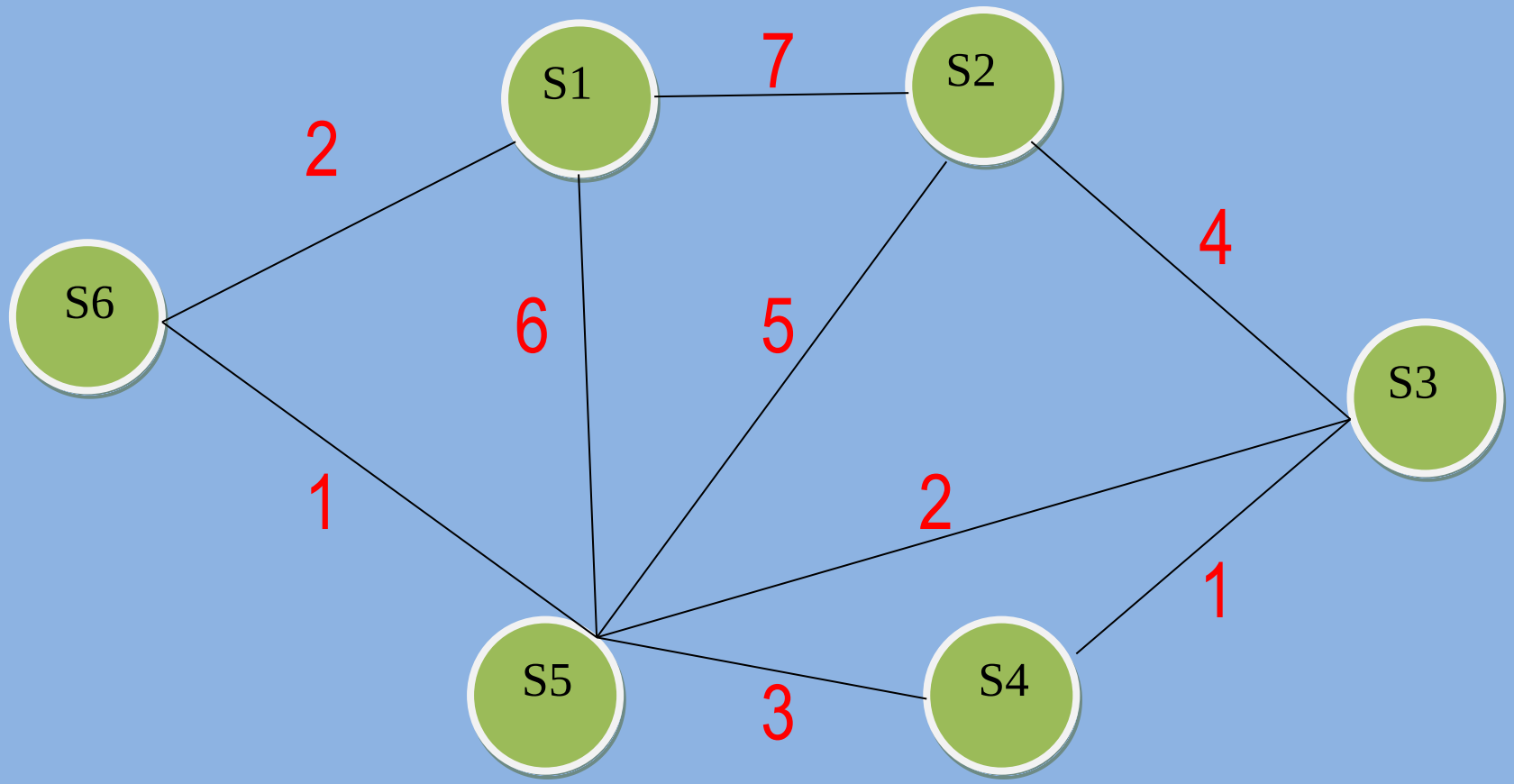
U.F.R SCIENCES ET TECHNIQUES
Département d'Informatique
B.P. 1155
64013 PAU CEDEX
Téléphone secrétariat : 05.59.40.79.64
Télécopie : 05.59.40.76.54

II-PROBLEME DE RECOUVREMENT MINIMUM

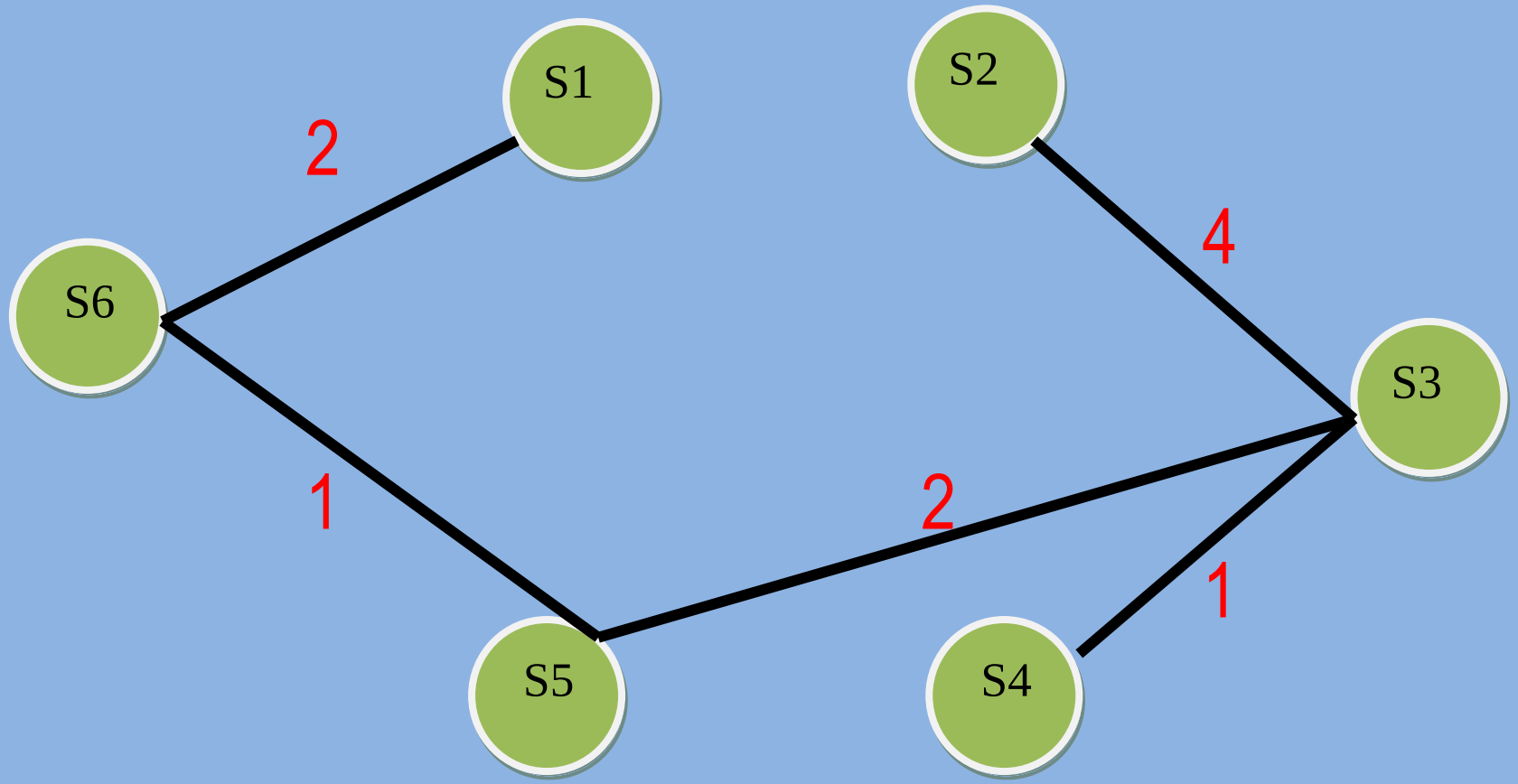
QUEL EST LE PROBLEME ?

Soit l'ensemble des sommets :

Graphe non orienté valué de connexité



Graphe de recouvrement minimum



I- ARBRE ET GRAPHE

Soit **n** la taille d'un graphe non orienté G = (S,A),

n = |S|.

Soit **m** le nombre d'arêtes de *G*,

m = |A|.

Le nombre **cyclomatique** de G est estimé à partir de **n**, **m** et du nombre **p** de ses composantes connexes:

$$\nu(G) = m-n+p$$

$\nu(G)$ estime le nombre de cycles que possède le graphe :

$$\nu(G) \geq 0$$

1- Arbre

Un arbre est un graphe **connexe :**

$$p=1 \Rightarrow \nu(G) = m-n+1 \geq 0$$

Un arbre est un graphe **sans cycle**.

$$\nu(G) = m-n+1$$

$$= 0$$

Le nombre d' arêtes d'un arbre est donc :

$$m = n-1$$

Un arbre est un graphe qui connecte tous les sommets entre eux avec un **minimum d'arêtes**.

2-Théorème d'arbre

Les propositions suivantes sont équivalentes pour tout graphe **non orienté** G à **n** sommets :

1- **G** est un **arbre**,
2- **G** est **sans cycles** et **connexe**,
3- **G** est **sans cycles** et comporte **n-1 arêtes**,
4- **G** est **connexe** et comporte **n-1 arêtes**,
5- chaque paire **(u, v)** de sommets distincts est reliée par une seule **chaîne simple**.

3-Arbre couvrant

Un **arbre couvrant** ou arbre maximal est un **graphe partiel** qui est aussi un **arbre**.

Conséquences

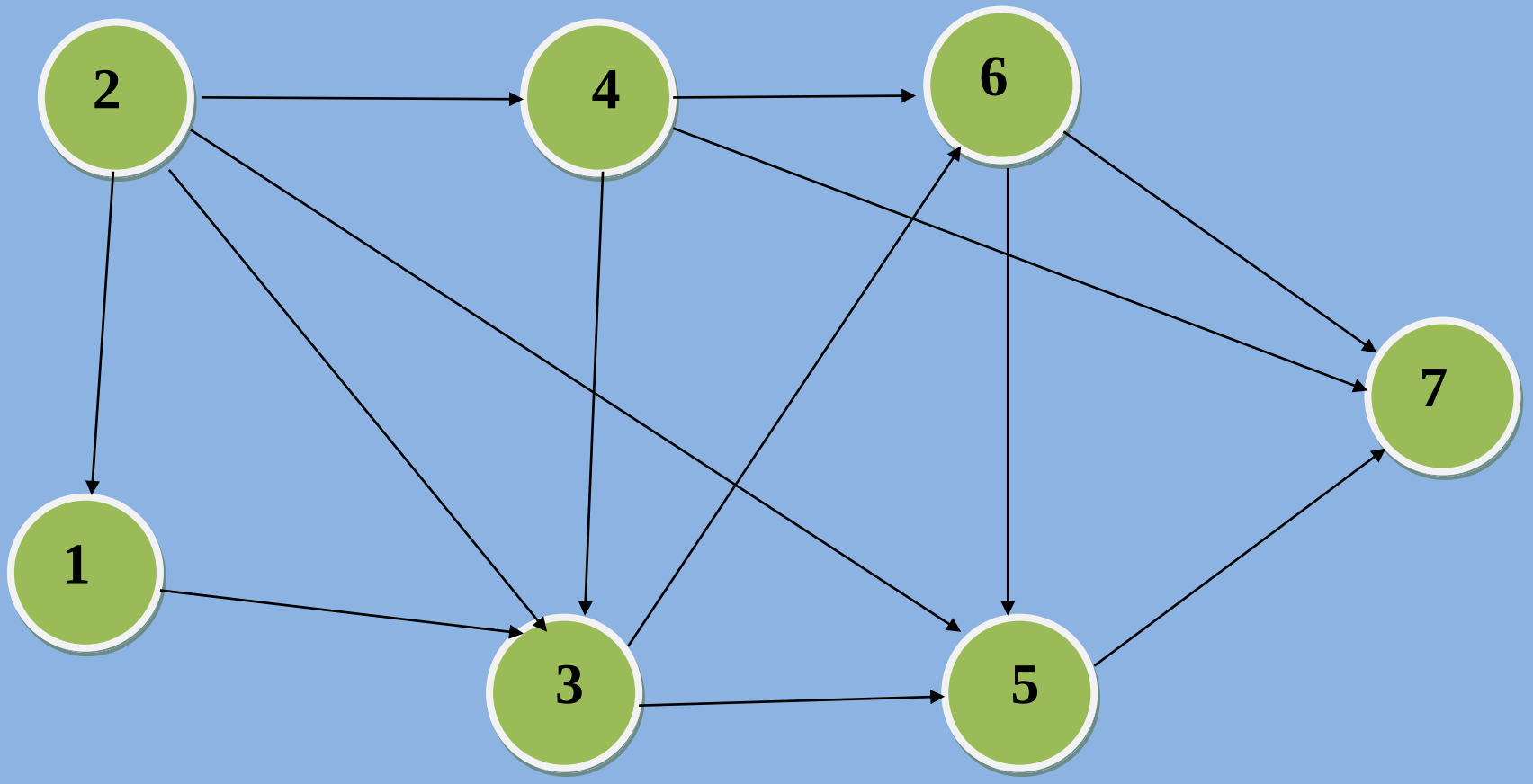
. L'ajout de la moindre arête supplémentaire dans un arbre crée un cycle.

- Un graphe **connexe** possède toujours un graphe partiel qui est un arbre.

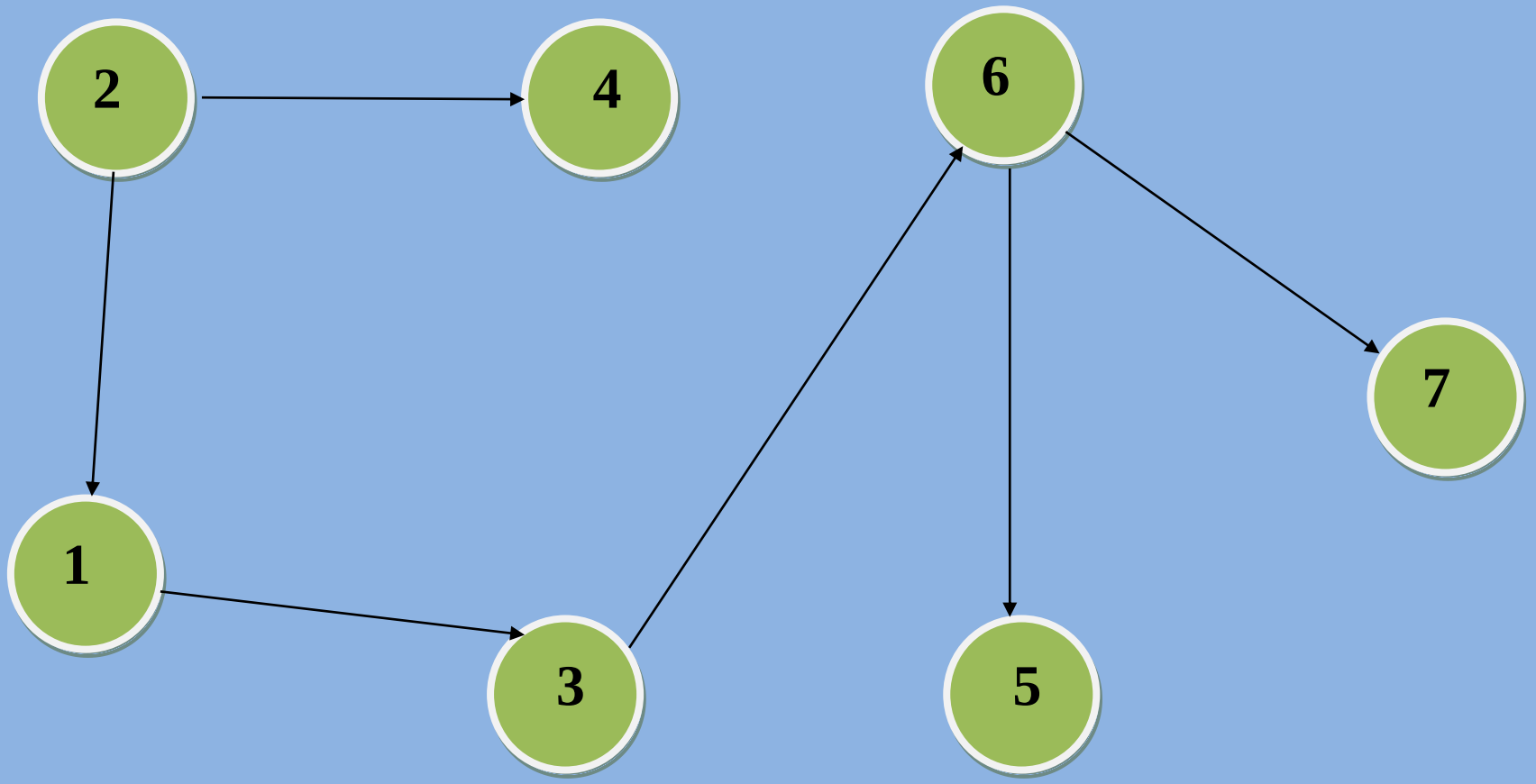
- Un arbre couvrant est construit en enlevant suffisamment d'arêtes de façon à supprimer **tous les cycles**.

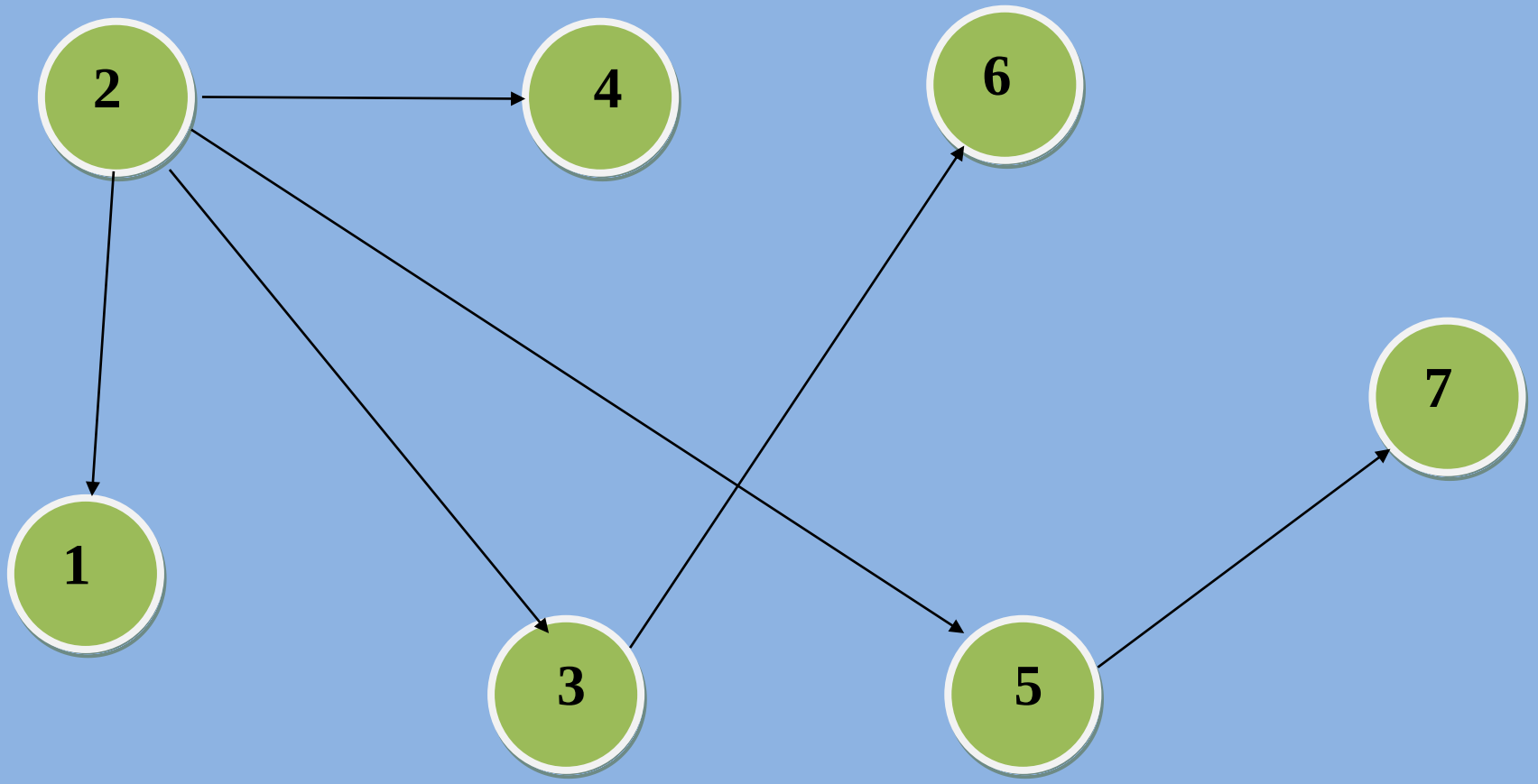
Exemple

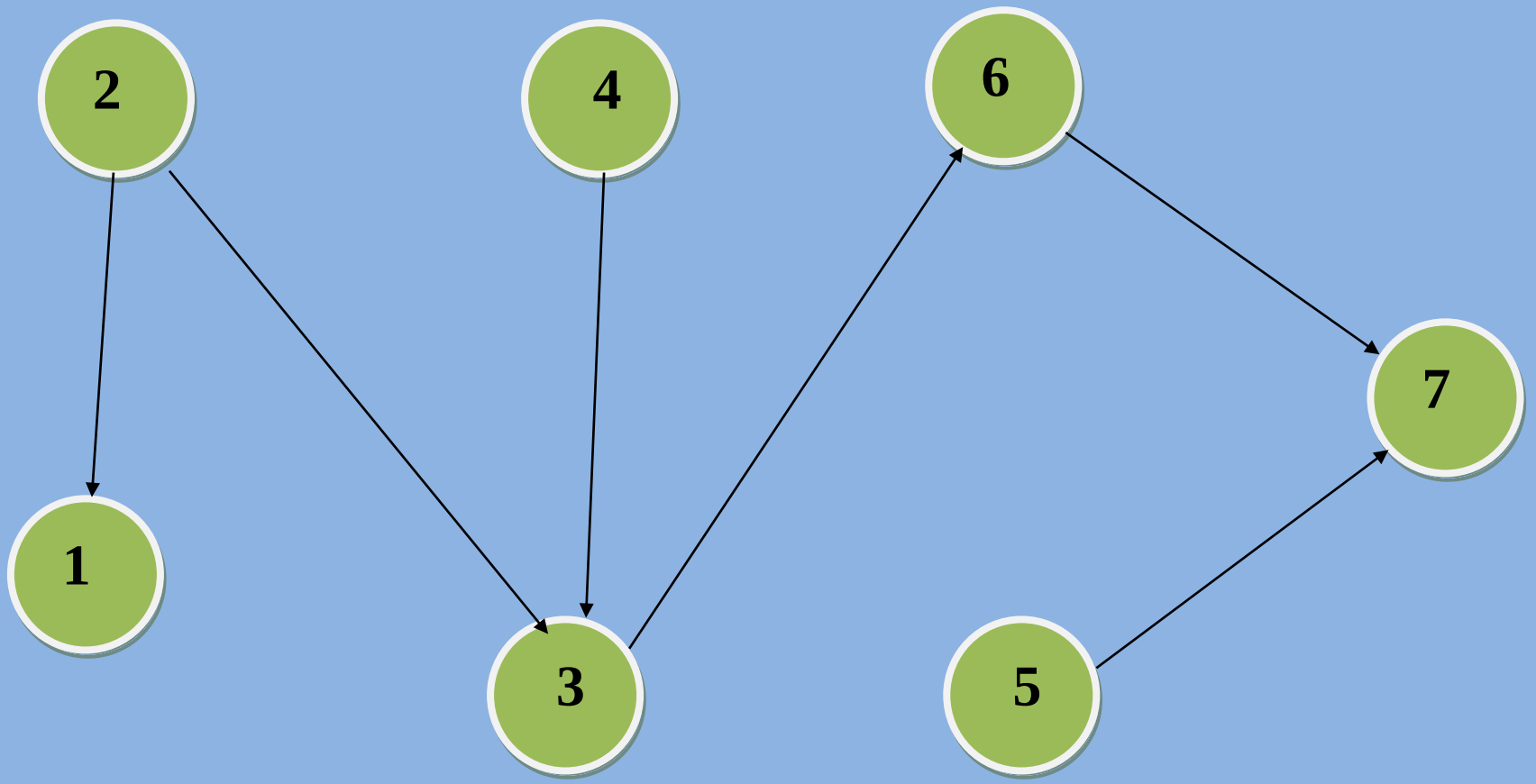
Soit le graphe orienté connexe:



On peut en extraire les **arbres couvrants** suivants :




2
4
6
7
1
3
5

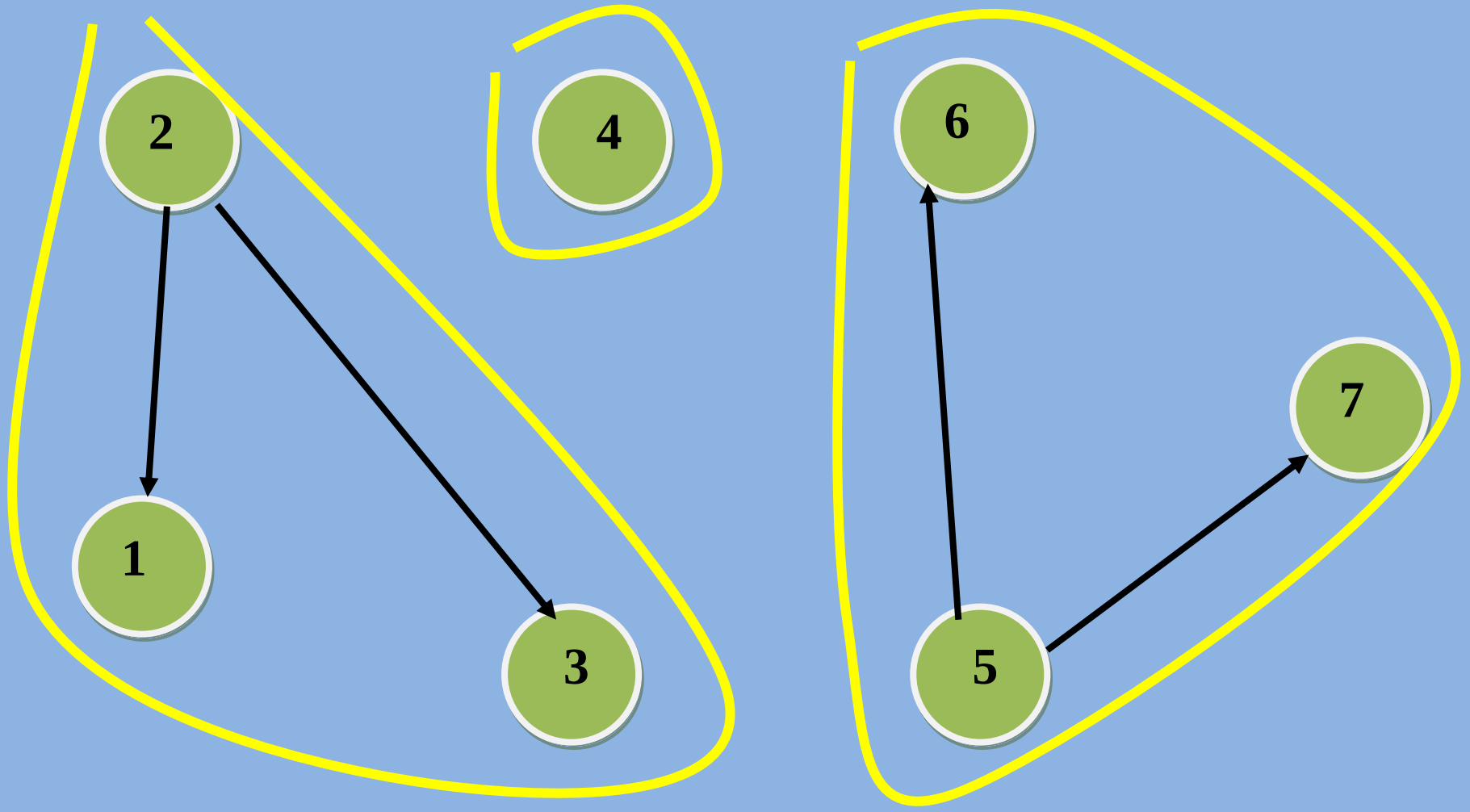

2
4
6
7
1
3
5

4- Forêt

On appelle **forêt** un graphe dont chaque composante connexe est un arbre.

