Arbres couvrants minimaux

Algorithmique – L3

François Laroussinie

1er décembre 2010

Plan

1. Définitions
2. Algorithme de Prim
3. Algorithme de Kruskal
4. Application au ≪ voyageur de commerce ≫

Plan

1. Définitions
2. Algorithme de Prim
3. Algorithme de Kruskal
4. Application au « voyageur de commerce »

Arbres couvrants minimaux

$G = (S, A, w)$: non-orienté, connexe et valué ($w : A \to \mathbb{R}$).

Définitions :

- un arbre couvrant de G est un graphe $T = (S, A')$ avec $A' \subseteq A$, connexe et acyclique.
- un arbre couvrant $T = (S, A')$ est dit minimal lorsque :

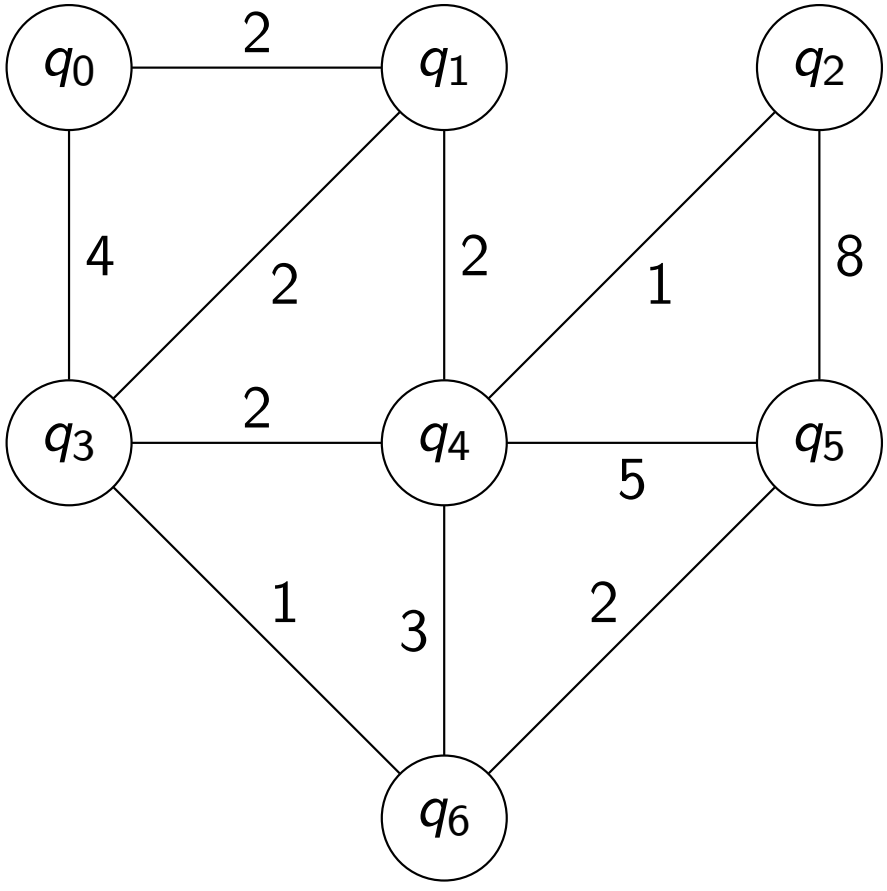
$$w(A') = \min\{w(A'') \mid T' = (G, A'') \text{ est un AC de } G\}$$

avec $w(A) \stackrel{\text{def}}{=} \sum_{(x,y)\in A} w(x, y)$

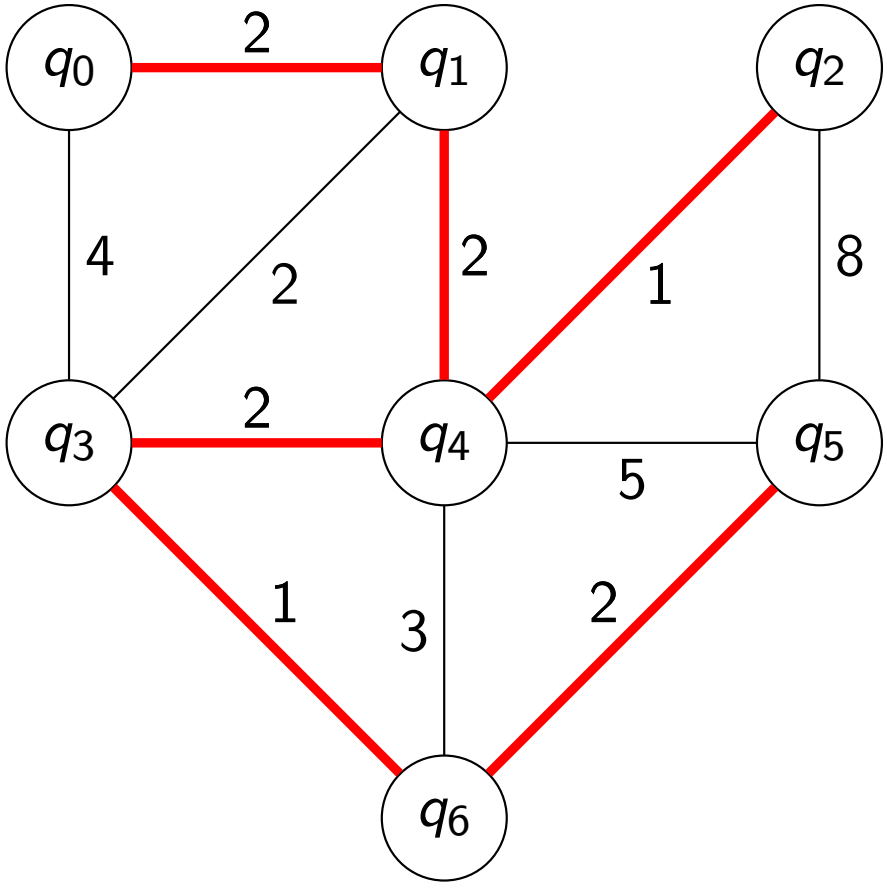
Propriété :Existence d'un ACM

Tout graphe non-orienté, valué et connexe admet un ou plusieurs ACM.

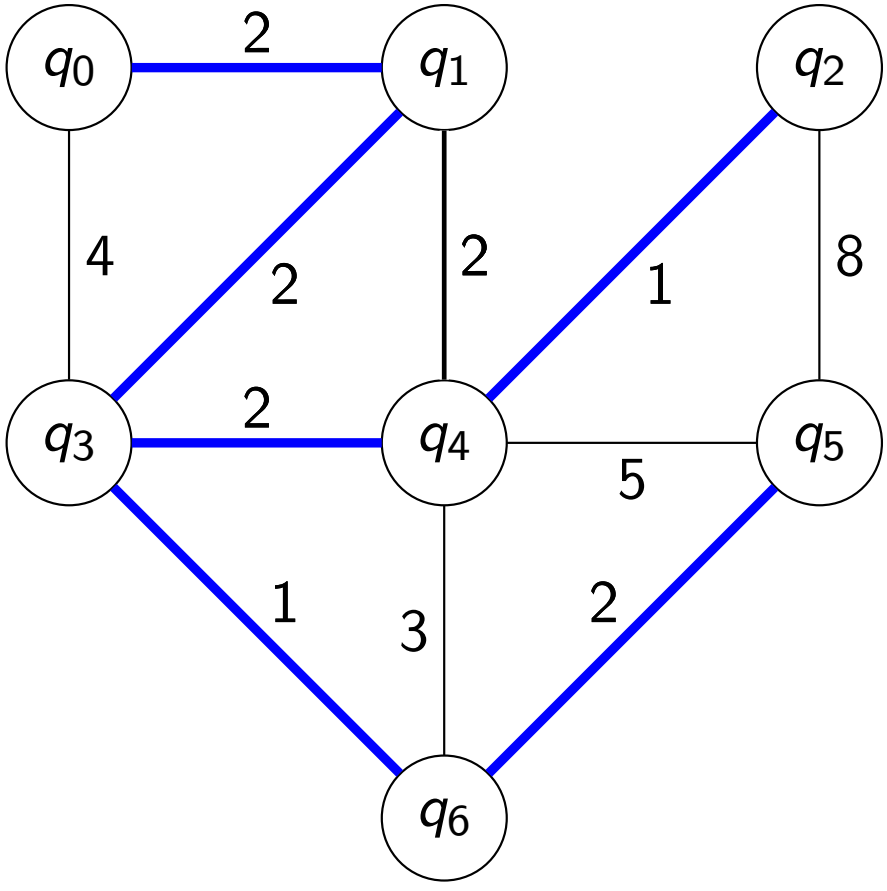
Exemple d'ACM



Exemple d'ACM



Exemple d'ACM



Construire un ACM

On va construire un ACM $T = (S, A')$ de manière incrémentale.

