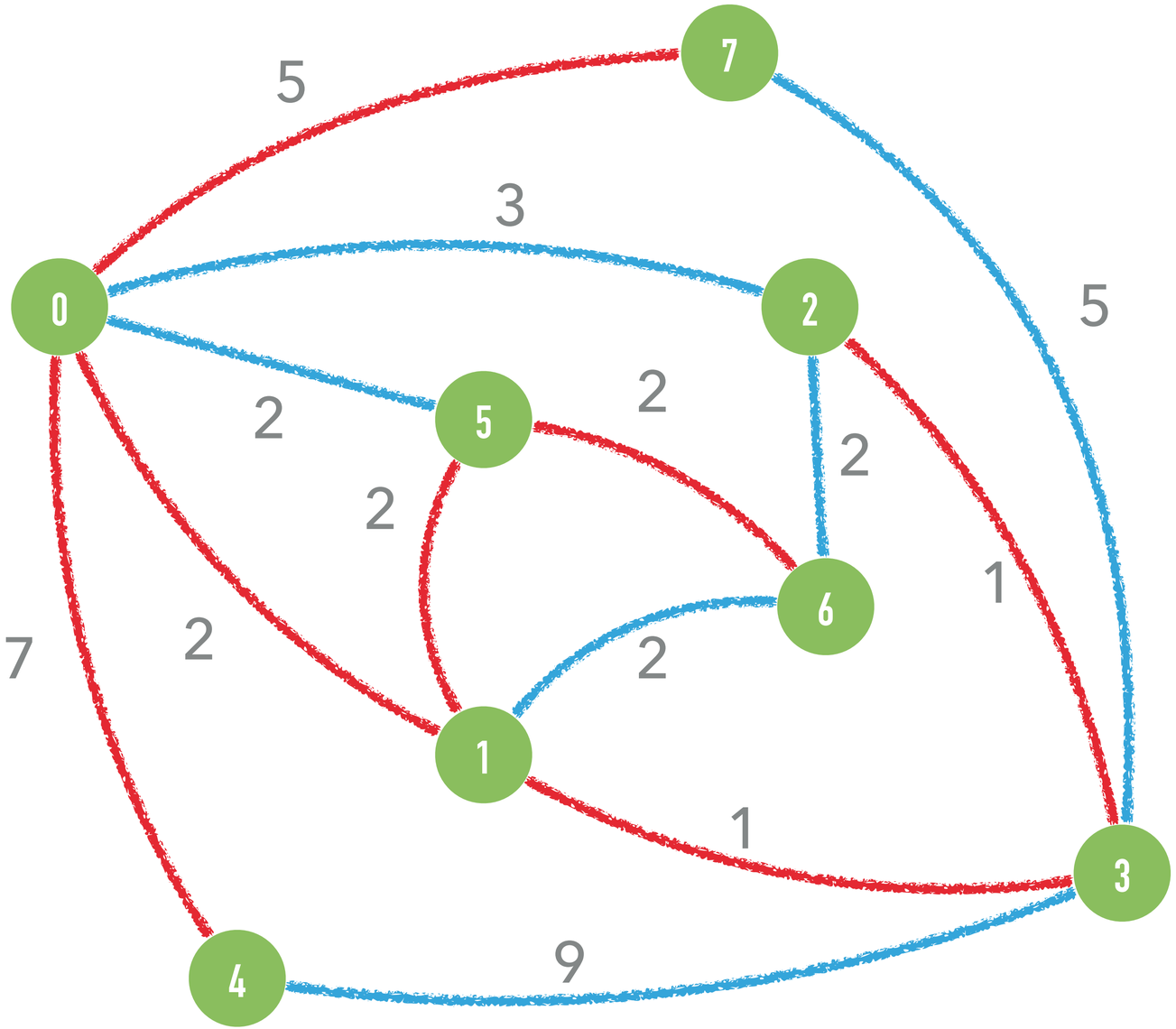


OPEN CLOSED UNVISITED

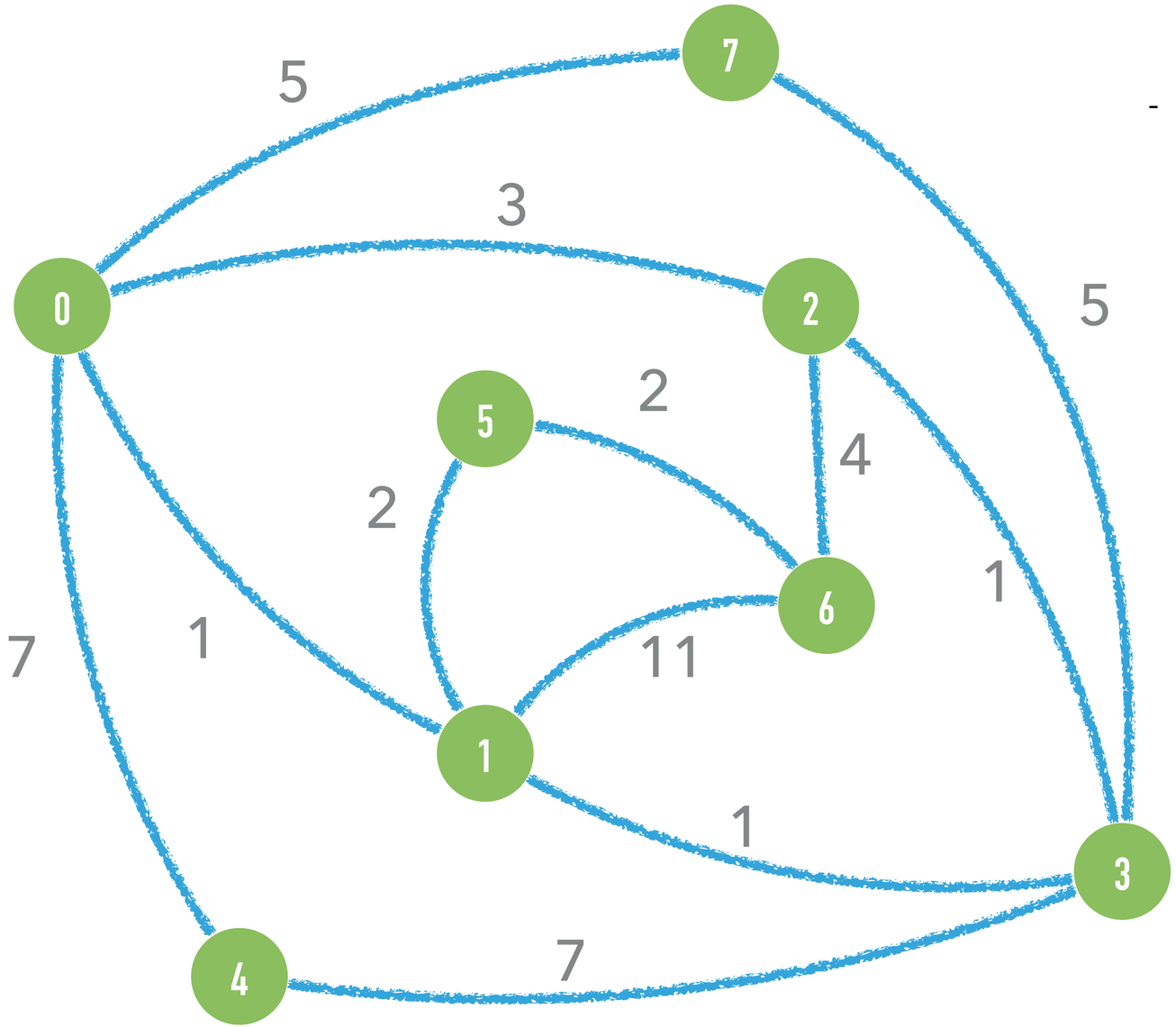
Question 6.1.7: MST



Question 6.1.7: MST

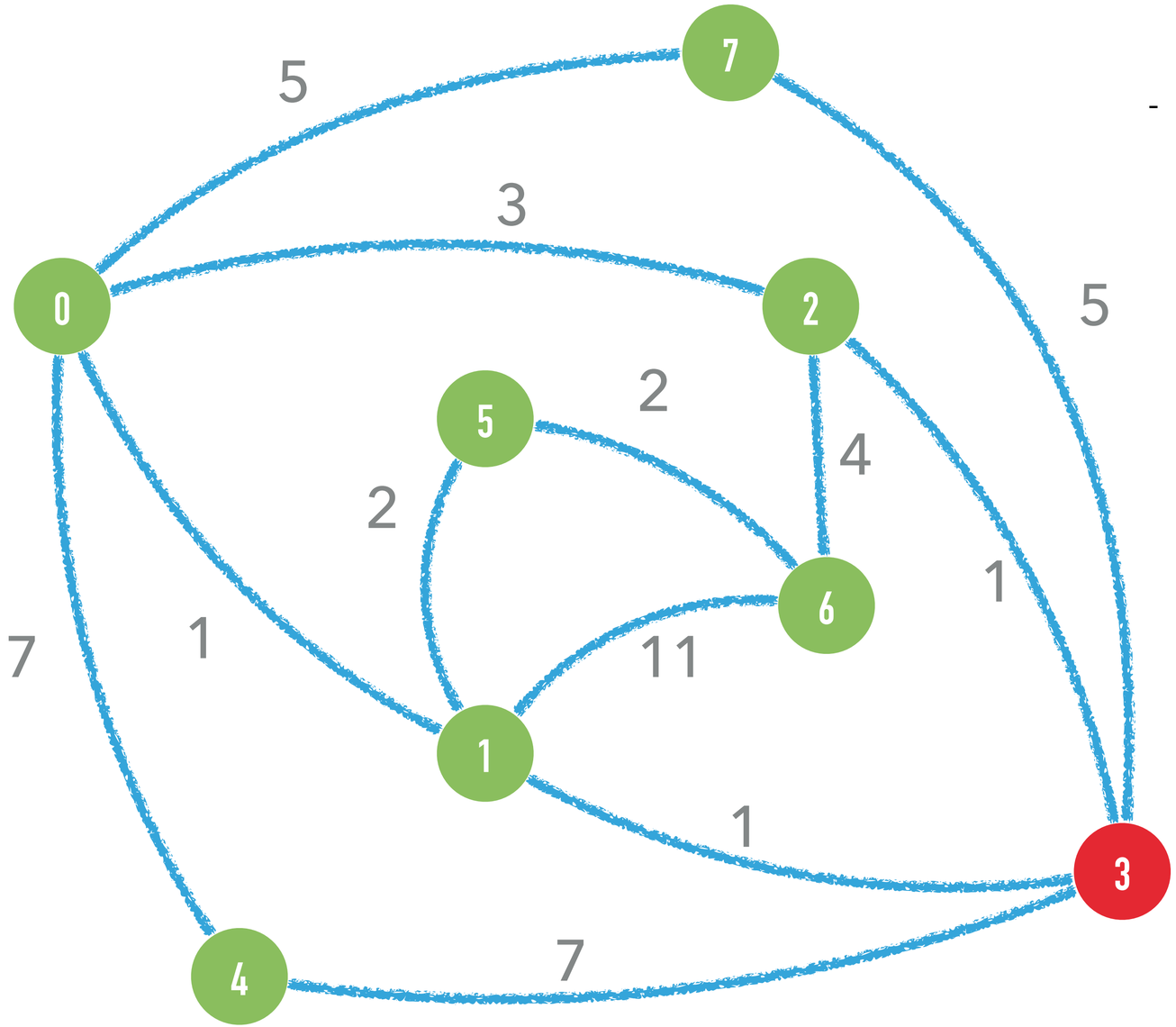


Question 6.1.7: MST - PRIM



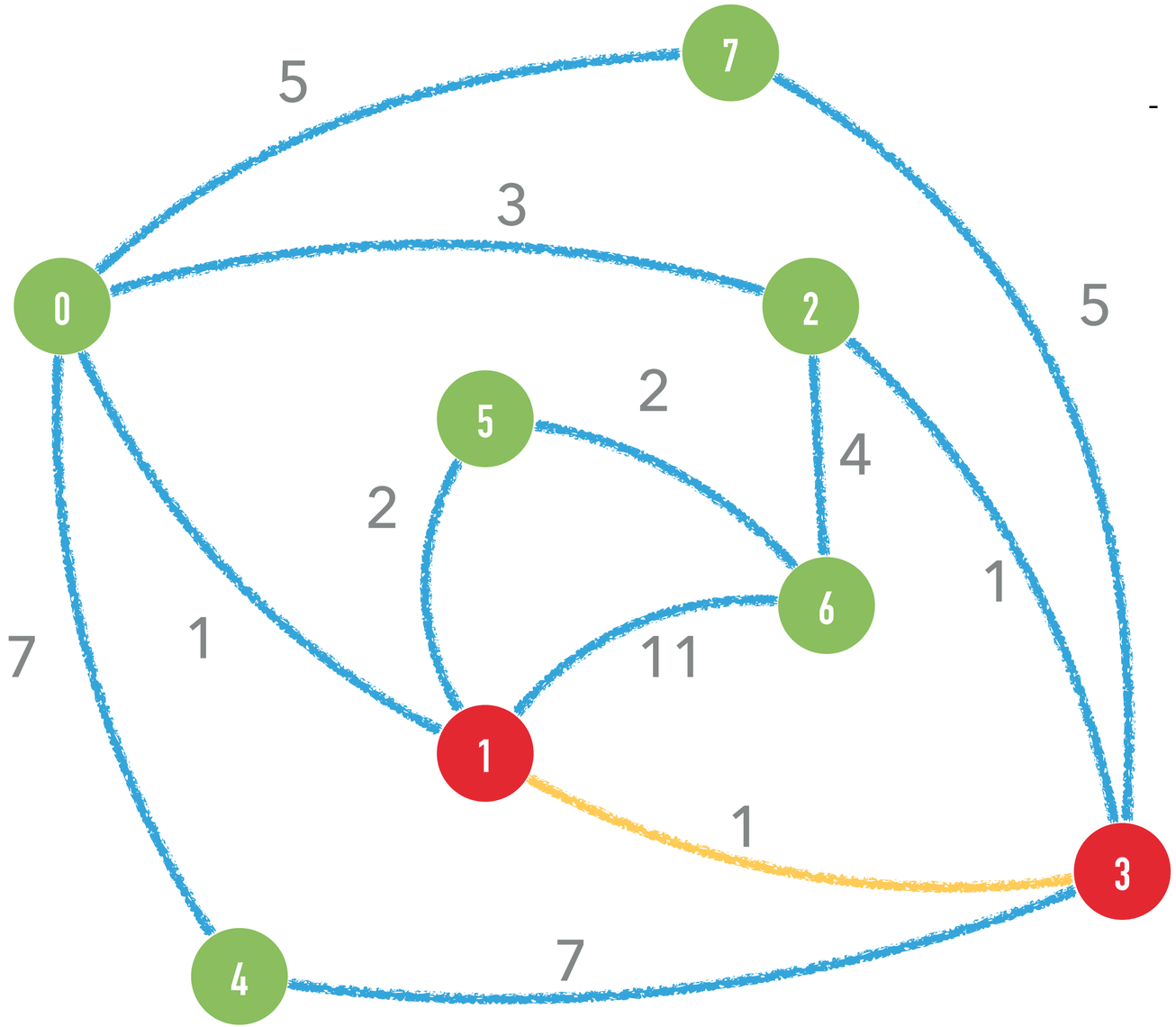
- On démarre d'un sommet quelconque
- On ajoute l'arête la plus petite adjacente aux noeuds qu'on a déjà pris qui n'ajoute pas de cycle

Question 6.1.7: MST - PRIM



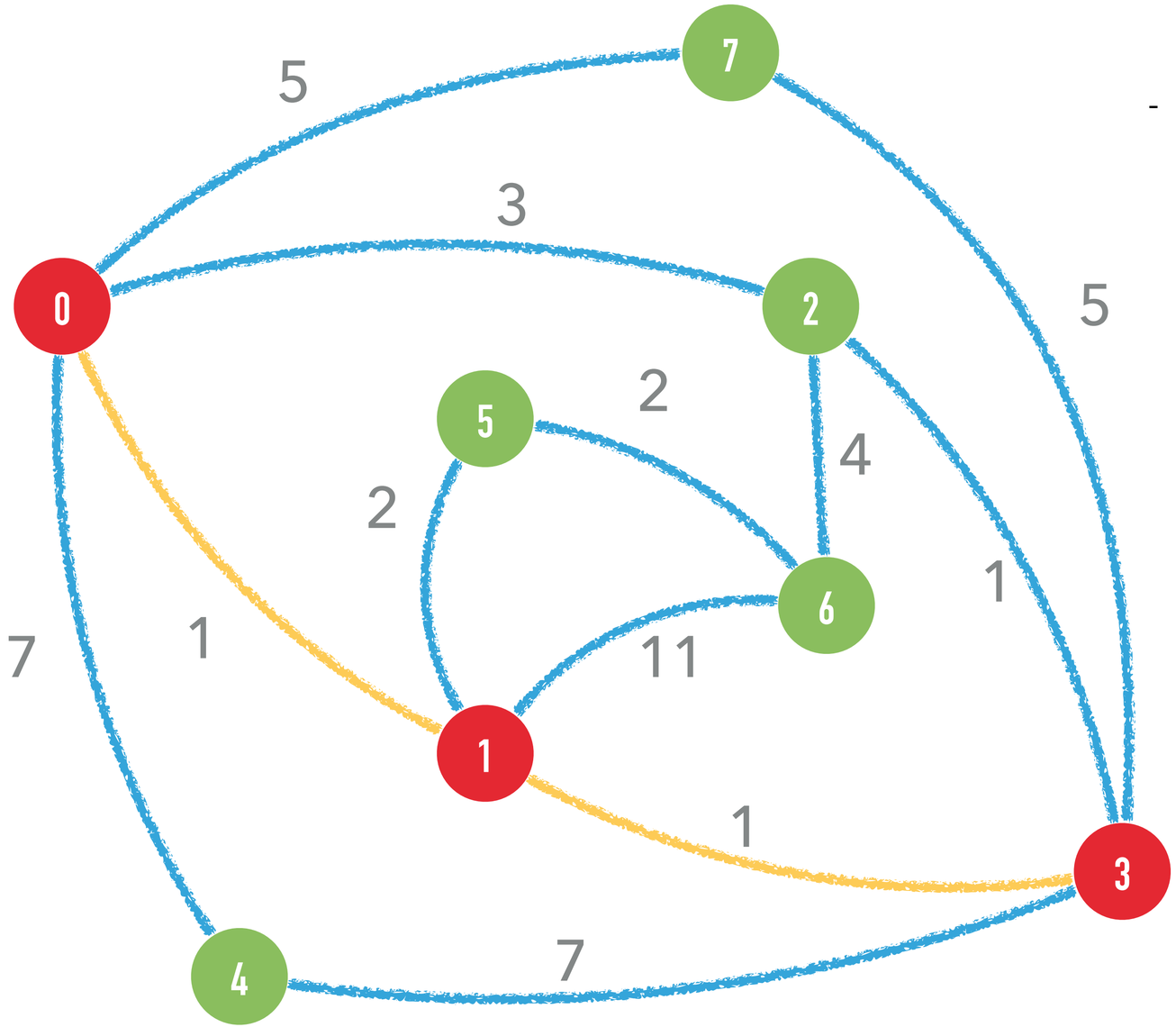
- On démarre d'un sommet quelconque
- On ajoute l'arête la plus petite adjacente aux noeuds qu'on a déjà pris qui n'ajoute pas de cycle