Un graphe sans cycle mais non connexe est appelé une **forêt**.

Exemple de forêt



5- Racine - Antiracine

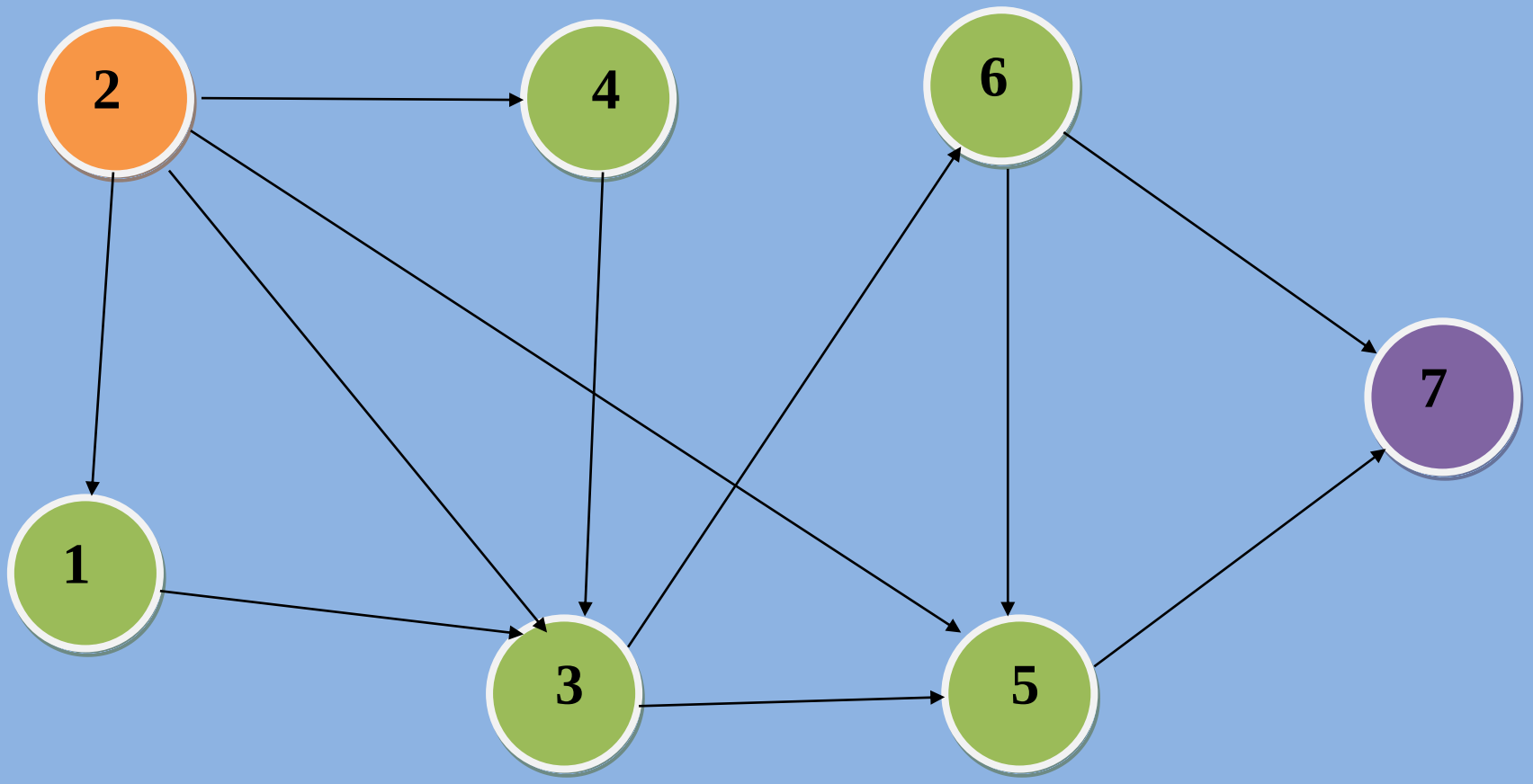
Un sommet **r** d'un graphe orienté *G* est une **racine** de *G :*

- s'il existe un chemin
- joignant **r** à chaque sommet du graphe *G*.

Un sommet **a** d'un graphe *G* est une **anti-racine** de *G* :

- s'il existe un chemin
- joignant chaque sommet du graphe *G* à ***a***.

Exemple



2 est une **racine** du graphe.

7 est une **anti-racine** du graphe

6- Arborescence, anti-arborescence

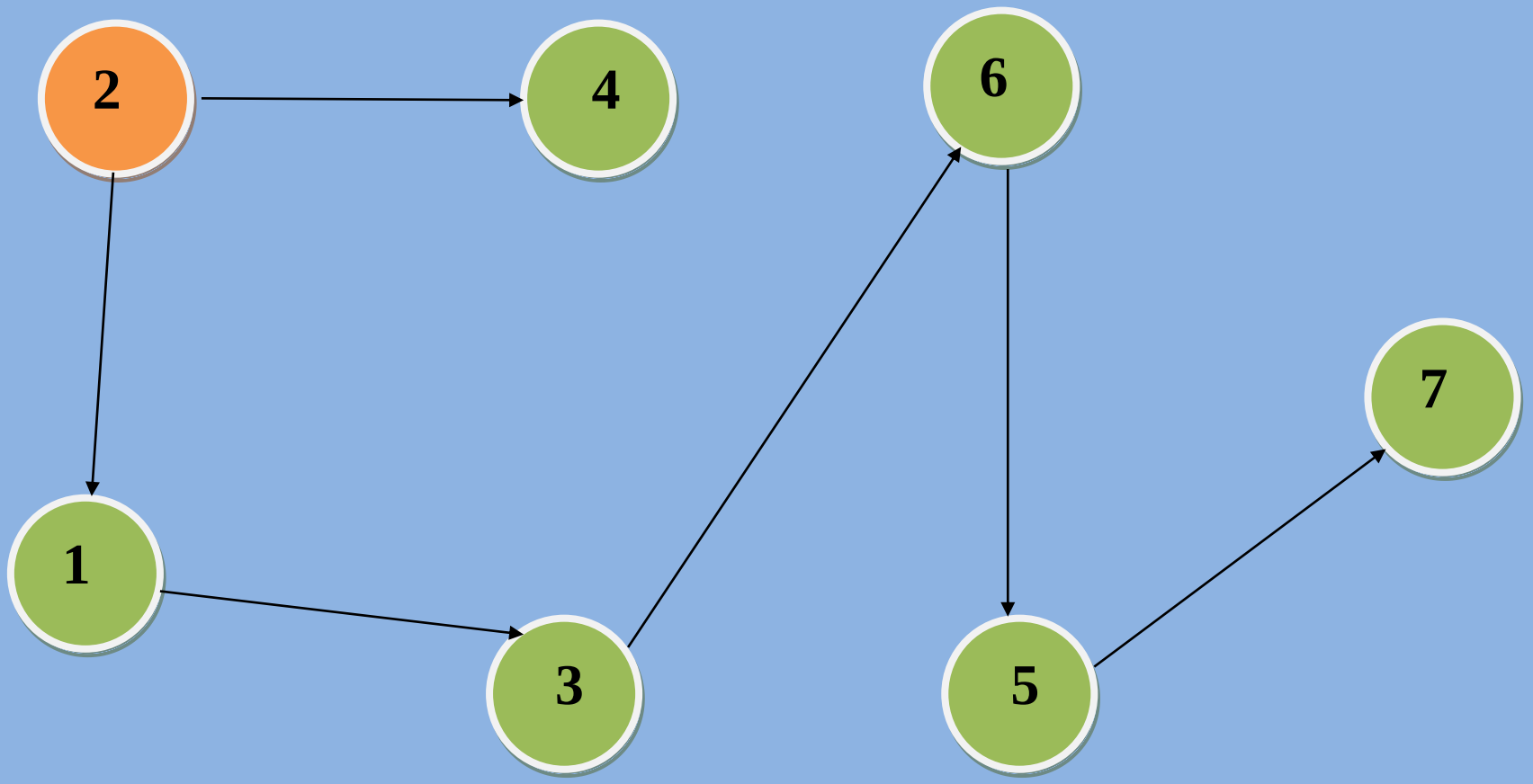
Un graphe *G* est une **arborescence** de racine **r** si :

- *G* est un arbre
- et si **r** est une racine.

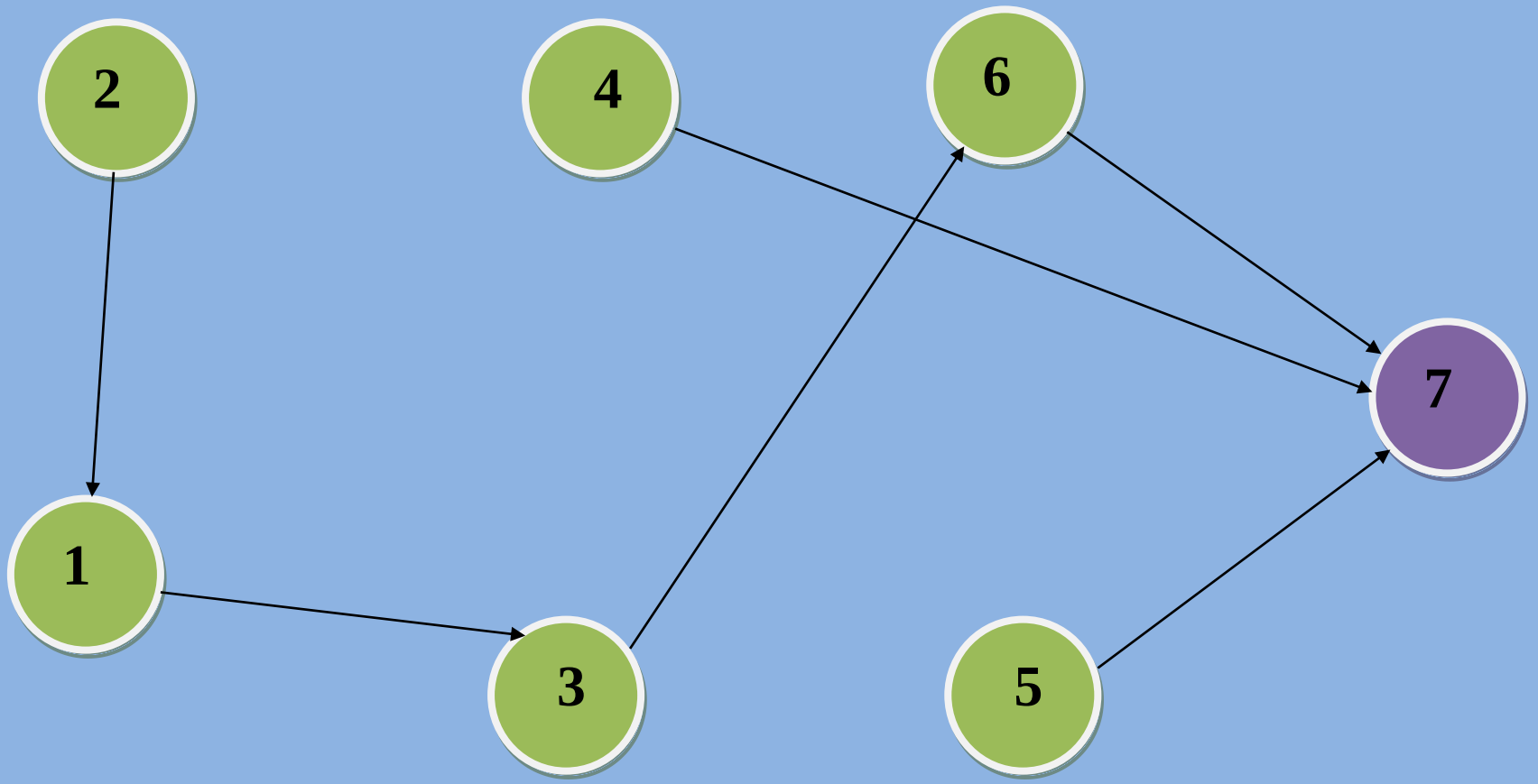
Un graphe *G* est une **anti-arborescence** d'**anti-racine a** si :

- *G* est un arbre
- et si **a** est une anti-racine.

Arborescence de racine 2



Anti-arborescence d’anti-racine 7



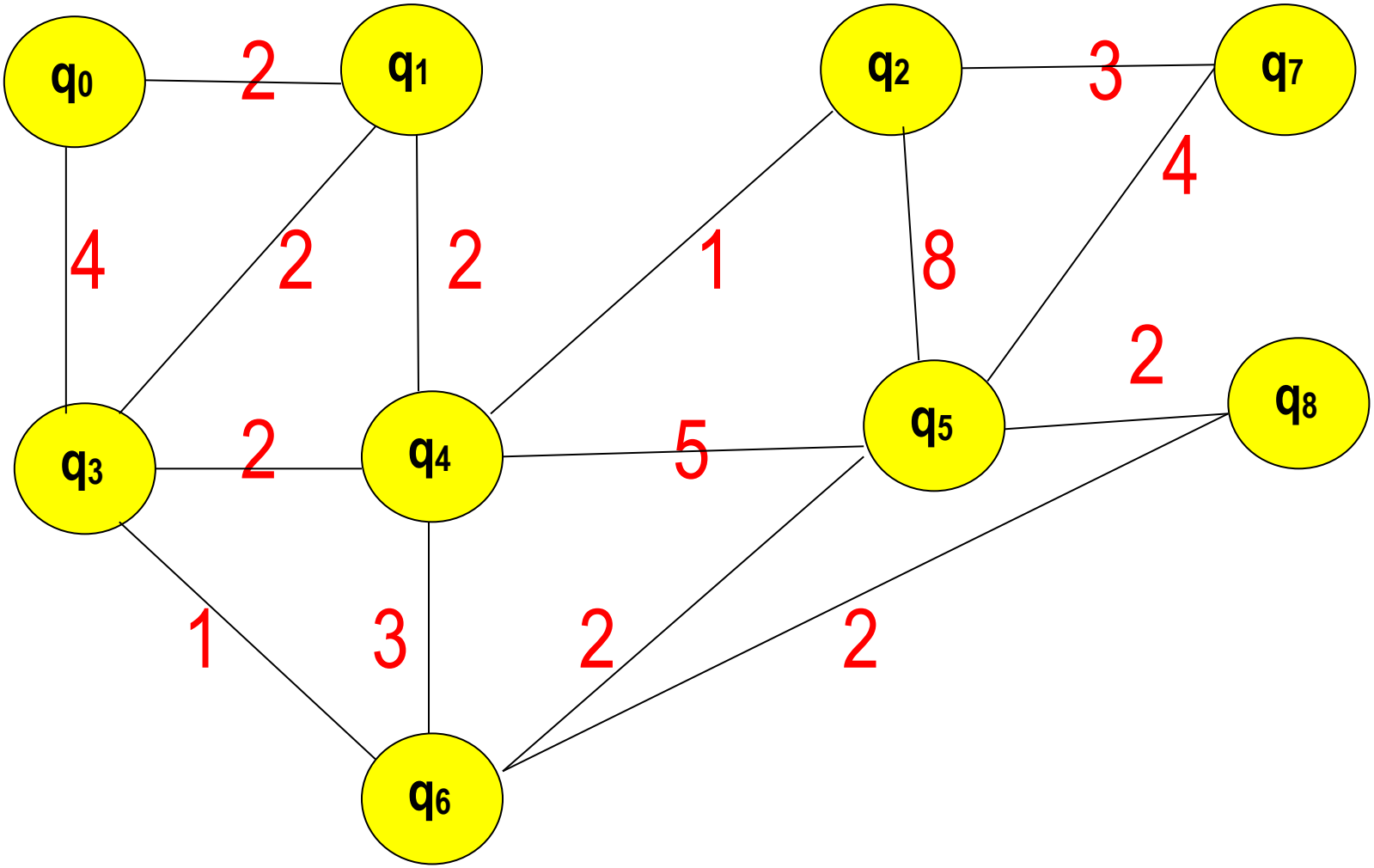
II- ARBRE DE RECOUVREMENT MINIMUM

Soit G est un graphe non orienté : G= (S,A)

Imaginons que l'on associe :

- à chaque arête **a** $\in$ A,
- une valeur, notée **c(a)**, appelée **coût** ou **poids**.

G est appelé graphe **valué**.


q0
q1
q2
q7
q3
q4
q5
q8
q6
2
4
2
2
1
8
3
4
2
2
5
1
3
2
2

On appelle **coût d'un graphe** partiel G' généré par :

$$A' = \{a_1, a_2, \ldots, a_p\}$$

et on note **coût (G')**, la somme :

$$c(a_1) + c(a_2) + \ldots + c(a_p)$$

des coûts des arêtes de G'.

Position du problème

Le problème de **recouvrement minimum** consiste à trouver :

-un **arbre couvrant** de *G*,

-dont le **coût est minimum**.

Exemple d’un cas réel

Minimiser le coût du câblage électrique pour alimenter les différents «postes» d’un avion peut se ramener à la recherche :

-d'un arbre couvrant,

-de coût minimum.

En effet, on cherche à:

- connecter tous les postes entres eux: **connexité**
- sans générer de lignes de câblage inutiles.

D'où la recherche d'un arbre : **absence de cycle**

Ensuite, on veut utiliser le moins de câble possible: **coût minimum**

Aussi on:

- associe à chaque possibilité de connexion la **longueur** de câble nécessaire,

- cherche à minimiser la **longueur totale** de câble utilisée.

Existence d'une solution

Soit un graphe non orienté connexe G; on **peut toujours** trouver un **arbre couvrant** en supprimant de G les arêtes qui forment un cycle.

Il existe un **nombre fini** d'arbres couvrants pour G.

Si G est valué, l'**existence** d'un arbre couvrant de **coût minimum** est donc assurée.

Condition d'unicité

En général, il peut y avoir, pour G, **plusieurs** arbres couvrants de coût minimum.

Si les coûts des arêtes satisfont la condition suivante:

$$\forall u,v \in A \bullet c(u) \neq c(v)$$

alors l'**unicité** est assurée.

Algorithme de construction

Deux algorithmes de construction d'arbre couvrant de coût minimum seront étudiés.

Leur **efficacité** dépend :

- du choix de **représentation** du graphe
- de la **structure même du graphe**.

Dans les deux cas :

- on part d'un **graphe vide**,
- on construit progressivement l'arbre couvrant de coût minimum par **adjonctions d'arêtes** (arcs).

La différence est que, pendant la construction :

- l'algorithme de **Kruskal** assure l'**absence de cycle**,
- alors que l'algorithme de **Prim** en assure la **connexité**.

1- Algorithme de KRUSKAL

Soit G= (S,A,C), un graphe non orienté **valué** et connexe tel que : |S| = n et |A|= m.

Le problème consiste à construire l'arbre, noté G', de recouvrement minimum.

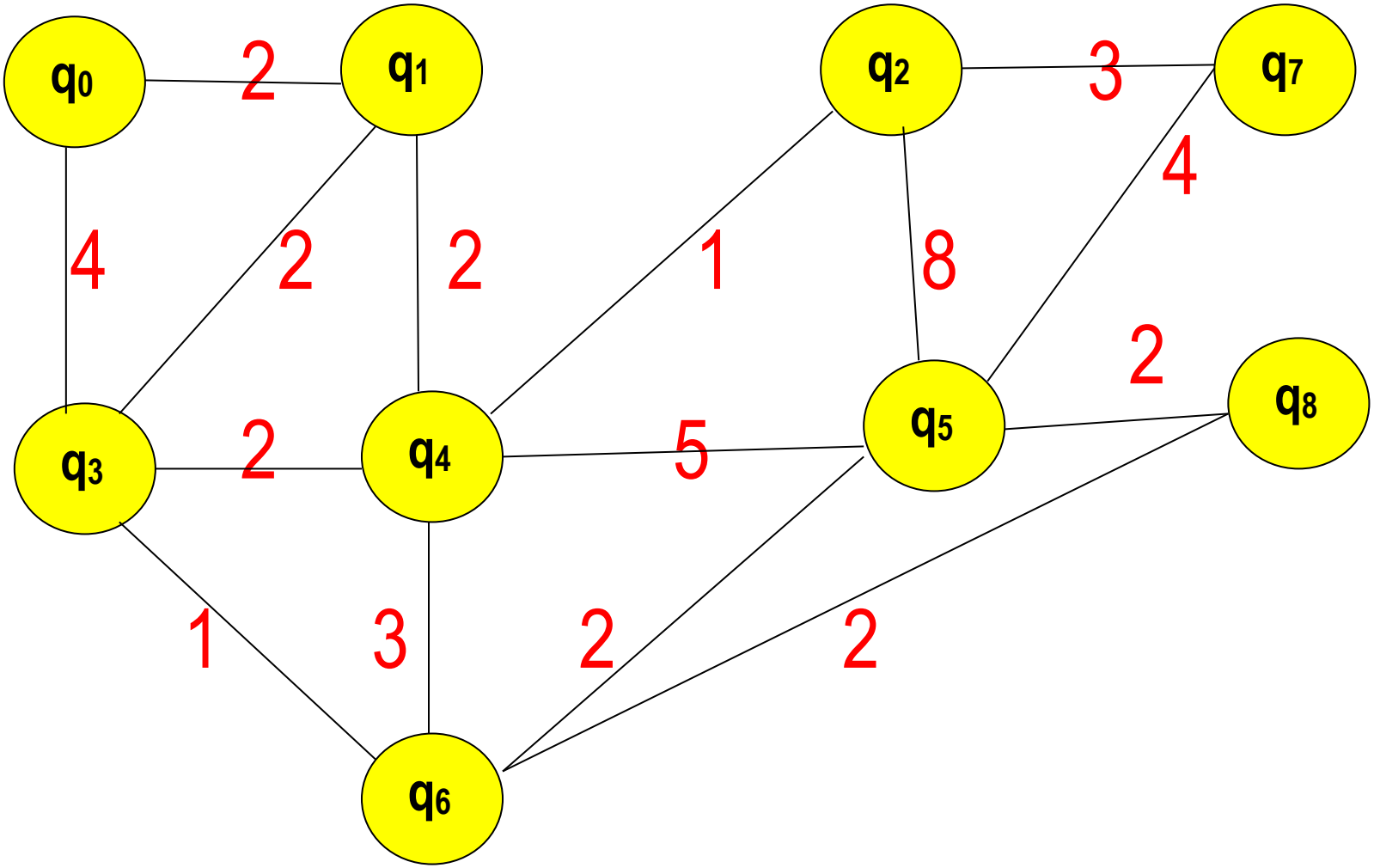
Idée

Deux points :

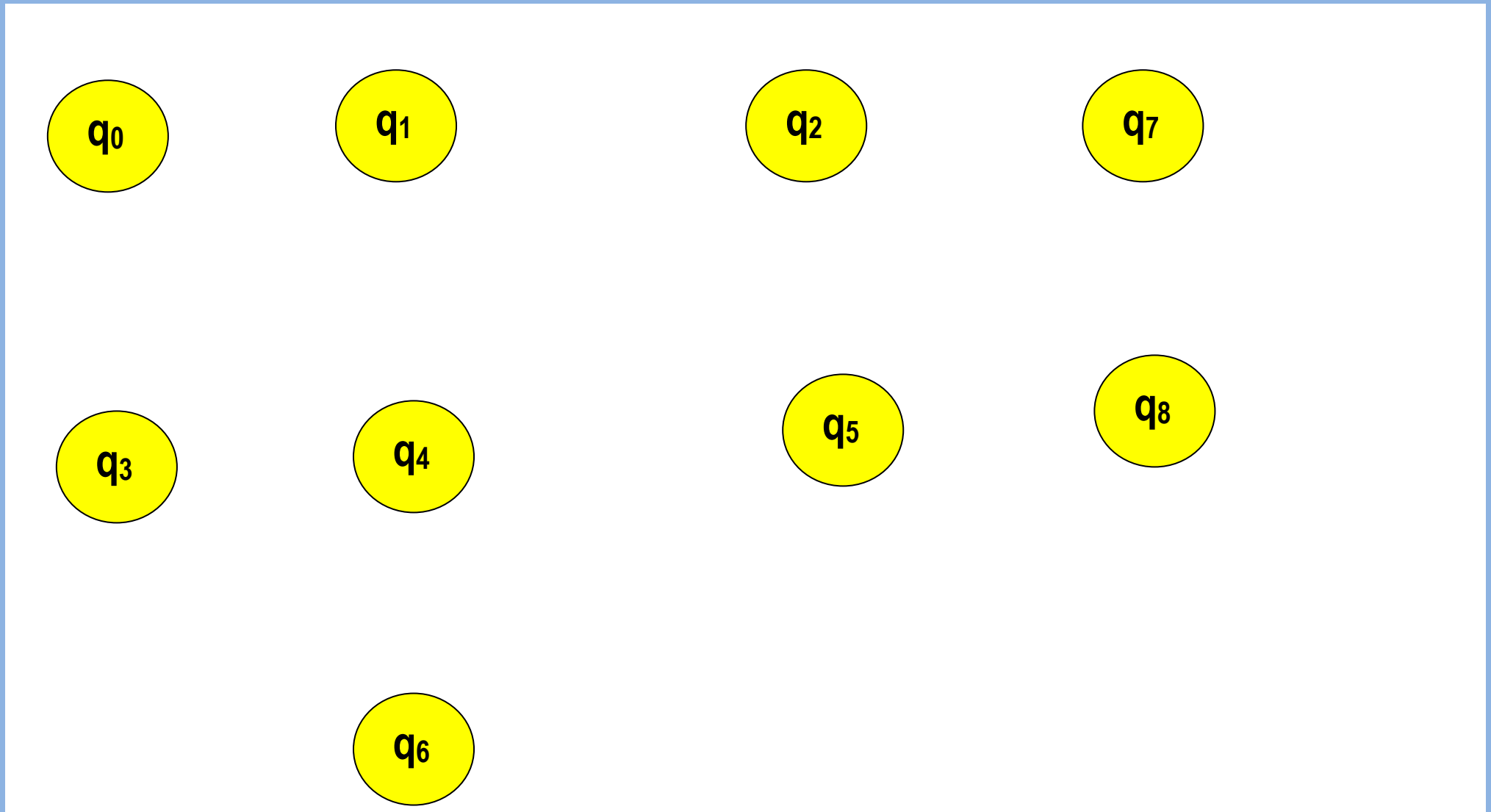
- **l'arbre** G' est construit partant d'une **forêt**,
- une arête **compatible** est une arête de coût minimum reliant deux arbres de la forêt.