- Au début, A' est vide.
- A chaque étape, on va choisir une nouvelle arête (u, v) tq $A' \cup \{(u, v)\}$ est toujours un sous-ensemble d'un ACM pour G.

Def : on dit alors que (u, v) est compatible avec A'.

Construire un ACM

On va construire un ACM $T = (S, A')$ de manière incrémentale.

- Au début, A' est vide.
- A chaque étape, on va choisir une nouvelle arête (u, v) tq $A' \cup \{(u, v)\}$ est toujours un sous-ensemble d'un ACM pour G.

Def : on dit alors que (u, v) est compatible avec A'.

Tout le problème est de pouvoir décider (efficacement) si une arête est compatible avec un A'...

Comment connecter à moindre coût des sommets non encore reliés par A' ?

Comment vérifier la compatibilité d'une arête ?

Soient $C_1, \ldots, C_k$ les k composantes connexes de (S, A').

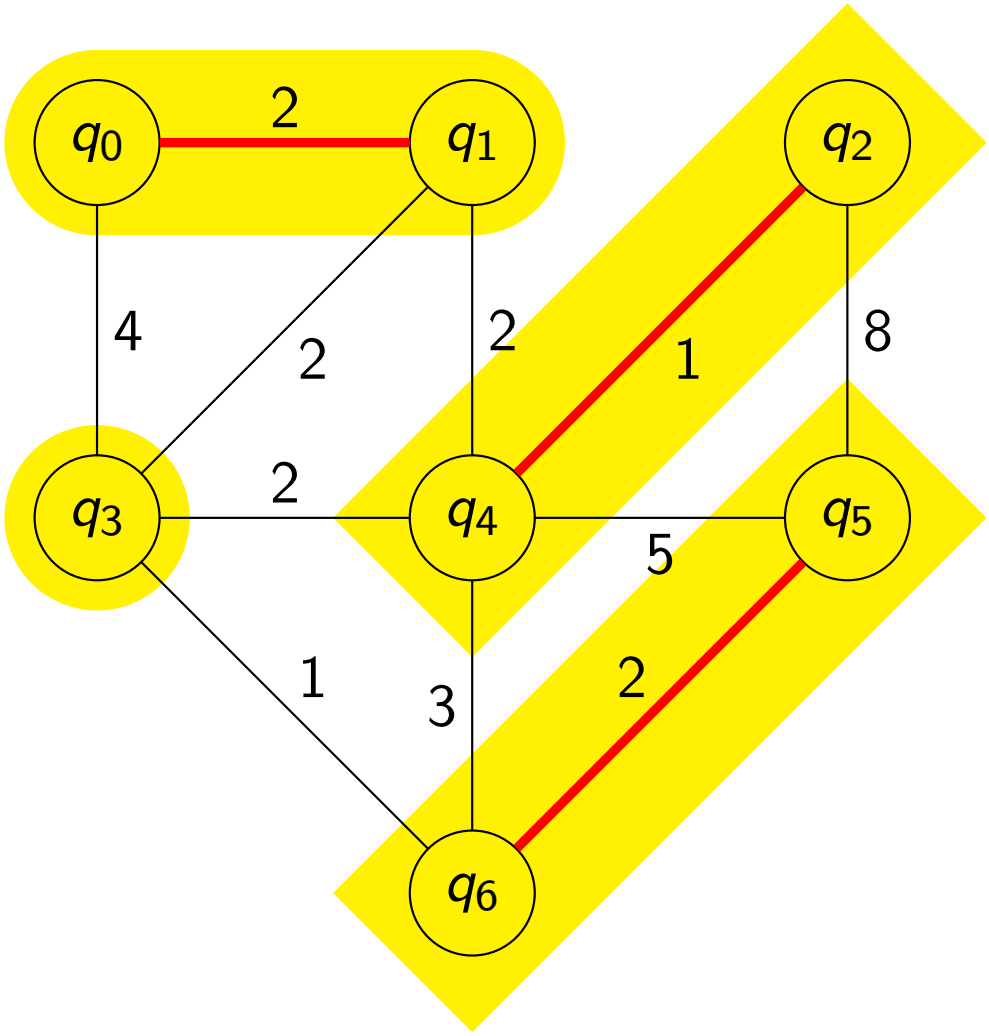
1- On prend une partition $[S_1, S_2]$ de S qui respecte A'.
(i.e. $\forall (x, y) \in A'$, x et y sont dans le même S_i)

$\Big($ ou chaque C_i est inclue soit dans S_1, soit dans S_2 $\Big)$

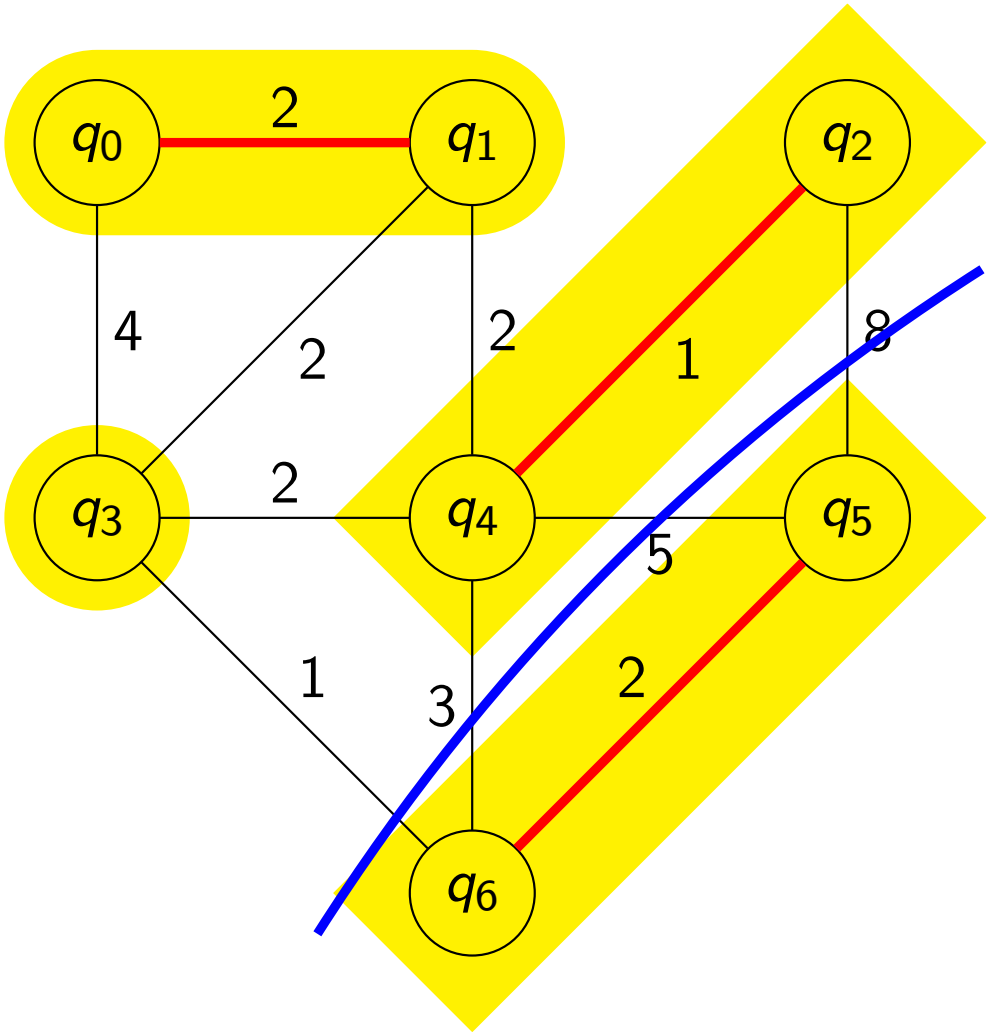
2- On choisit une arête (x, y) telle que :