Question 6.1.7: MST - PRIM



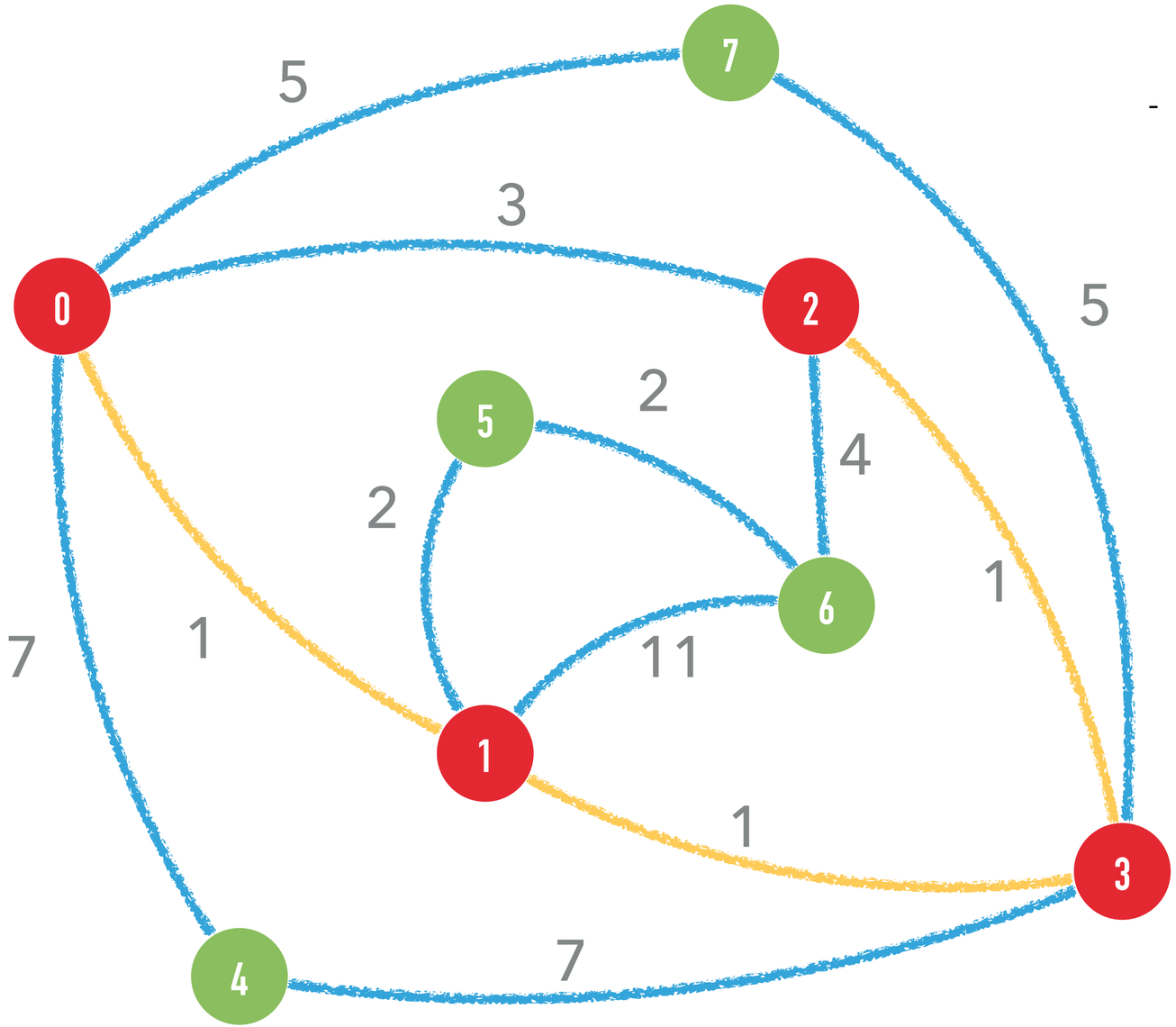
- On démarre d'un sommet quelconque
- On ajoute l'arête la plus petite adjacente aux noeuds qu'on a déjà pris qui n'ajoute pas de cycle

Question 6.1.7: MST - PRIM



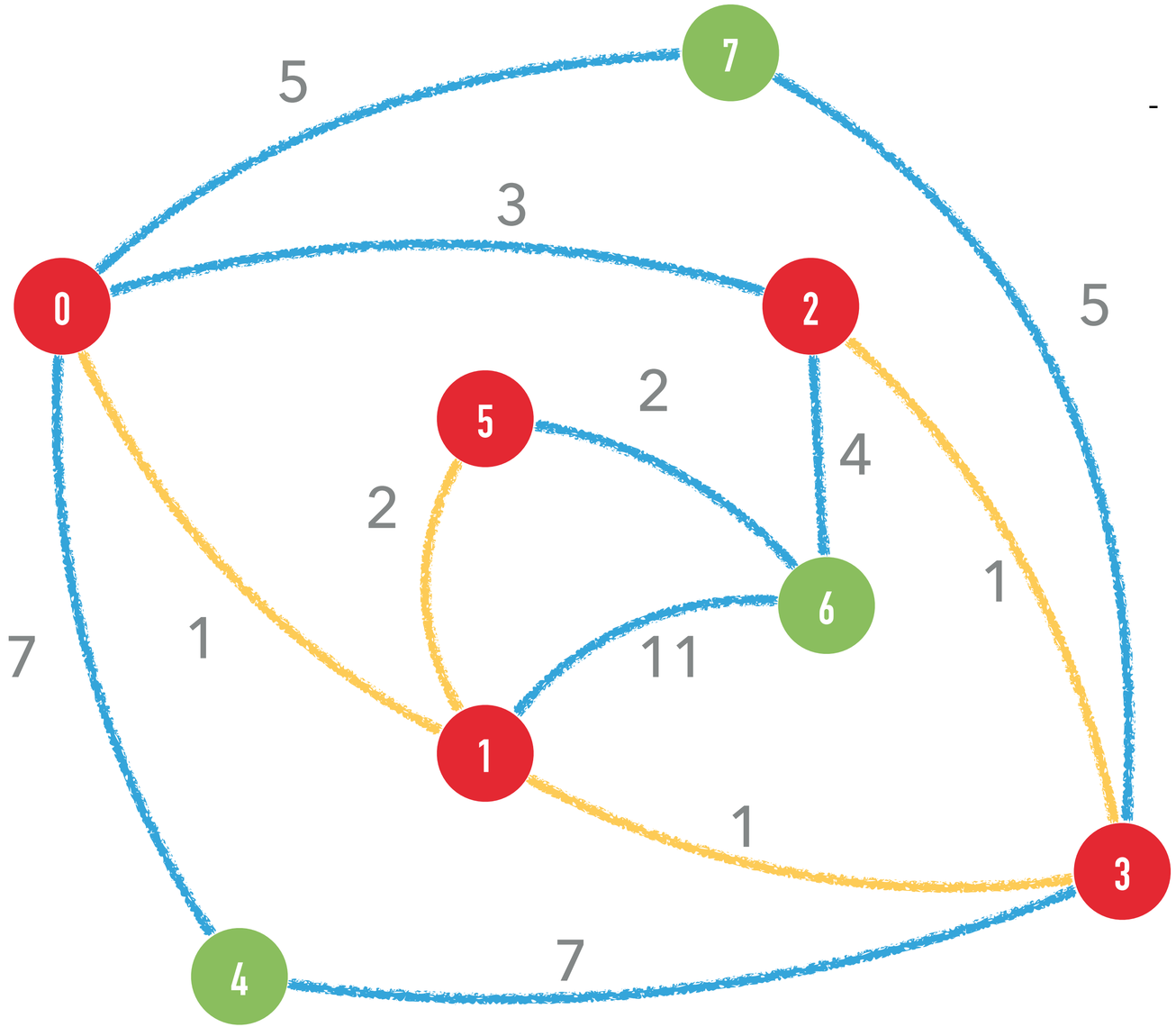
- On démarre d'un sommet quelconque
- On ajoute l'arête la plus petite adjacente aux noeuds qu'on a déjà pris qui n'ajoute pas de cycle

Question 6.1.7: MST - PRIM



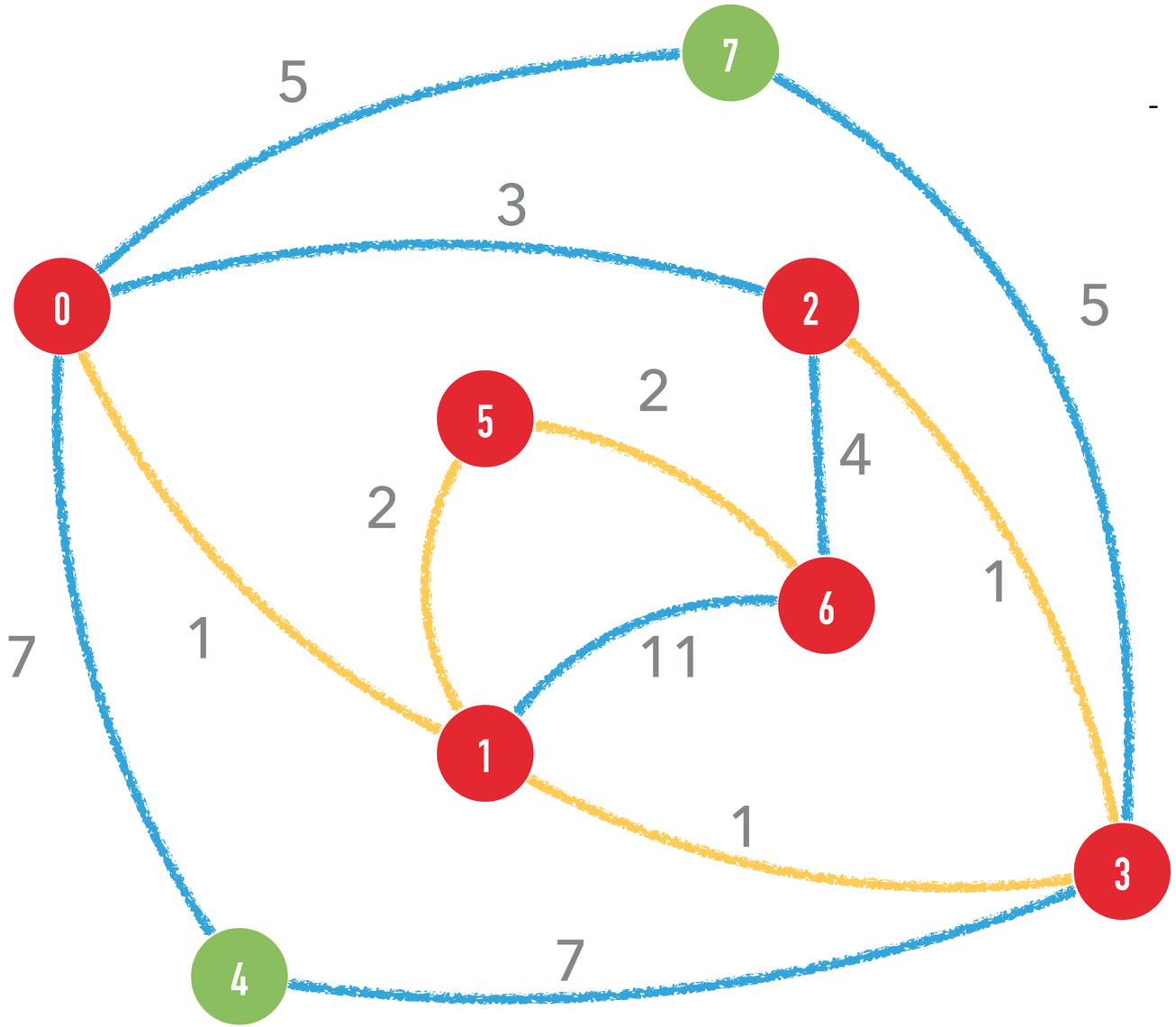
- On démarre d'un sommet quelconque
- On ajoute l'arête la plus petite adjacente aux noeuds qu'on a déjà pris qui n'ajoute pas de cycle

Question 6.1.7: MST - PRIM



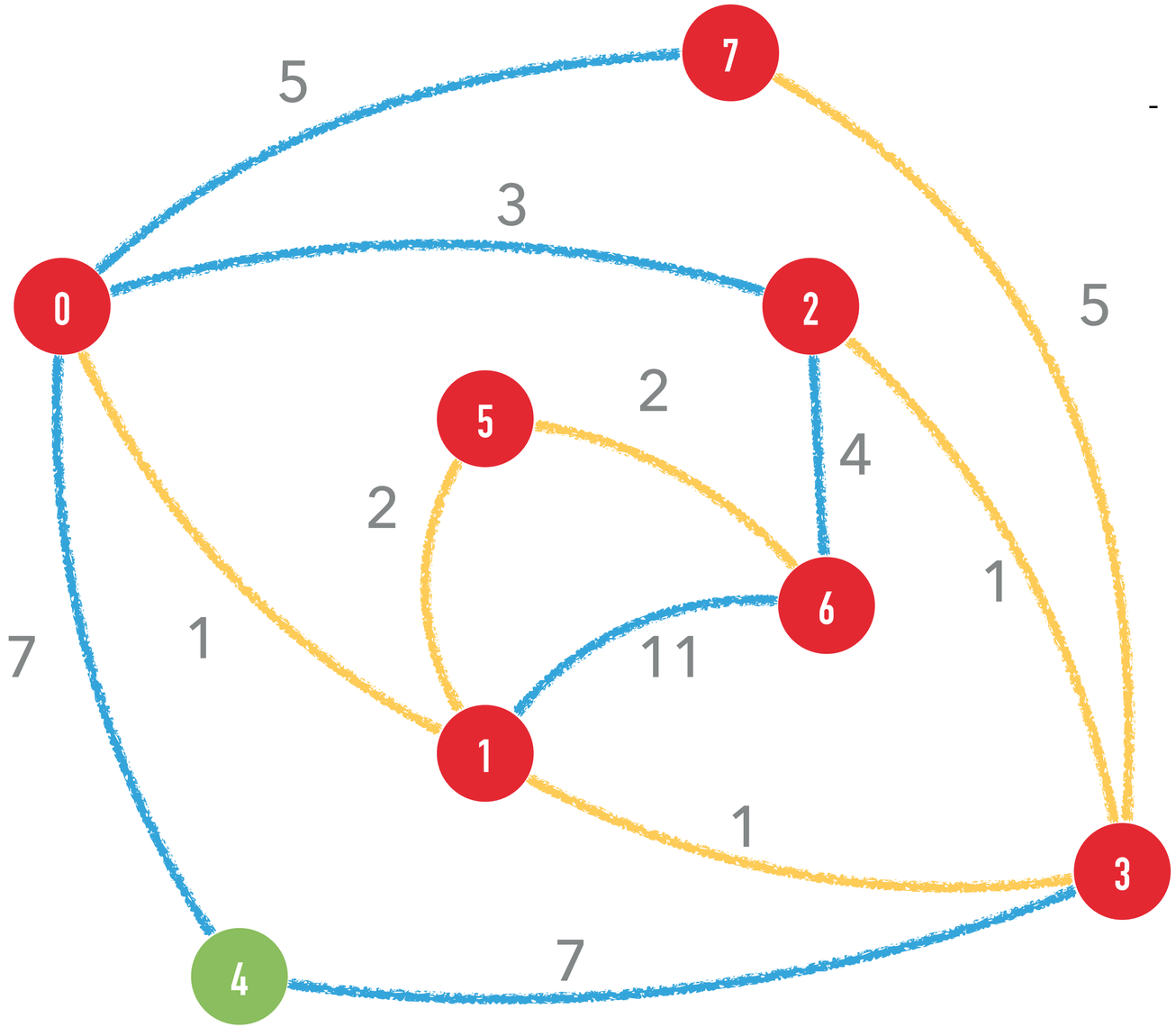
- On démarre d'un sommet quelconque
- On ajoute l'arête la plus petite adjacente aux noeuds qu'on a déjà pris qui n'ajoute pas de cycle

Question 6.1.7: MST - PRIM



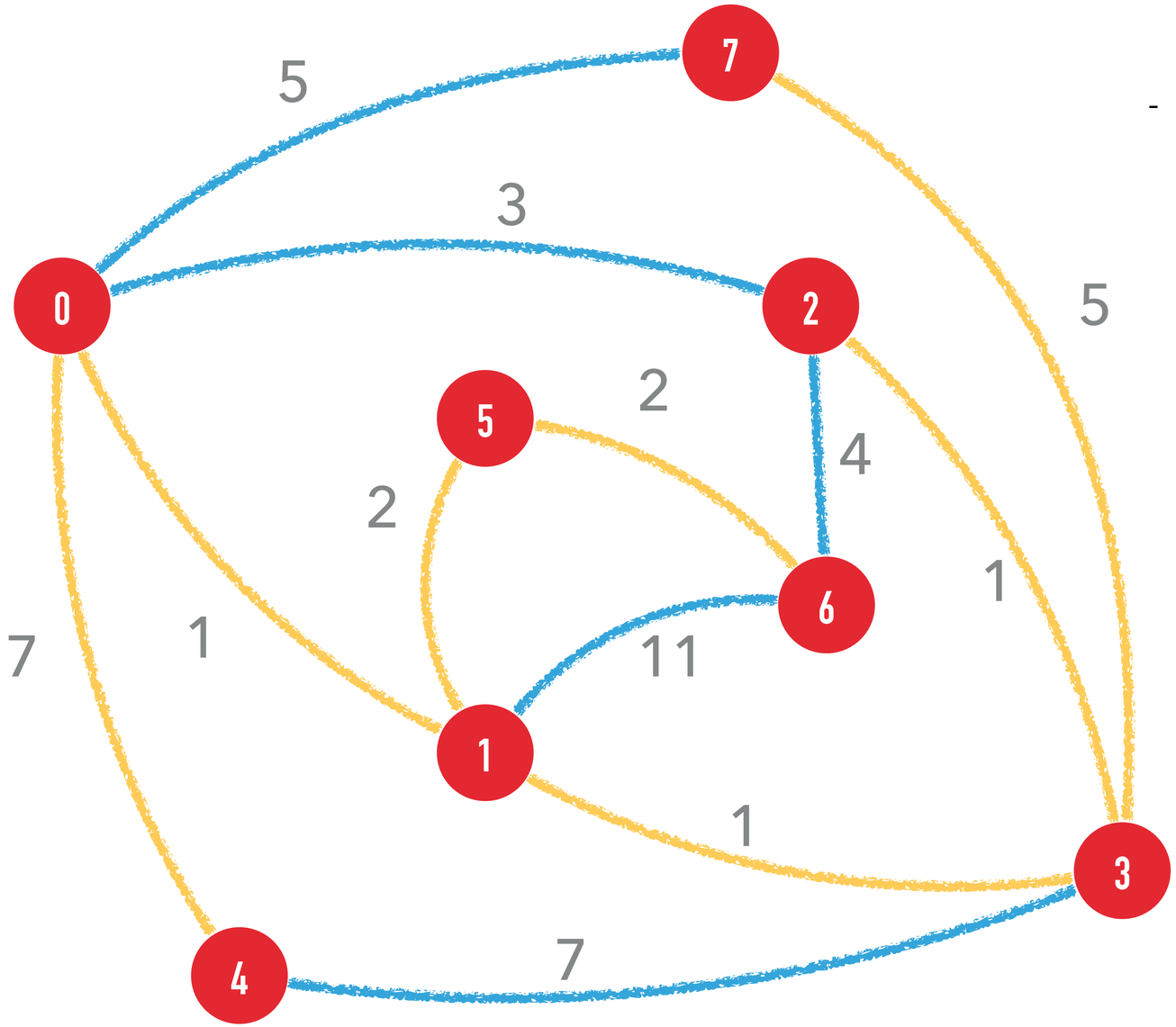
- On démarre d'un sommet quelconque
- On ajoute l'arête la plus petite adjacente aux noeuds qu'on a déjà pris qui n'ajoute pas de cycle