Comment trouver la (une) arête de coût minimum reliant deux arbres de la forêt G' ?

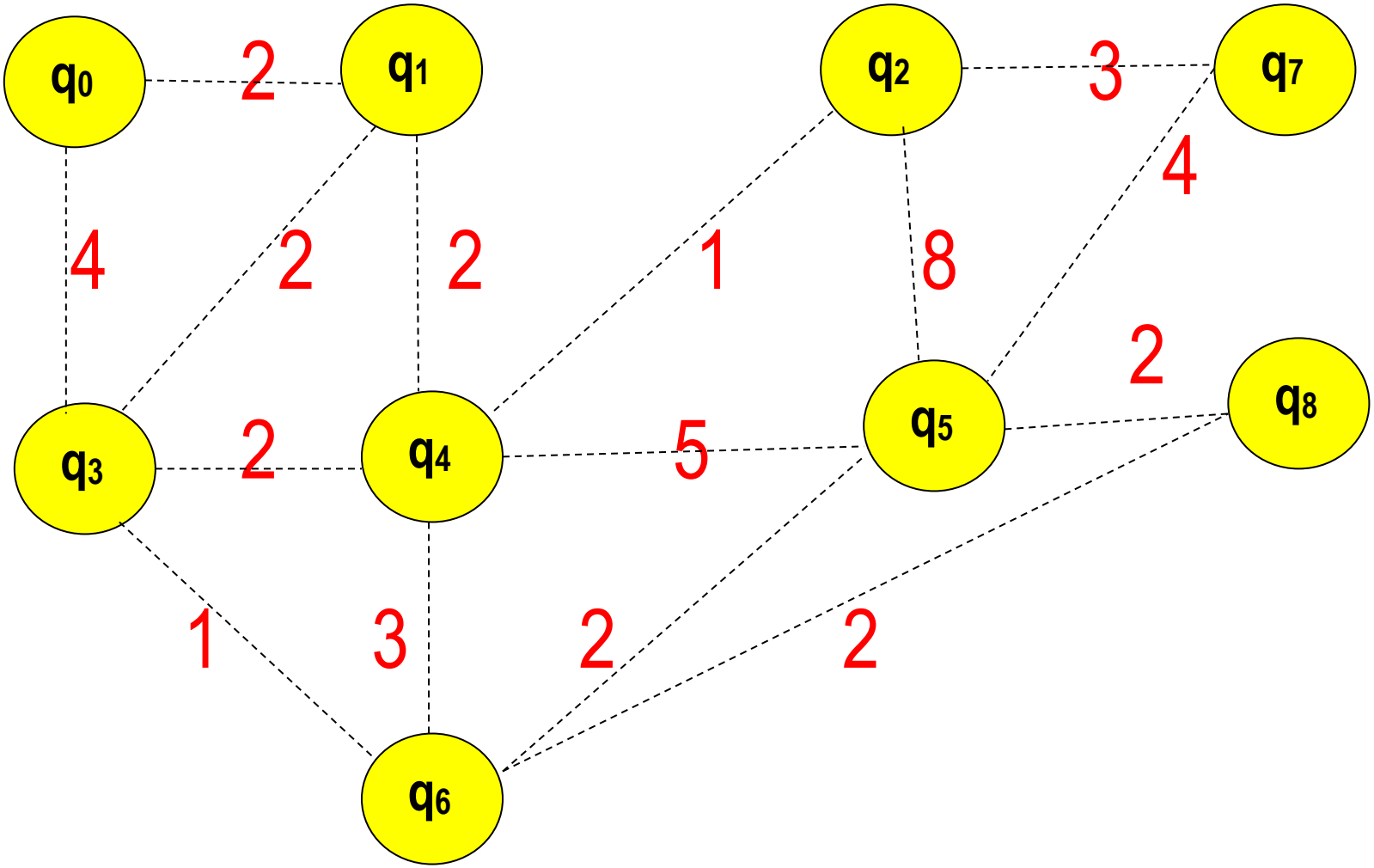
Graphe original

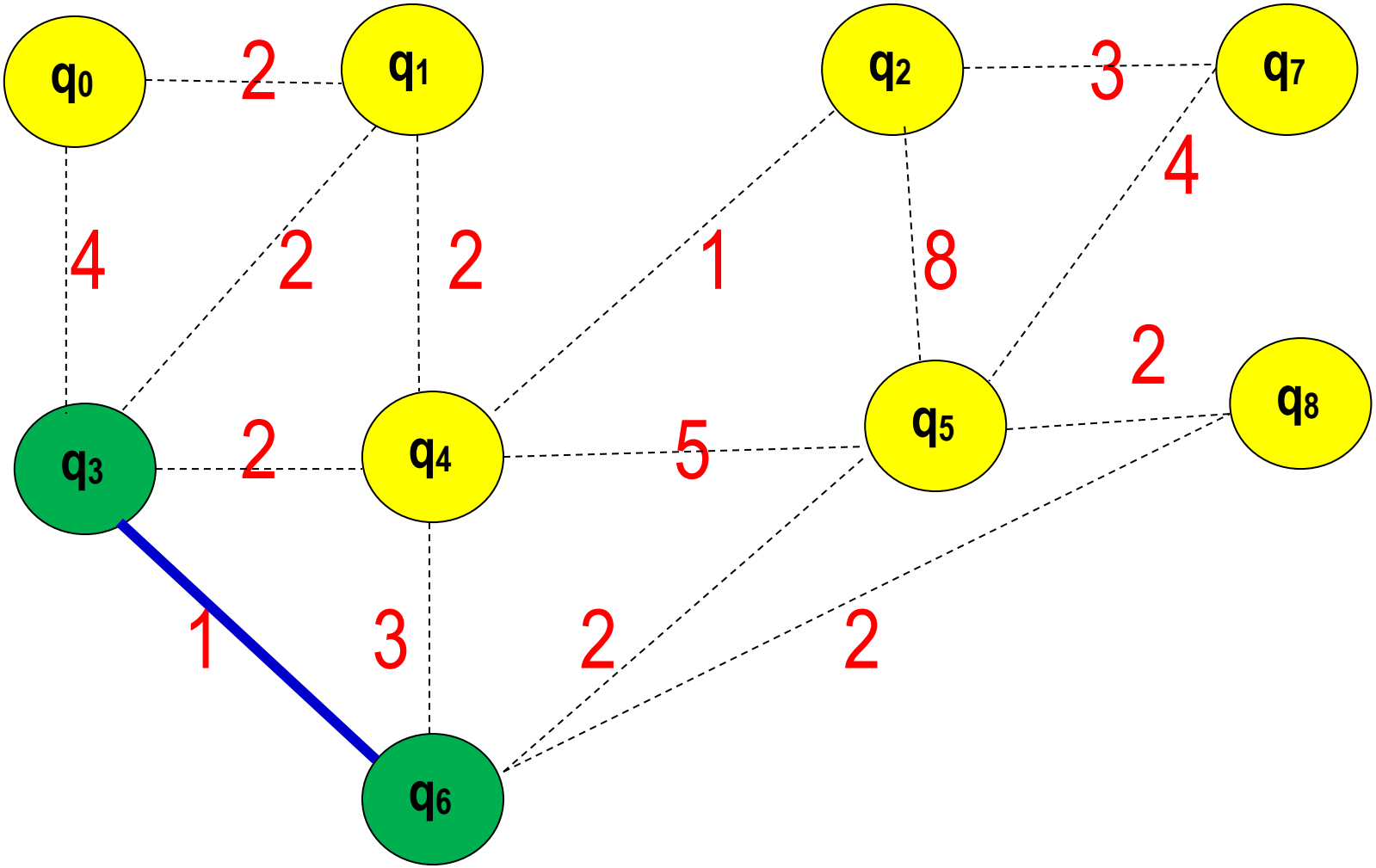


Forêt initiale selon Kruskal

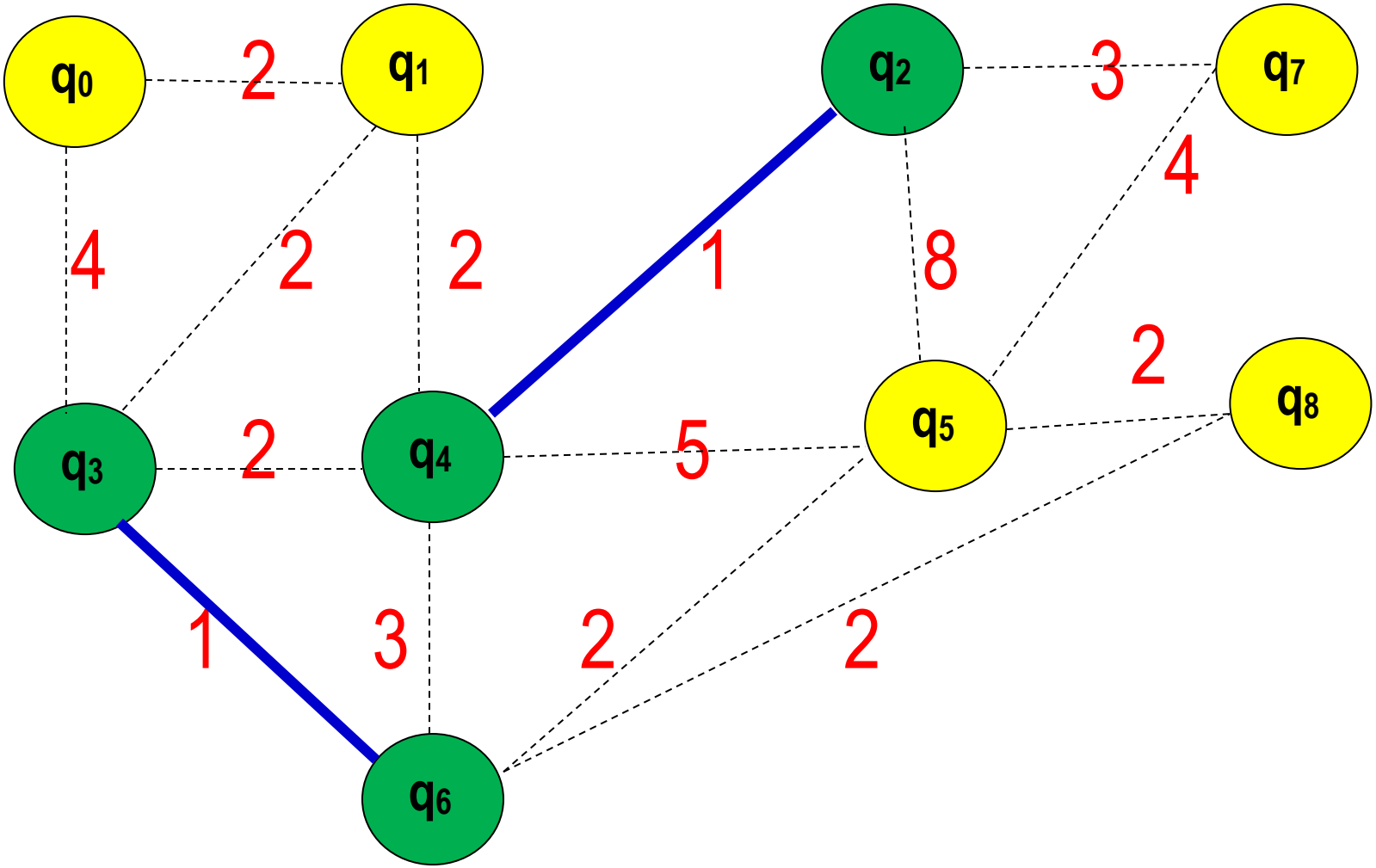


Au départ : graphe **vide**

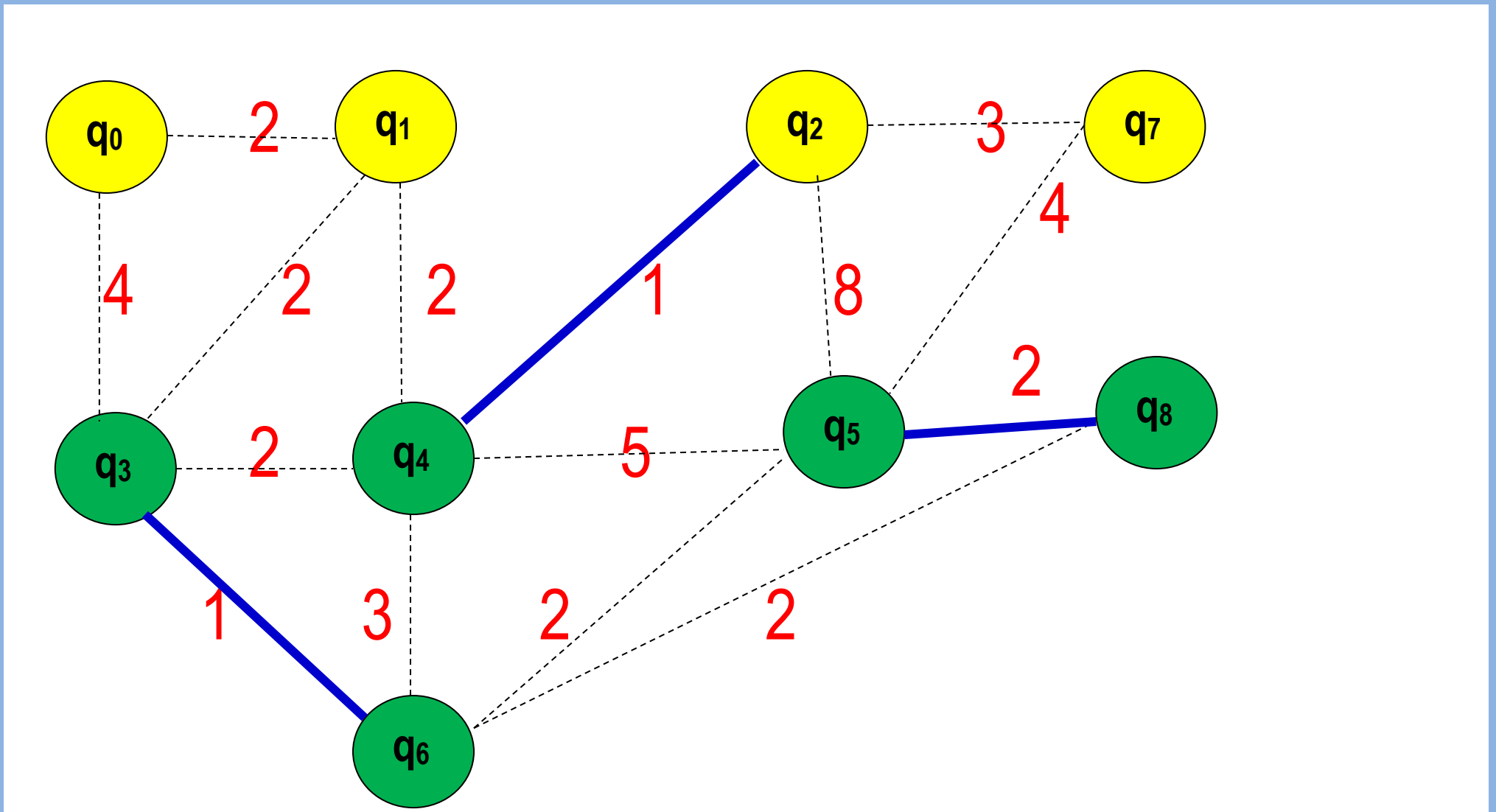




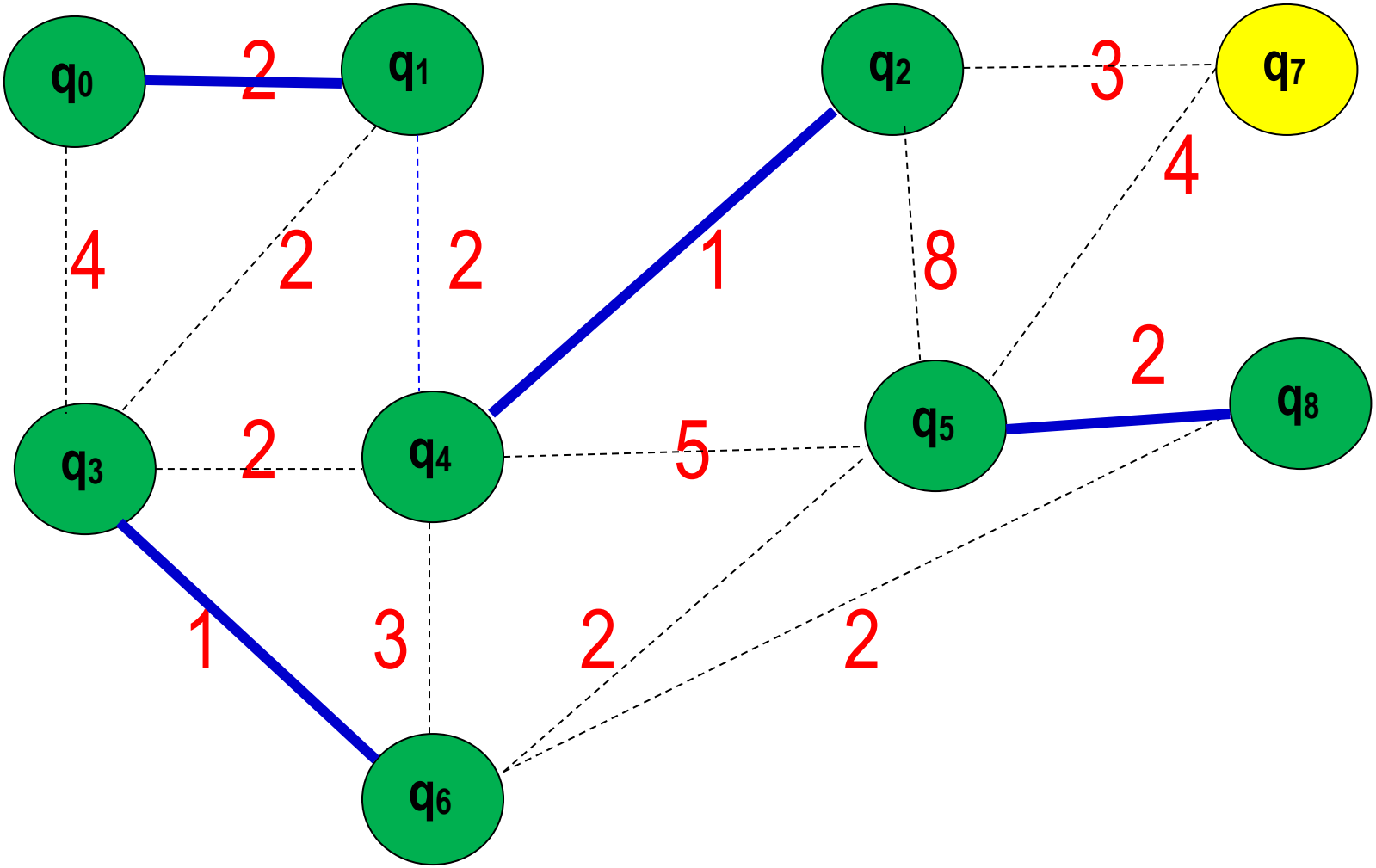
Forêt après ajout de arête (q_3,q_6)



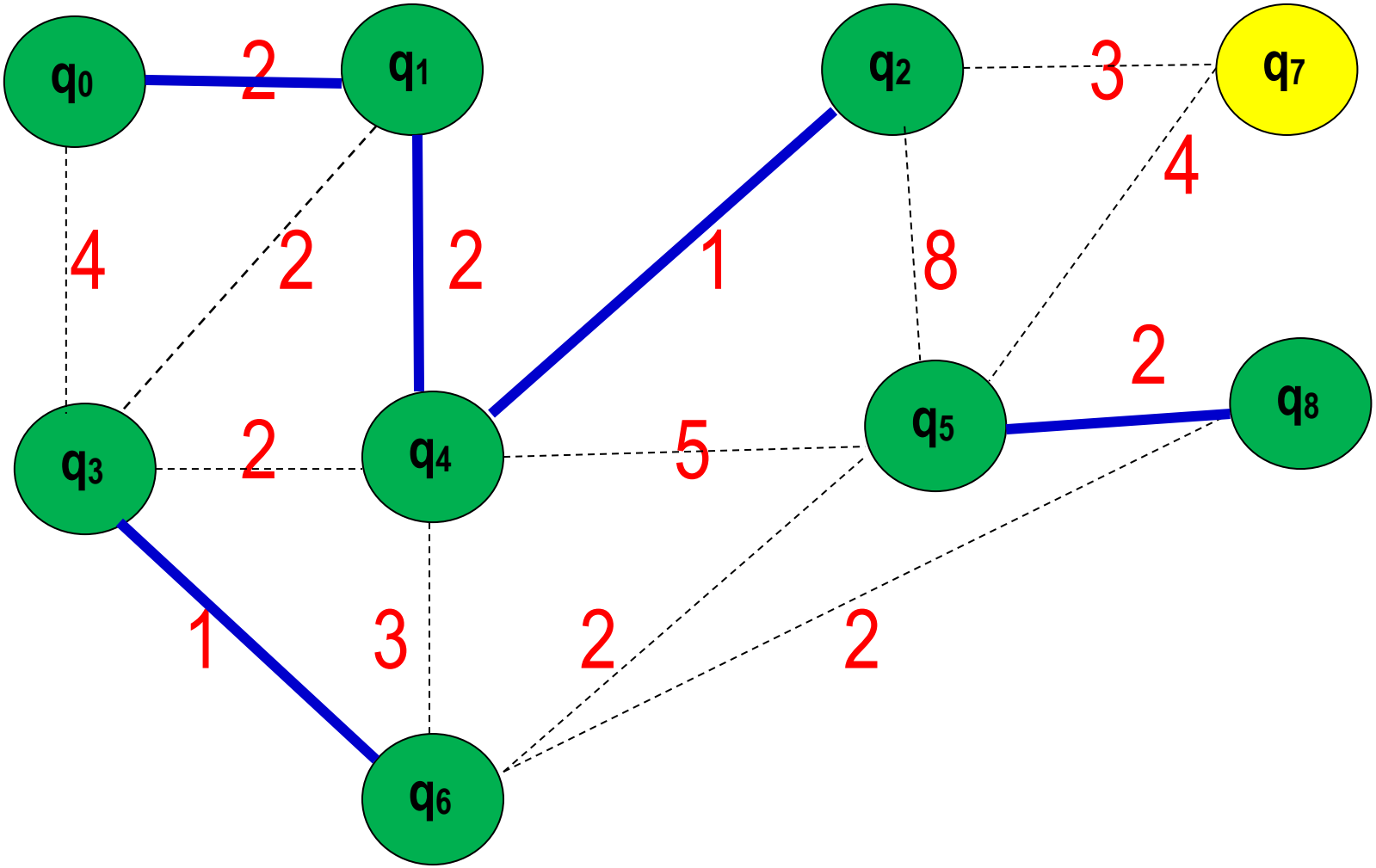
Forêt après ajout de arête (q_2,q_4)