- (x, y) traverse la partition $[S_1, S_2]$,
  $\Big(\overset{\text{def}}{=} \quad (x \in S_1 \text{ et } y \in S_2) \quad \text{ou} \quad (x \in S_2 \text{ et } y \in S_1) \Big)$
  et
- $w(x, y)$ est minimale parmi ces arêtes traversantes.

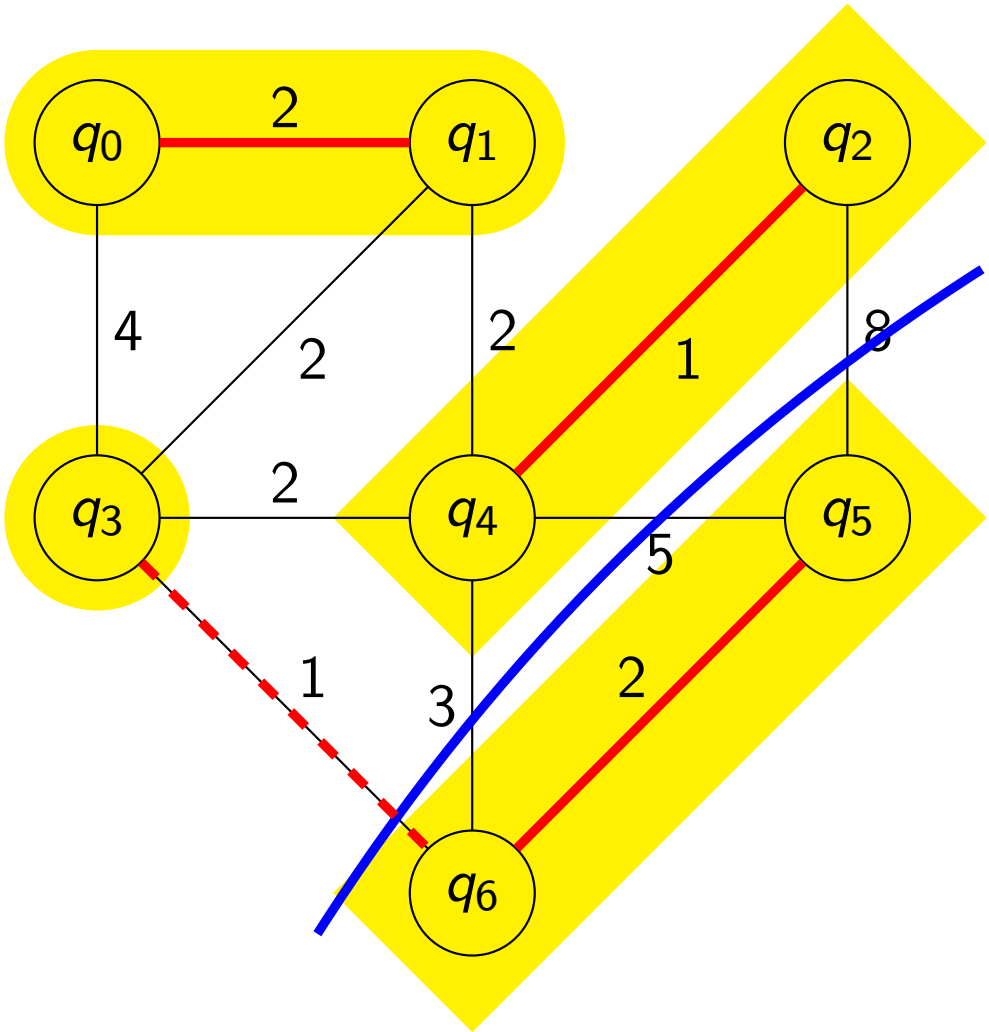
Exemple



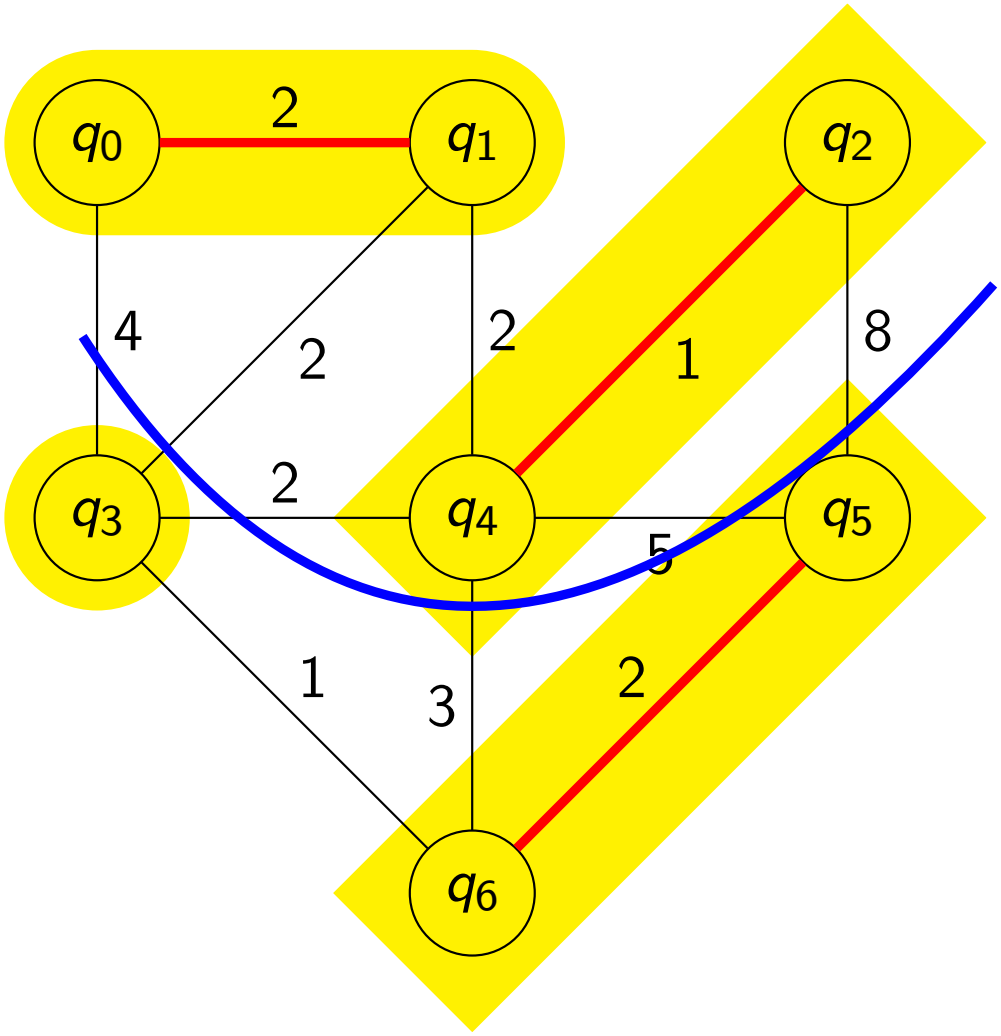
Exemple



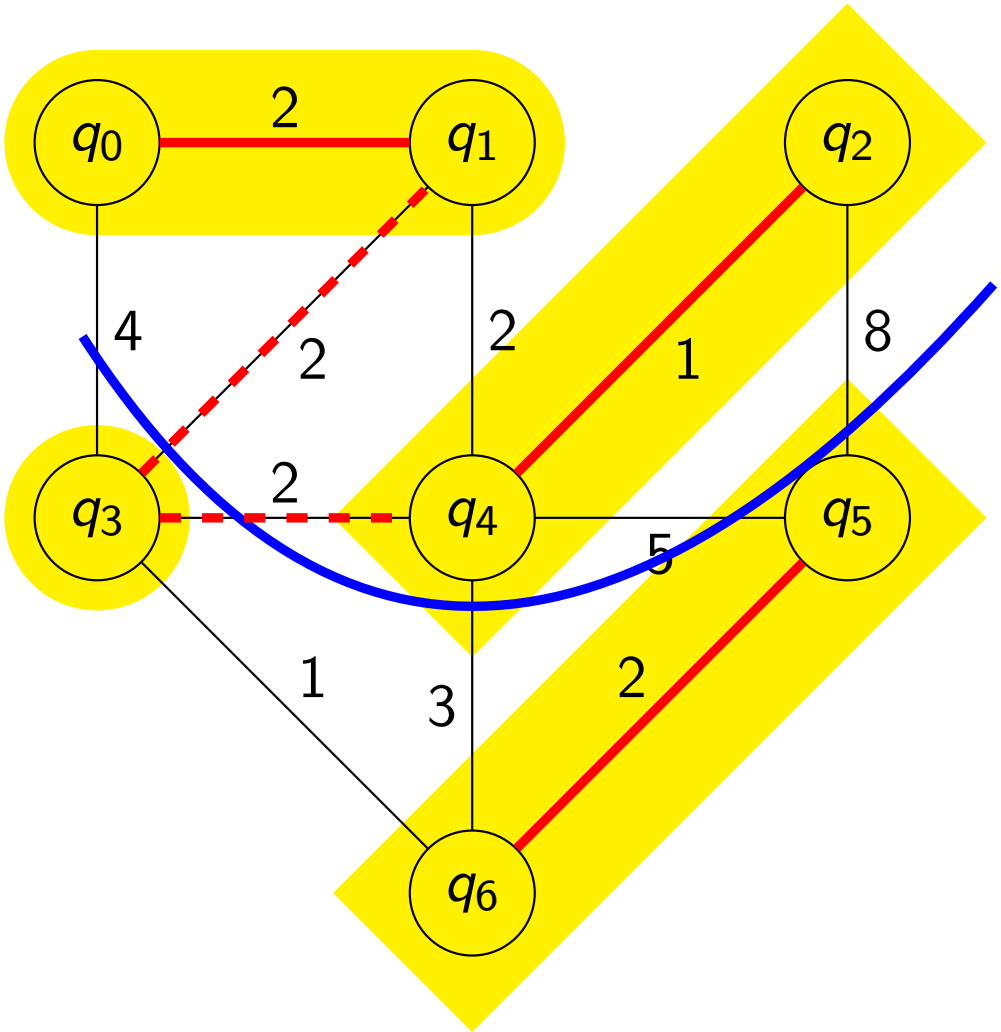
Exemple



Exemple



Exemple



Compatibilité

Théorème 4

Étant donnés :

- $G = (S, A, w)$ un graphe non-orienté, valué et connexe,
- $A' \subseteq A$ tel qu'il existe un ACM de G contenant A',
- (S_1, S_2) une partition qui respecte A', et
- (u, v) une arête traversante minimale de (S_1, S_2),

alors (u, v) est compatible avec A'.
(*i.e.* il existe un ACM contenant $A' \cup \{(u, v)\}$).

Plan

1. Définitions