Question 6.1.7: MST - PRIM



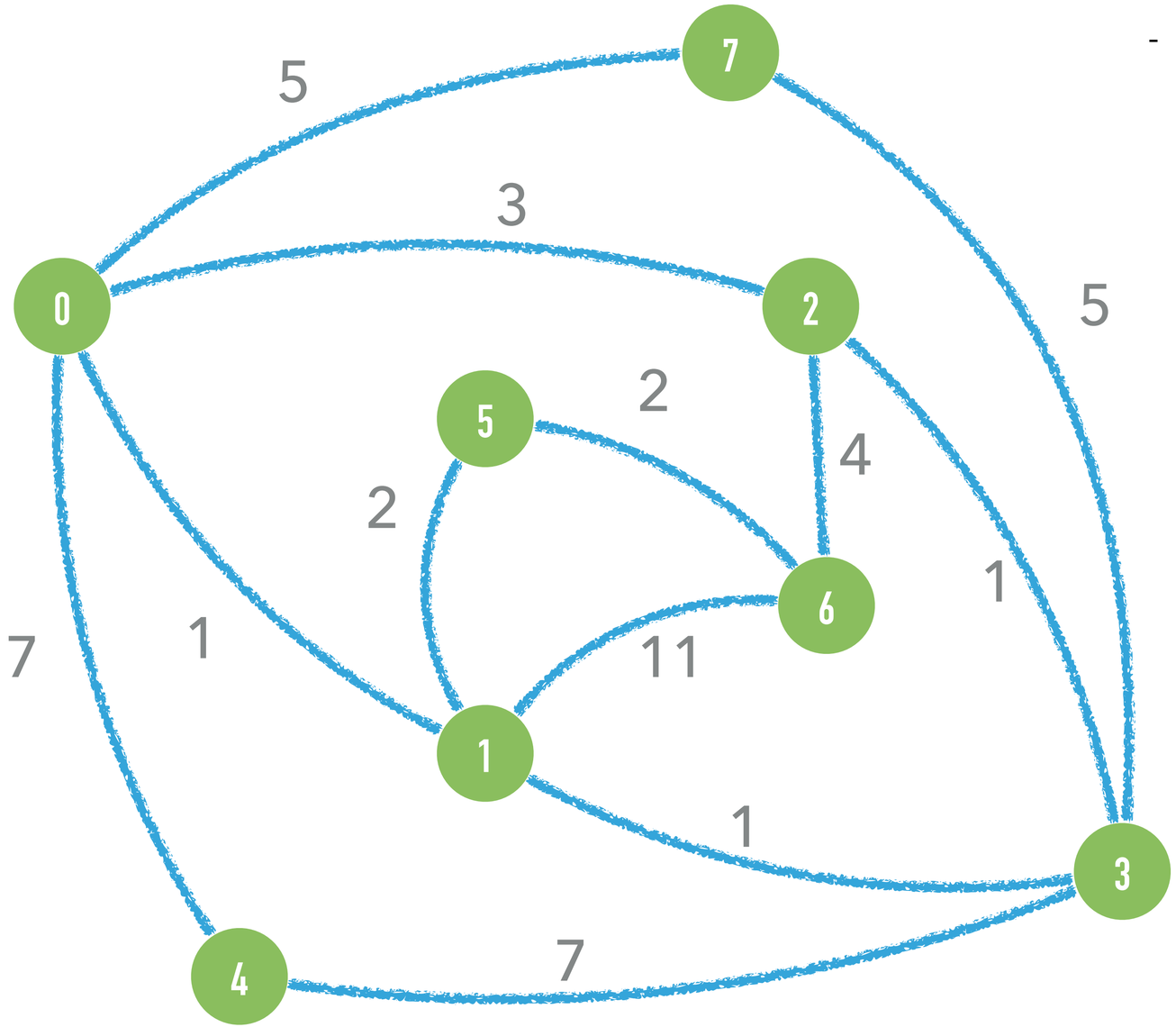
- On démarre d'un sommet quelconque
- On ajoute l'arête la plus petite adjacente aux noeuds qu'on a déjà pris qui n'ajoute pas de cycle

Question 6.1.7: MST - PRIM



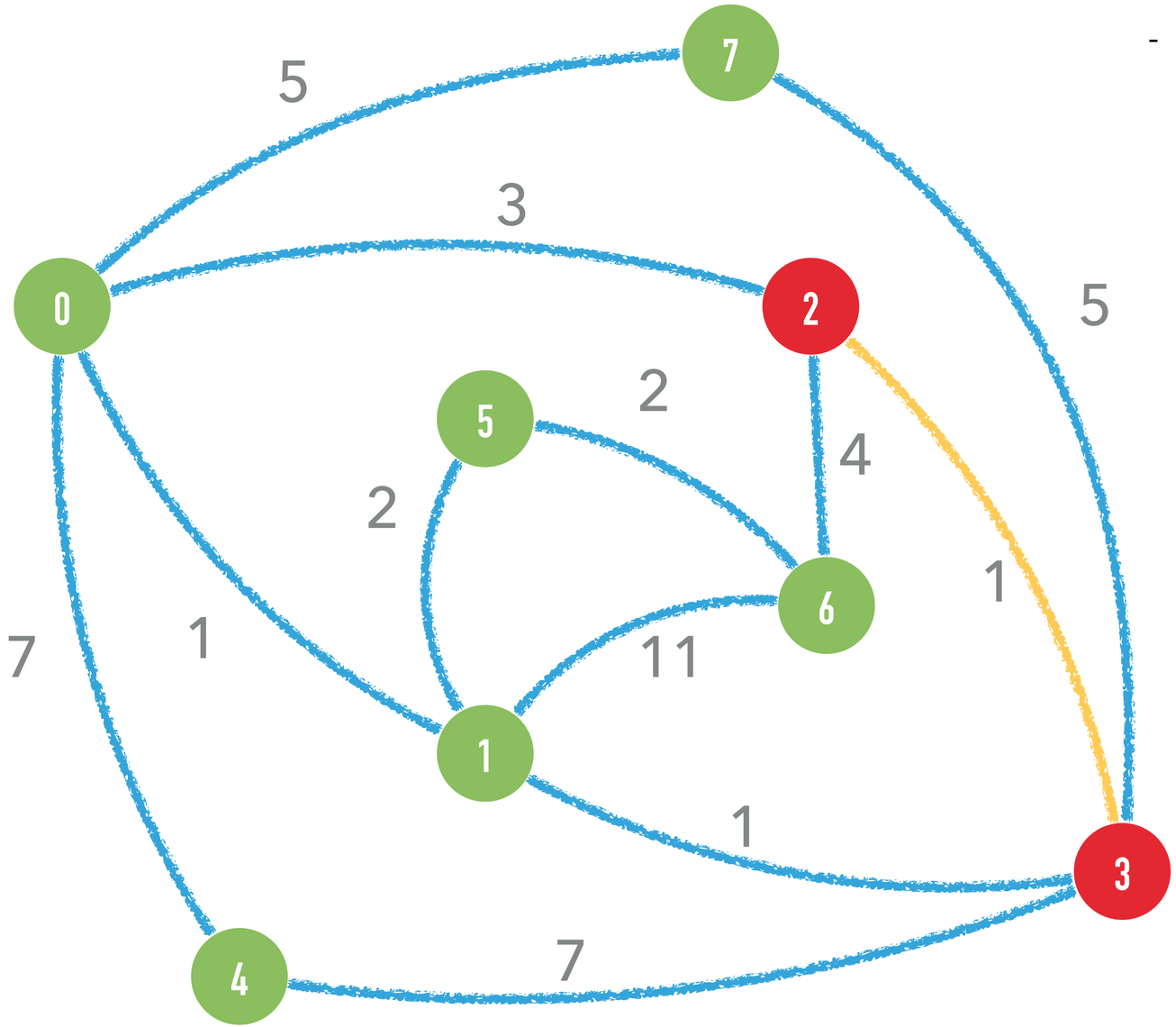
- On démarre d'un sommet quelconque
- On ajoute l'arête la plus petite adjacente aux noeuds qu'on a déjà pris qui n'ajoute pas de cycle

Question 6.1.7: MST - KRUSKAL



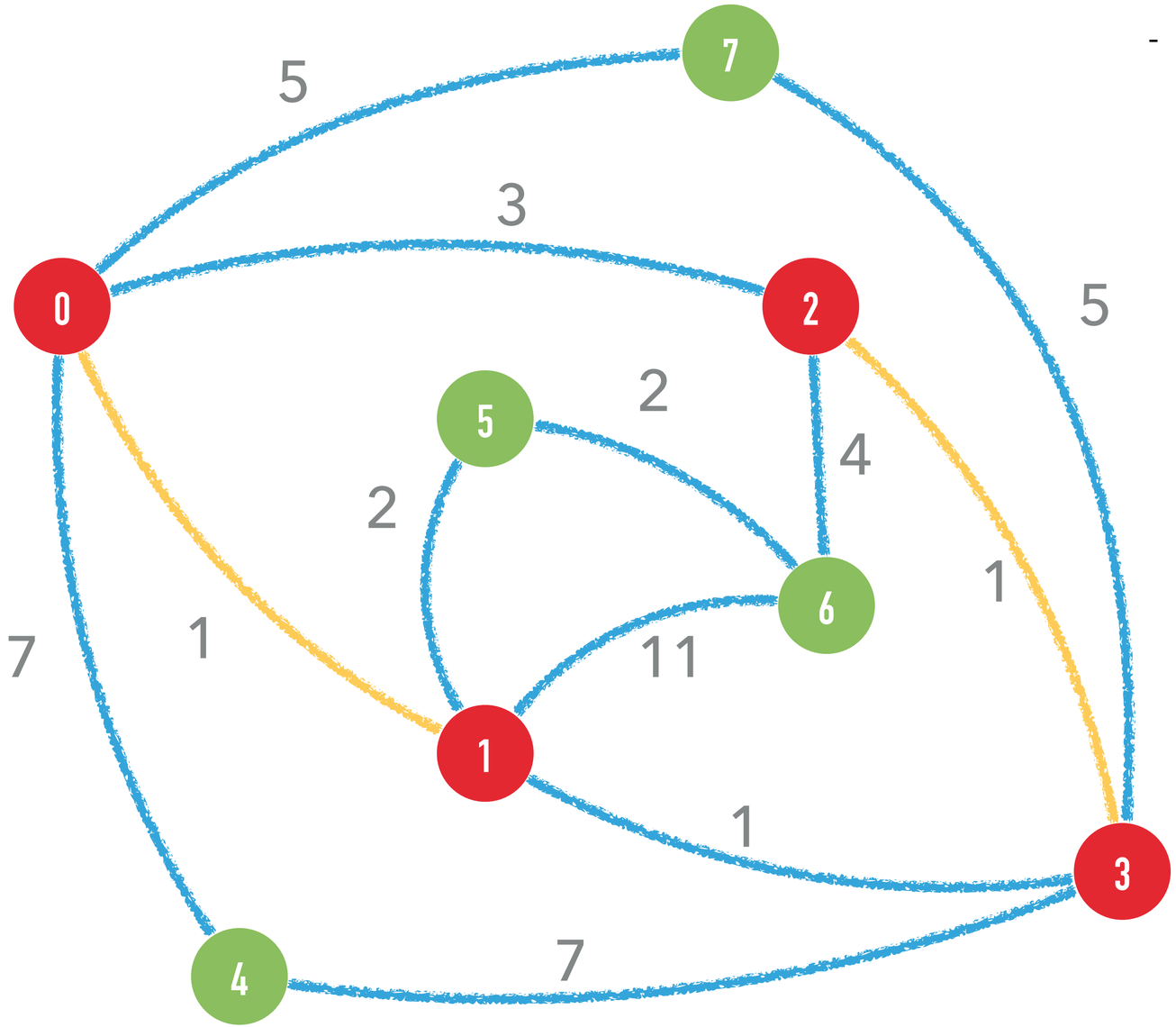
- On ajoute toujours l'arête la plus petite (de tout le graphe) si elle ne fait pas de cycle

Question 6.1.7: MST - KRUSKAL



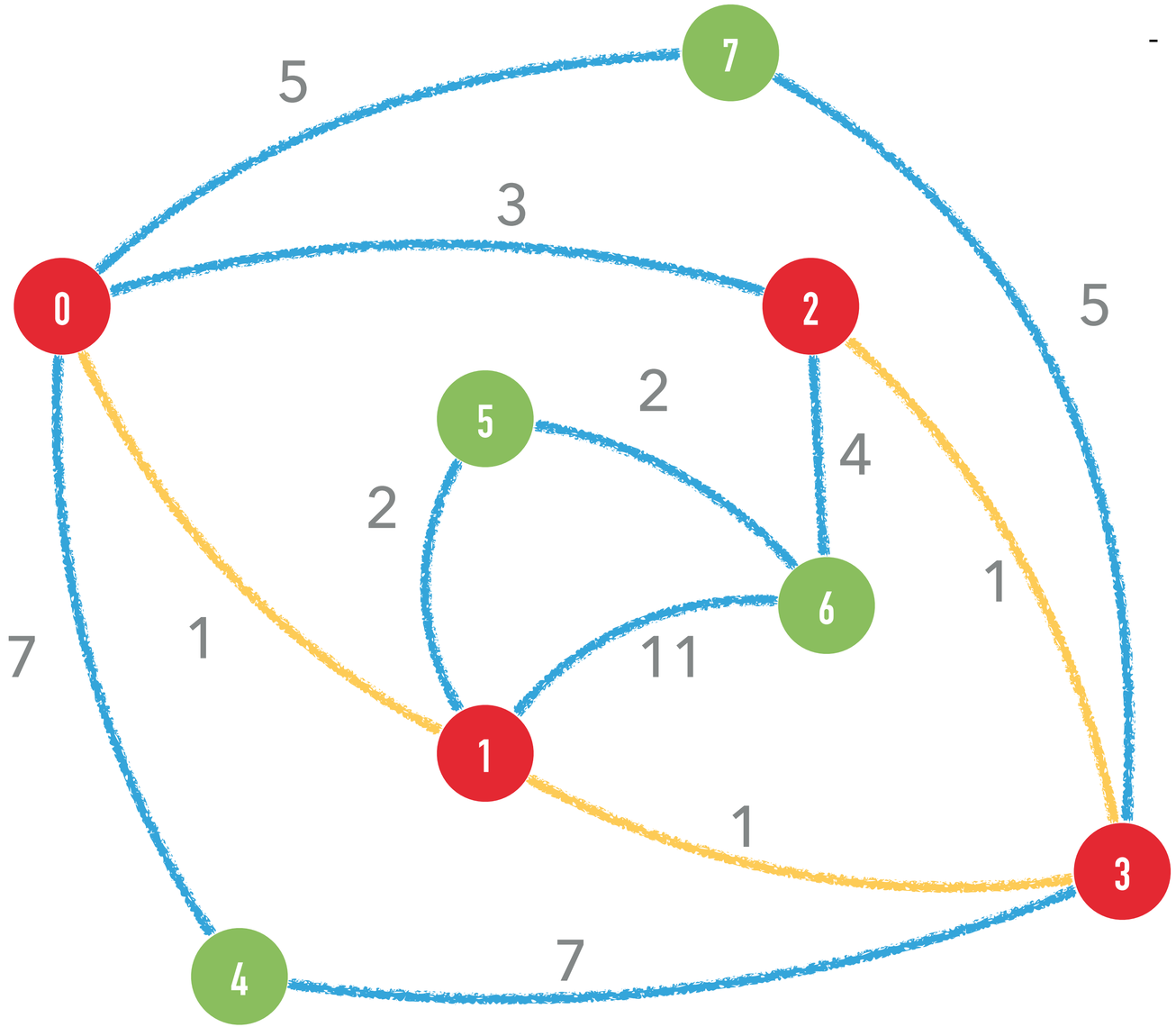
- On ajoute toujours l'arête la plus petite (de tout le graphe) si elle ne fait pas de cycle