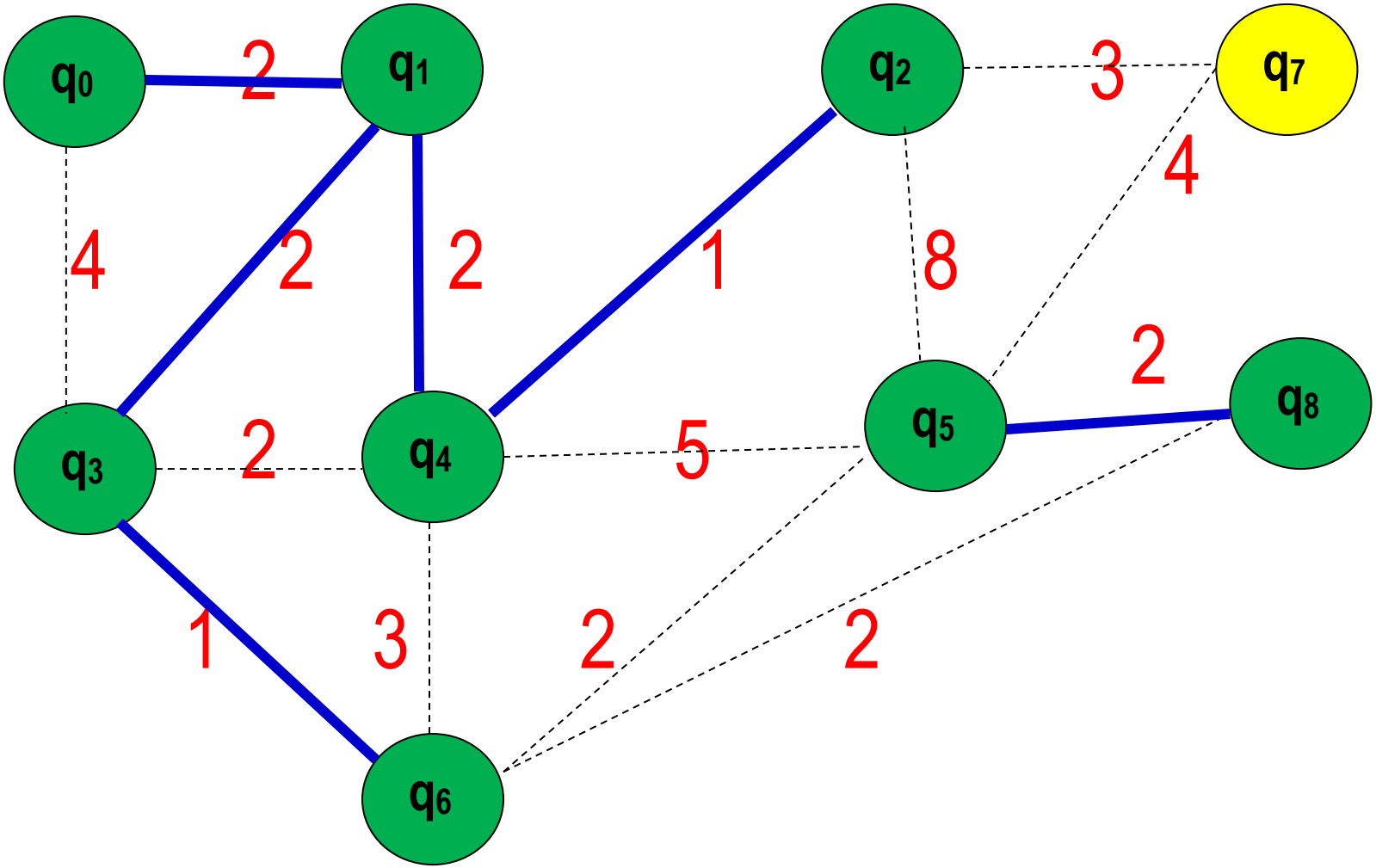
Forêt après ajout de arête (q_5,q_8)



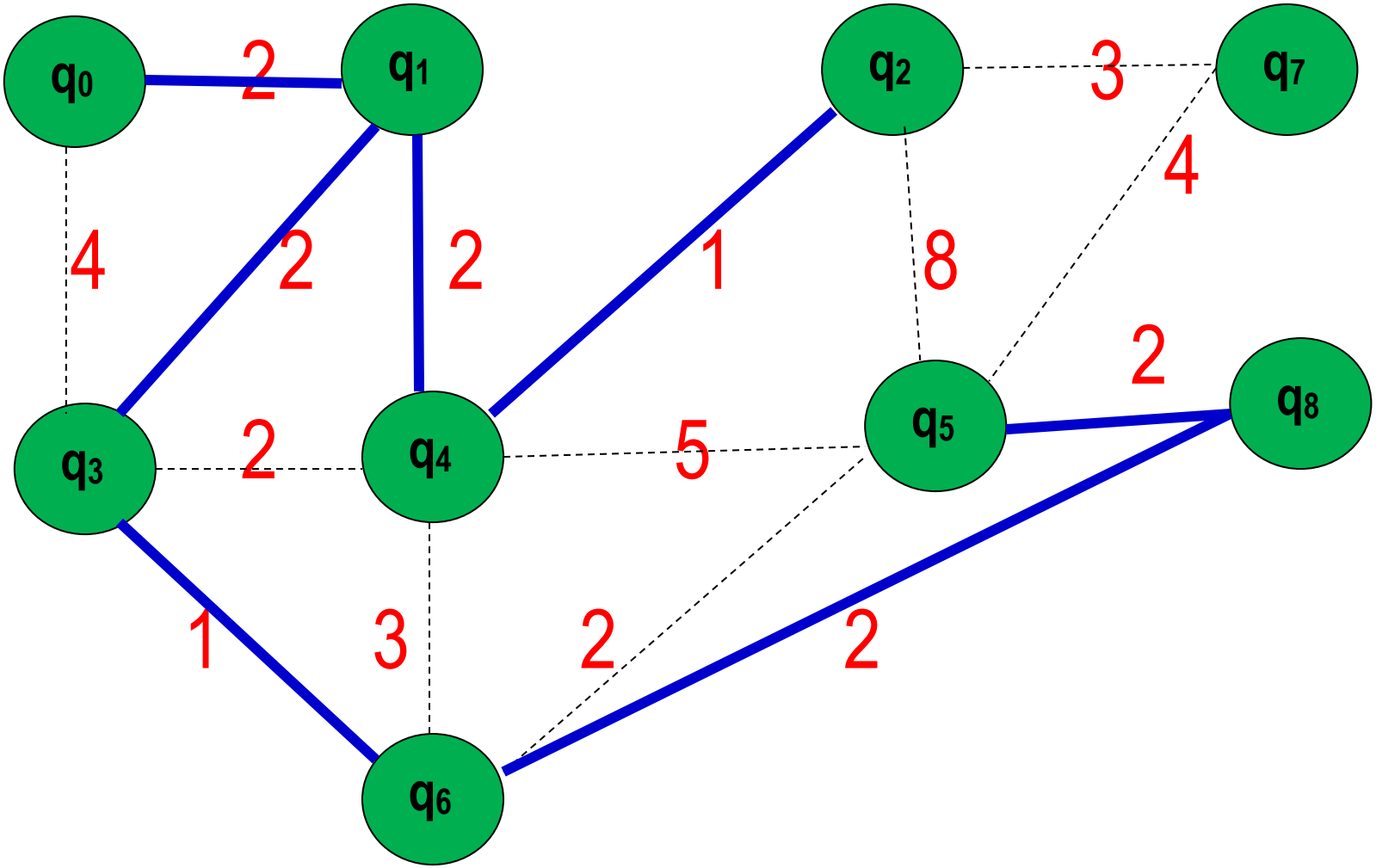
Forêt après ajout de arête (q_0,q_1)



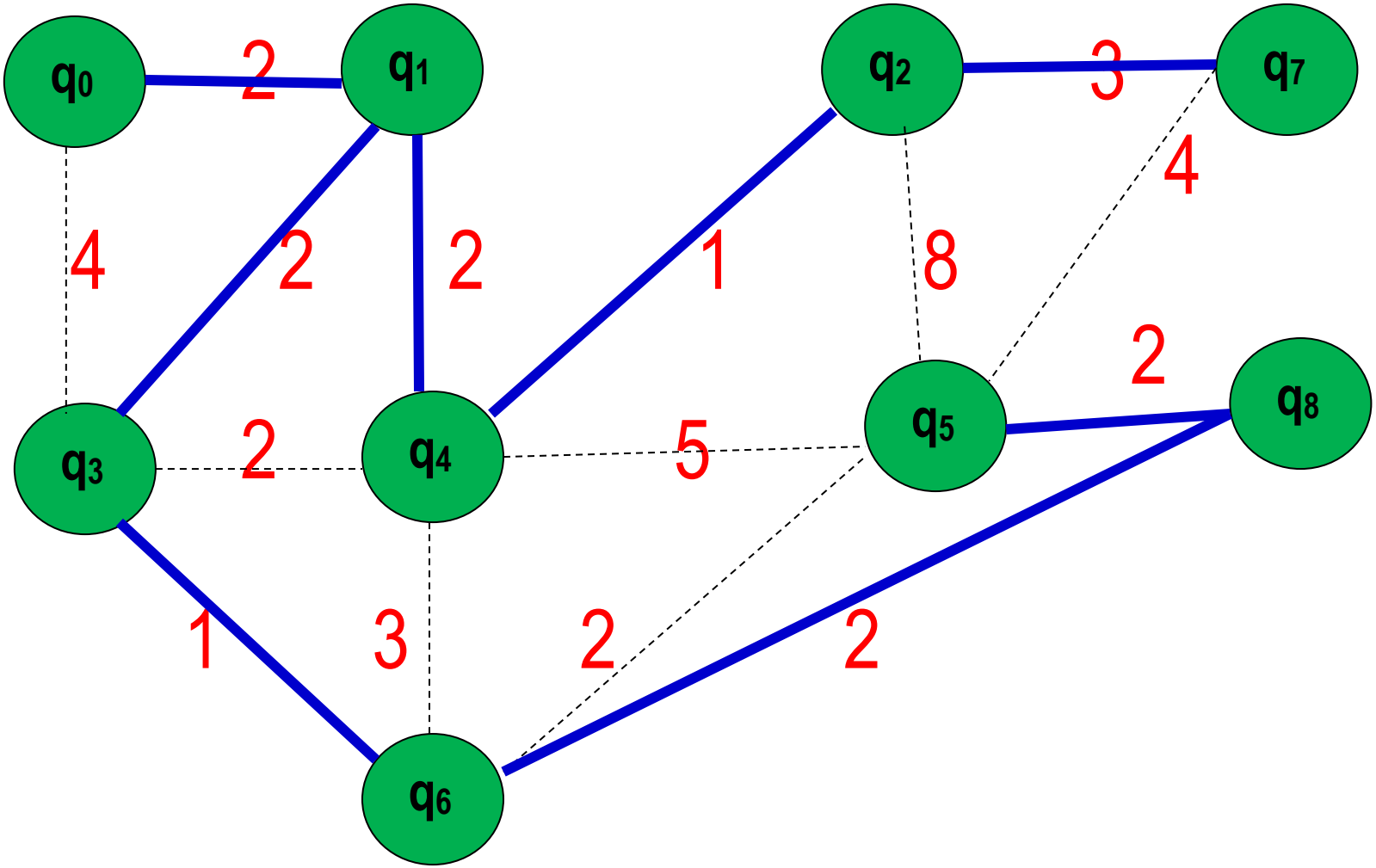
Forêt après ajout de arête (q_1,q_4)



Forêt après ajout de arête (q_1,q_3)

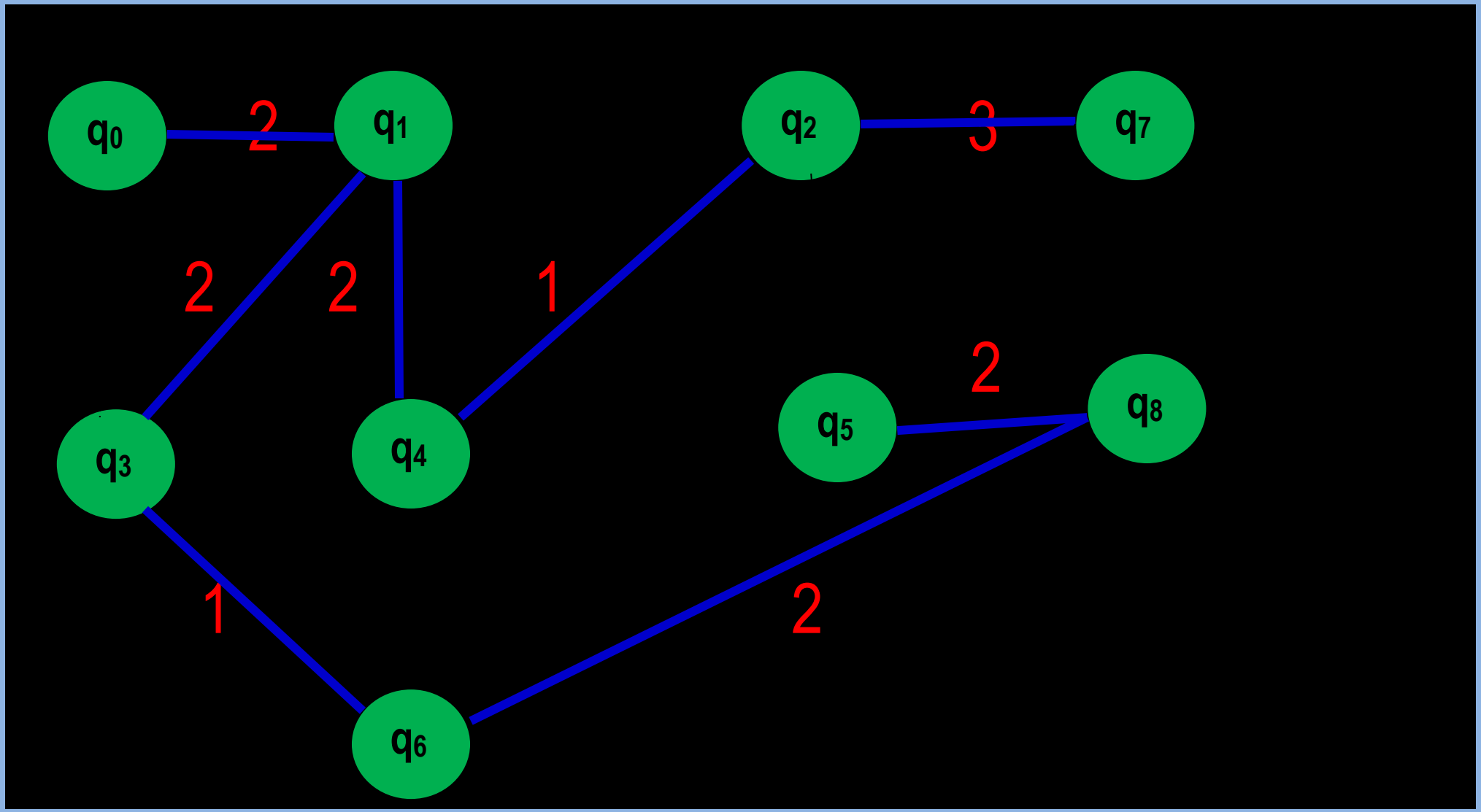


Forêt après ajout de arête (q_6,q_8)



Forêt après ajout de arête (q_2,q_7)

Arbre couvrant minimum



Principe

L'algorithme impose d'abord de **trier les m arêtes** de G par **ordre croissant** de leur coût.

Soit σ la suite induite:

$$\sigma = a_1, a_2, \ldots, a_i, a_{i+1}, \ldots, a_m$$

La procédure pour construire l'arbre de recouvrement minimum part d'un **graphe vide** G':

G'← **GrapheVide()**

Ensuite, les arêtes sont considérées, une par une, dans l'**ordre du tri**.

Si l'ajout d'une arête a_i dans G' n'introduit pas de **cycle:**

- alors on l'ajoute : G'← **AddArc(x,y, a_i ,G)**
- sinon, on passe à l'arête suivante a_{i+1}.

Ainsi l'arbre de recouvrement minimum est construit, **progressivement**, par ajout d'arêtes.

Lorsqu'on a ajouté **n-1** arêtes, **sans créer de cycle**, on a fini de construire l'arbre de recouvrement minimum.

Procédure

Procédure: **Kruskal**
Entrées: G = (S,A) : GRAPHE.
Sortie: G' = (S,A') : GRAPHE

Kruskal(G: GRAPHE) G': GRAPHE

Début

/*Initialisation */

n← |S| ; m← |A| ;

trier(A) */*trier les m arêtes de G dans l'ordre croissant des coûts;*/*

/ On les notera* : $a_1, a_2, \ldots, a_m$ avec: $c(a_1) \leq c(a_2) \leq \ldots c(a_{m-1}) \leq c(a_m)$ **/*

A' ← ∅; */* on part d'un graphe* G' vide **/*

pour i ← 1 **à** m

si A' ∪ {a_i} ne génère pas de cycle

alors A' ← A' ∪ {a_i};

fin_si

fin_pour;

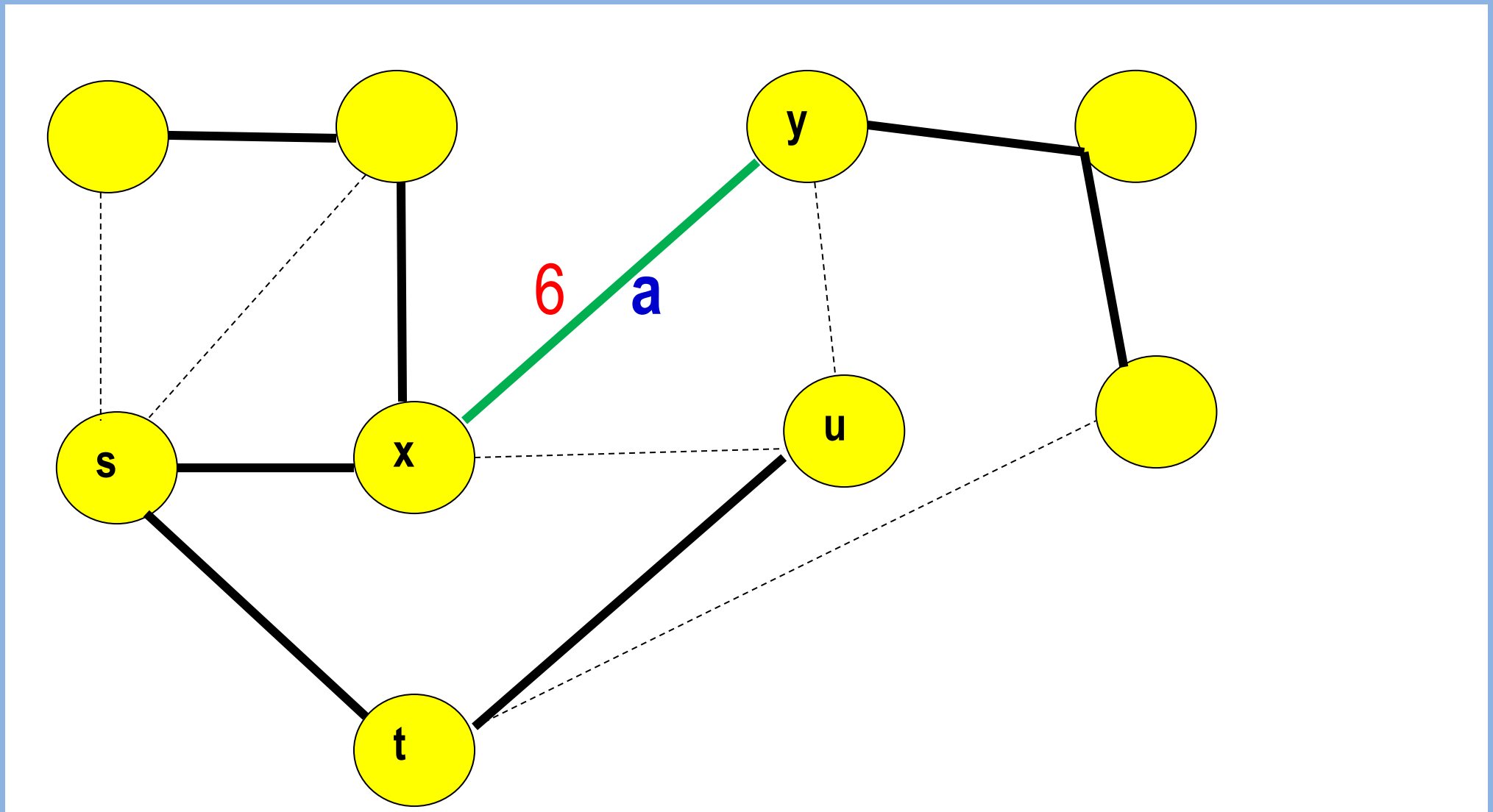
Fin

Justification de l’algorithme

Supposons que l'algorithme avait effectué quelques itérations pour construire l’arbre **G’**.

Considérons maintenant l'ajout de la prochaine arête **a** = (**x**,**y**) dans l'arbre G’.

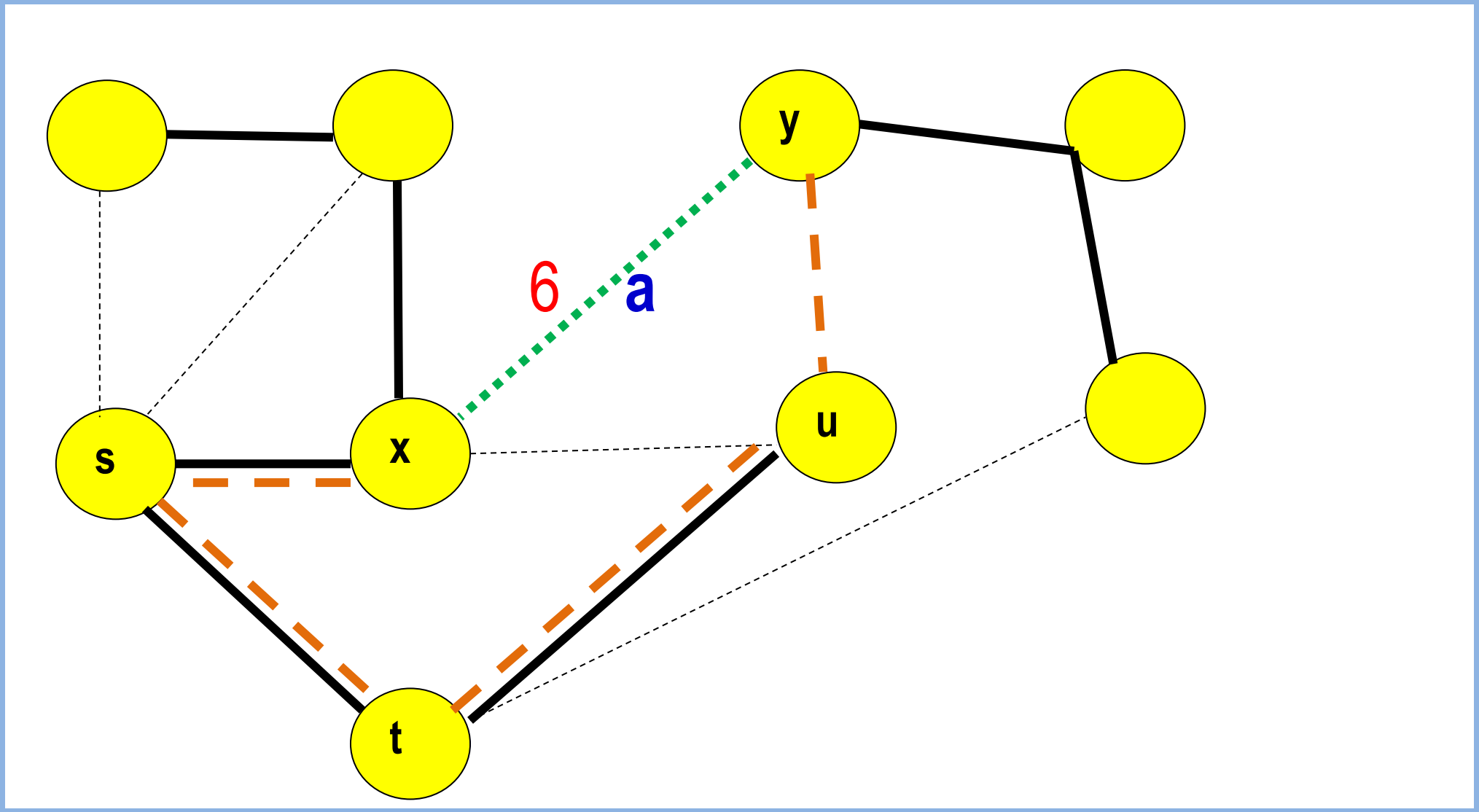
On suppose que cet ajout de **a** n'introduit pas de cycle: G’ demeure alors un arbre.


y
6
a
s
x
u
t

Cependant, est-on certain que **a** garantit la construction de l'arbre avec un **coût minimum** ?

En fait, **x** et **y** doivent être connectés d'une manière ou d'une autre.

Car dans un arbre, chaque paire de sommets distincts est reliée par une seule **chaîne simple**.(propriété P5)


y
6
a
s
x
u
t

Si ce n'est pas **a** qui relie **x** et **y**, ce sera une certaine chaîne C.