2. Algorithme de Prim
3. Algorithme de Kruskal
4. Application au « voyageur de commerce »

Algorithme de Prim – idée

A chaque étape de la construction de A' :

- A' correspond à un arbre et un ensemble de sommets isolés ;
- L'algorithme choisit une arête de poids minimal qui relie l'arbre à un sommet isolé.

Algorithme de Prim – idée

A chaque étape de la construction de A' :

- A' correspond à un arbre et un ensemble de sommets isolés ;
- L'algorithme choisit une arête de poids minimal qui relie l'arbre à un sommet isolé.

Propriété [correction de l'algorithme de Prim]

A chaque étape, l'arête choisie est compatible : on applique le théorème 4 avec la partition (S_1, S_2) suivante :

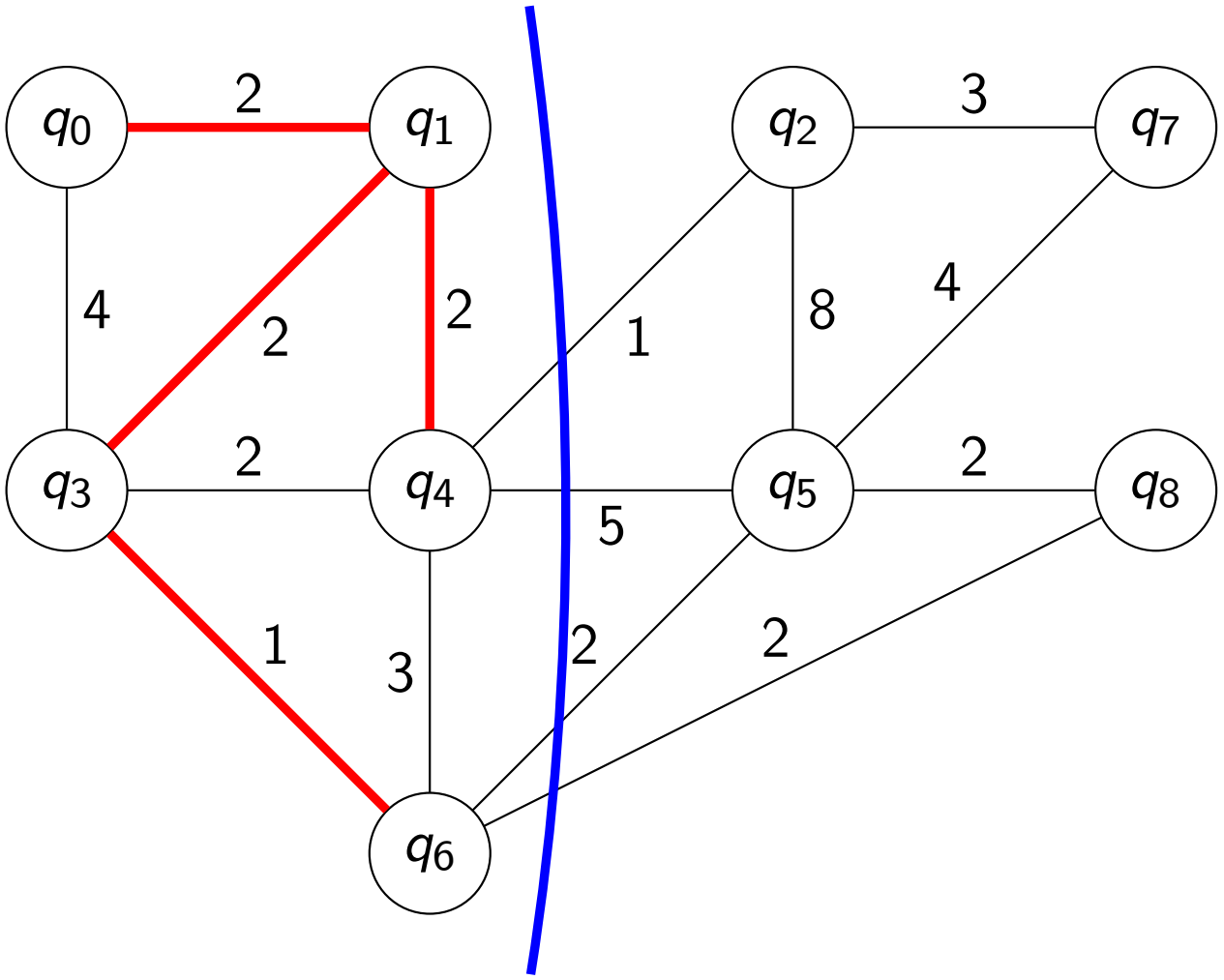
- S_1 les sommets de l'arbre ;
- S_2 les sommets isolés.