Question 6.1.7: MST - KRUSKAL



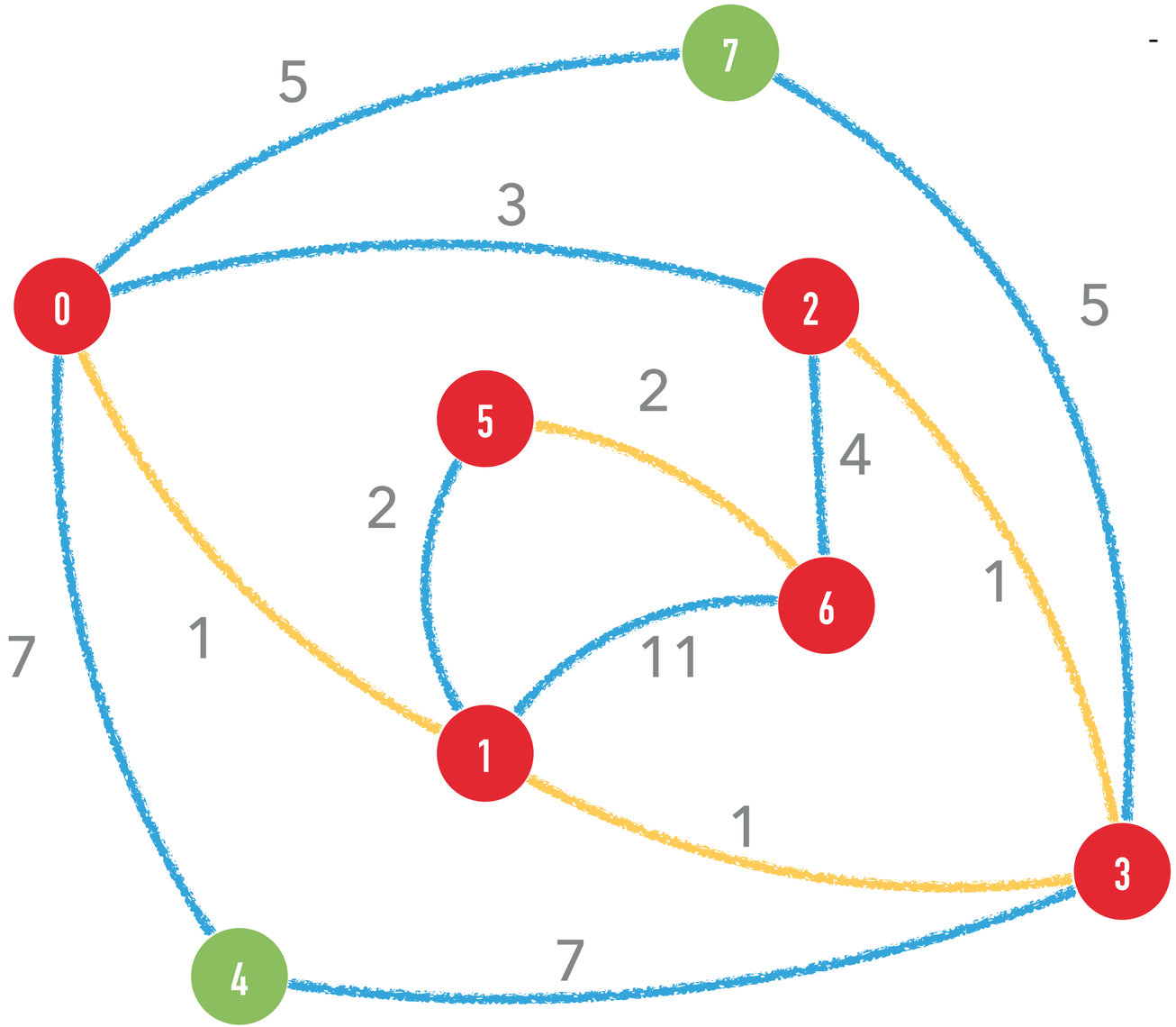
- On ajoute toujours l'arête la plus petite (de tout le graphe) si elle ne fait pas de cycle

Question 6.1.7: MST - KRUSKAL



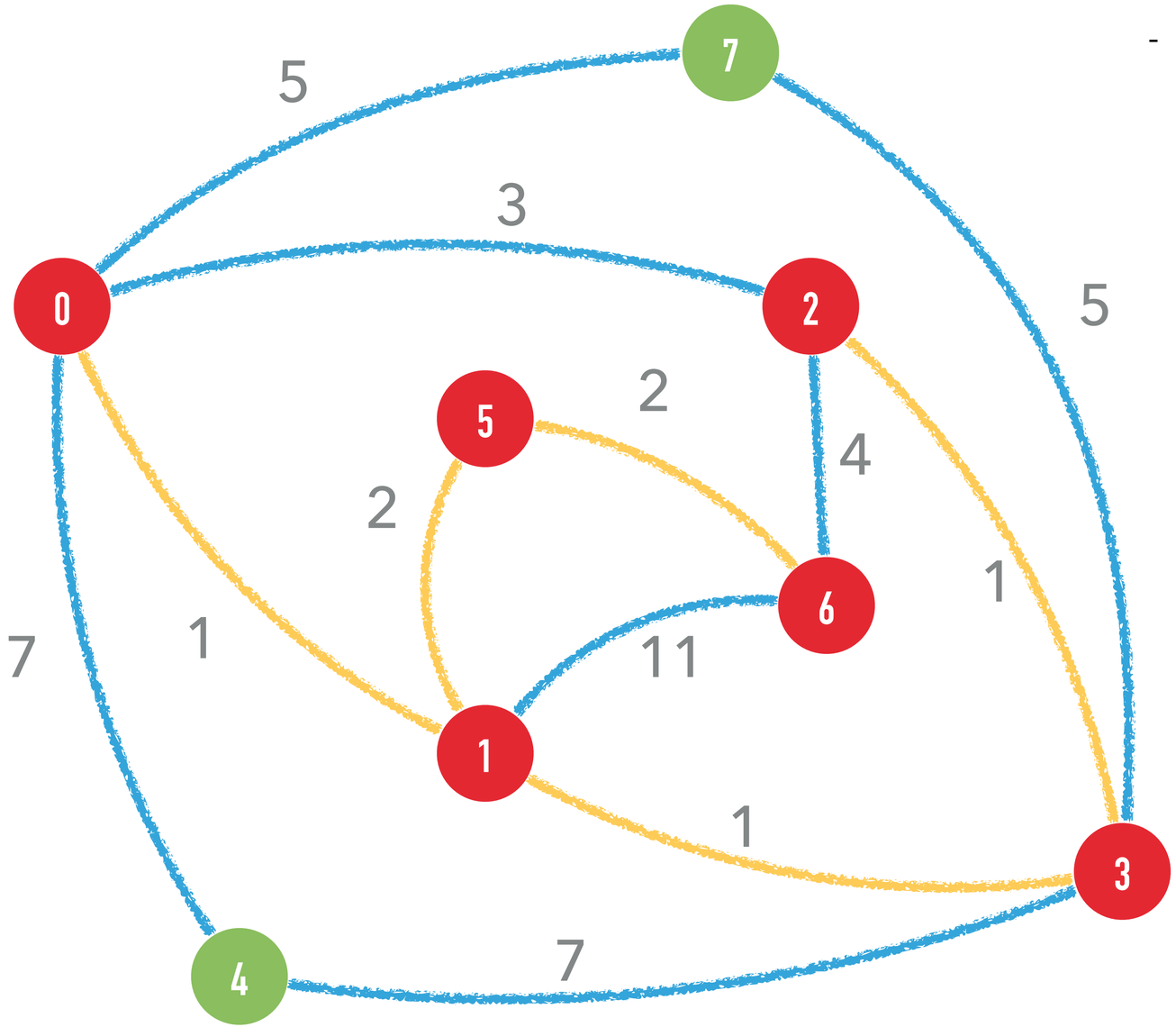
- On ajoute toujours l'arête la plus petite (de tout le graphe) si elle ne fait pas de cycle

Question 6.1.7: MST - KRUSKAL



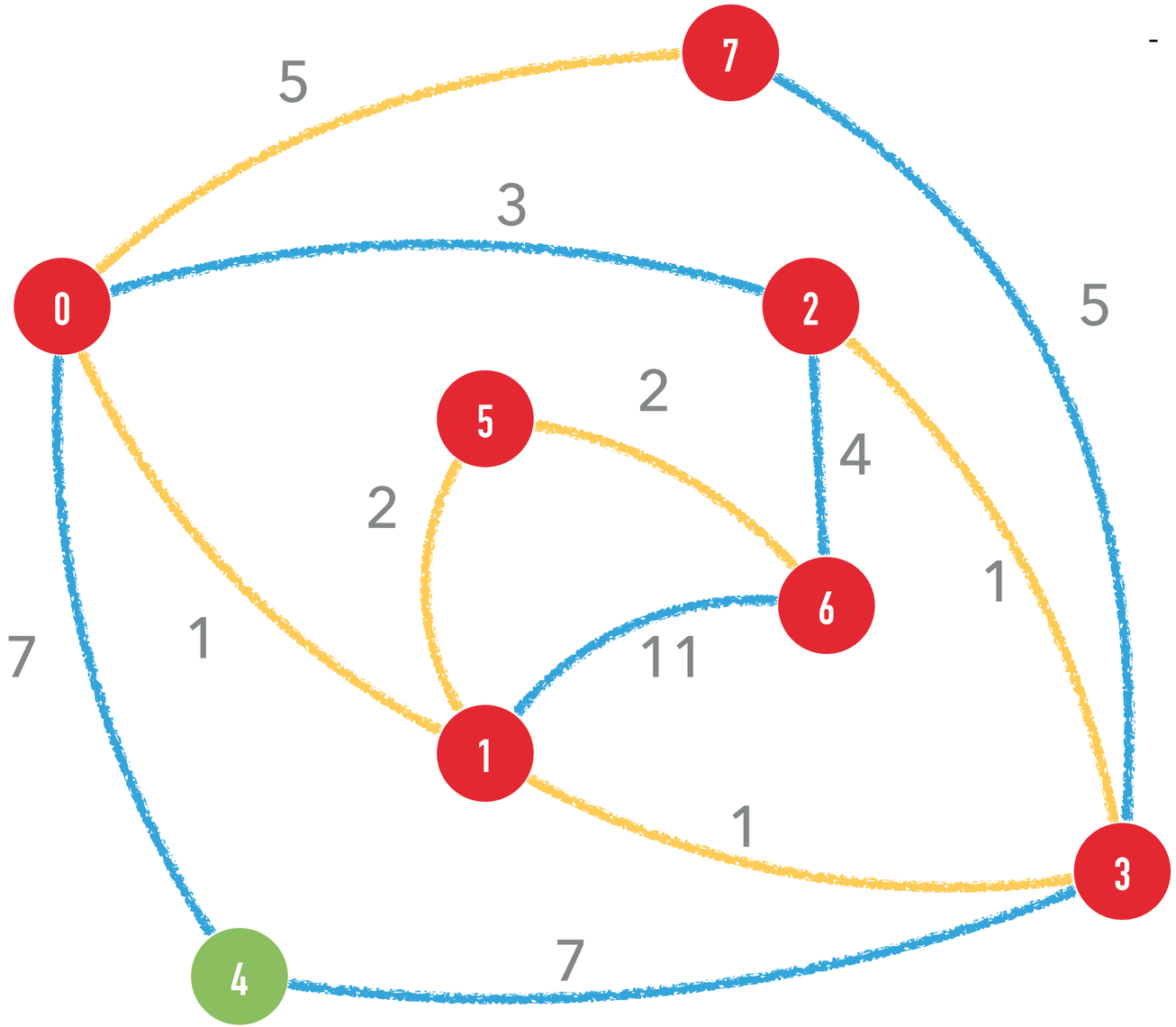
- On ajoute toujours l'arête la plus petite (de tout le graphe) si elle ne fait pas de cycle

Question 6.1.7: MST - KRUSKAL



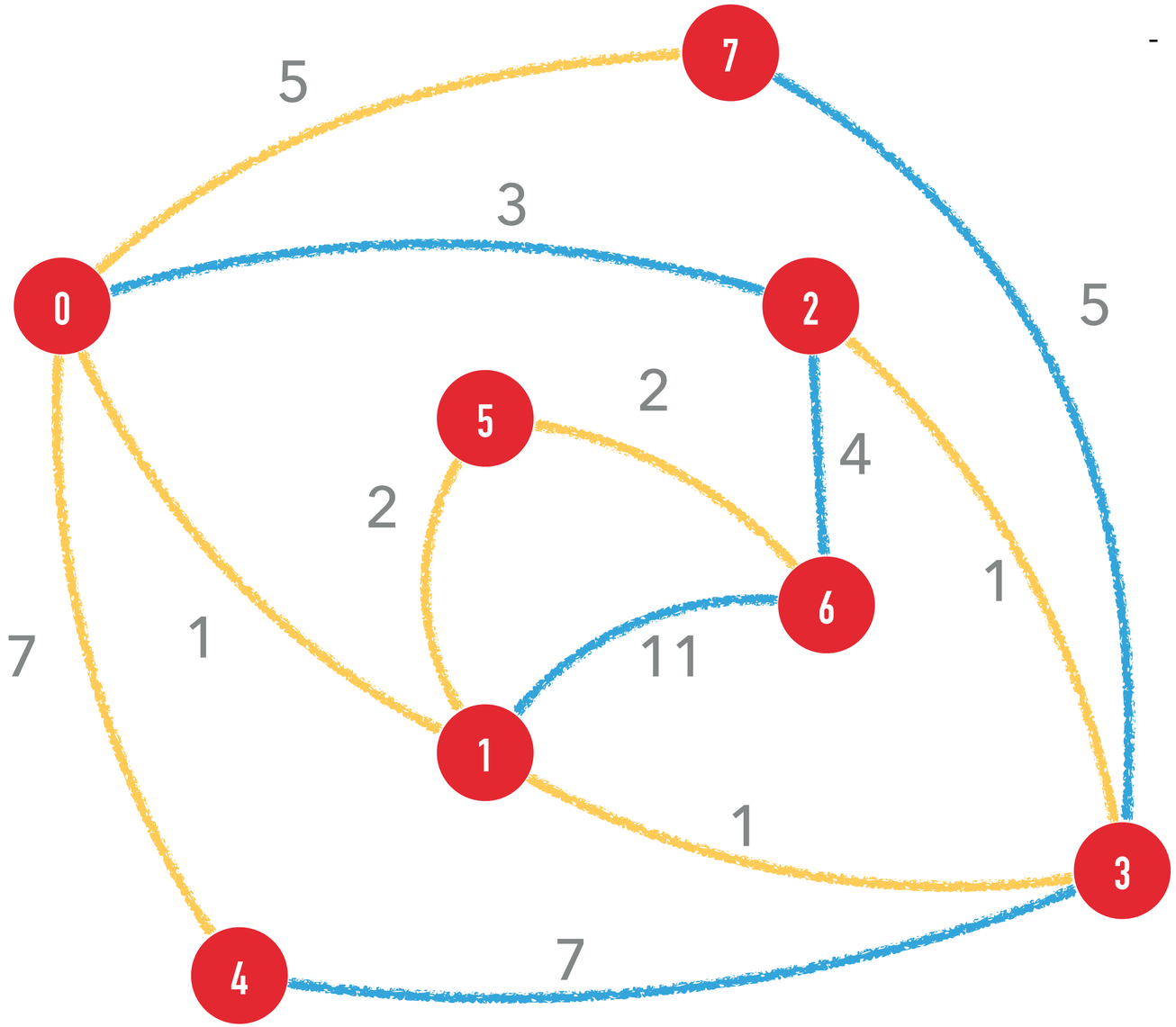
- On ajoute toujours l'arête la plus petite (de tout le graphe) si elle ne fait pas de cycle

Question 6.1.7: MST - KRUSKAL



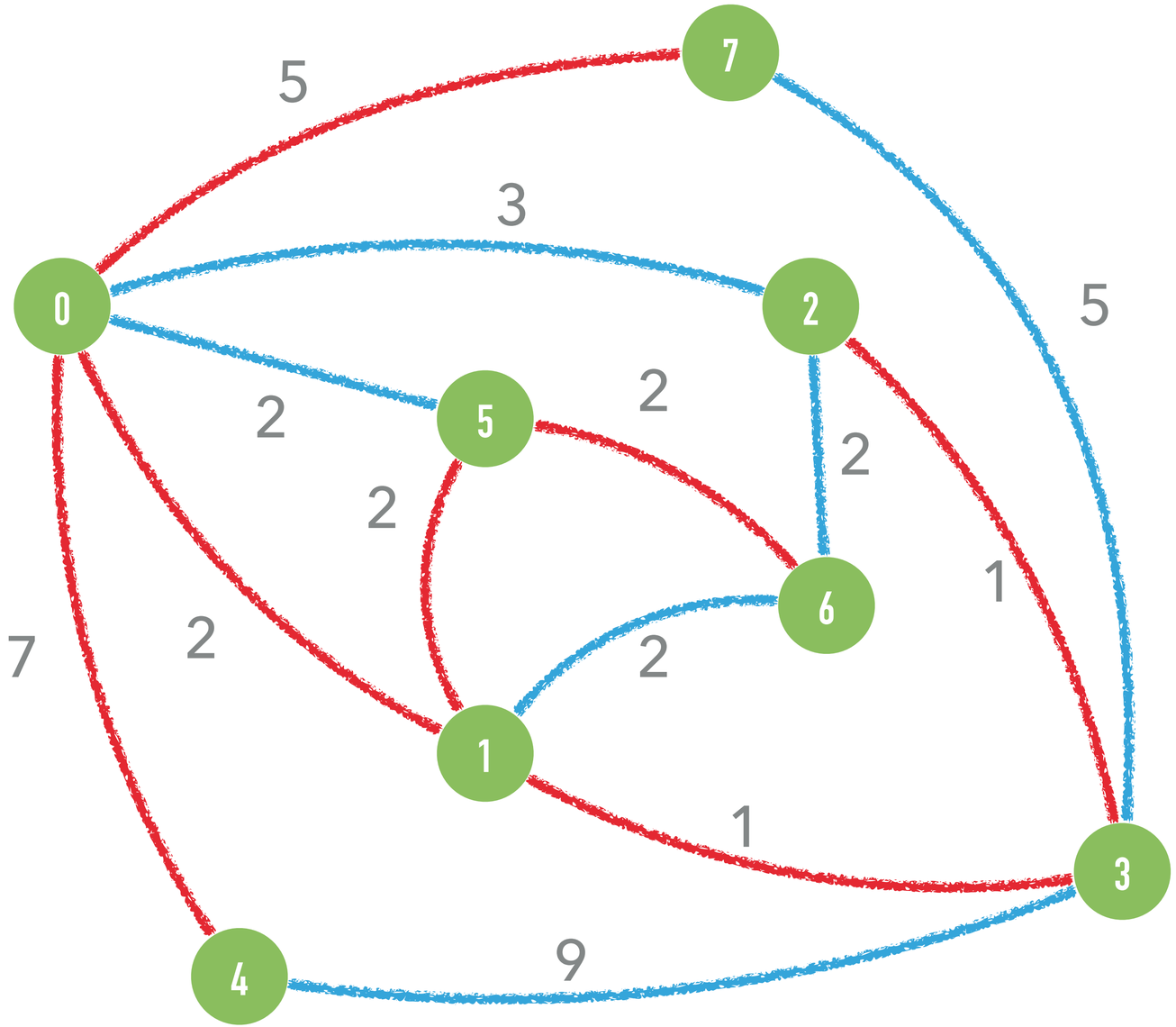
- On ajoute toujours l'arête la plus petite (de tout le graphe) si elle ne fait pas de cycle