Soit :

$$\boldsymbol{C} = (x,s,t,u,y)$$

la chaîne qui relie *x* à *y* .

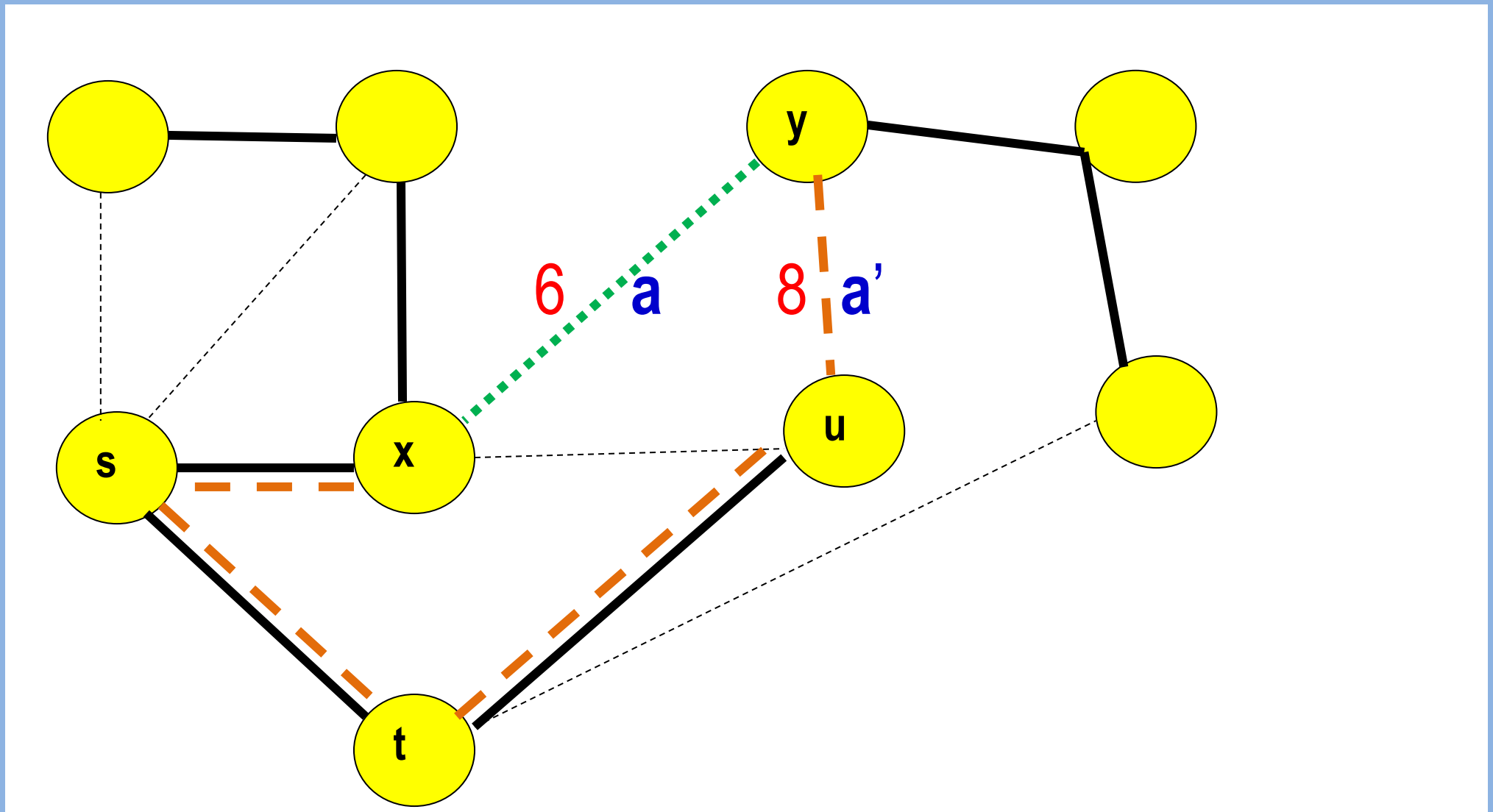
Comme ***a*** n'introduit pas de cycle cela signifie que la chaîne **C** n'est pas encore construite: **elle ne le sera que plus tard** !

Cela signifie qu'une arête, **au moins**, de cette chaîne, soit **a'**, n'a pas été encore prélevée de la liste

$$\sigma = a_1, a_2, \ldots, a_i, a_{i+1}, \ldots, a_m$$

Donc **a' qui** sera choisie ultérieurement à **a** a un coût supérieur ou égal à celui de **a** :

$$\text{cout}(\mathbf{a'}) \geq \text{cout}(\mathbf{a})$$


y
6
a
8
a’
s
x
u
t

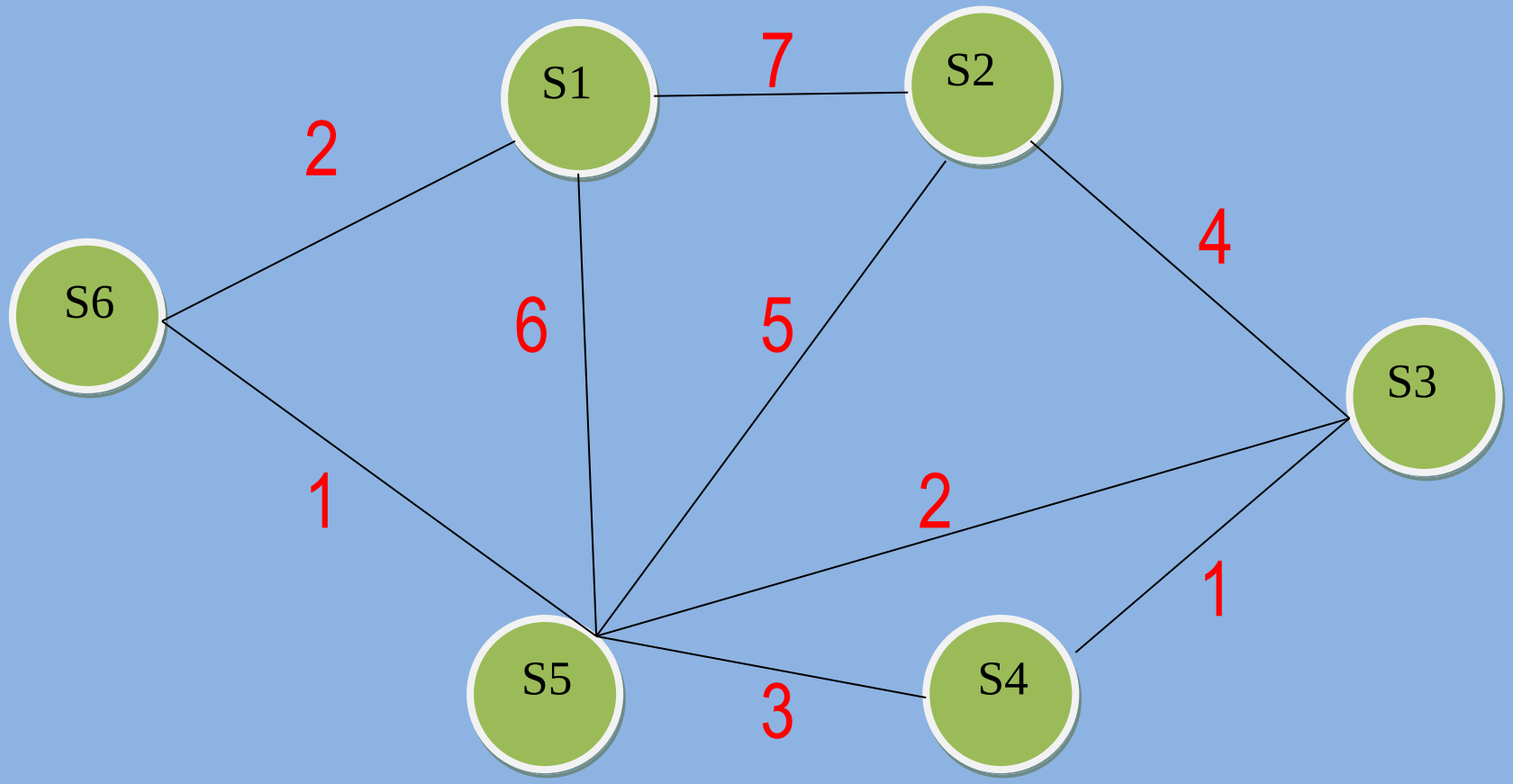
A fortiori, le fait de choisir la chaîne ***C*** pour connecter ***x*** à ***y*** est donc plus coûteux que de connecter ***x*** à ***y*** par ***a***.

En conclusion, l'arête choisie ***a*** garantit d'obtenir un arbre de **coût plus faible**.

Cela **justifie** le choix de ***a*** et donc la démarche globale de l'algorithme de Kruskal.

Exemple

Soit le graphe non orienté valué connexe:



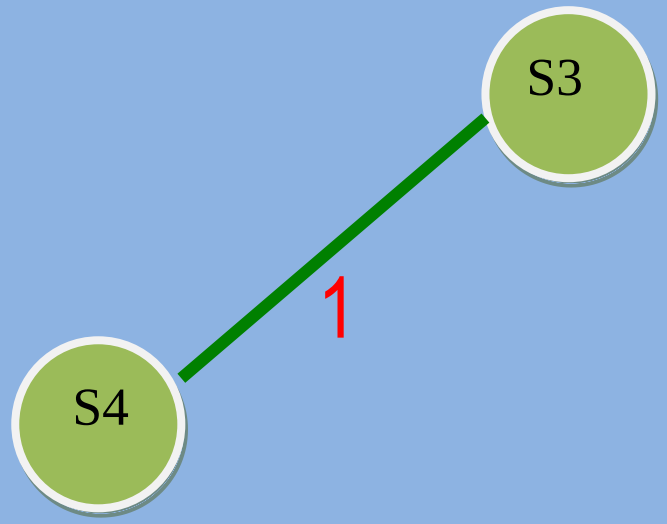
La liste des arêtes est triée dans l'ordre de coût croissant:

$$\sigma = [(S_3, S_4), (S_5, S_6), (S_1, S_6), (S_3, S_5), (S_4, S_5), (S_2, S_3), (S_2, S_5), (S_1, S_5), (S_1, S_2)]$$

On sélectionne l’arête de (S3,S4) de coût:

c(S3,S4) = 1

On l’ajoute car il n’y pas de cycle.



On sélectionne ensuite l'arête (S5,S6) de coût :

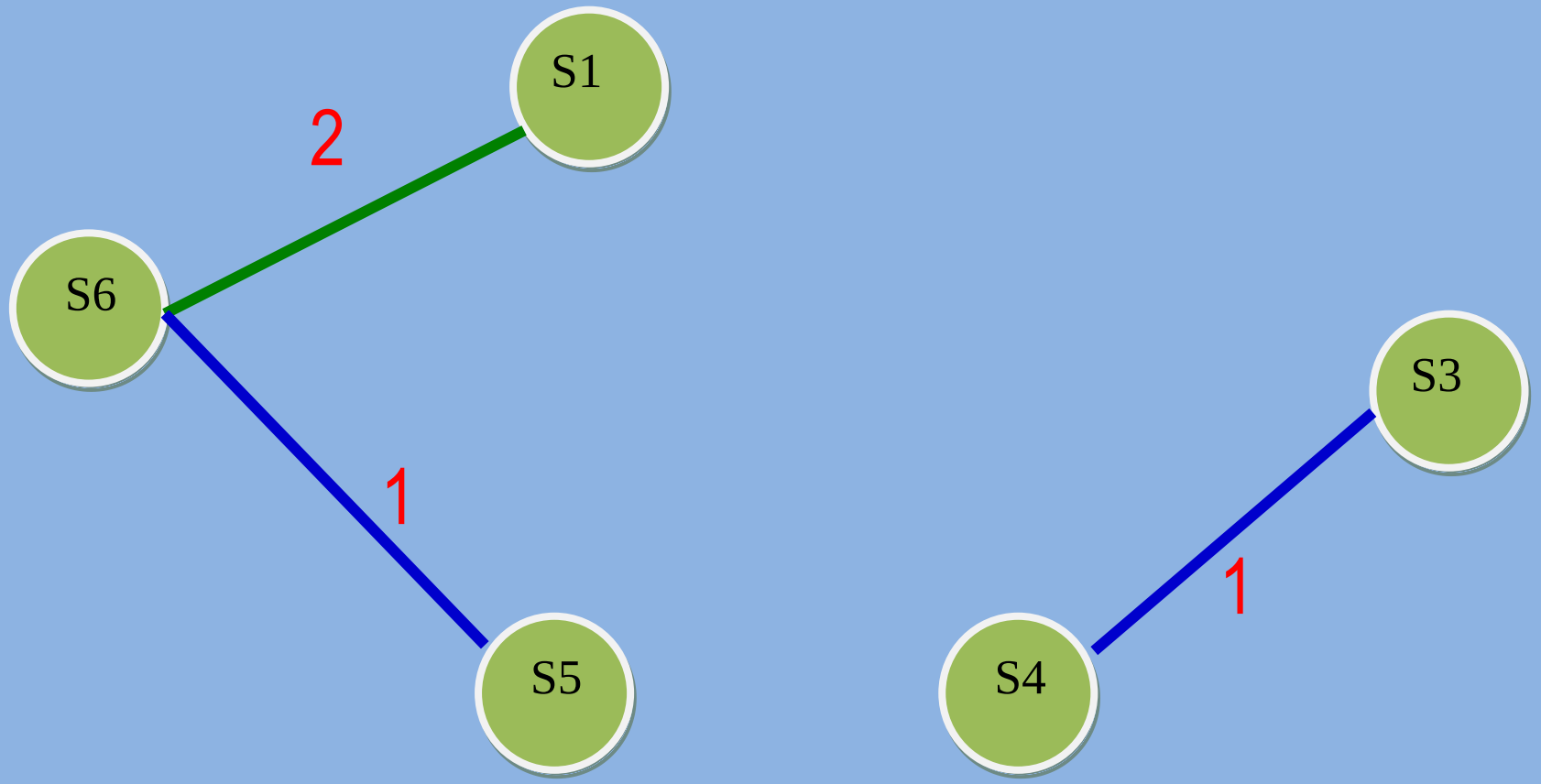
c(S5,S6) = 1

Pas de cycle, donc ajout de l'arête (S5,S6)

On sélectionne ensuite l'arête (S1,S6) de coût :

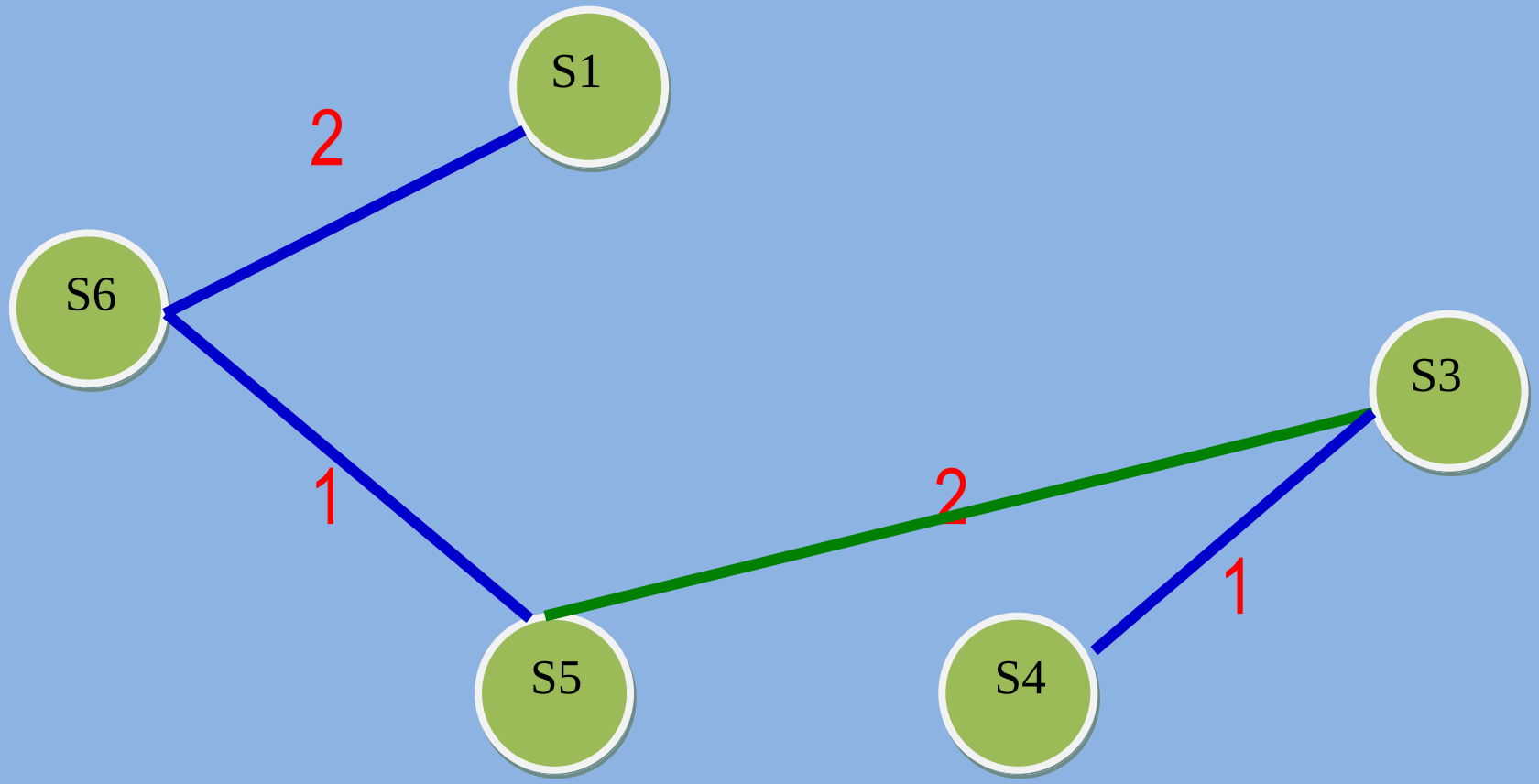
c(S1,S6) = 2

Pas de cycle, donc ajout de l'arête (S1,S6)



On sélectionne ensuite l'arête (S3,S5) de coût :
c(S3,S5) = 2

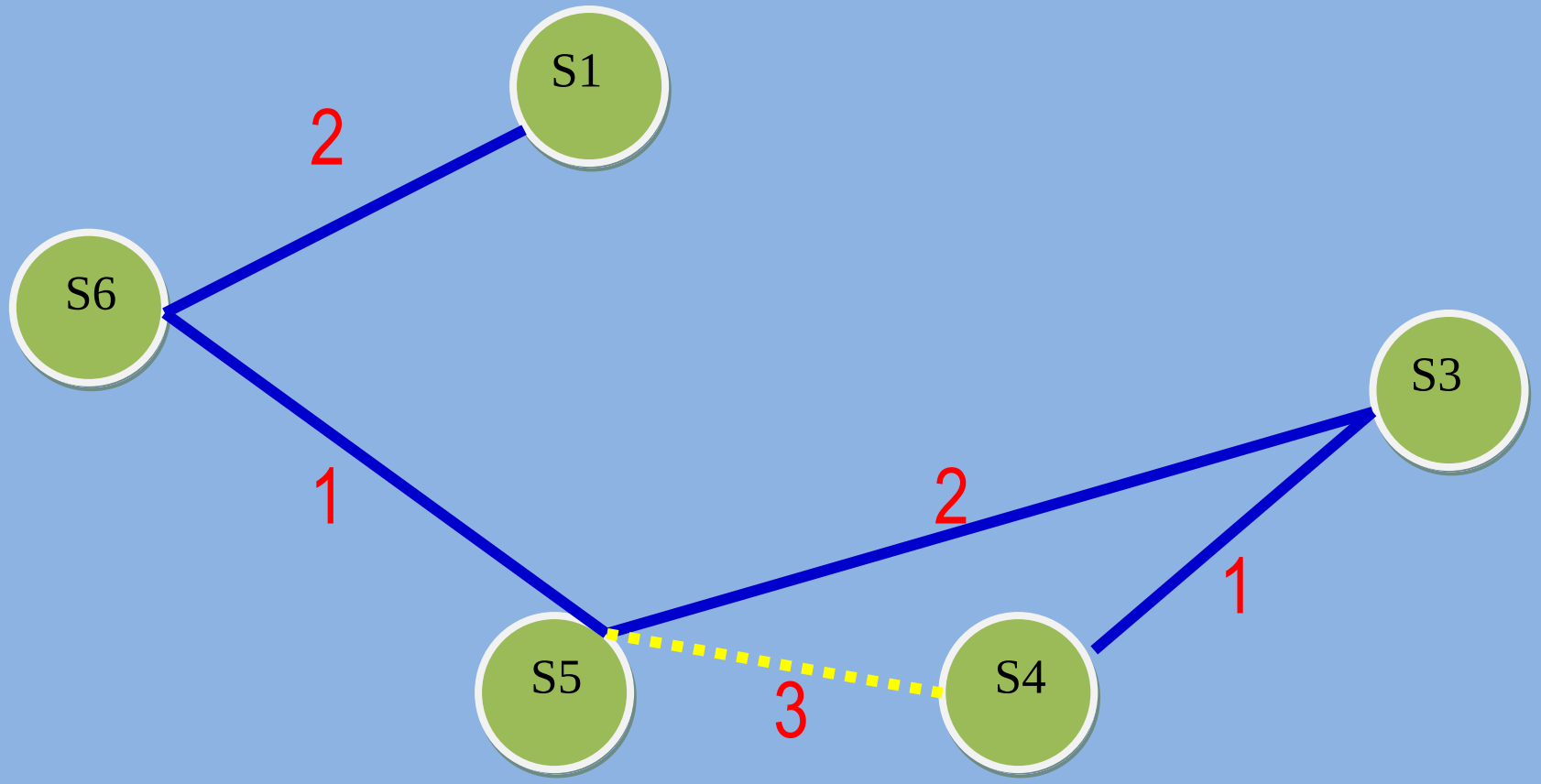
Pas de cycle, donc ajout de l'arête (S3,S5)



On sélectionne ensuite l'arête (S4,S5) de coût :

c(S4,S5) = 3

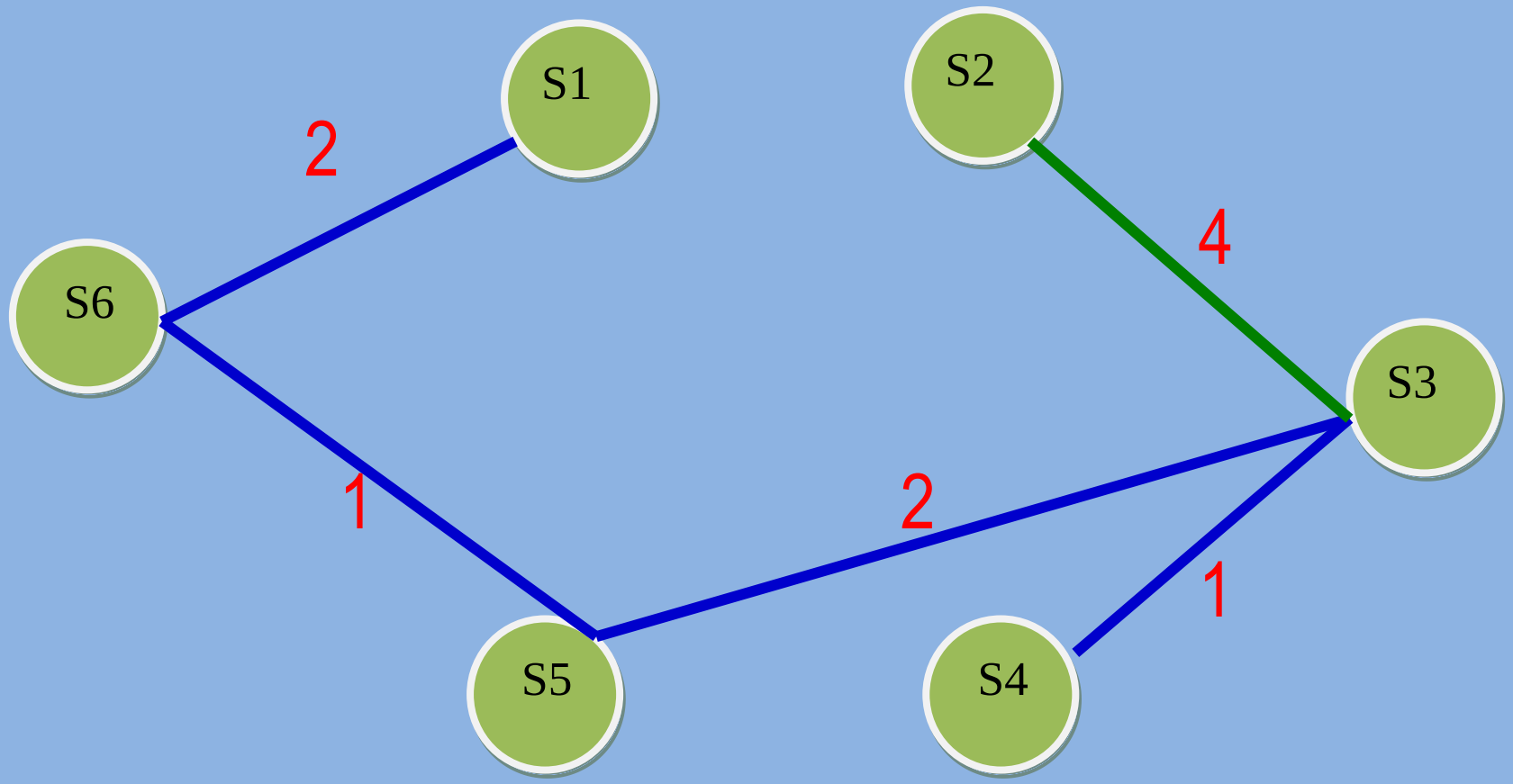
Formation d'un cycle donc pas d'ajout de l'arête (S4,S5)



On sélectionne ensuite l'arête (S2,S3) de coût :

c(S2,S3) = 4

Pas de cycle, donc ajout de l'arête (S2,S3)



A ce stade, la construction de l'arbre de recouvrement minimum est **terminée**.