Algorithme de Prim

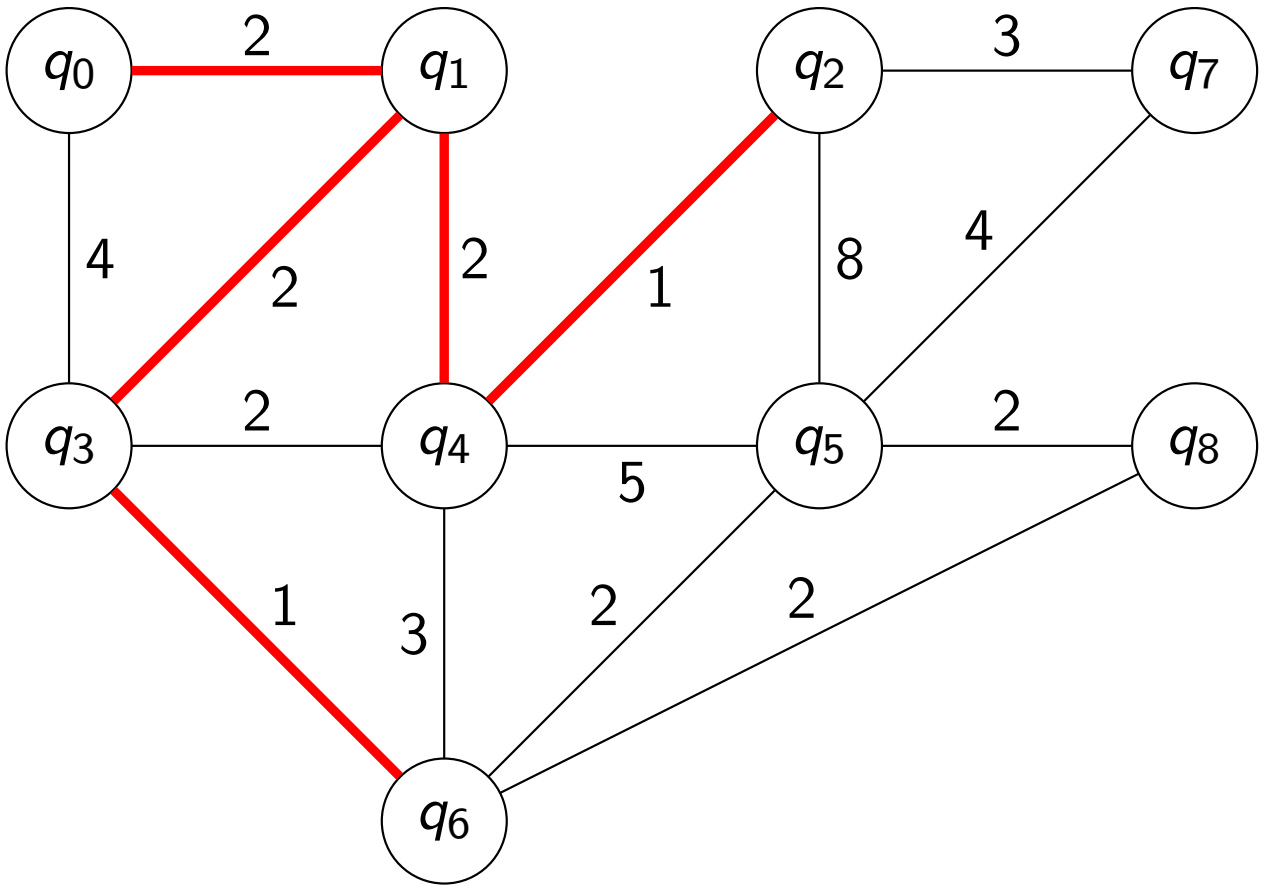
1. $A' = \emptyset$
2. On choisit un sommet de départ s.
3. On ajoute à A' une arête (s, x) de poids minimal,
4. On ajoute à A' une arête (q, q') reliant un sommet isolé à l'arbre en construction et de poids minimal,
5. recommencer le point **4** jusqu'à la connection de tous les sommets.

A' est une solution !