Question 6.1.7: MST - KRUSKAL

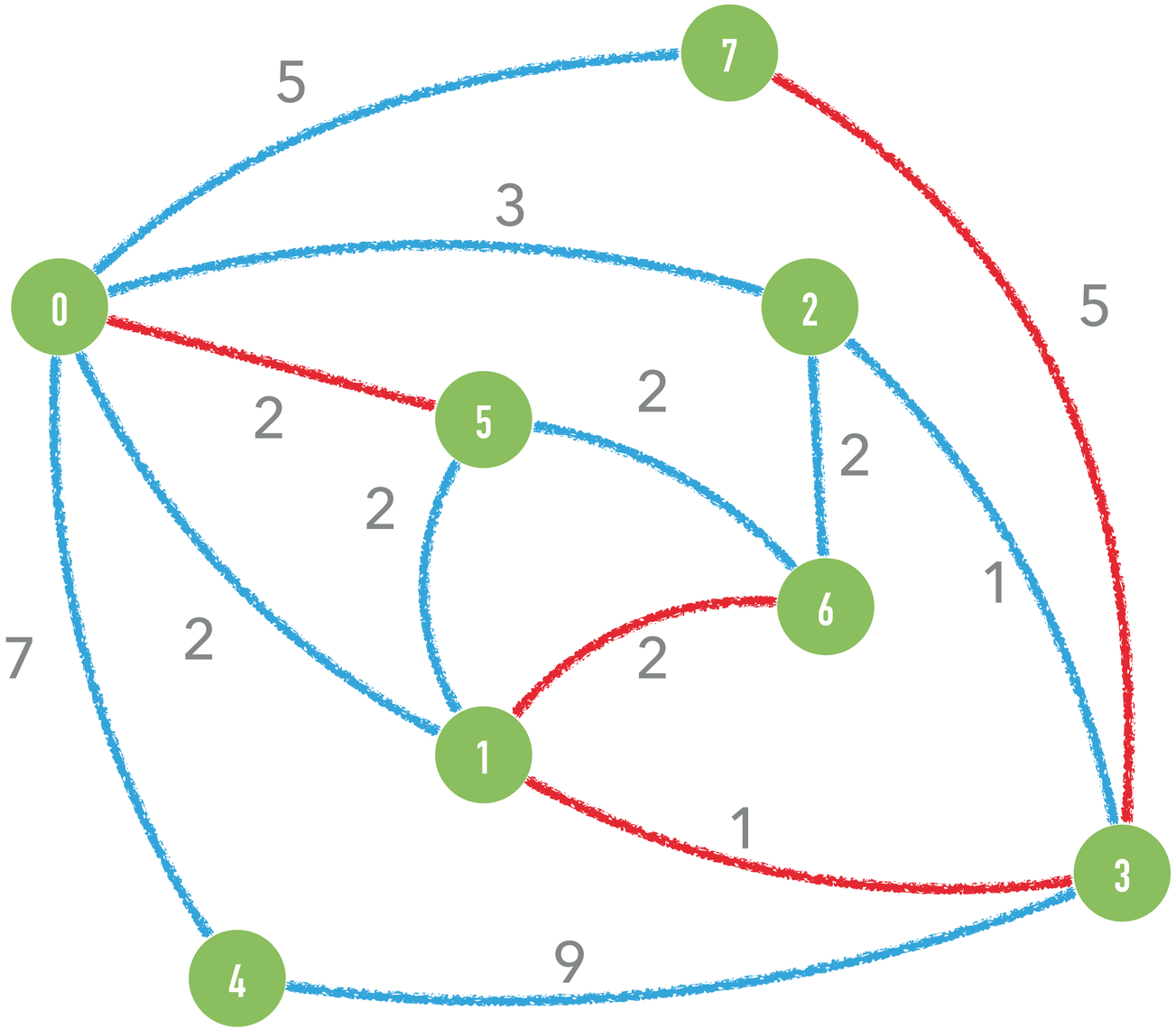


- On ajoute toujours l'arête la plus petite (de tout le graphe) si elle ne fait pas de cycle

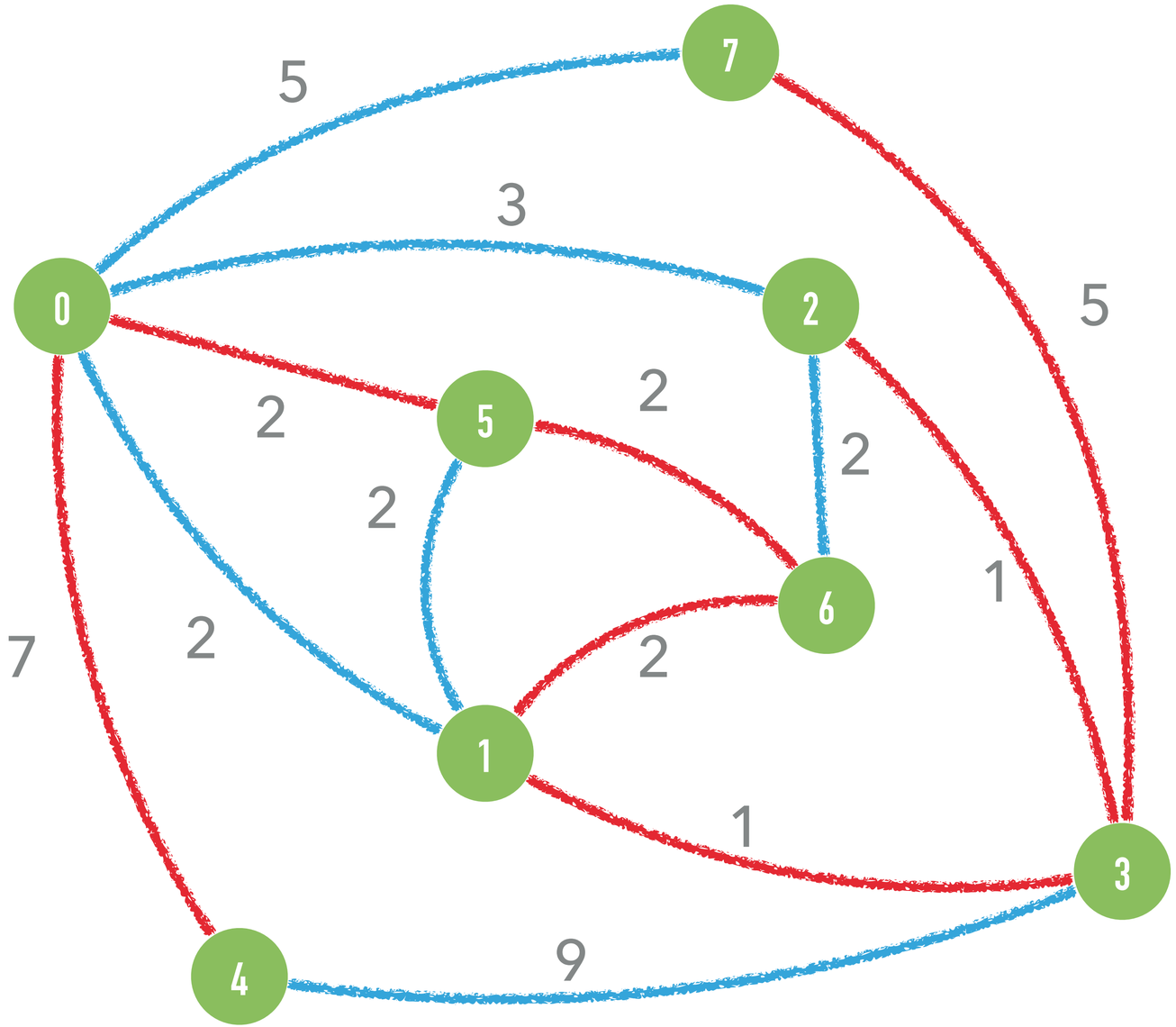
Question 6.1.7: MST - partir d'une partie de MST



Question 6.1.7: MST



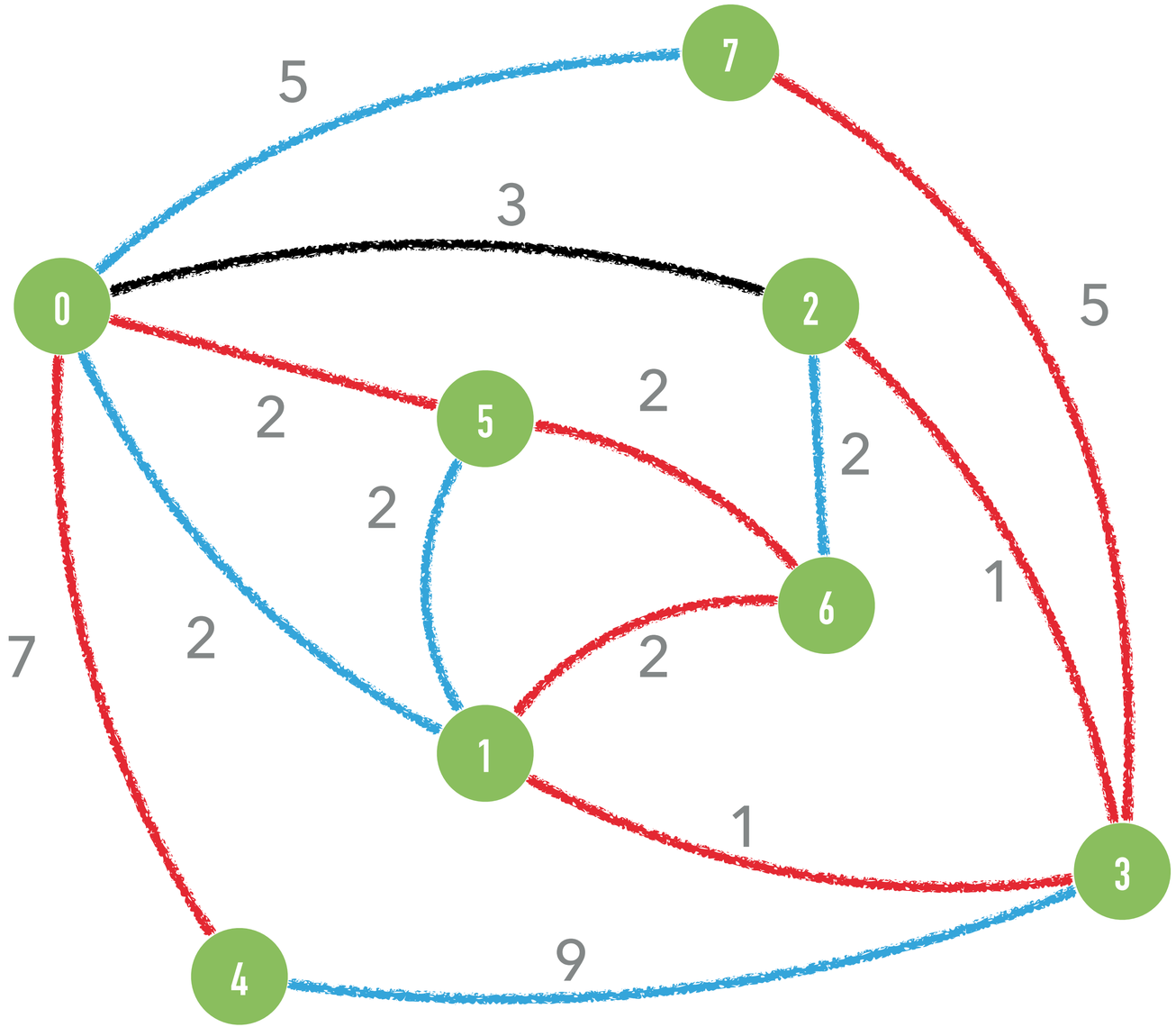
Question 6.1.7: MST



Les algorithmes de MST sont greedy:

Ils prennent toujours localement une décision optimale.

Question 6.1.8: MST AVEC ARETES SPECIFIEE



Comment insérer l'arête de poids 3?

Ajouter une arête ajoute un et un seul cycle

On trouve le cycle, retire l'arête la plus grande.

Question 6.1.9: DIJKSTRA

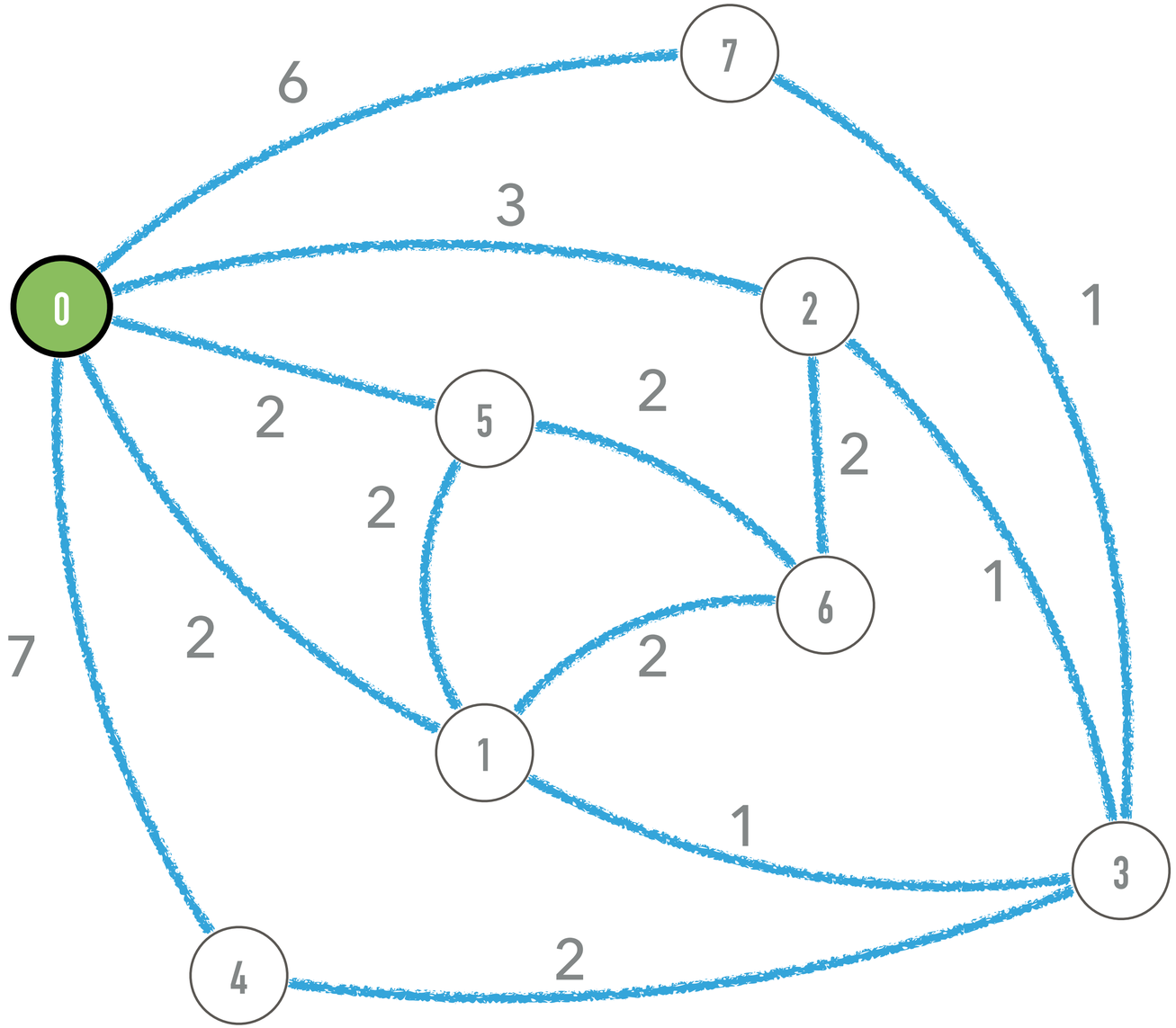


TABLEAU DES DISTANCE

Noeud	Distance
0	0
1	
2	
3	
4	
5	
6	
7	

PRIORITY QUEUE (TODO)

Noeud	Distance
0	0

Question 6.1.9: DIJKSTRA

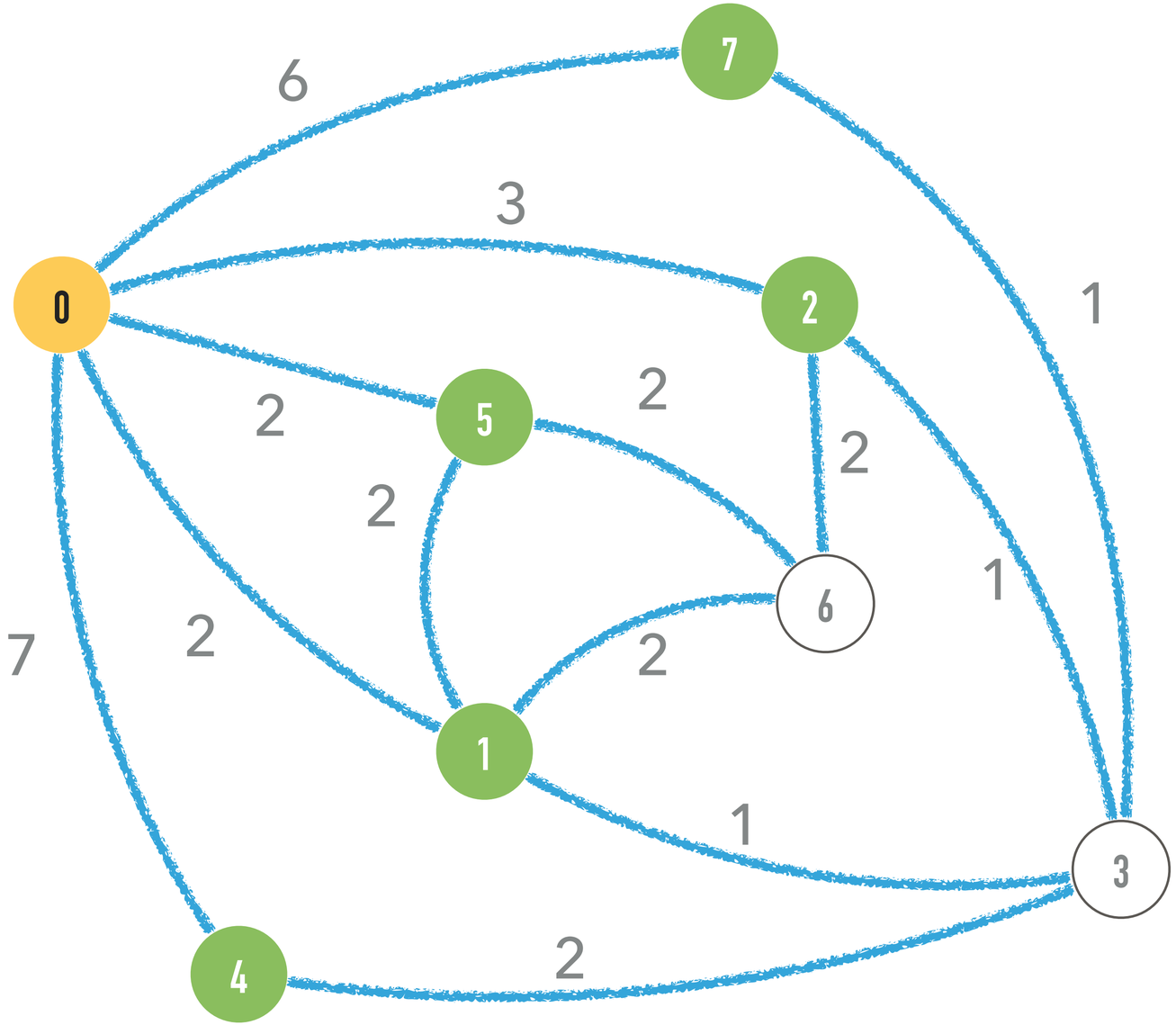


TABLEAU DES DISTANCE

Noeud	Distance
0	0
1	2
2	3
3	
4	7
5	2
6	
7	6

PRIORITY QUEUE (TODO)