Pourquoi ?:

Le nombre d'arêtes pouvant être ajoutées est m':

$$m' = n-1 = 6-1 = 5$$

2- Algorithme de PRIM

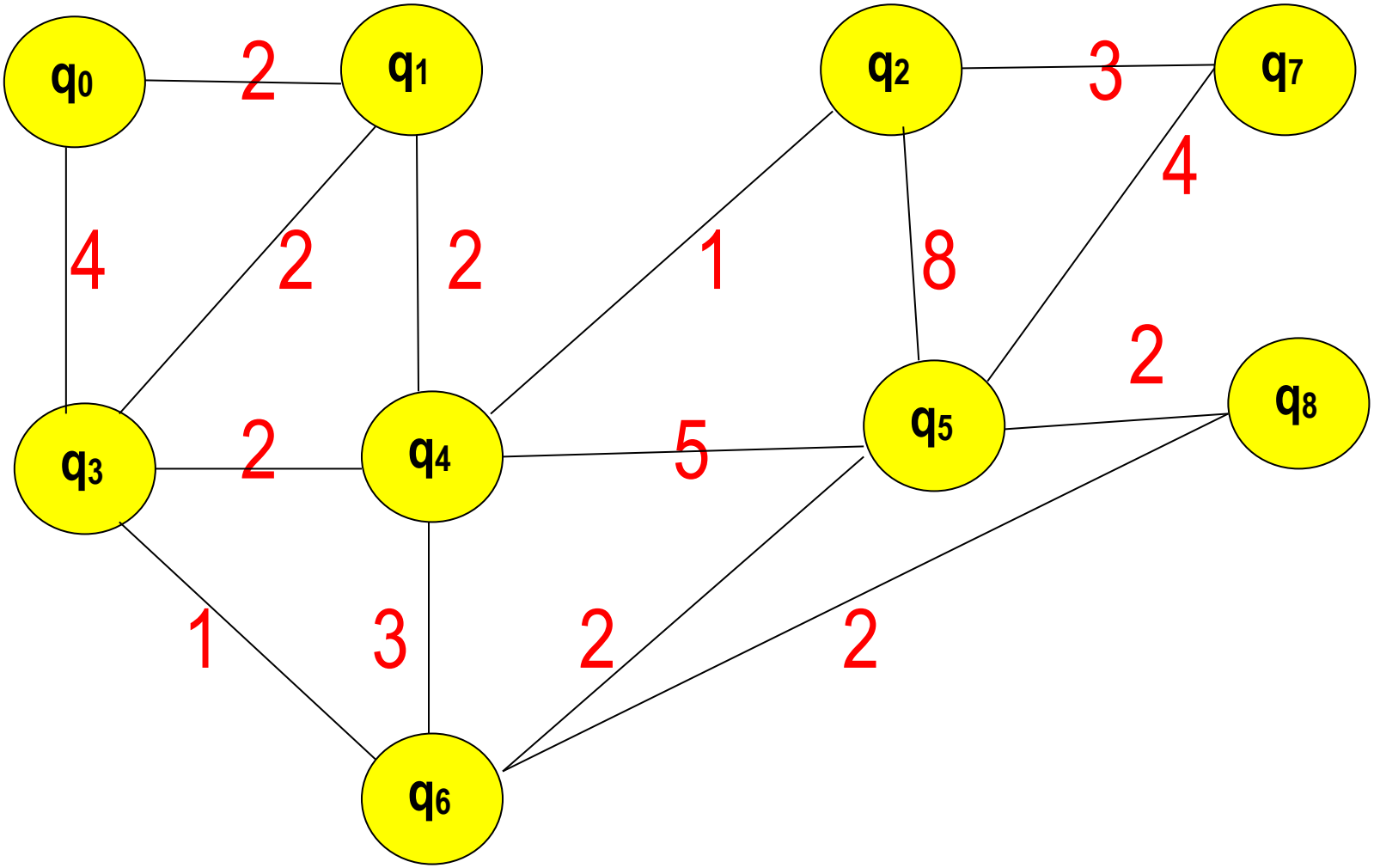
Idée

A chaque étape de la construction de G':

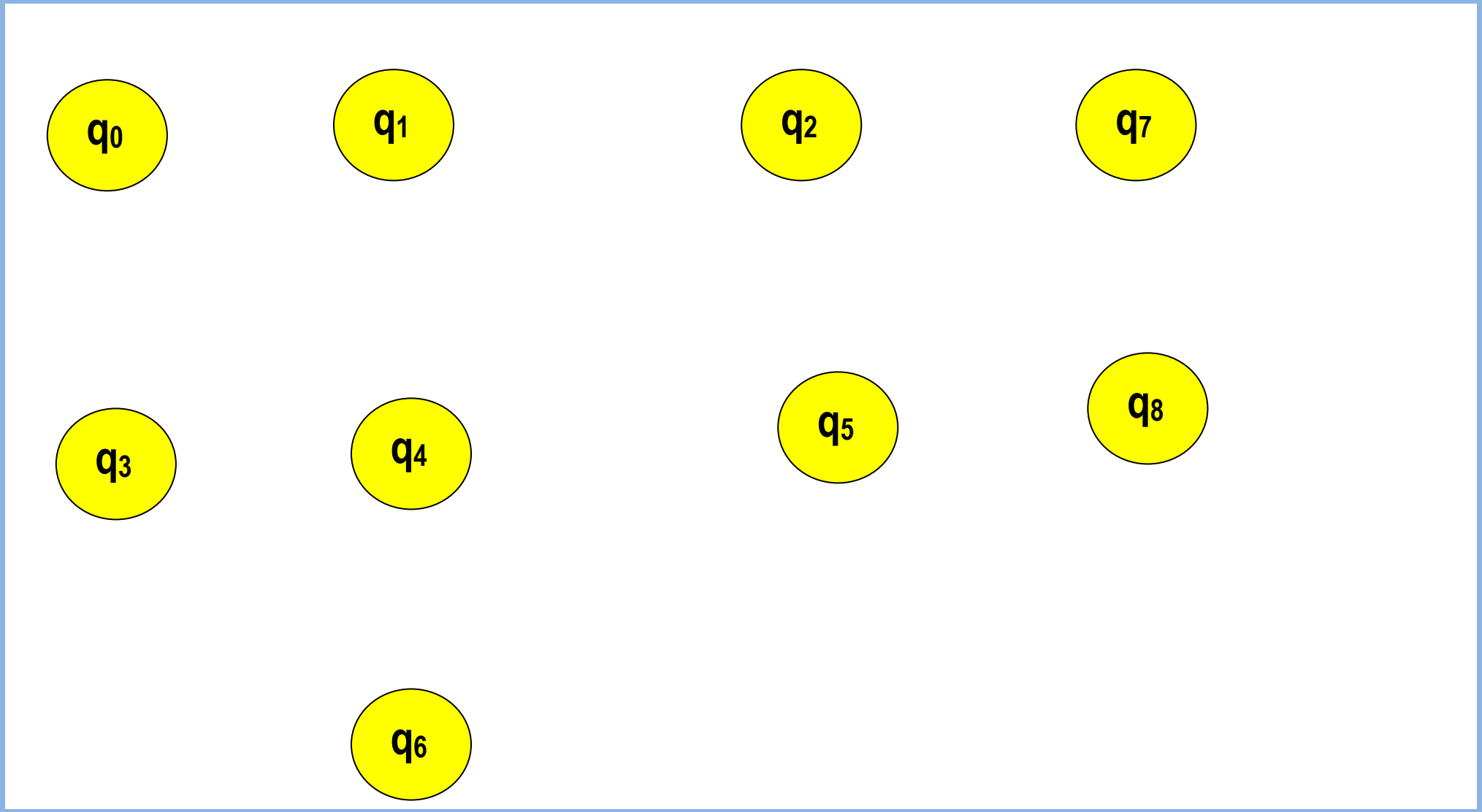
- G' est formé d'**un arbre** et un ensemble de **sommets isolés** ;

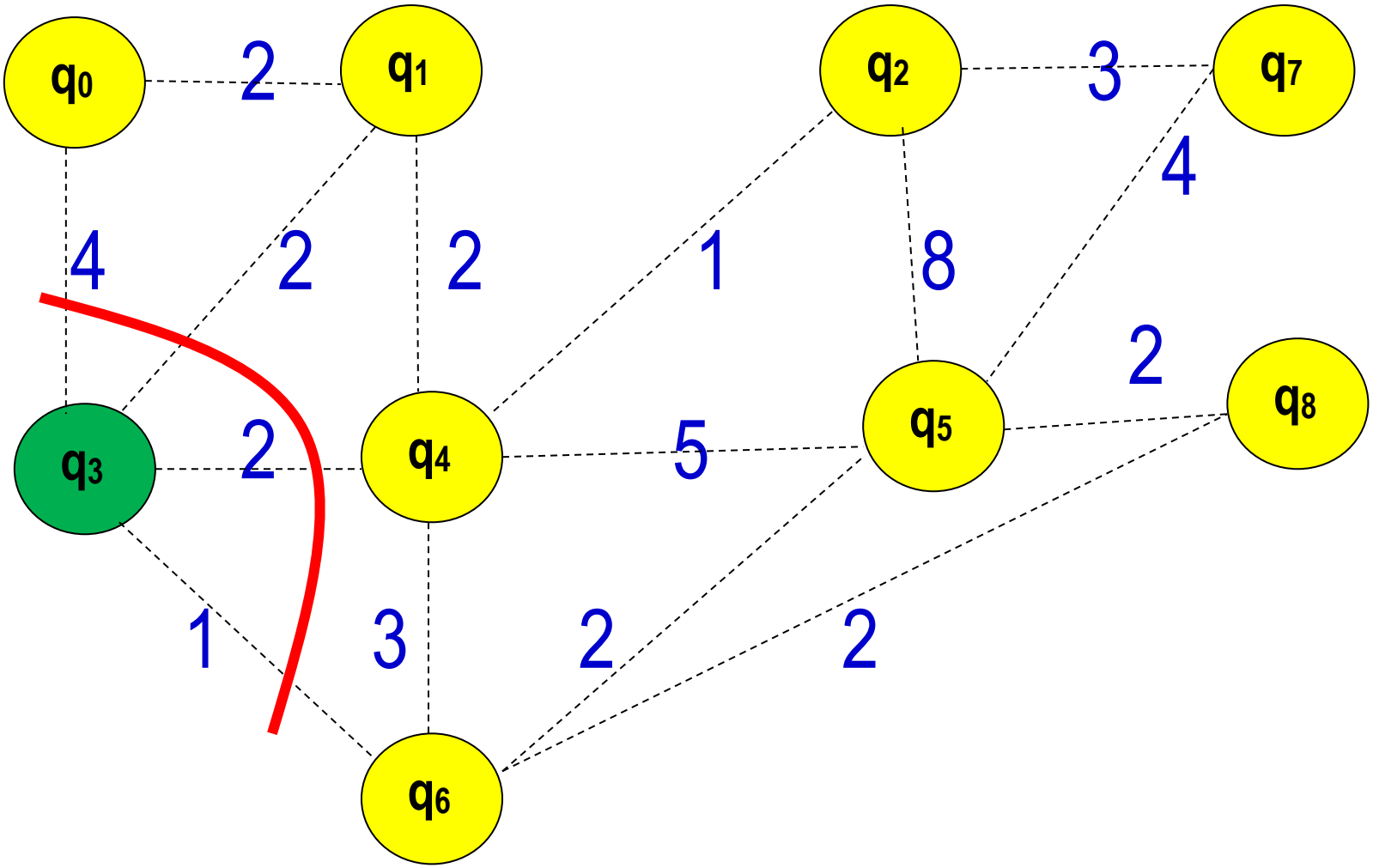
- l'algorithme doit choisir une arête de coût minimal qui relie l'arbre à l'un des sommets isolés.