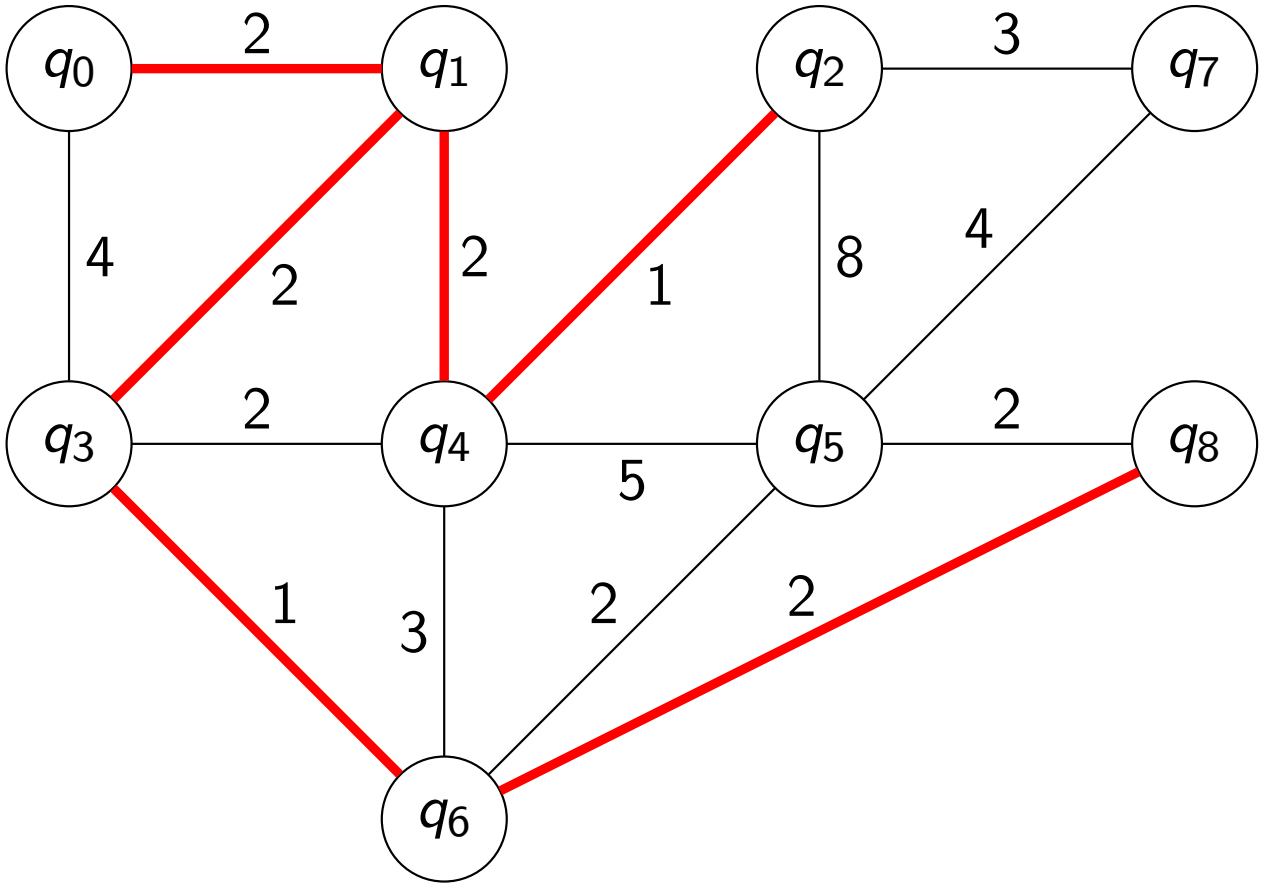
Exemple



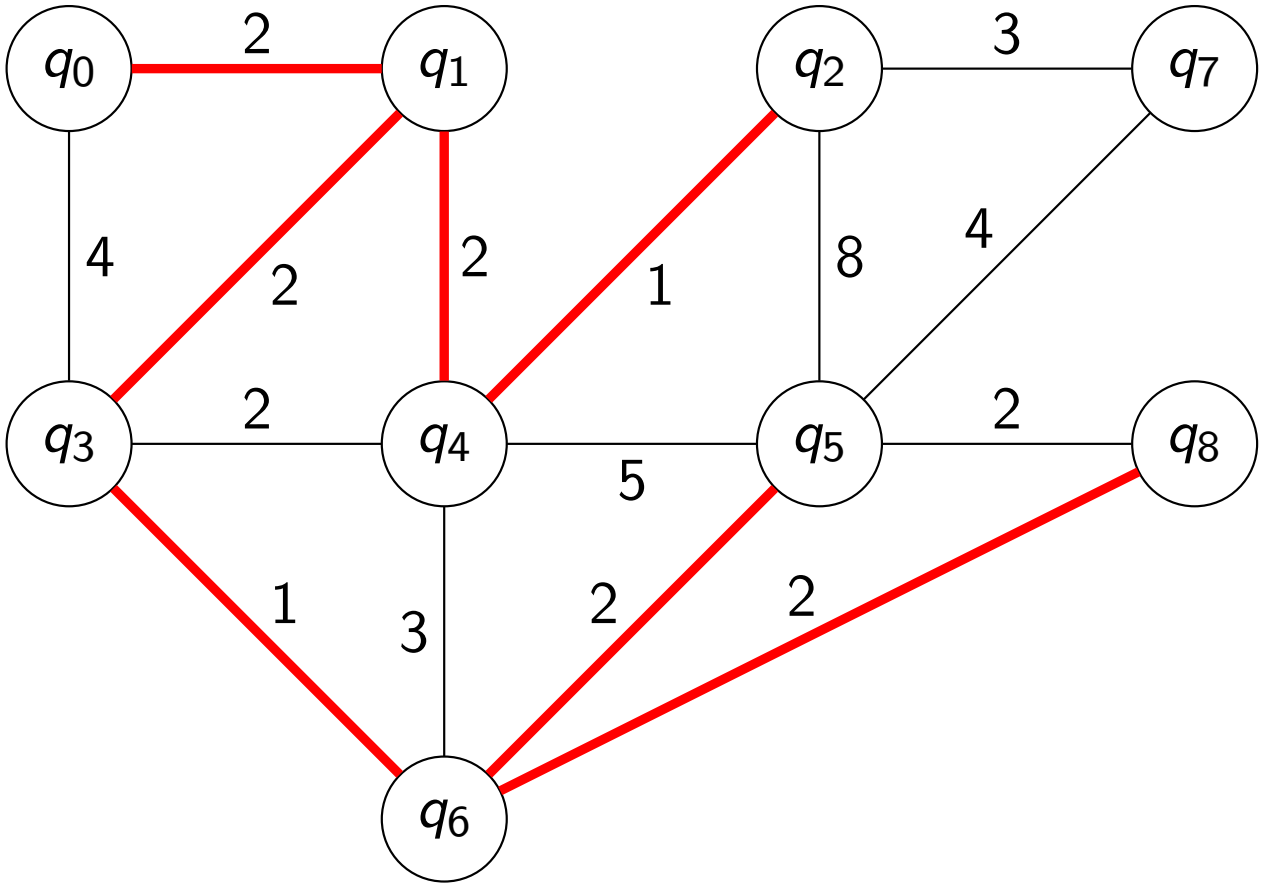
Exemple



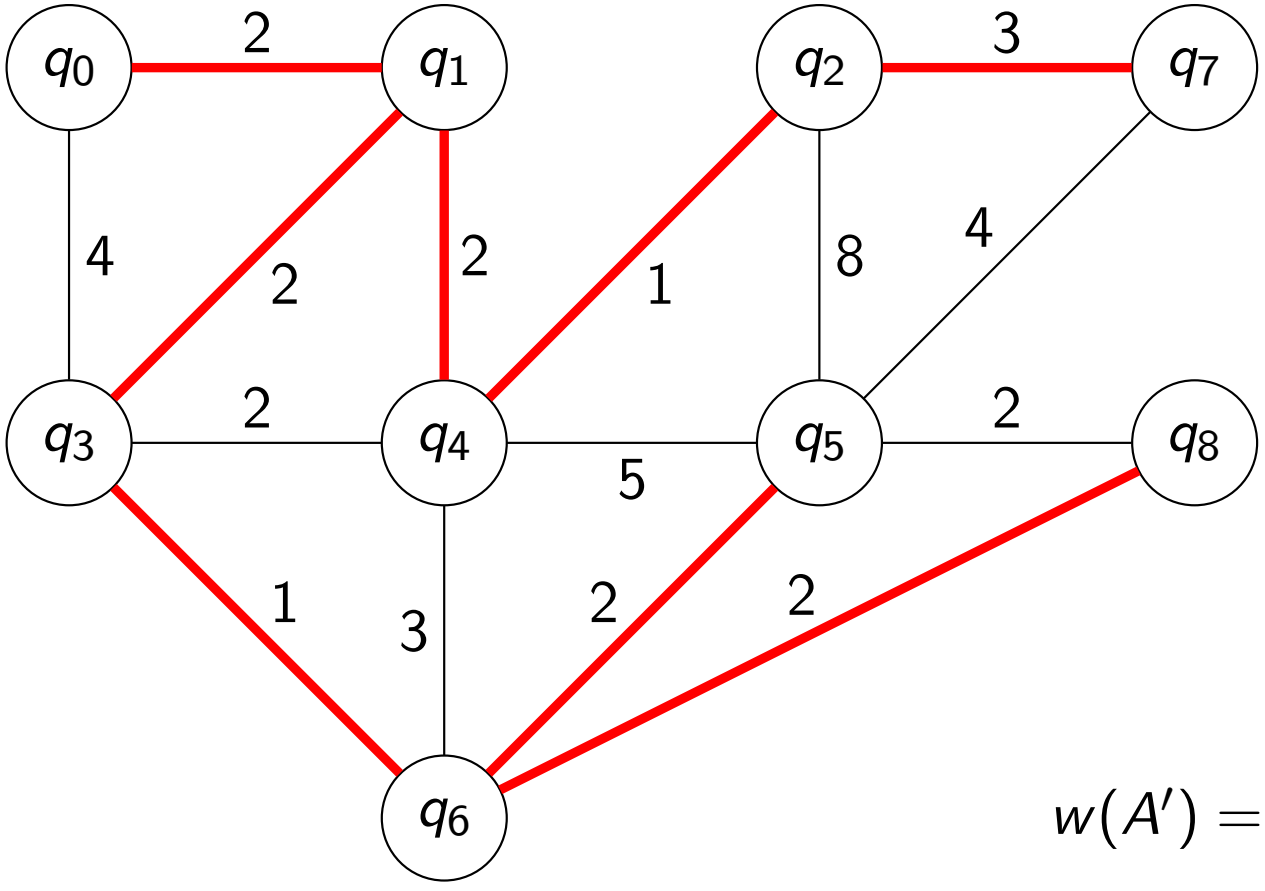
Exemple



Exemple



Exemple



$w(A') = 15$

Choix du plus proche sommet...