Noeud	Distance
1	2
5	2
2	3
7	6
4	7

Question 6.1.9: DIJKSTRA

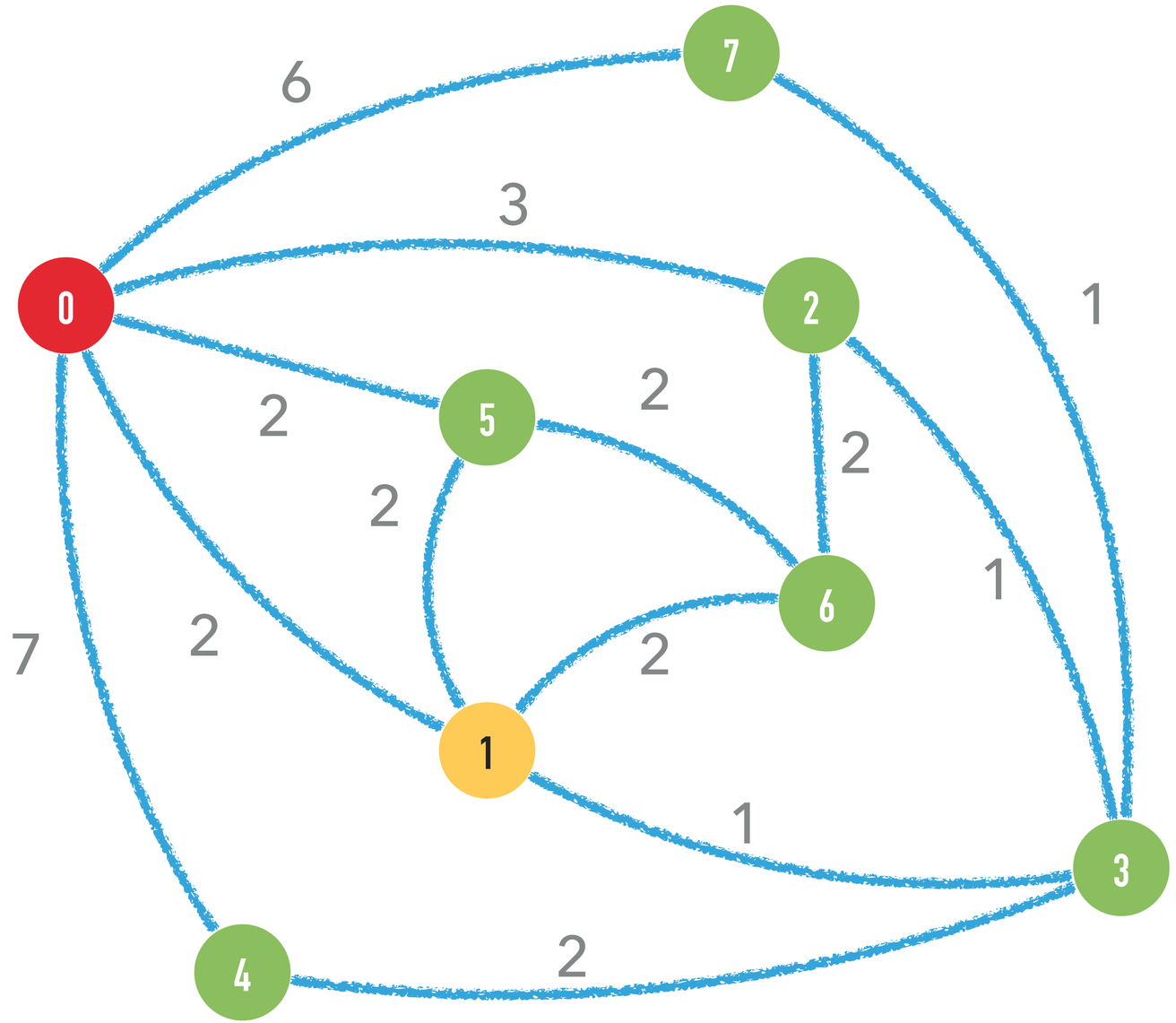


TABLEAU DES DISTANCE

Noeud	Distance
0	0
1	2
2	3
3	3
4	7
5	2
6	4
7	6

PRIORITY QUEUE (TODO)

Noeud	Distance
5	2
2	3
3	3
6	4
7	6
4	7

Question 6.1.9: DIJKSTRA

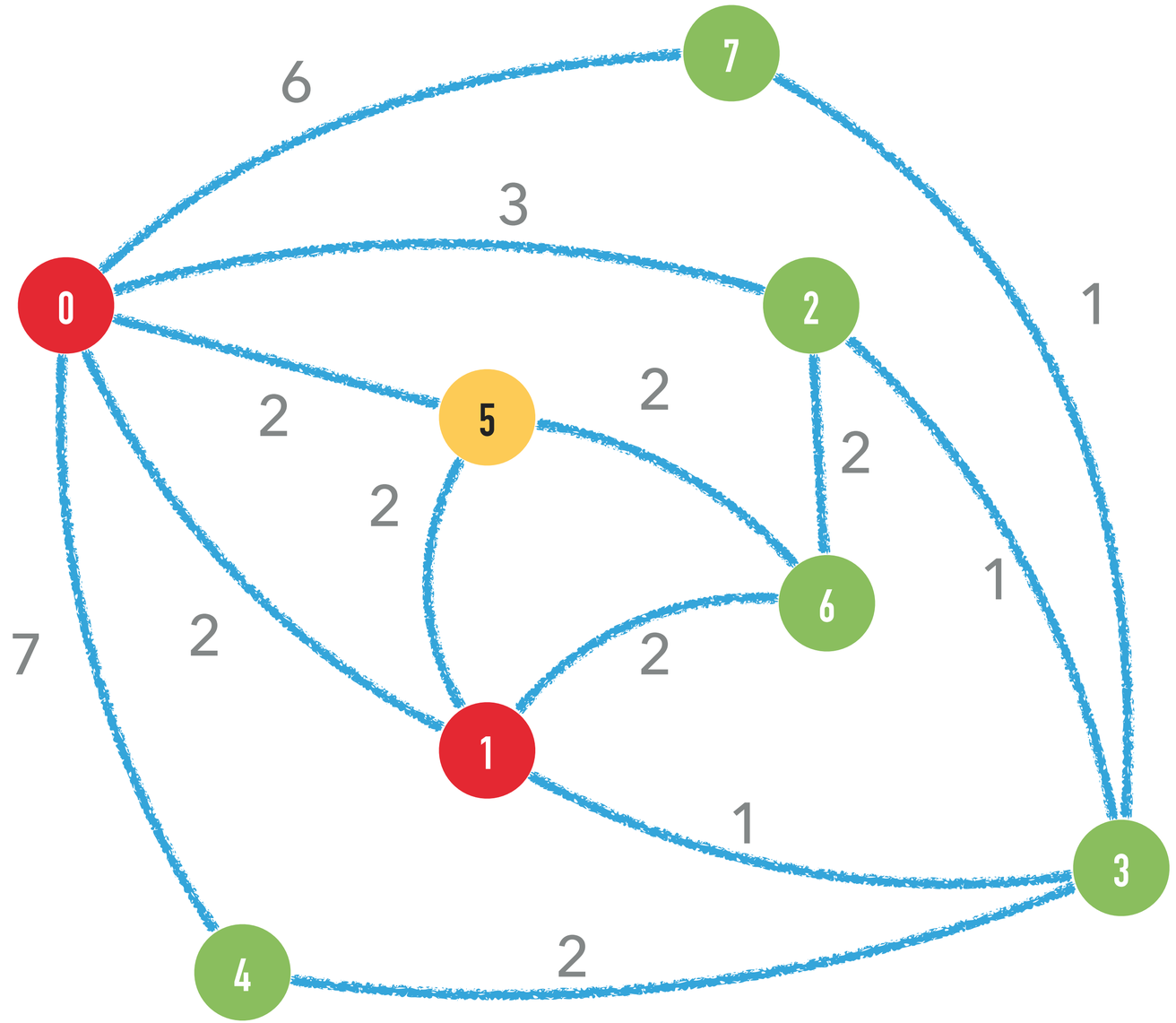


TABLEAU DES DISTANCE

Noeud	Distance
0	0
1	2
2	3
3	3
4	7
5	2
6	4
7	6

PRIORITY QUEUE (TODO)

Noeud	Distance
2	3
3	3
6	4
7	6
4	7

Question 6.1.9: DIJKSTRA

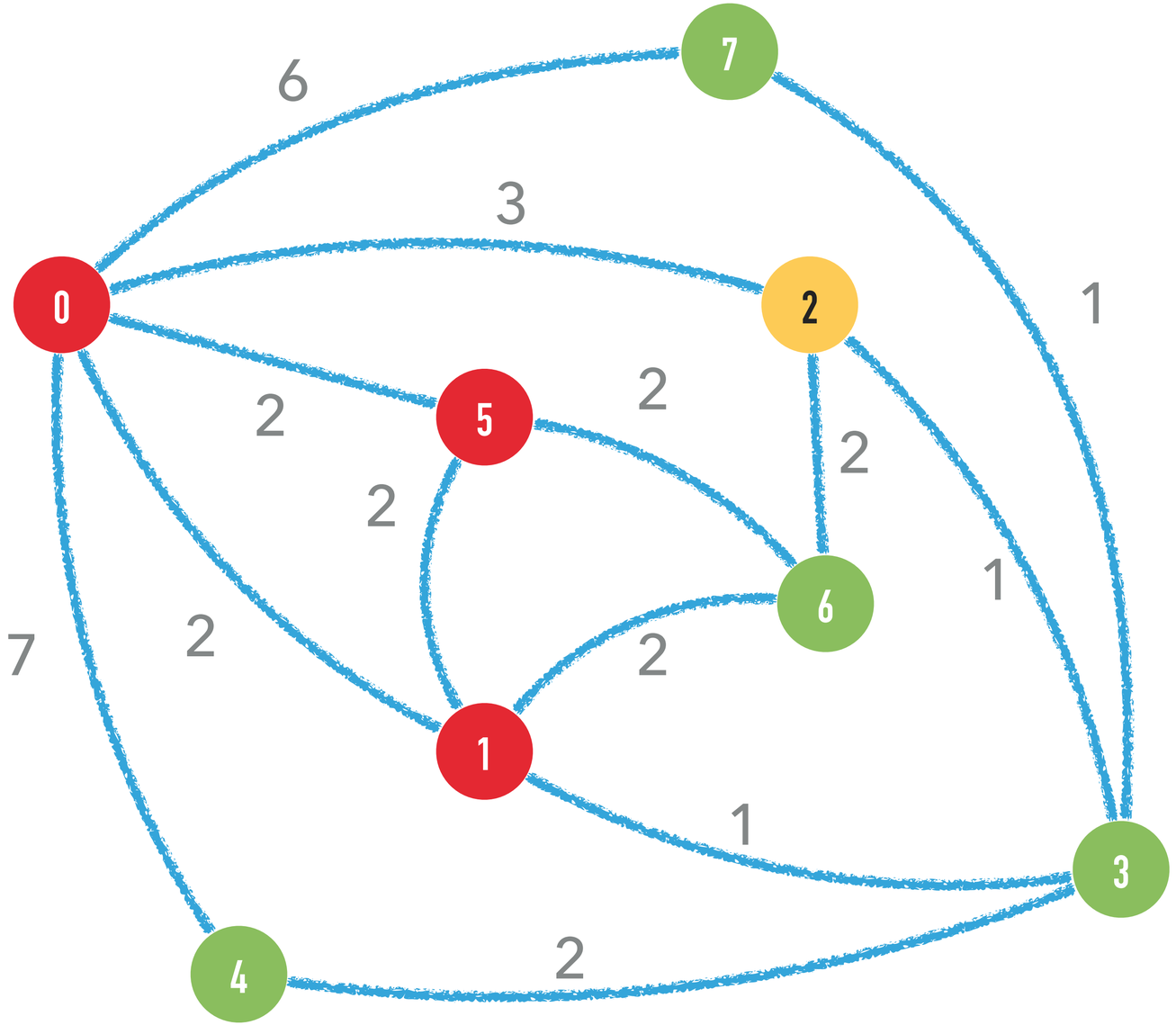


TABLEAU DES DISTANCE

Noeud	Distance
0	0
1	2
2	3
3	3
4	7
5	2
6	4
7	6

PRIORITY QUEUE (TODO)

Noeud	Distance
3	3
6	4
7	6
4	7

Question 6.1.9: DIJKSTRA

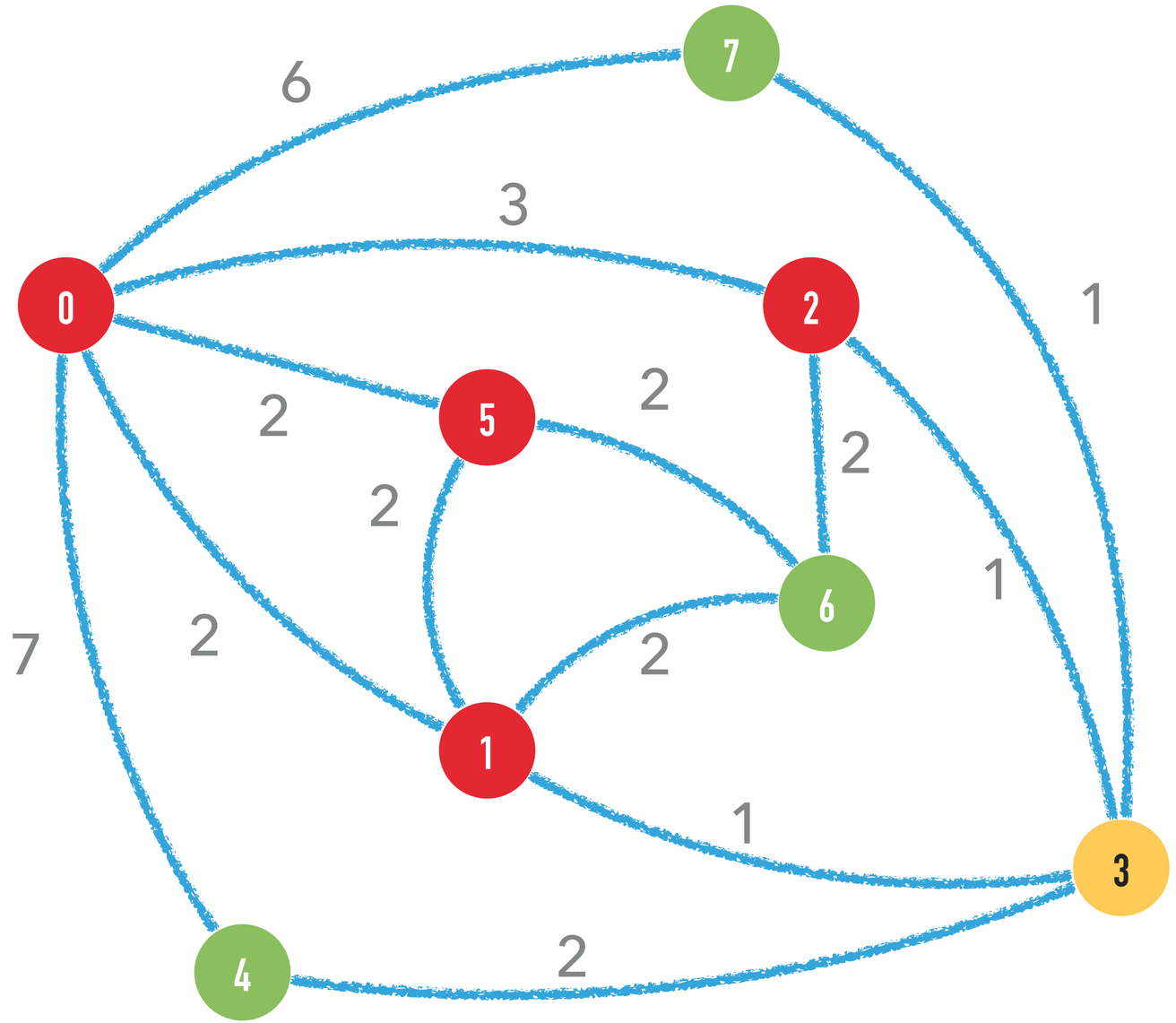


TABLEAU DES DISTANCE

Noeud	Distance
0	0
1	2
2	3
3	3
4	5
5	2
6	4
7	4

PRIORITY QUEUE (TODO)

Noeud	Distance
6	4
7	4
4	5

Question 6.1.9: DIJKSTRA

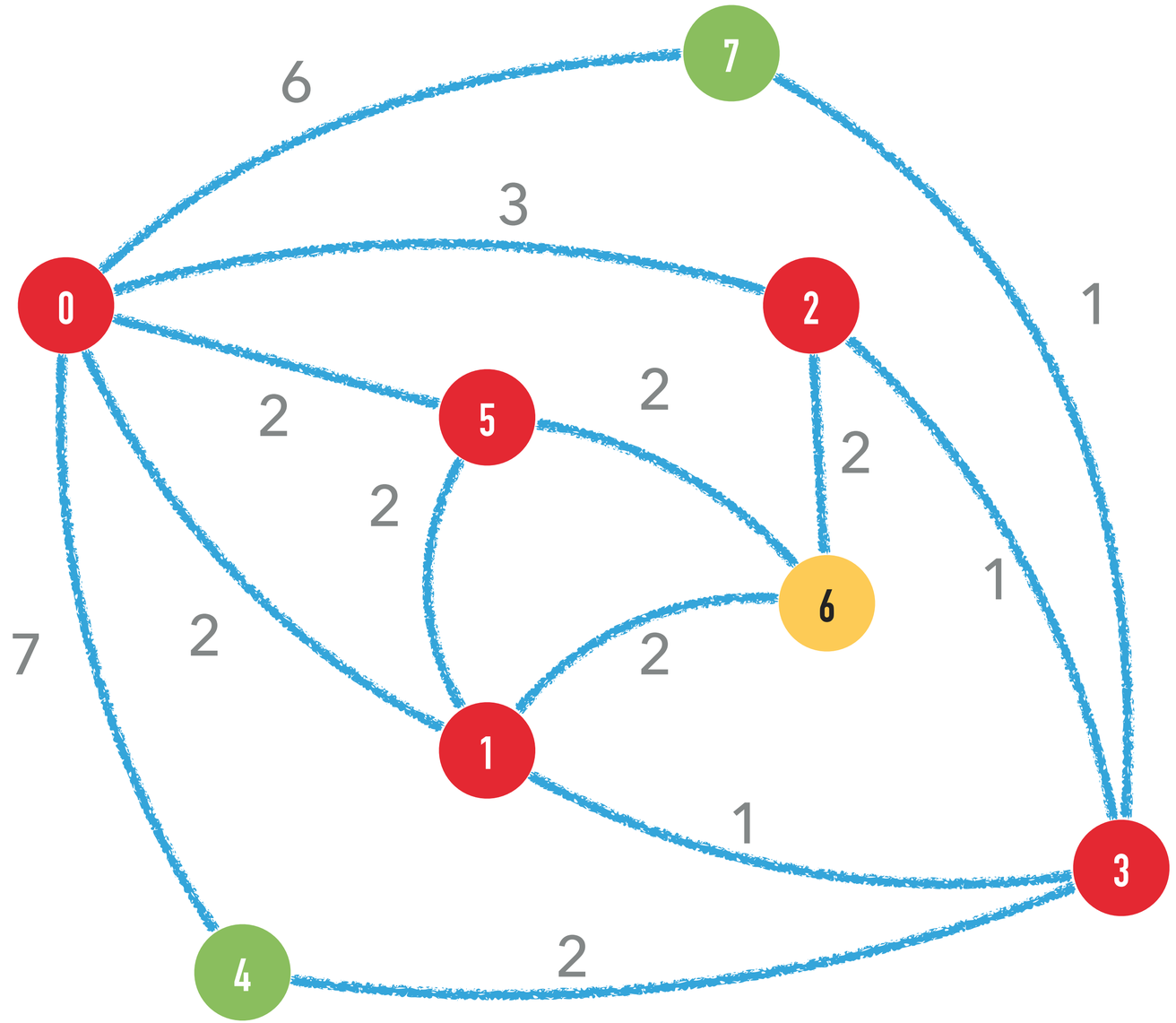


TABLEAU DES DISTANCE

Noeud	Distance
0	0
1	2
2	3
3	3
4	5
5	2
6	4
7	4

PRIORITY QUEUE (TODO)