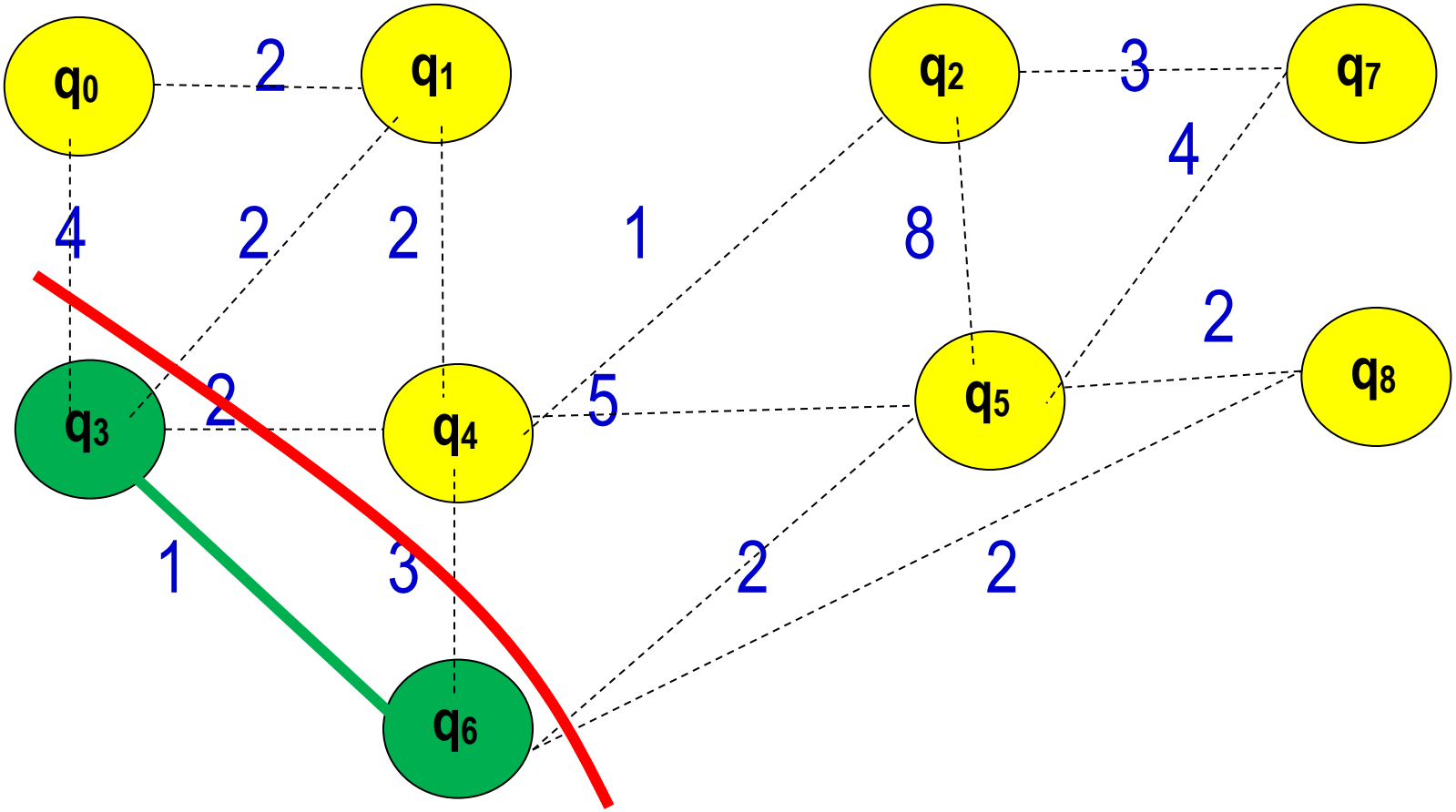
Graphe initial

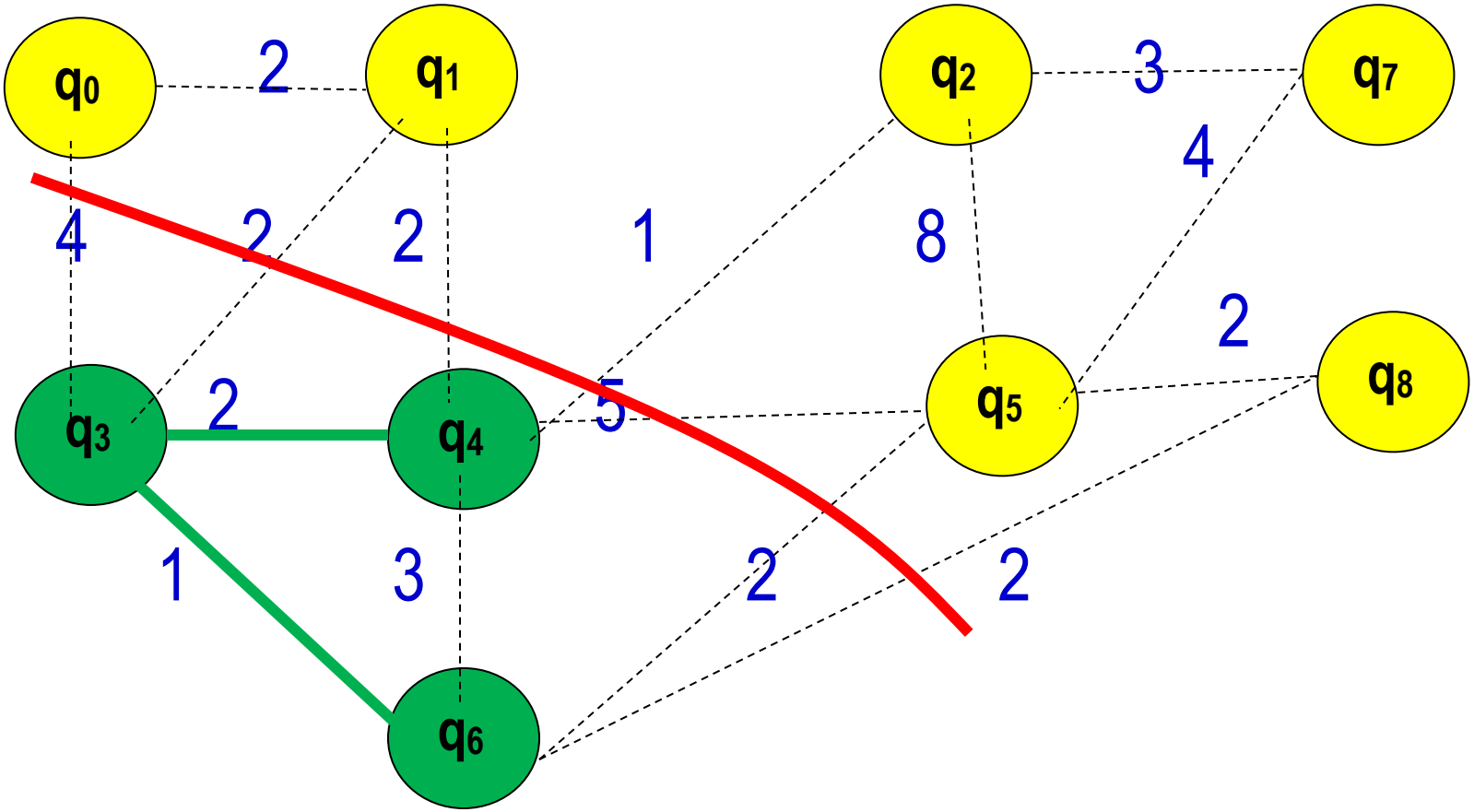


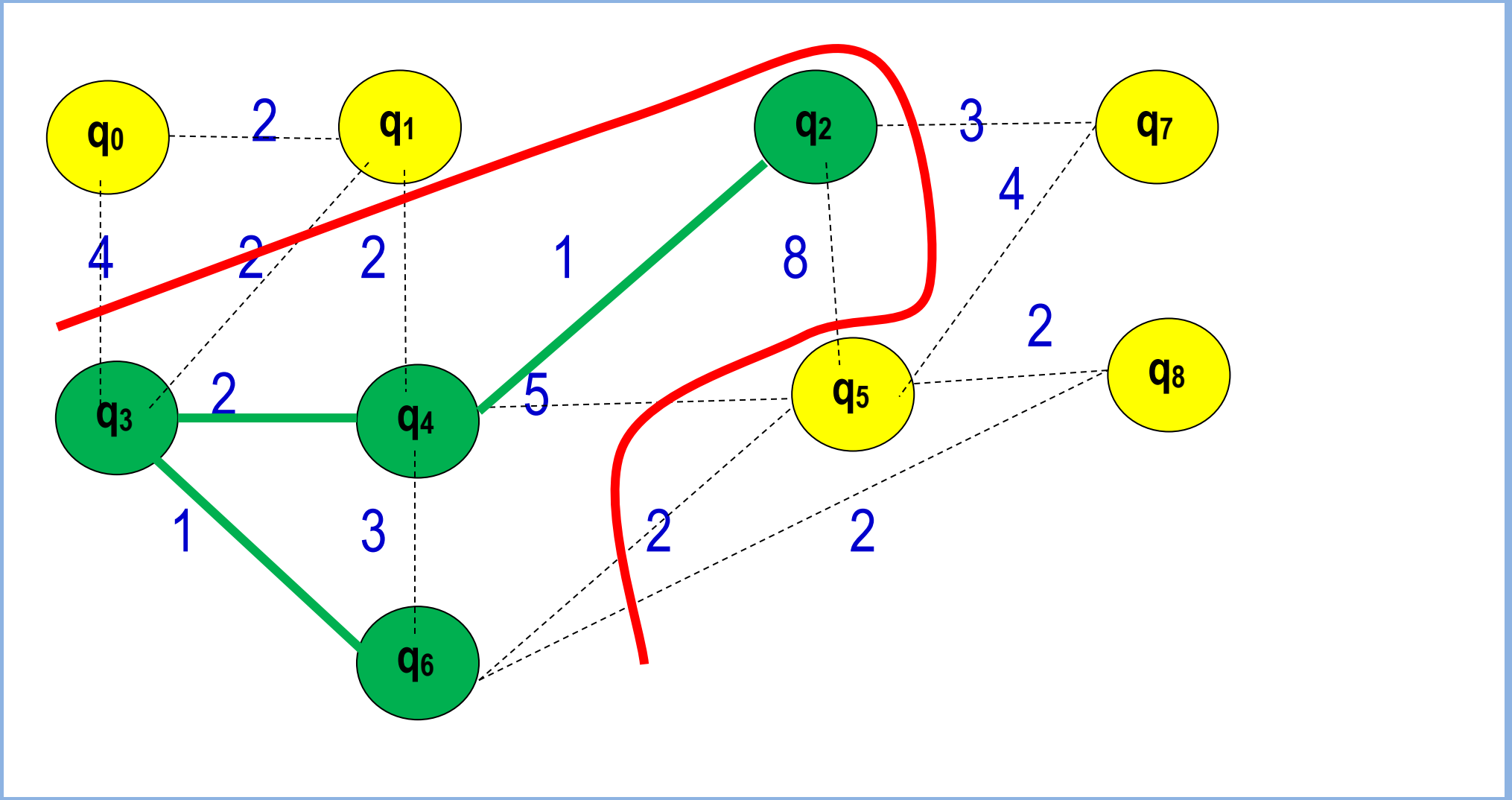
Graphe de départ selon Prim

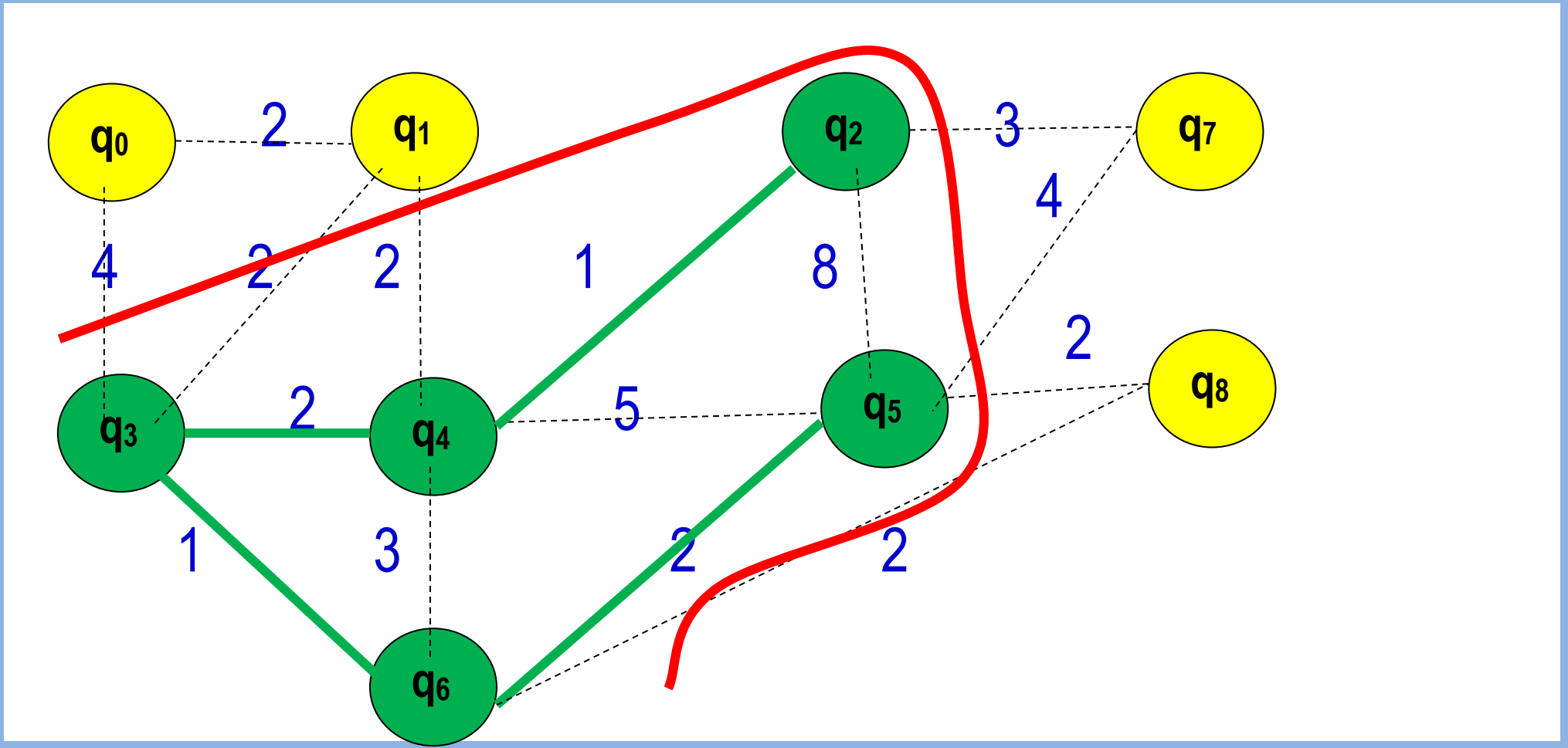


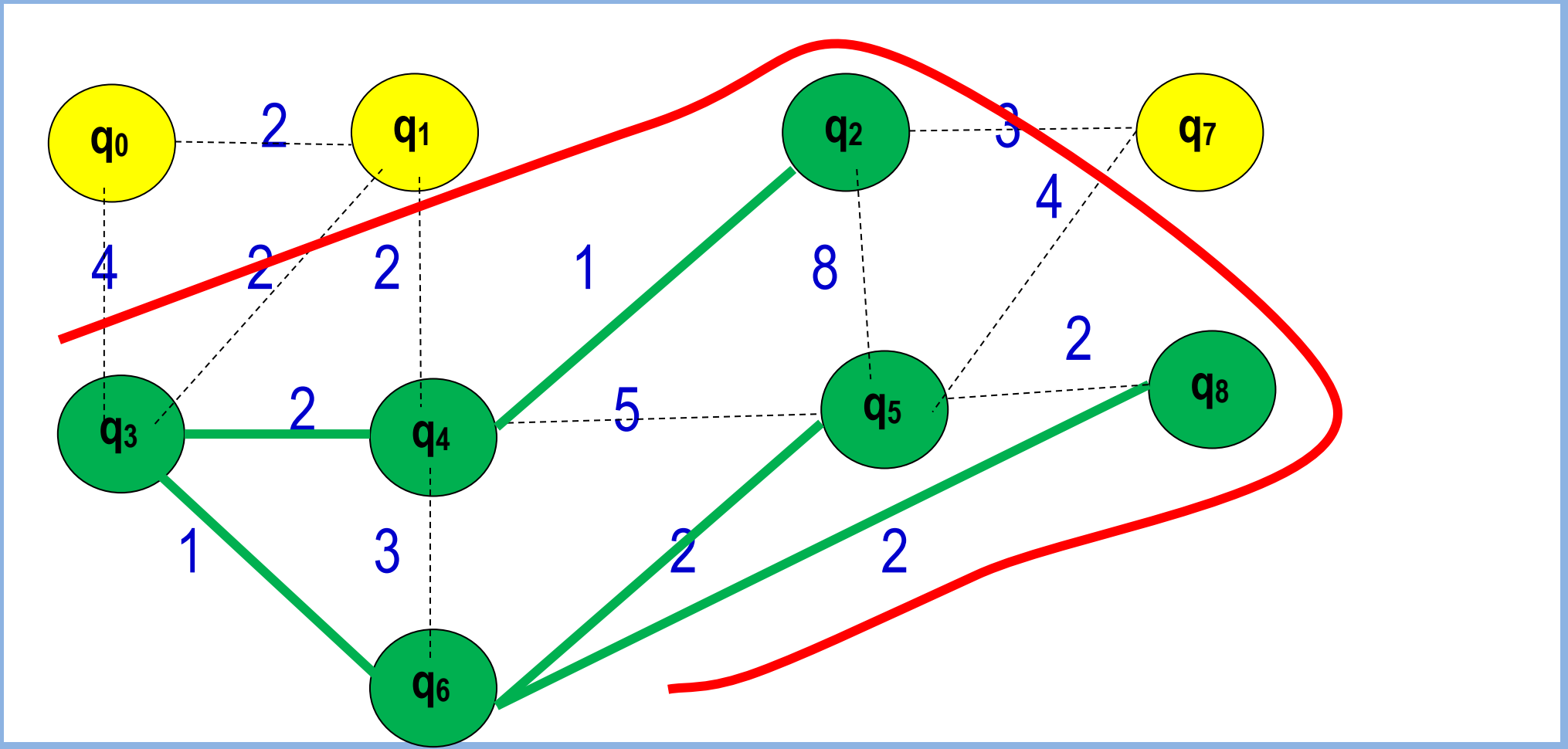

q0
q1
q2
q7
q3
q4
q5
q8
q6
2
4
2
2
1
8
4
2
2
5
1
3
2
2

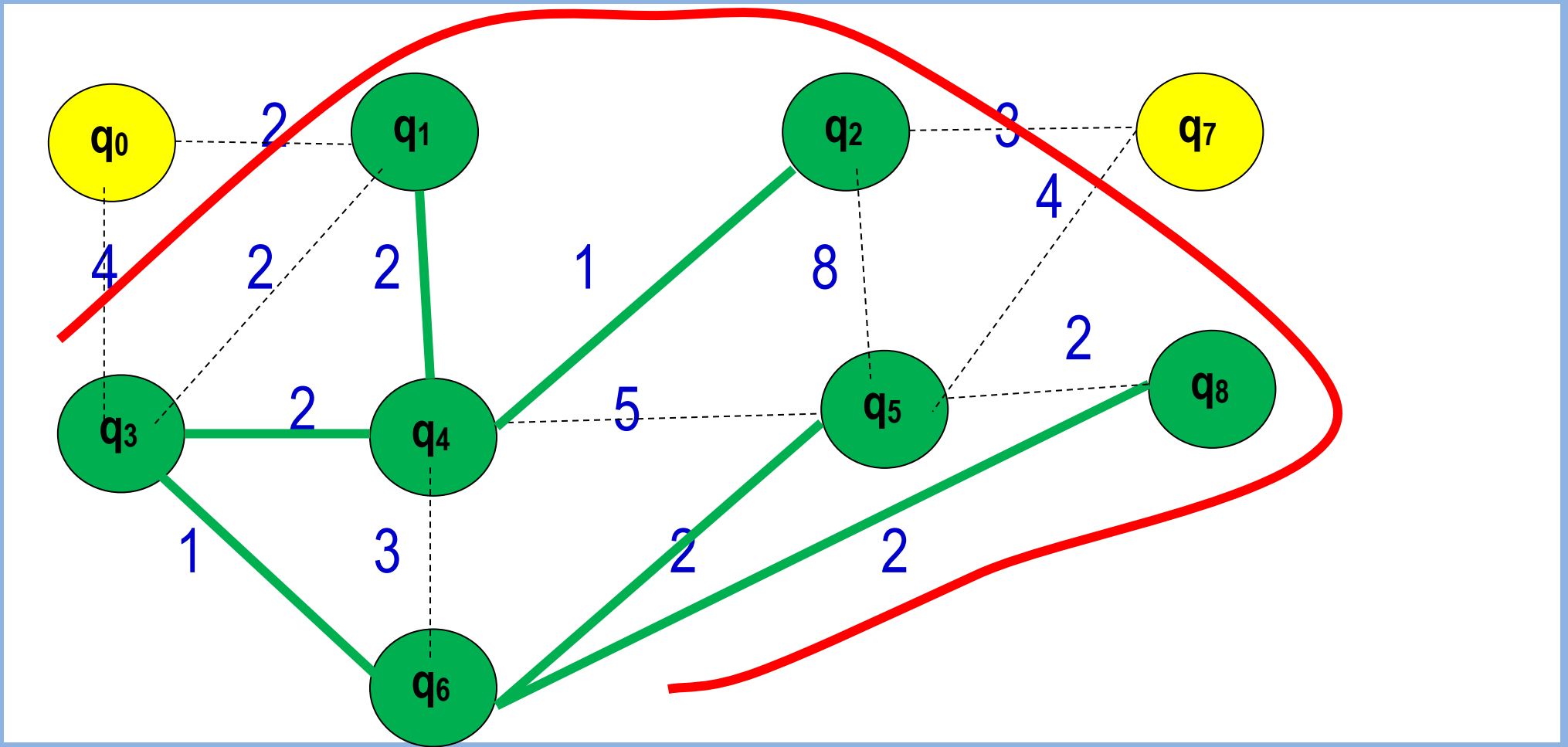

q0
q1
q2
q7
q3
q4
q5
q8
q6
2
4
2
2
1
8
4
2
2
5
1
3
2
2

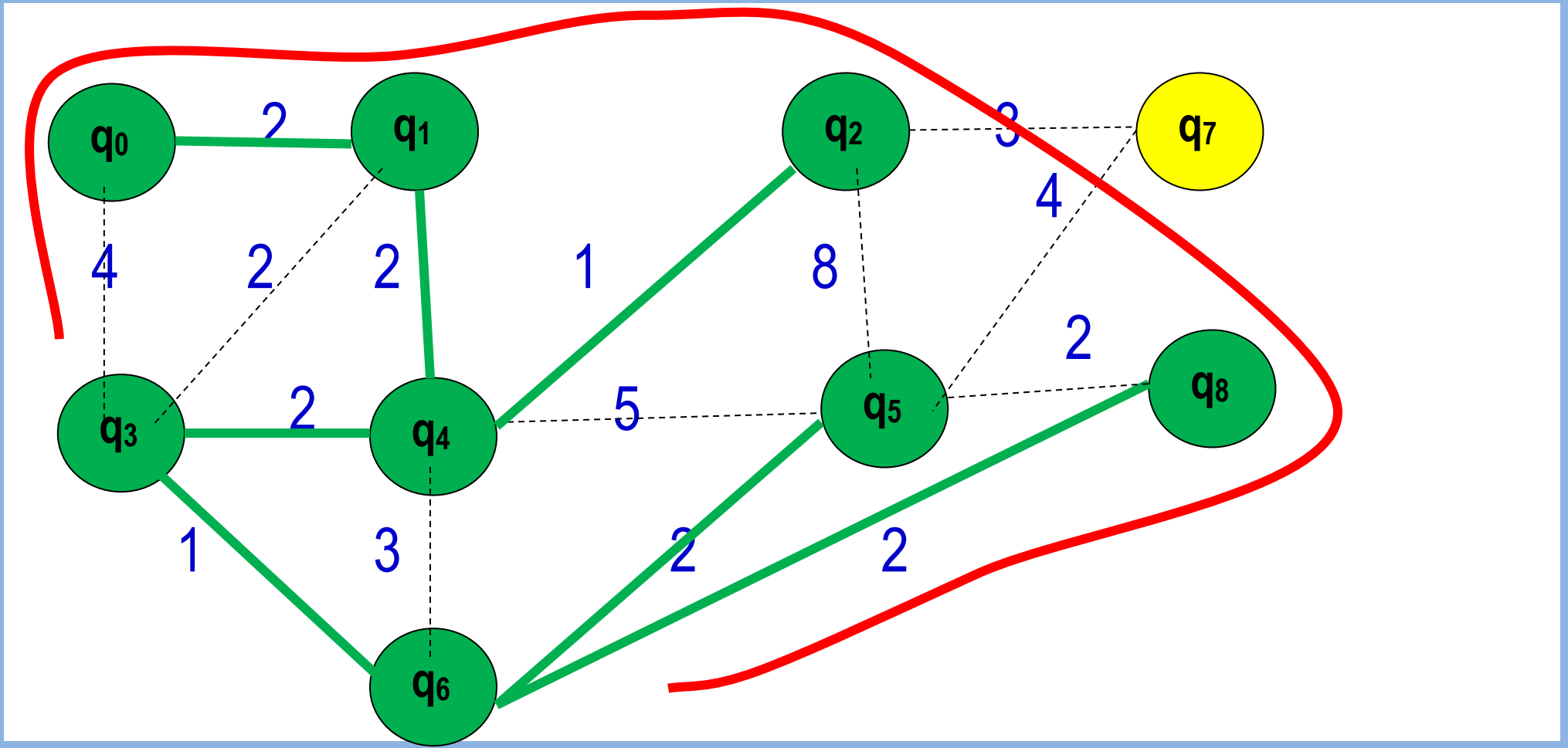

q0
q1
q2
q7
q3
q4
q5
q8
q6
2
4
2
2
1
8
3
4
2
2
5
1
3
2
2

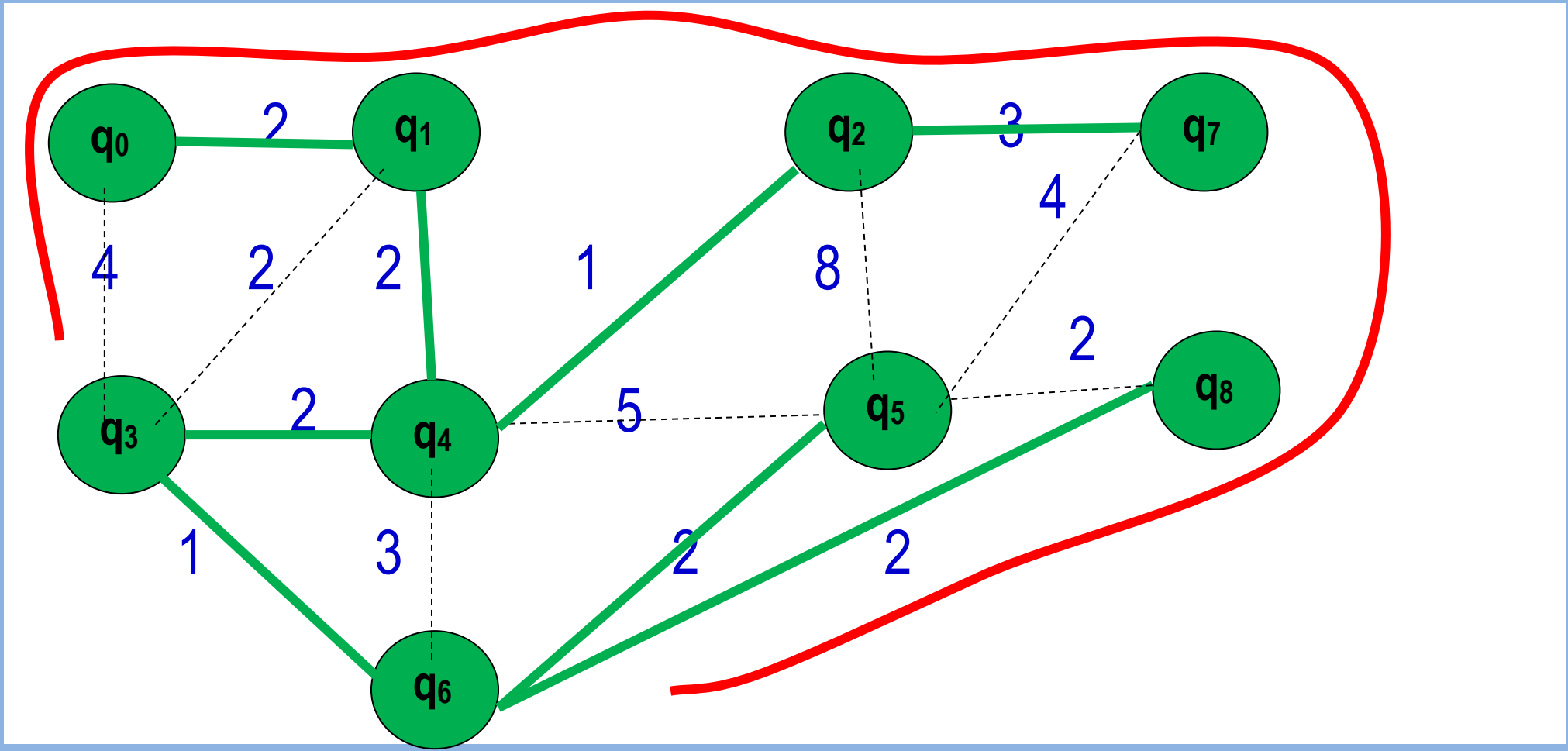

q0
q1
q2
q7
q3
q4
q5
q8
q6
2
3
4
4
2
2
1
8
2
2
5
1
3
2
2

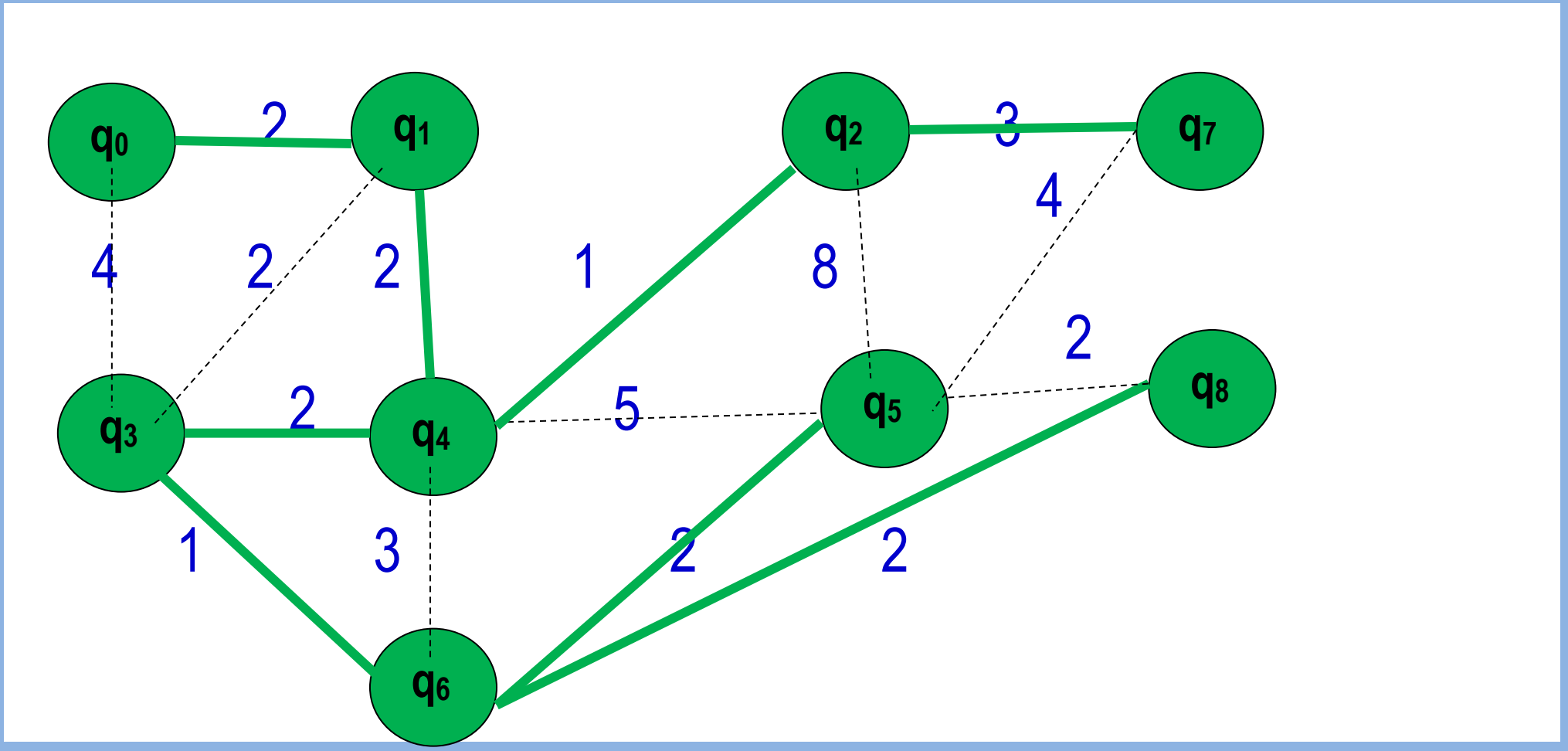

q0
q1
q2
q7
q3
q4
q5
q8
q6
2
3
4
4
2
2
1
8
2
2
5
1
3
2
2


q0
q1
q2
q7
q3
q4
q5
q8
q6
2
4
2
2
1
8
3
4
2
2
5
1
3
2
2


q0
q1
q2
q7
q3
q4
q5
q8
q6
2
4
2
2
1
8
3
4
2
2
5
1
3
2
2


q0
q1
q2
q7
q3
q4
q5
q8
q6
2
4
2
2
1
8
3
4
2
2
5
1
3
2
2


q0
q1
q2
q7
q3
q4
q5
q8
q6
2
4
2
2
1
8
3
4
2
5
1
3
2
2


q0
q1
q2
q3
q4
q5
q6
q7
q8
2
4
2
2
1
8
3
4
2
2
5
1
3
2
2

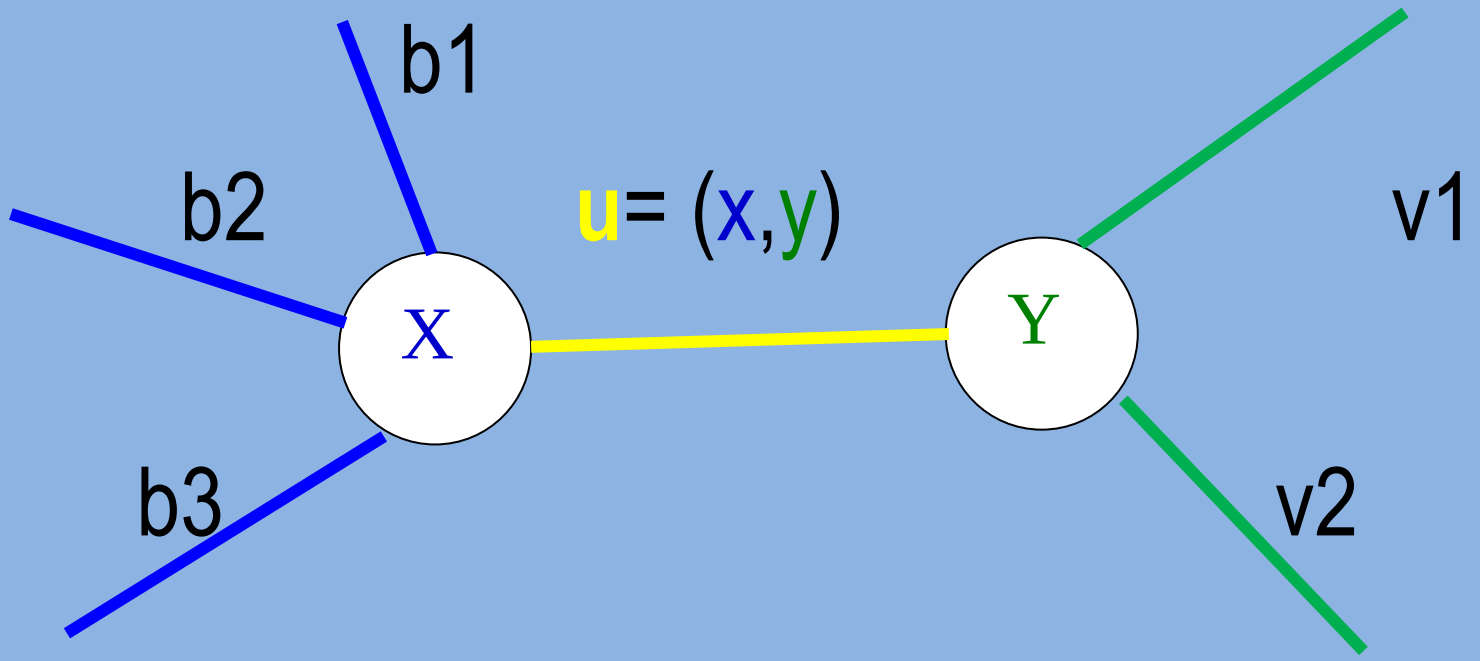
Principe

Le principe de l'algorithme de **Prim** consiste à :

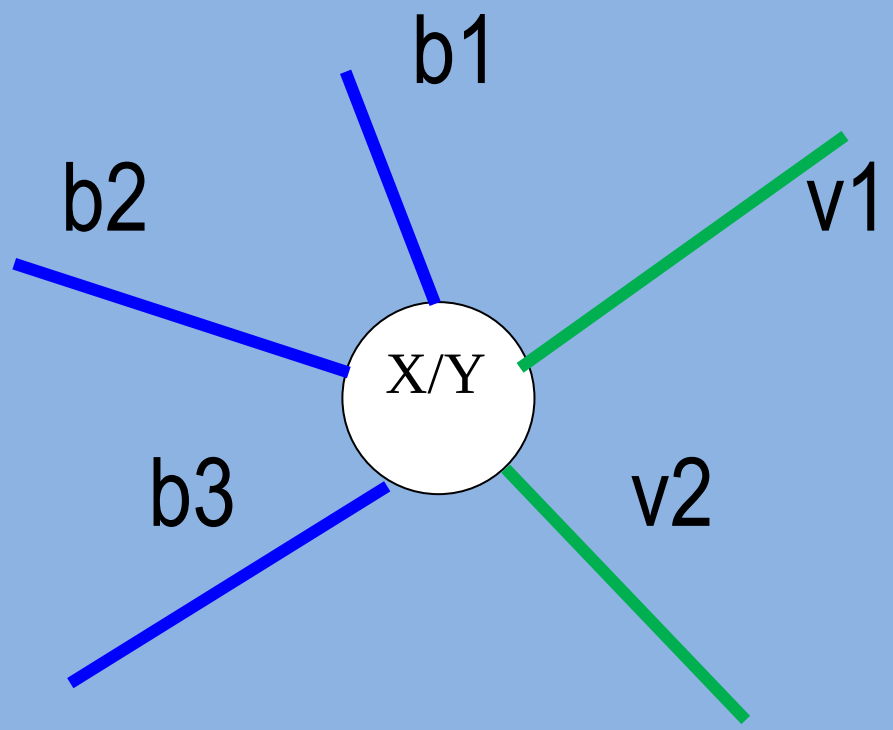
- « **fusionner»**, deux par deux, les sommets de G

- pour obtenir finalement un seul sommet représentant l'**arbre couvrant** à construire.