A chaque étape, l'algorithme doit choisir un sommet isolé le plus proche possible (en une transition) de l'arbre A'.

Choix du plus proche sommet...

A chaque étape, l'algorithme doit choisir un sommet isolé le plus proche possible (en une transition) de l'arbre A'.

On stocke les sommets isolés dans une file de priorité F :
la priorité de x sera sa distance à l'arbre A'.

Choix du plus proche sommet...

A chaque étape, l'algorithme doit choisir un sommet isolé le plus proche possible (en une transition) de l'arbre A'.

On stocke les sommets isolés dans une file de priorité F :
la priorité de x sera sa distance à l'arbre A'.

En plus de « Extraire-Min », on a besoin des fonctions suivantes :

- `IndiceDansF`$[x]$: donne l'indice de x dans F.
- $\Pi[x]$: indique par quelle arête x est le plus proche possible de l'arbre.
- MaJ-F-Prim$(F, \mathrm{d}, G, x, \Pi,$ `IndiceDansF`$)$: met à jour F après l'ajout de x dans A'.

Algorithme de Prim

Procédure `Recherche-ACM-Prim`(G, s_0)
begin
  pour chaque $s \in S$ **faire**
    $\Pi[s] := \text{nil}$
    $\text{d}[s] := \begin{cases} 0 & \text{si } s = s_0 \\ \infty & \text{sinon} \end{cases}$
  $A' := \emptyset$
  $F := \text{File}(S, \text{d}, \text{IndiceDansF})$
  tant que $F \neq \emptyset$ **faire**
    $s := \text{Extraire-Min}(F)$
    $\text{IndiceDansF}[s] := -1$
    si $s \neq s_0$ **alors** $A' := A' \cup \{(\Pi(s), s)\}$
    $\text{MaJ-F-Prim}(F, \text{d}, G, s, \Pi, \text{IndiceDansF})$
  return A'
end

Algorithme de Prim (suite)

Procédure MaJ-F-Prim(F, d, G, s, Π, IndiceDansF)
begin
  pour chaque $(s, u) \in A$ **faire**
    si $(\mathit{IndiceDansF}[u] \neq -1) \wedge (w(s, u) < d[u])$ **alors**
      $\Pi(u) := s$
      d$[u] := w(s, u)$
      //On réorganise...
      //$F[i]$: le sommet à la position i dans F.
      $i :=$ IndiceDansF$[u]$
      tant que $(i/2 \geq 1) \wedge (d[F[i/2]] > d[F[i]])$ **faire**
        $F[i] \leftrightarrow F[i/2]$
        IndiceDansF$[F[i]] := i$
        IndiceDansF$[F[i/2]] := i/2$
        $i := i/2$;
end

Complexité de l'algorithme de Prim

La complexité totale :

- la construction de la file : $O(|S|)$,
- chaque appel de ExtraireMin : $O(\log(|S|))$
- le coût total des MaJ$(F, \mathrm{d}, G, s, \Pi)$ est en $O(|A| \cdot \log(|S|))$.

$$O\Big(|S| \cdot \log(|S|) + |A| \cdot \log(|S|)\Big)$$

$$\Rightarrow \quad O(|A| \cdot \log(|S|))$$

Plan

1. Définitions
2. Algorithme de Prim
3. Algorithme de Kruskal
4. Application au « voyageur de commerce »

Algorithme de Kruskal – idée

Deux caractéristiques :

- A' décrit une forêt,
- une arête compatible est une arête de poids minimal reliant deux arbres de la forêt.

Comment trouver la (une) plus petite arête reliant deux arbres de la forêt A' ?

Algorithme de Kruskal – idée

Deux caractéristiques :

- A' décrit une forêt,
- une arête compatible est une arête de poids minimal reliant deux arbres de la forêt.

Comment trouver la (une) plus petite arête reliant deux arbres de la forêt A' ?

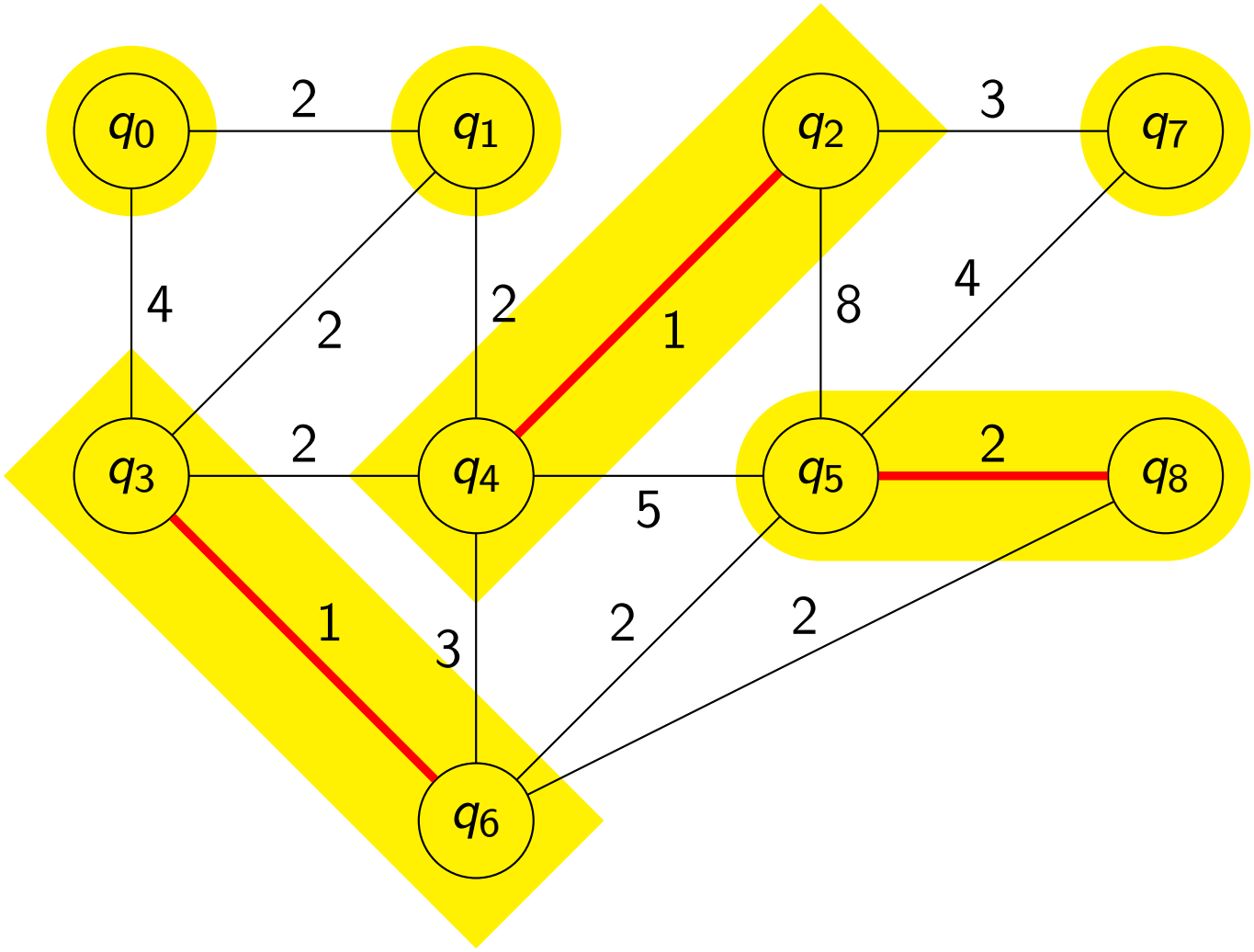
1. $A' = \emptyset$
2. On trie les arêtes par poids croissant.
3. Pour chaque arête (x, y) :