Noeud	Distance
7	4
4	5

Question 6.1.9: DIJKSTRA

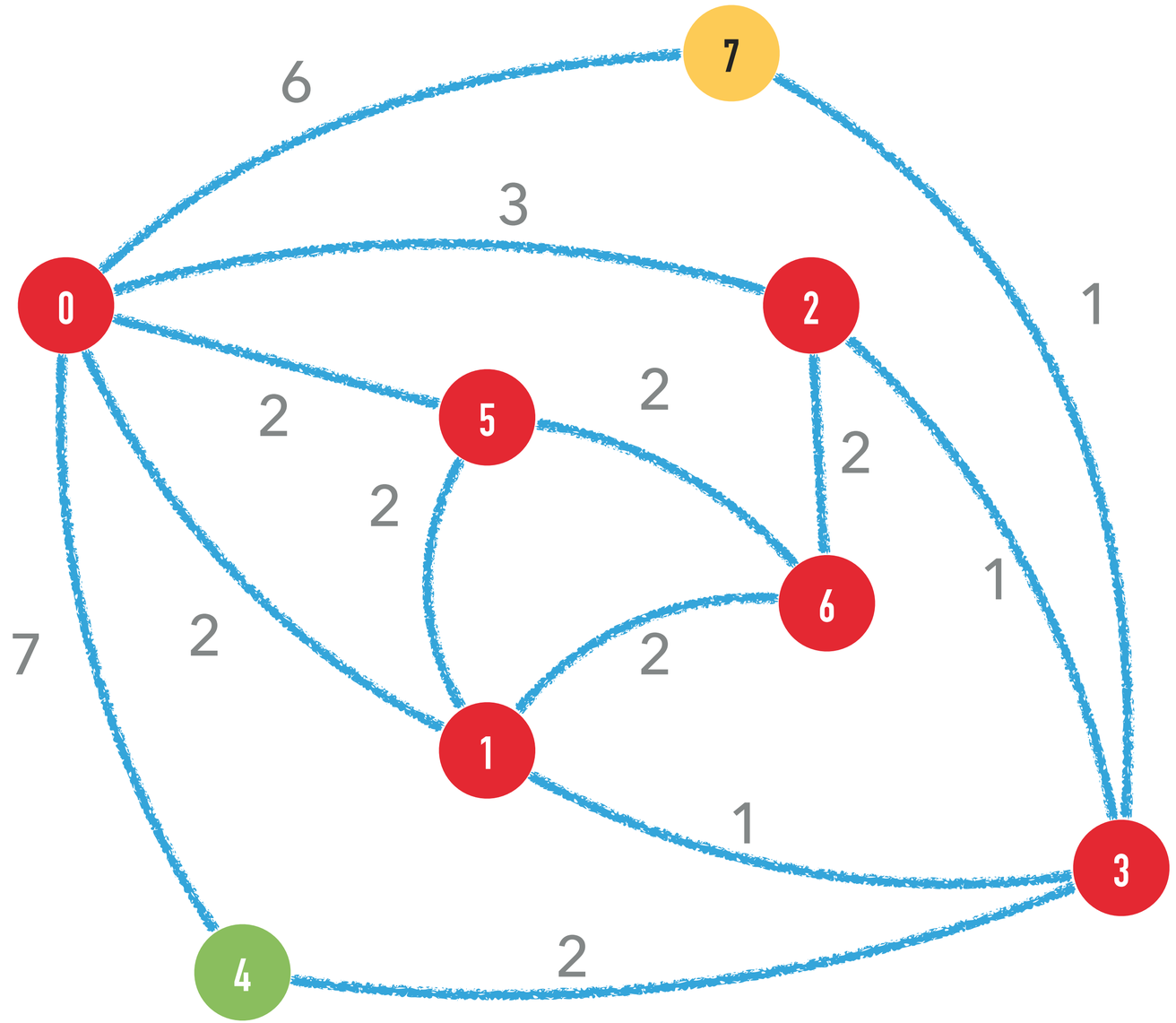


TABLEAU DES DISTANCE

Noeud	Distance
0	0
1	2
2	3
3	3
4	5
5	2
6	4
7	4

PRIORITY QUEUE (TODO)

Noeud	Distance
4	5

Question 6.1.9: DIJKSTRA

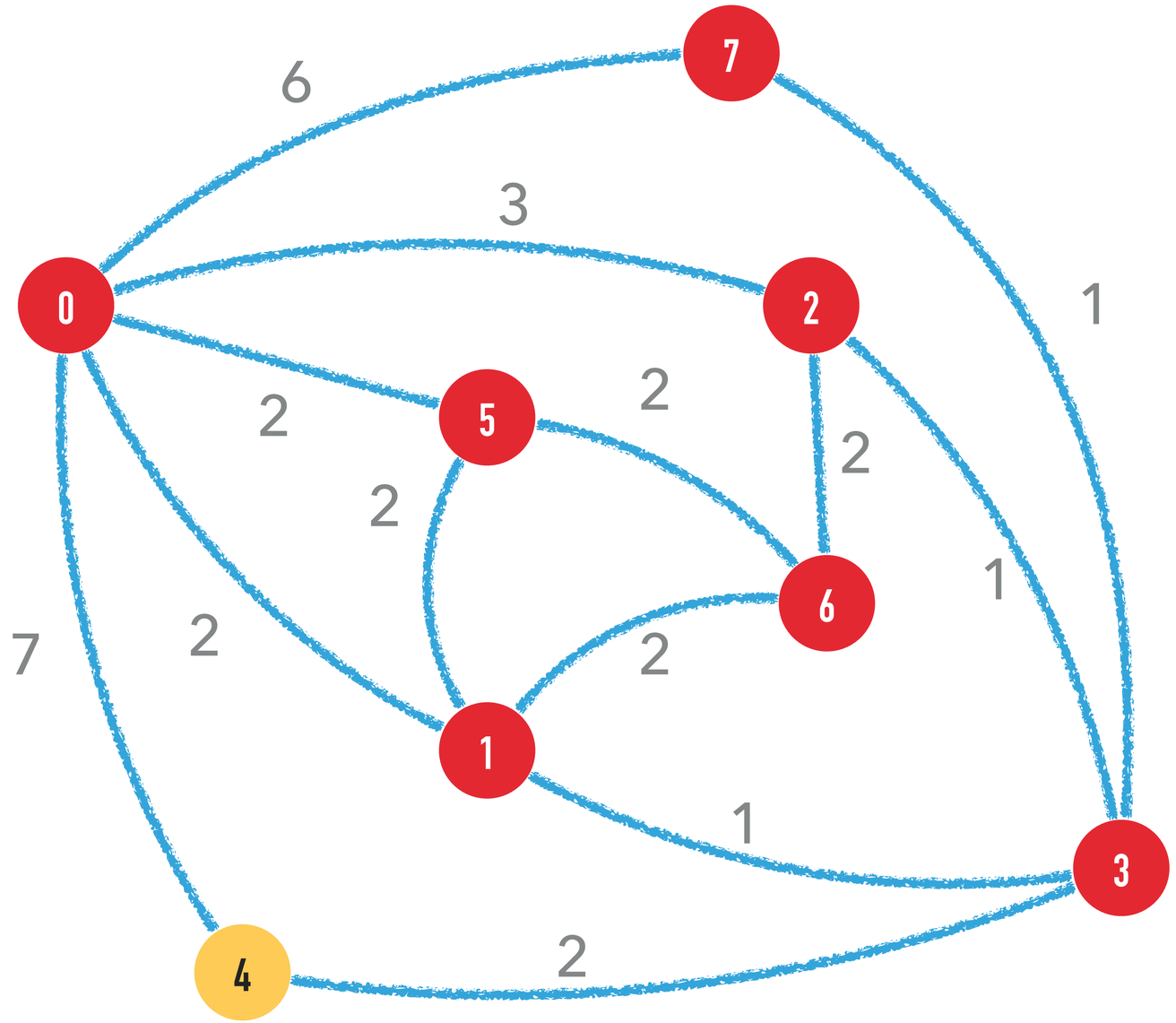


TABLEAU DES DISTANCE

Noeud	Distance
0	0
1	2
2	3
3	3
4	5
5	2
6	4
7	4

PRIORITY QUEUE (TODO)

Noeud	Distance

Question 6.1.9: DIJKSTRA

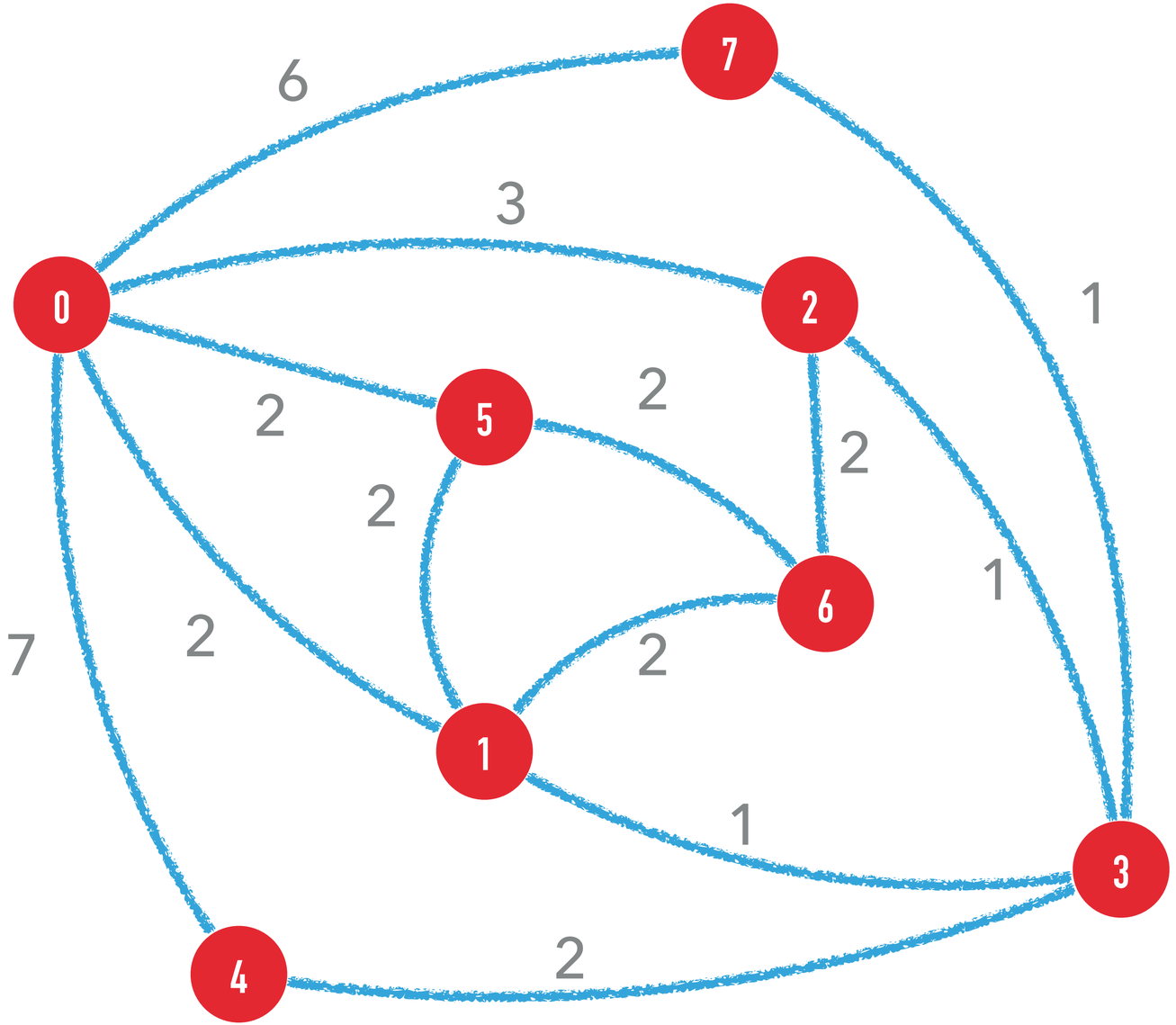


TABLEAU DES DISTANCE

Noeud	Distance
0	0
1	2
2	3
3	3
4	5
5	2
6	4
7	4

PRIORITY QUEUE (TODO)

Noeud	Distance

Question 6.1.9: DIJKSTRA

Complexité de Dijkstra:

- Visite des V noeuds $\mathcal{O}(V)$
- Lors de chaque visite de chaque noeud, on visite chaque arête $\mathcal{O}(E)$
- On pop V fois une PQ de taille V $\mathcal{O}(V \log V)$
- On « modifie » le poids au plus E fois des noeuds dans la PQ de taille V $\mathcal{O}(E \log V)$

$$\mathcal{O}((V + E)\log V)$$

Question 6.1.9: BESOIN D'UNE PQ « modifiable » ?

Peut-on utiliser une PQ Java, sans la possibilité de modifier les poids déjà à l'intérieur du heap?

Oui:

- Simplement réinsérer le noeud à chaque fois qu'il change
- Verifier quand on pop que l'élément n'a pas encore été pop

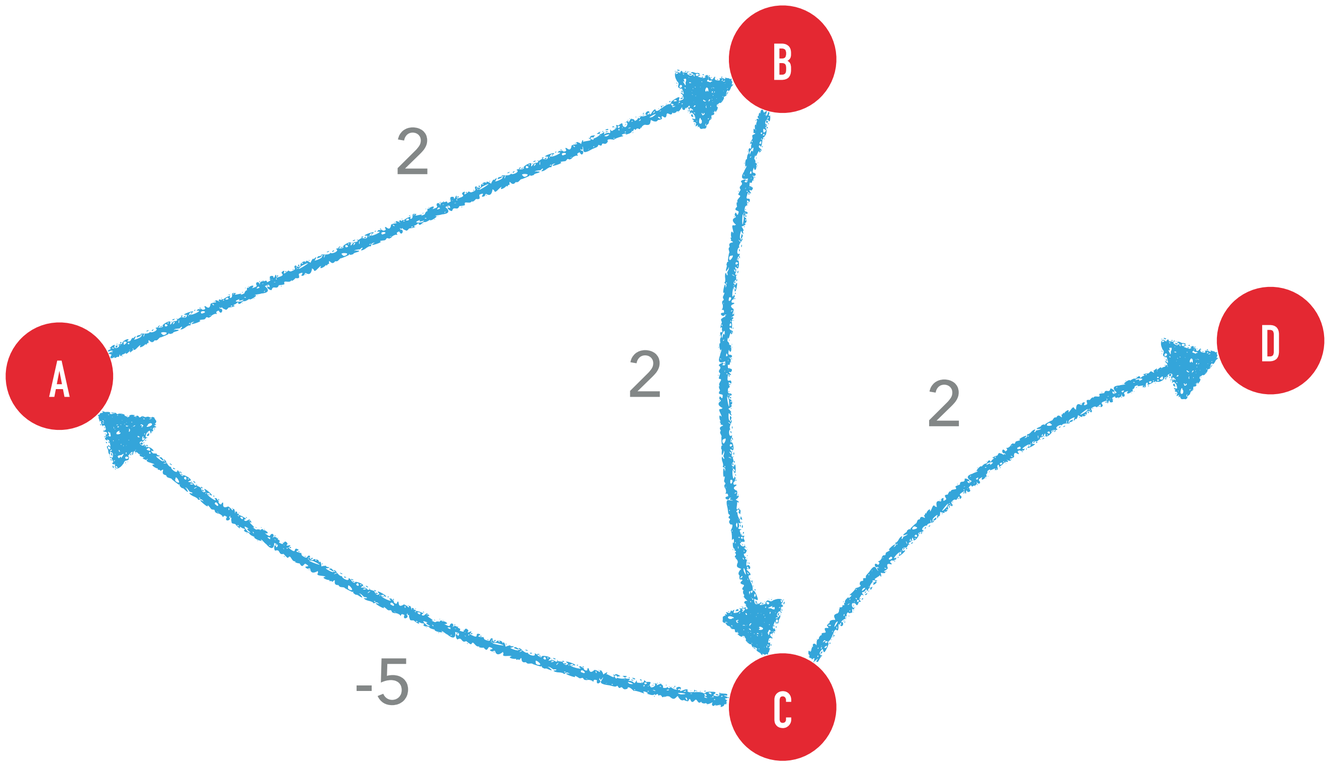
Complexité?

$$\mathcal{O}((V+E)\log E) = \mathcal{O}((V+E)\log V)$$

car $E < V^2$

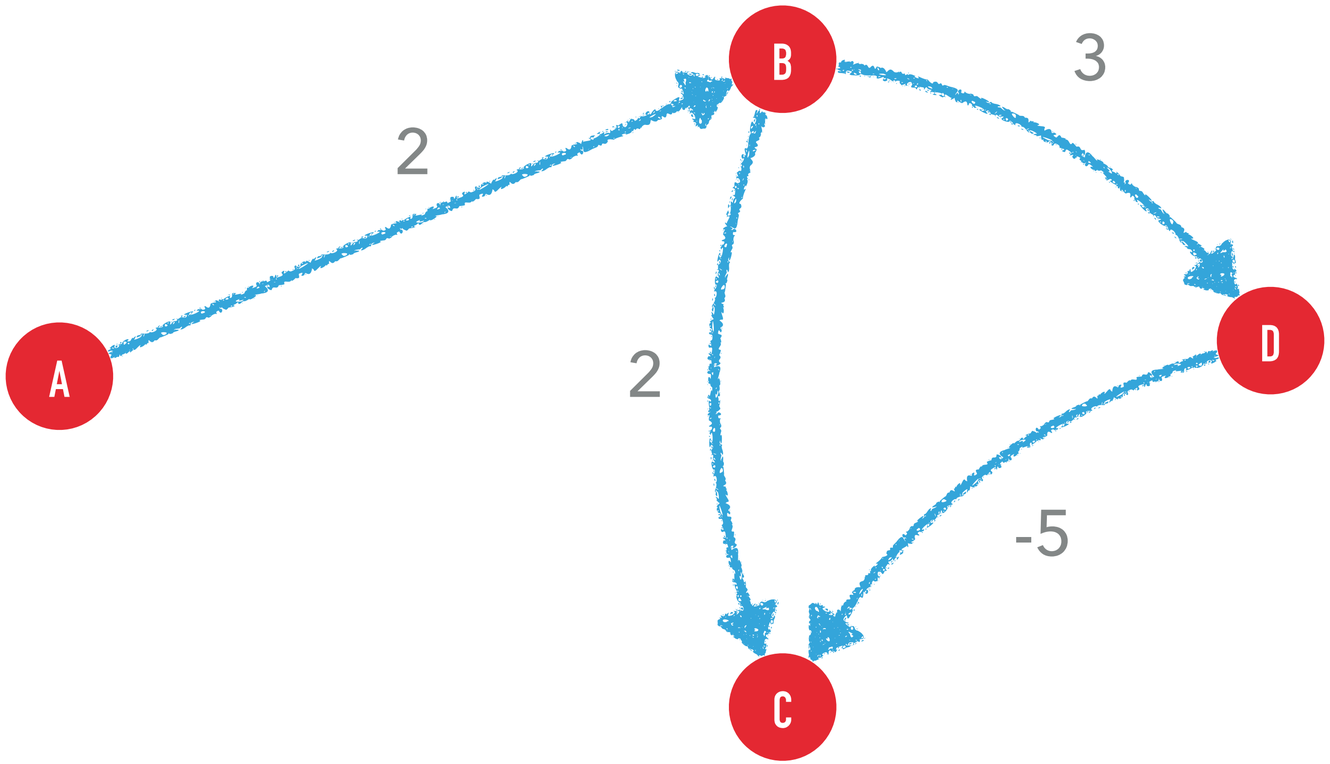
Question 6.1.10: les poids négatifs, c'est mal

Dijkstra fonctionne t'il avec des poids négatifs?