Par **fusionner**, on entend remplacer deux sommets par un seul.



Toutes les arêtes adjacentes à l'un ou l'autre des anciens sommets deviennent adjacentes au nouveau sommet.



Procédure de fusion

Le choix des sommets que l'on fusionne est fait en :

- en choisissant au hasard un sommet x,
- en cherchant, ensuite, une arête adjacente u= (x,y) de **coût mimimun**.

L'autre extrémité, soit y, de l'arête u fournit le deuxième sommet de la fusion.

Procédure

Titre: Prim
Entrées: G = (S, A): GRAPHE.
Sortie: G' = (S, A'): GRAPHE.
Variables intermédiaires: x: SOMMET , a:ARETE.

```
Prim (G :GRAPHE)  G': GRAPHE
Début
   /* initialisation */
   n← |S| ; m←|A|
   A' ←∅;    / * l'arbre recherché G' est initialement vide !  */
```

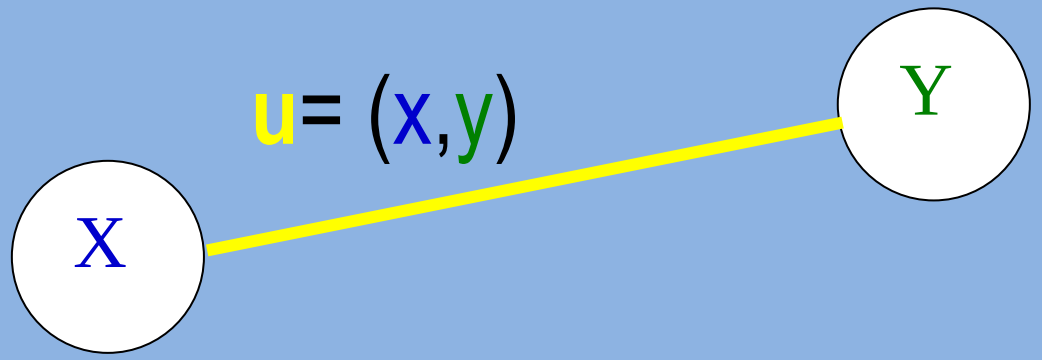
```
tant_que    |A'| < n - 1    / * car  G' sera un arbre */
        faire
         choisir x ∈S;     / * le choix de x est arbitraire  */
         choisir a ∈ A tel que  c(a)= min{c(x-y) | (x-y)∈A ∧  y≠x};
        A' ←A'∪ {a};
        fusionner(x,y); /*x et y deviennent un seul sommet */
  fin_tant_que;
Fin
```

Justification

Nous allons raisonner par récurrence.

Initialisation

Tout d'abord, on choisit une arête u = (x-y) où *x* et *y* sont deux sommets ne résultant pas d'une fusion.



On peut affirmer que le graphe résultant:

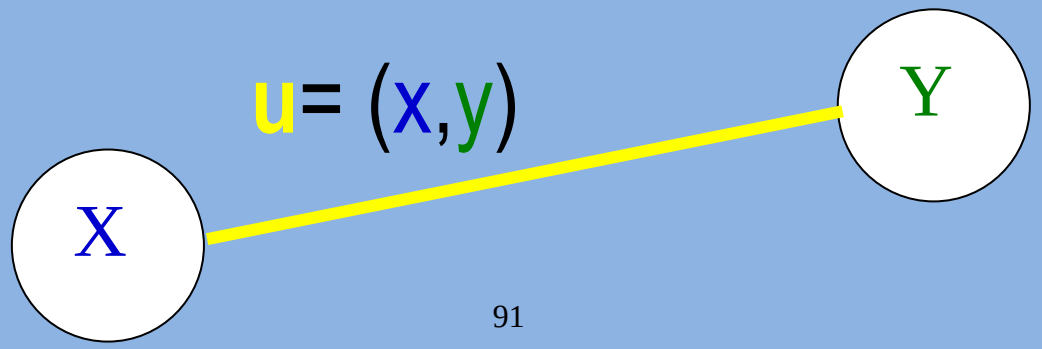
({x,y},{u})

est un arbre de coût minimum.

Hypothèse de récurrence :

Supposons, ensuite, par hypothèse **(de récurrence**) qu'à une étape donnée, les deux sommets ***x*** et ***y*** résultent des fusions précédentes.

En fait, **x** *et* **y** représentent selon cette hypothèse des arbres de coût minimum:



Si on choisit une arête **u** = (x-y) de coût minimum alors le graphe obtenu sera un arbre de coût minimum.

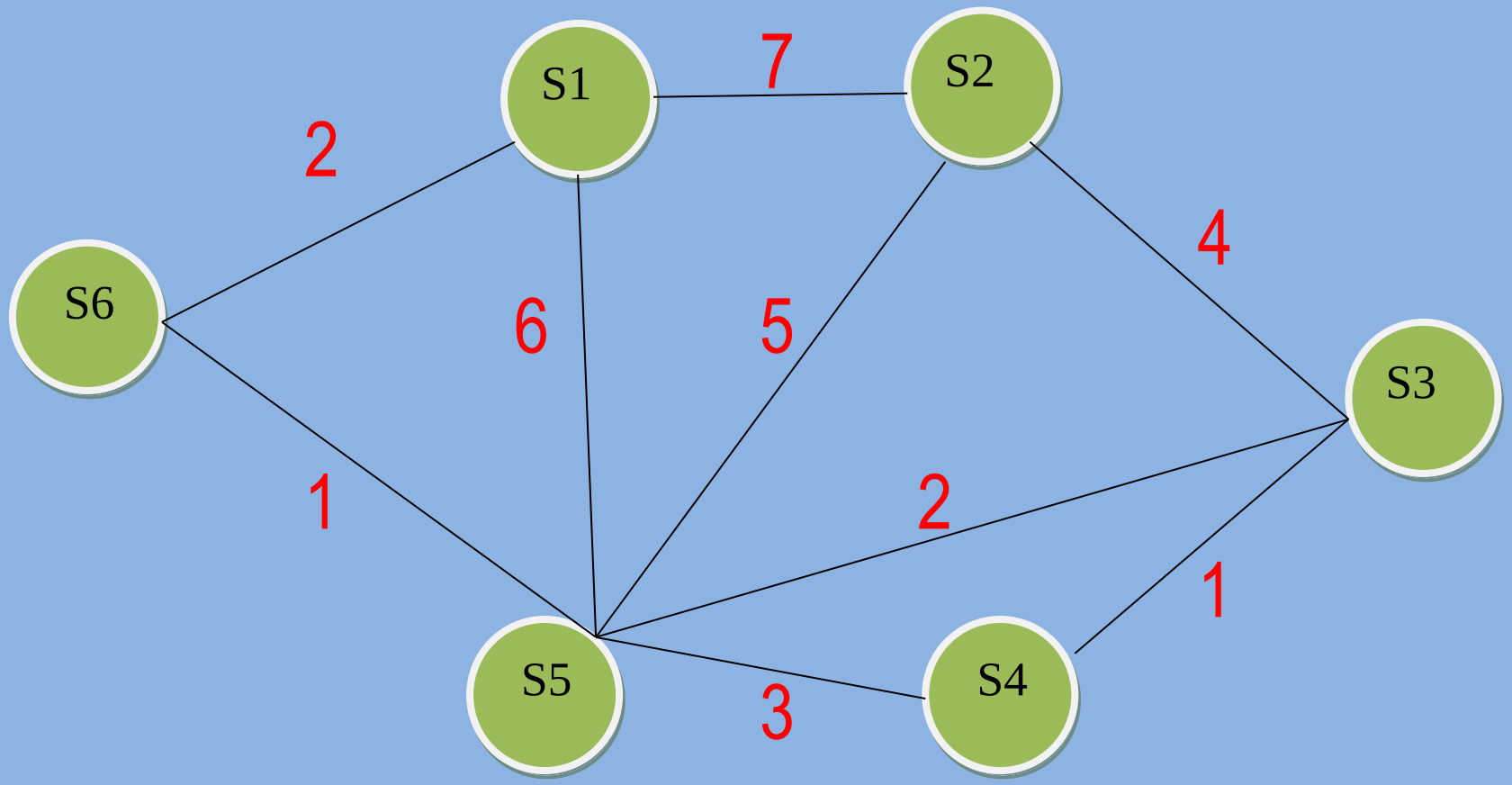
Donc à la dernière étape (lorsque tous les sommets seront fusionnés) de l'algorithme, on obtiendra bien :

- un **arbre de coût minimum**

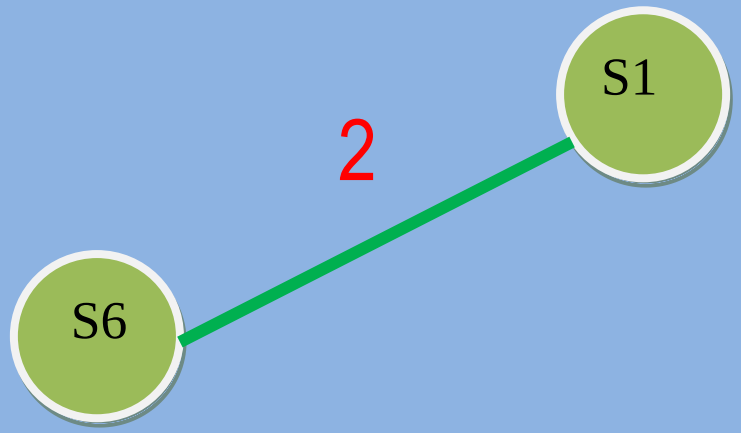
-qui est un **graphe partiel** de G incluant **n-1** arêtes

Exemple

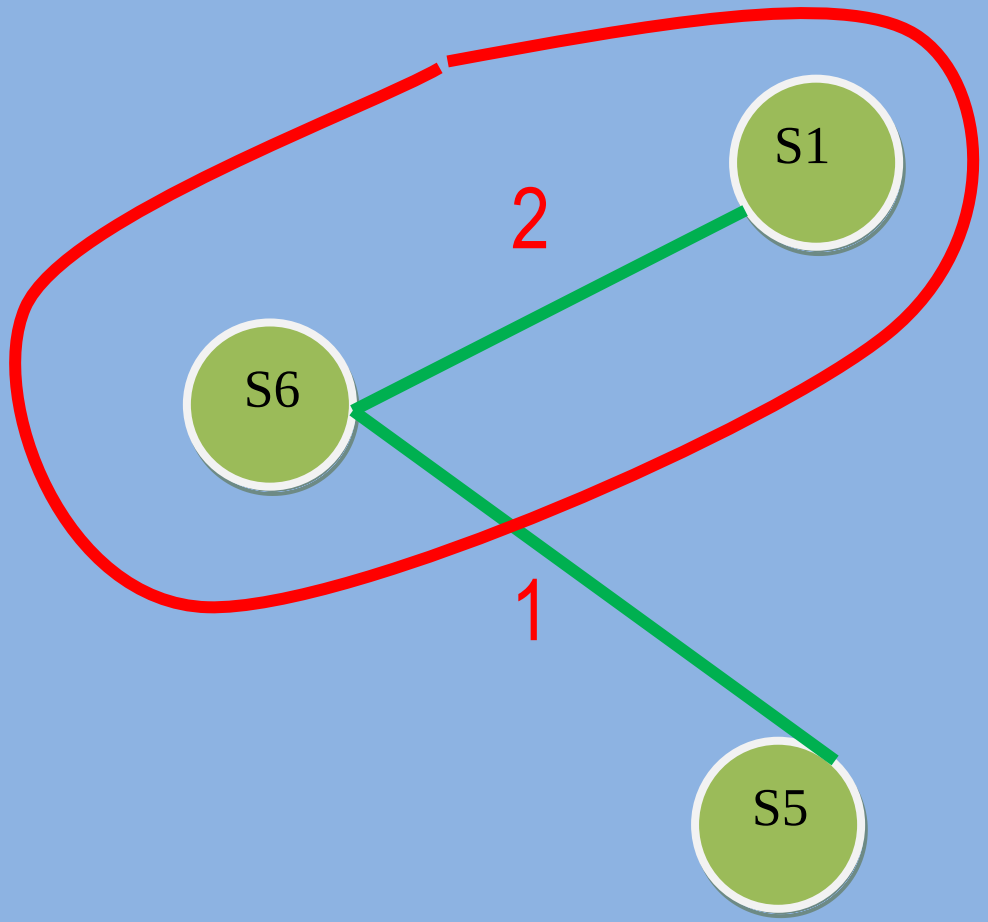
Soit le graphe non orienté valué connexe:



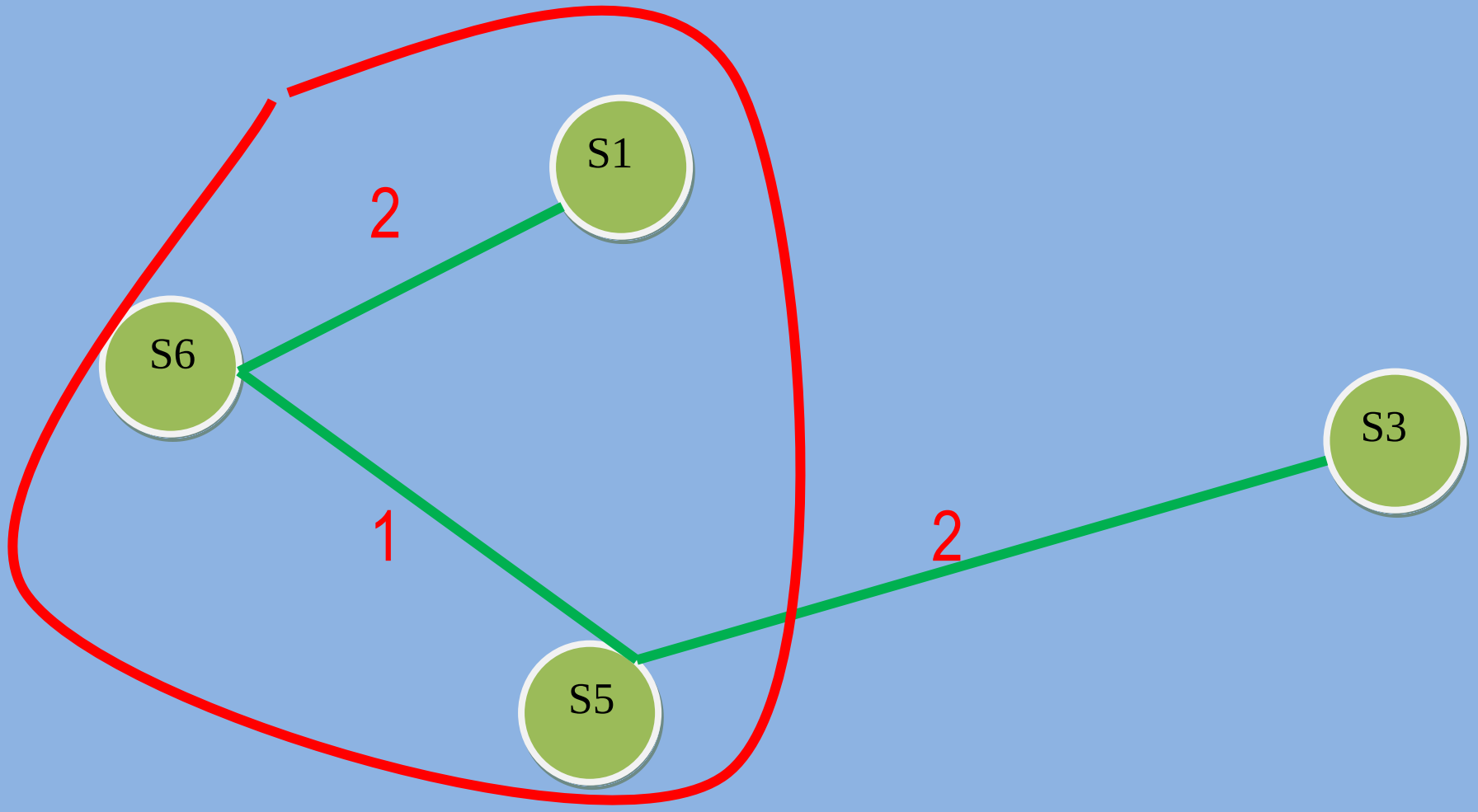
Choisir le sommet x=S1, le sommet le plus proche est y= S6



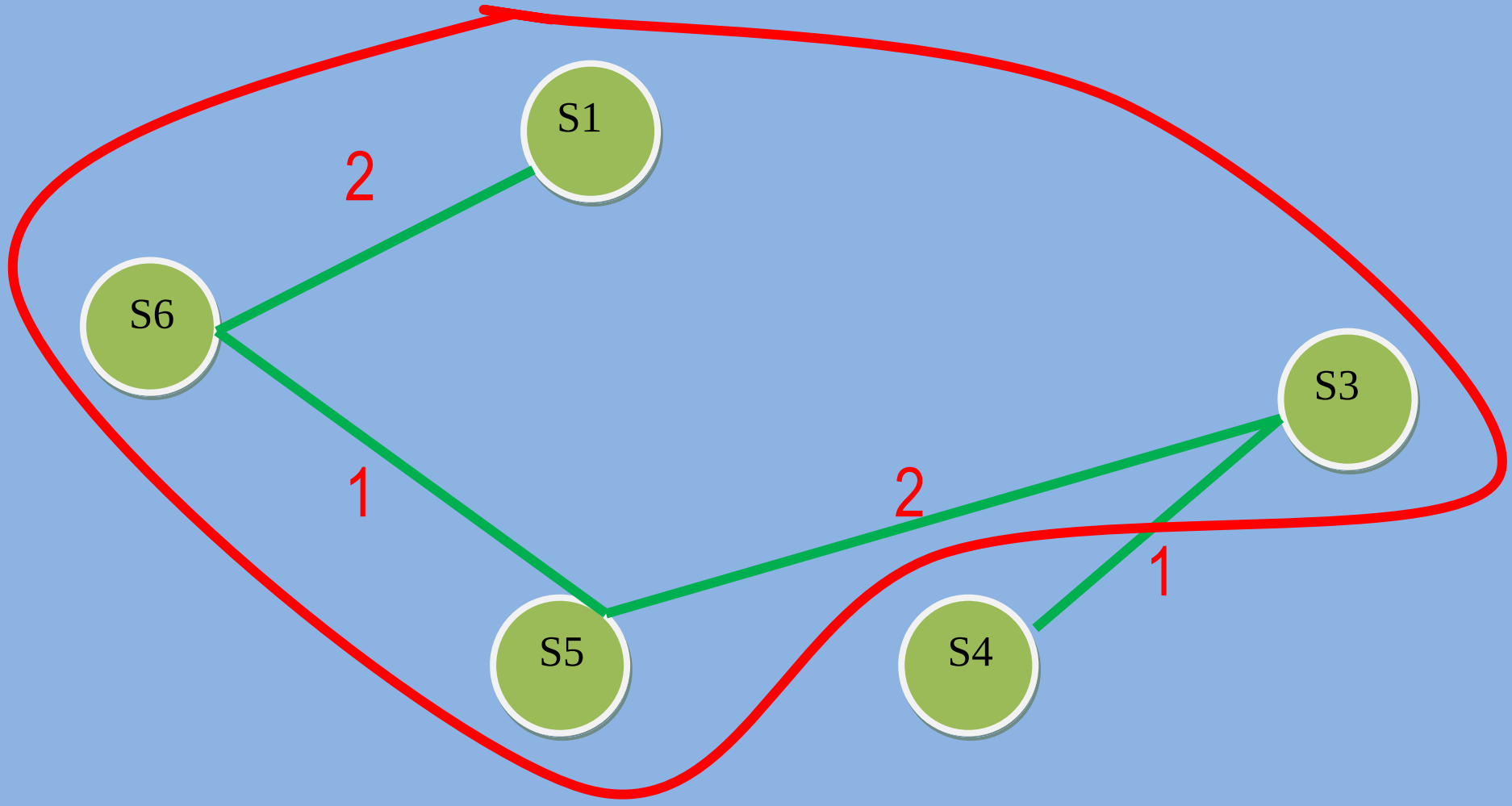
Le sommet le plus proche de {S1, S6} est y= S5



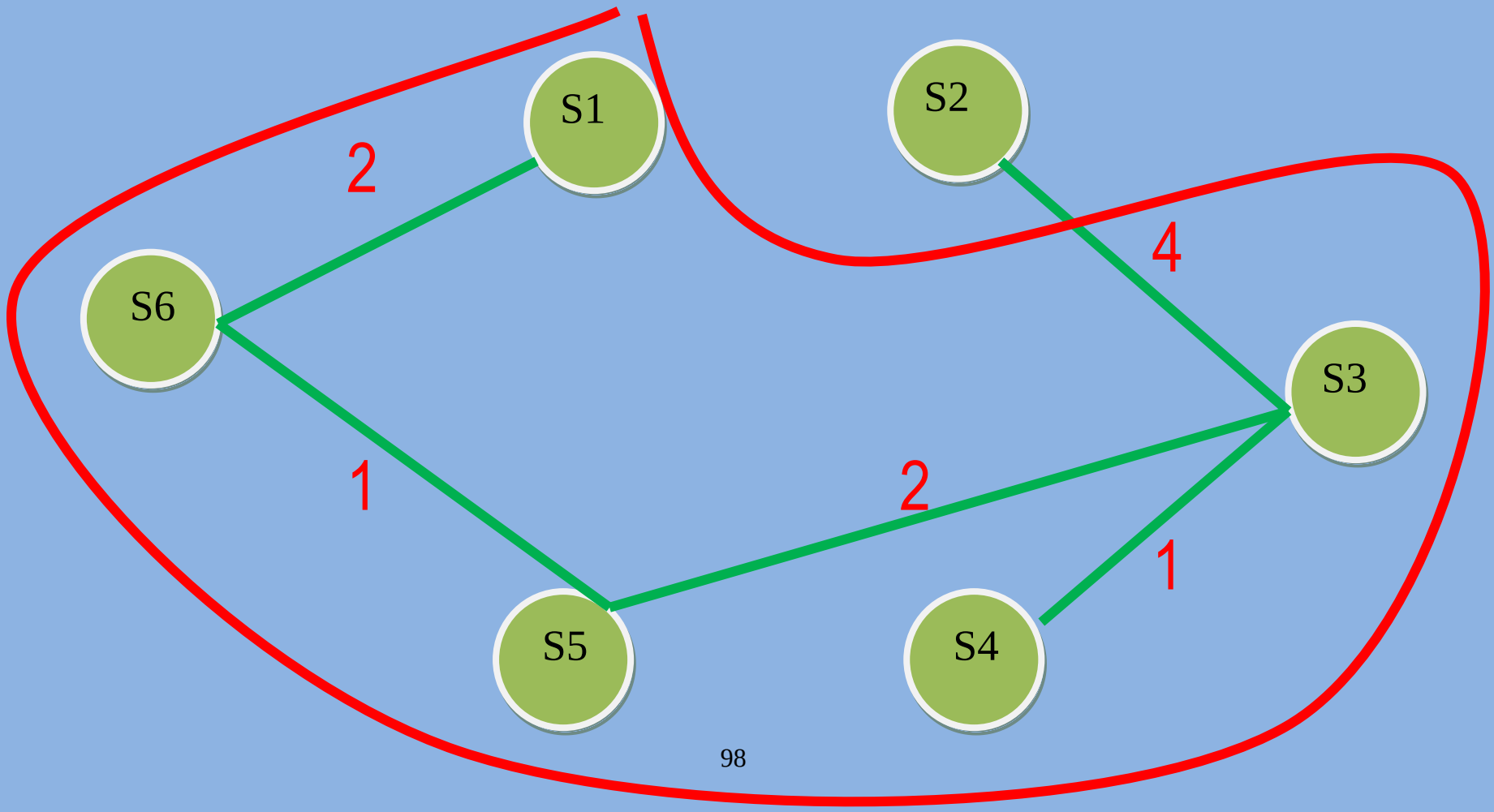
Le plus sommet le proche de x= {S1,S5, S6} est y= S3



Le sommet le plus proche de x= {S1,S3, S5,S6} est y= S4



Le sommet le plus proche de x= {S, S3,S4, S5,S6} est y= S2



Comme on a finalement:

$X = \{S1,S2,S3,S4,S5,S6\}$

$= S$

la construction de l'arbre de recouvrement minimum est **terminée**.

Remarques:

L'**analyse de la complexité** des deux algorithmes sera abordée en TP lors de leur application.

1-Complexité de l'algorithme de Kruskal

L'algorithme de Kruskal est dominé par le tri. Sa complexité est donc celle d'un tri de **m** arêtes: elle est en **O(mlog(m)).**

La rapidité de l’algorithme de Kruskal est fonction de la **structure du graphe**.

Le nombre **m** d'arêtes d'un graphe connexe peut varier de **n-1** à **n(n-1)/2**.

L'algorithme de Kruskal est d'autant plus rapide que le graphe connexe est pauvre en arêtes.(graphe de degré faible)

Complexité de l'algorithme de Prim

Il y a exactement **n** itérations principales et une boucle d'au plus **n** itérations à l'intérieur, c'est donc un algorithme polynomial en **$O(n^2)$.**

L'algorithme de Kruskal est donc plus intéressant quand il y a **peu d'arêtes** dans le graphe G.

L'algorithme de Prim **ne dépend pas du nombre d'arêtes** et a une complexité constante lorsque le nombre **n** de sommets est fixé.