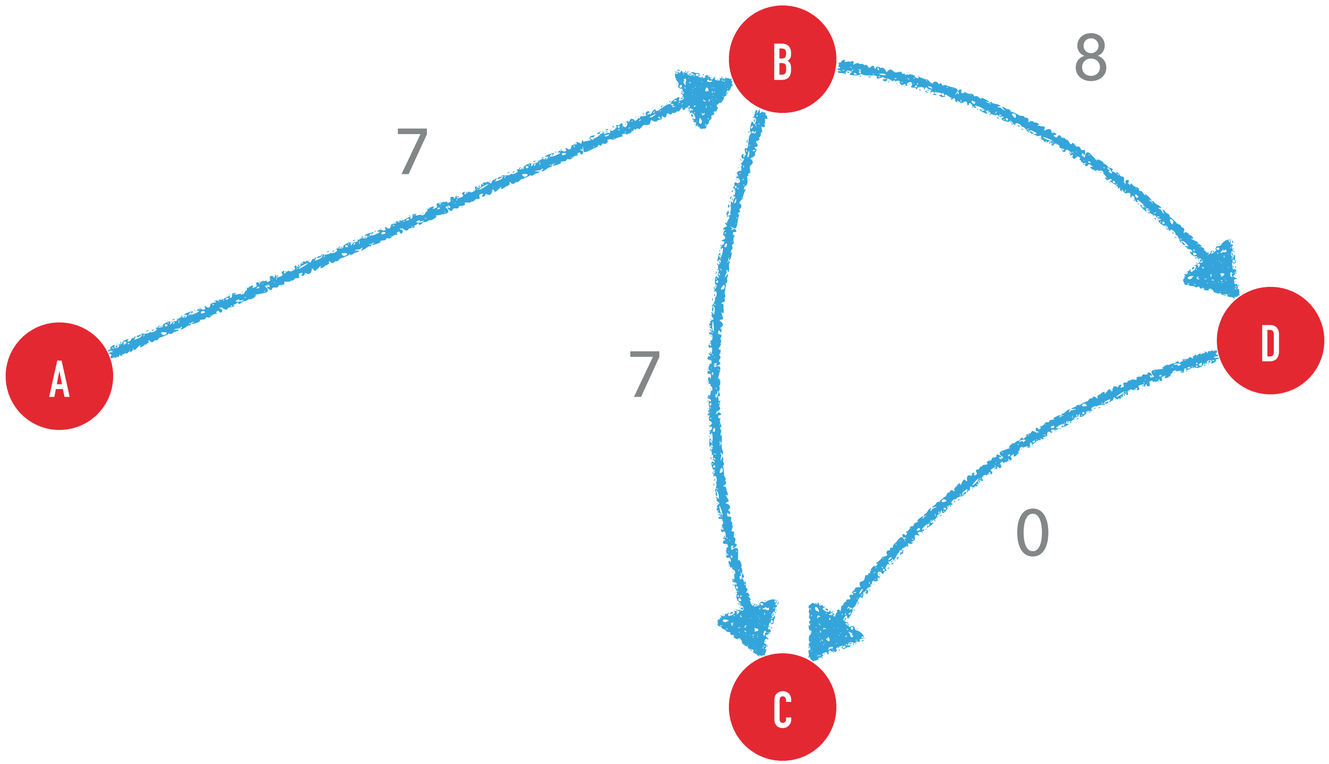
Question 6.1.10: les poids négatifs, c'est mal

Dijkstra fonctionne t'il avec des poids négatifs?



Question 6.1.11: les poids négatifs, c’est mal

Et en ajoutant 5 partout?



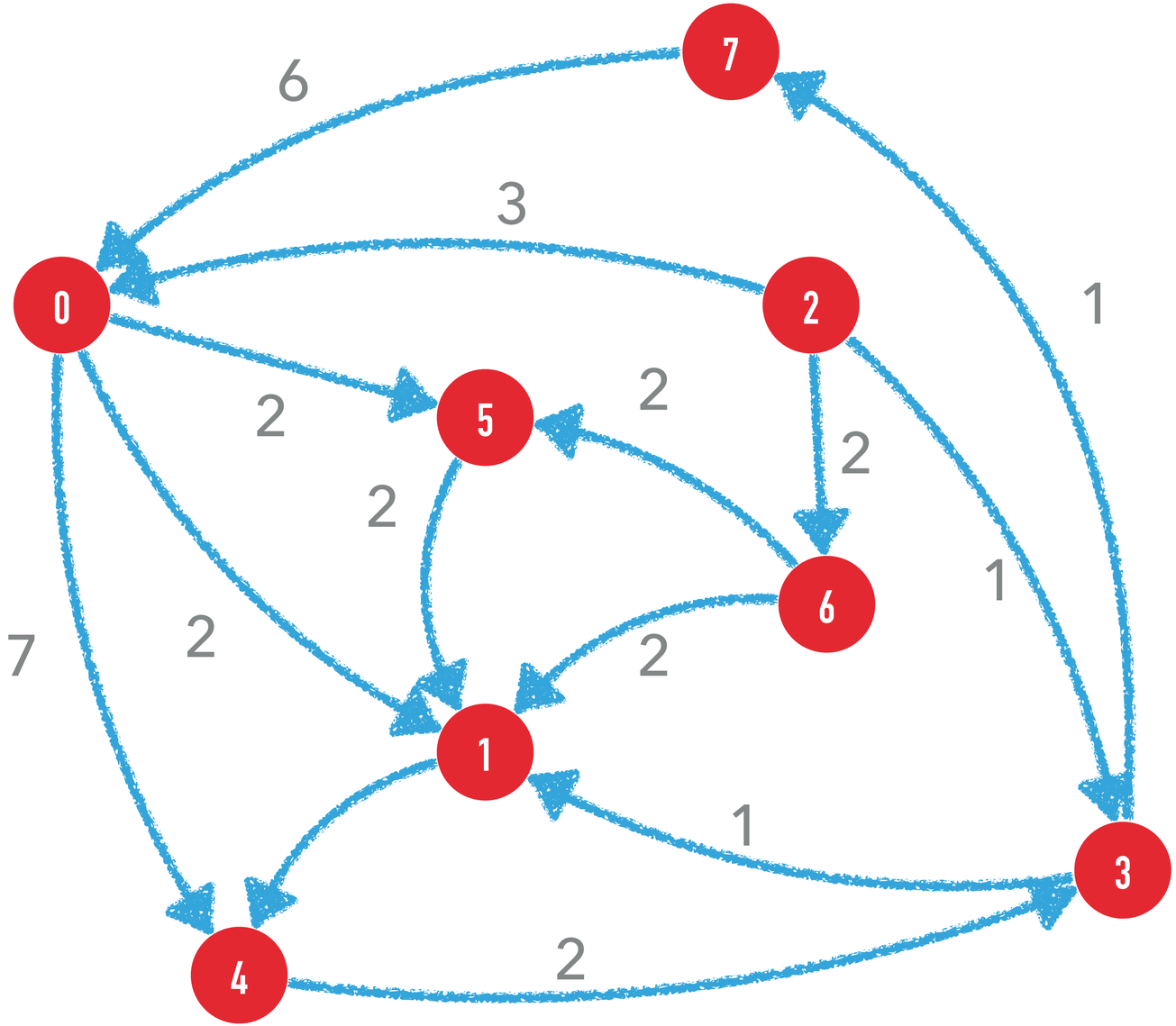
Question 6.1.11 BELLMAN FORD

$d[t,k]$ = distance du sommet source à t avec un chemin de k arcs

$$d[t,0] = \begin{cases} 0 & \text{if } t = s \\ \infty & \text{else} \end{cases}$$

$$d[t,k] = \min(d[t,k-1], \min_j(d[j,k-1] + w[j,t]))$$

Question 6.1.12 Chemin le plus long avec BELLMAN-FORD

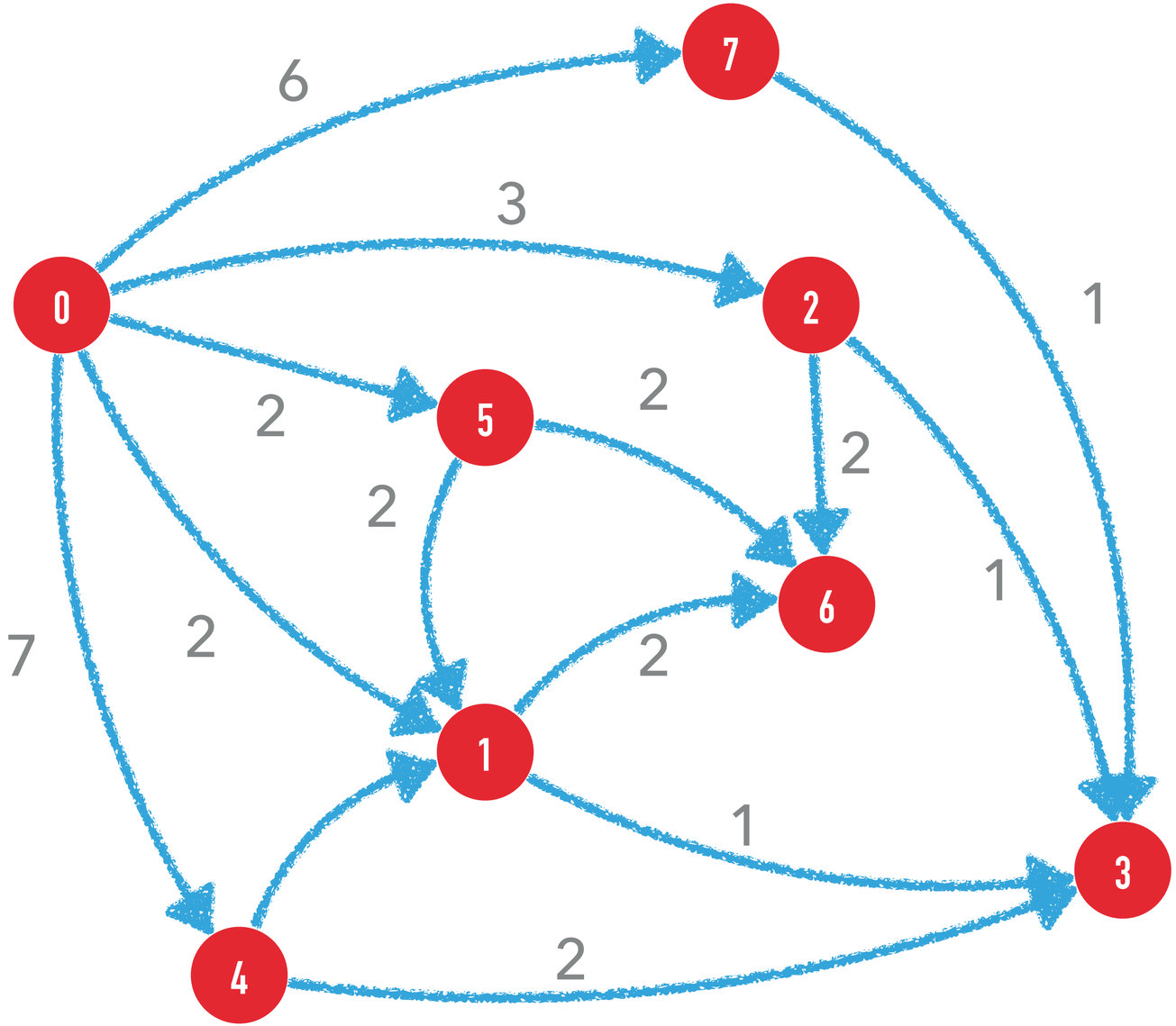


Comment trouver le chemin le plus long?

Peut-on mettre le poids des arêtes à leur opposé et lancer Bellman-ford?

Non, car ce graphe a des cycles! Le résultat de Bellman-ford est donc invalide.

Question 6.1.12 Chemin le plus long avec BELLMAN-FORD



CE GRAPHE EST UN DAG

Comment trouver le chemin le plus long?

Peut-on mettre le poids des arêtes à leur opposé et lancer Bellman-ford?

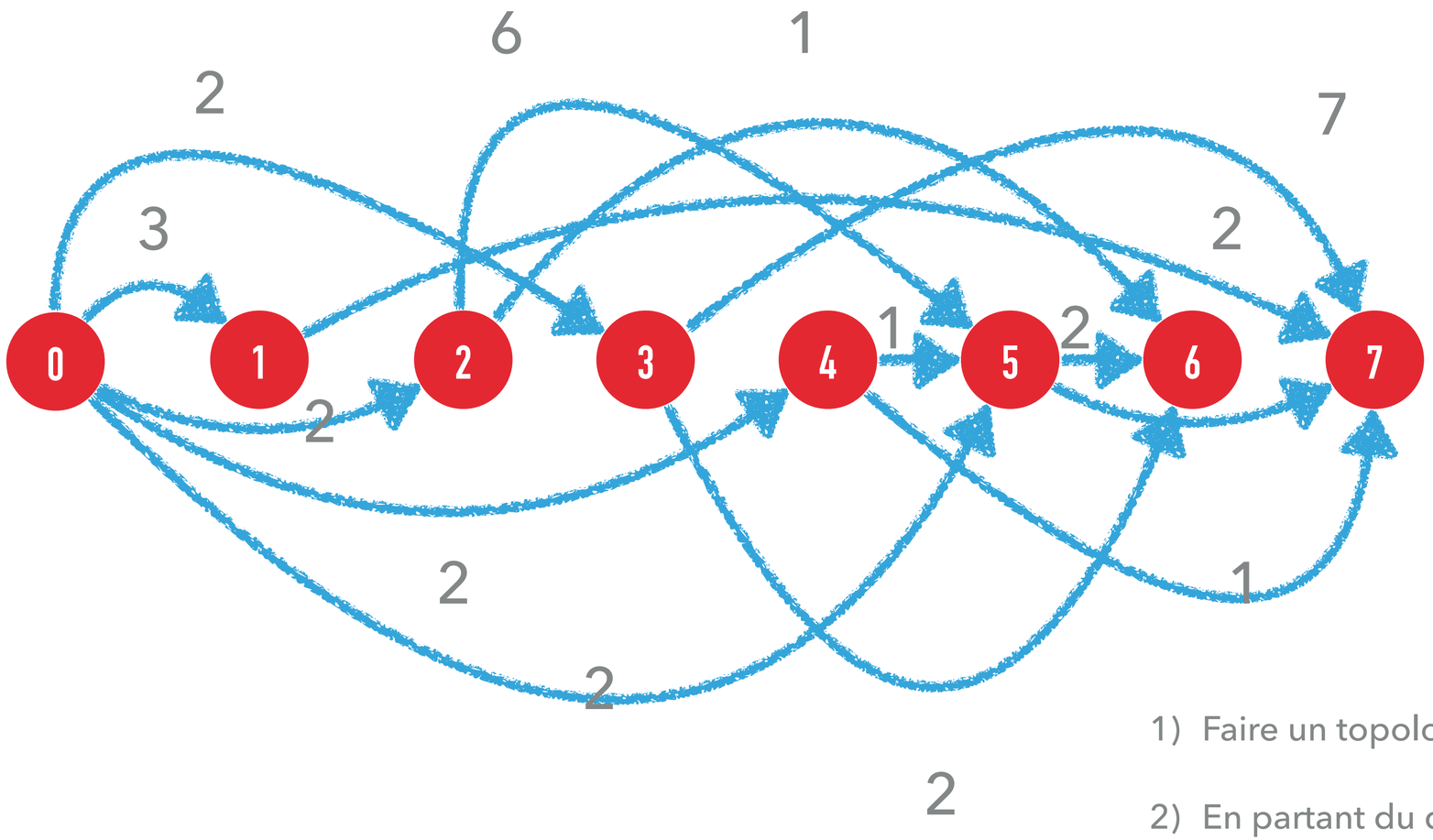
Ca marche!

Complexité?

$$\mathcal{O}(VE)$$

Question 6.1.12 Chemin le plus long avec BELLMAN-FORD

(nb: le graphe est différent par rapport au slide précédent)



1) Faire un topological sort
2) En partant du dernier noeud:

$$\mathcal{O}(V+E)$$

$$\text{longuest}[i] = \begin{cases} \max_{\text{voisin } j} \text{longuest}[j] + \text{weight}(i,j) & \text{si i a des voisins} \\ 0 & \text{sinon} \end{cases}$$