   Si (x, y) ne crée pas de cycle,

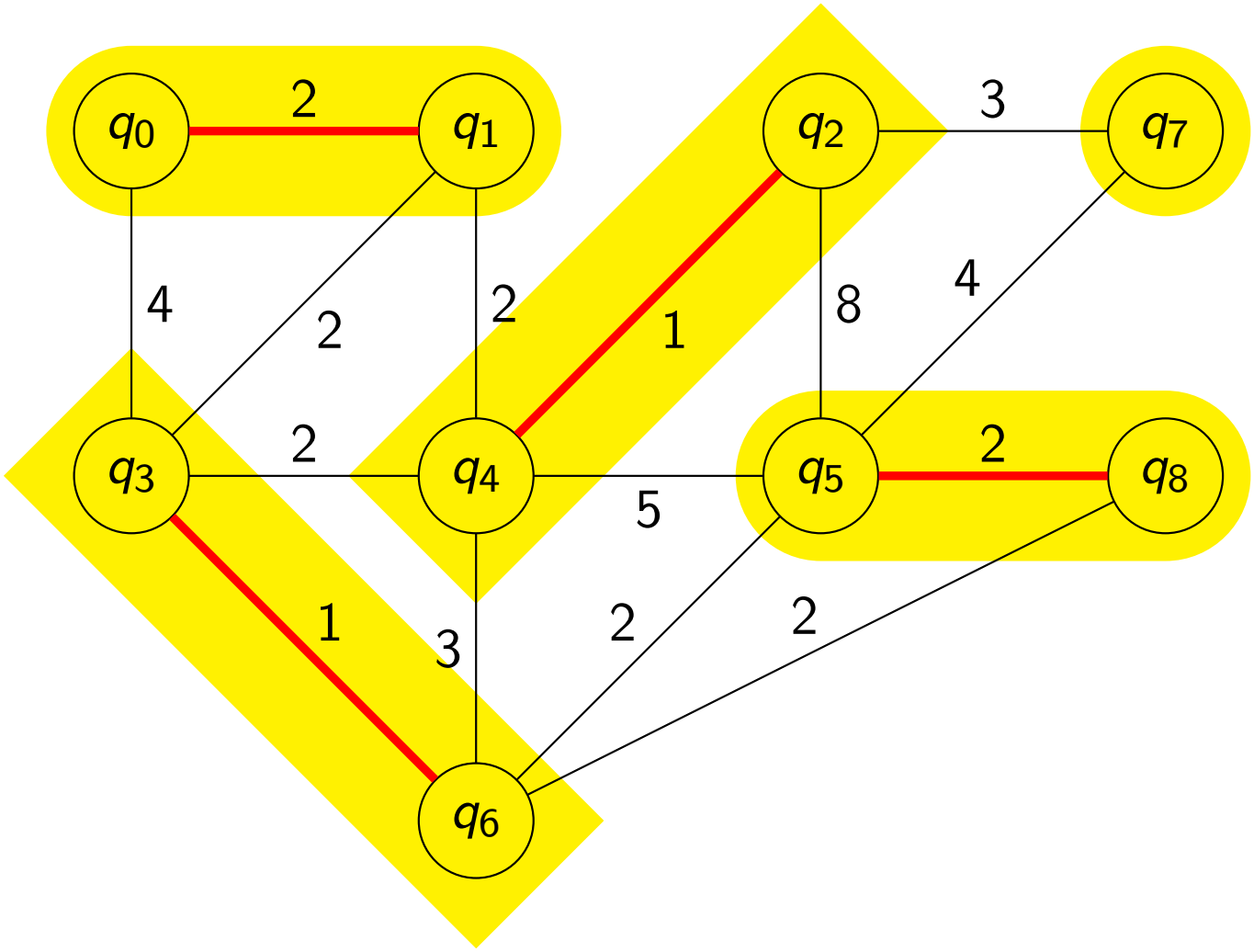
   alors on l'ajoute à A'

A' est une solution !

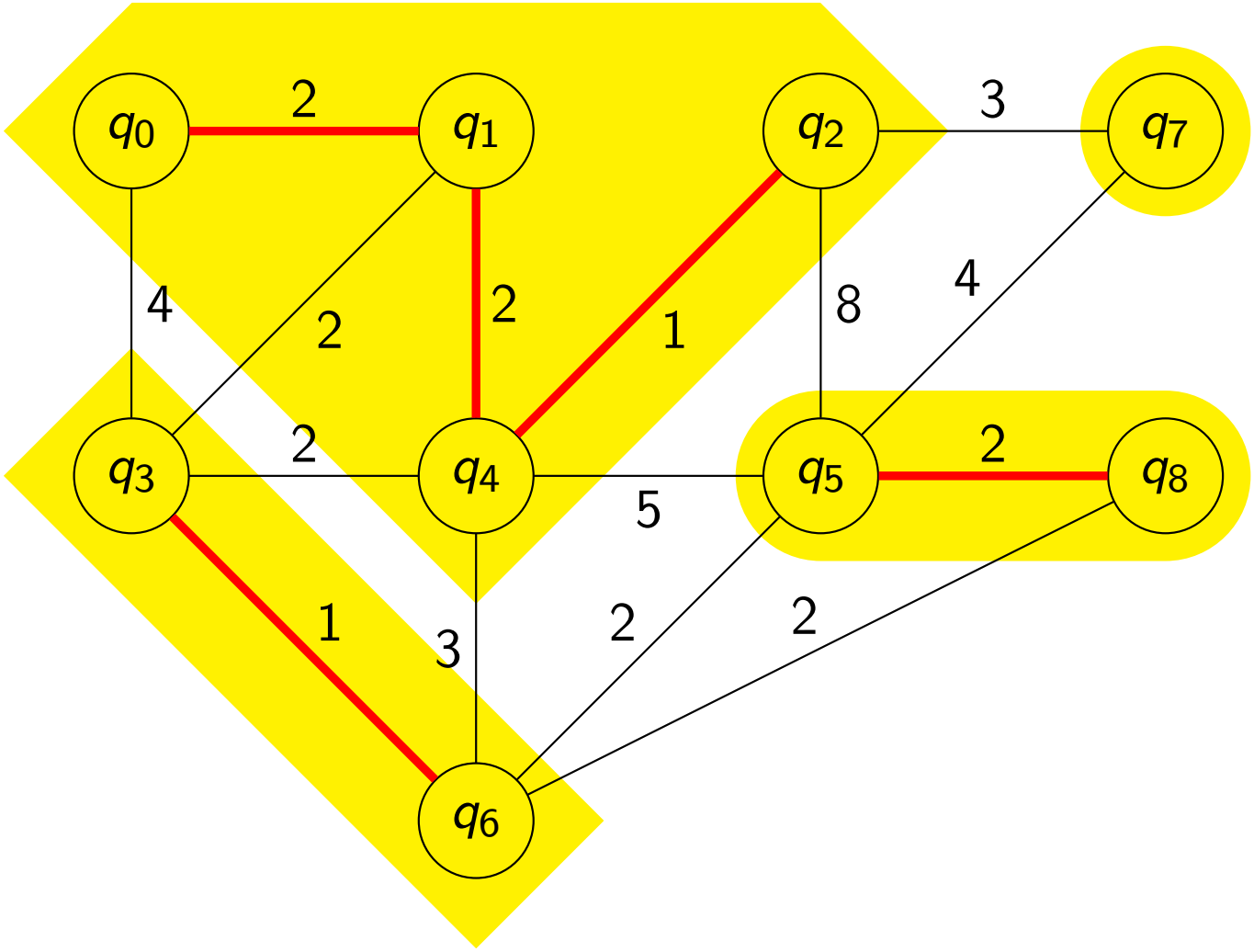
Algorithme de Kruskal



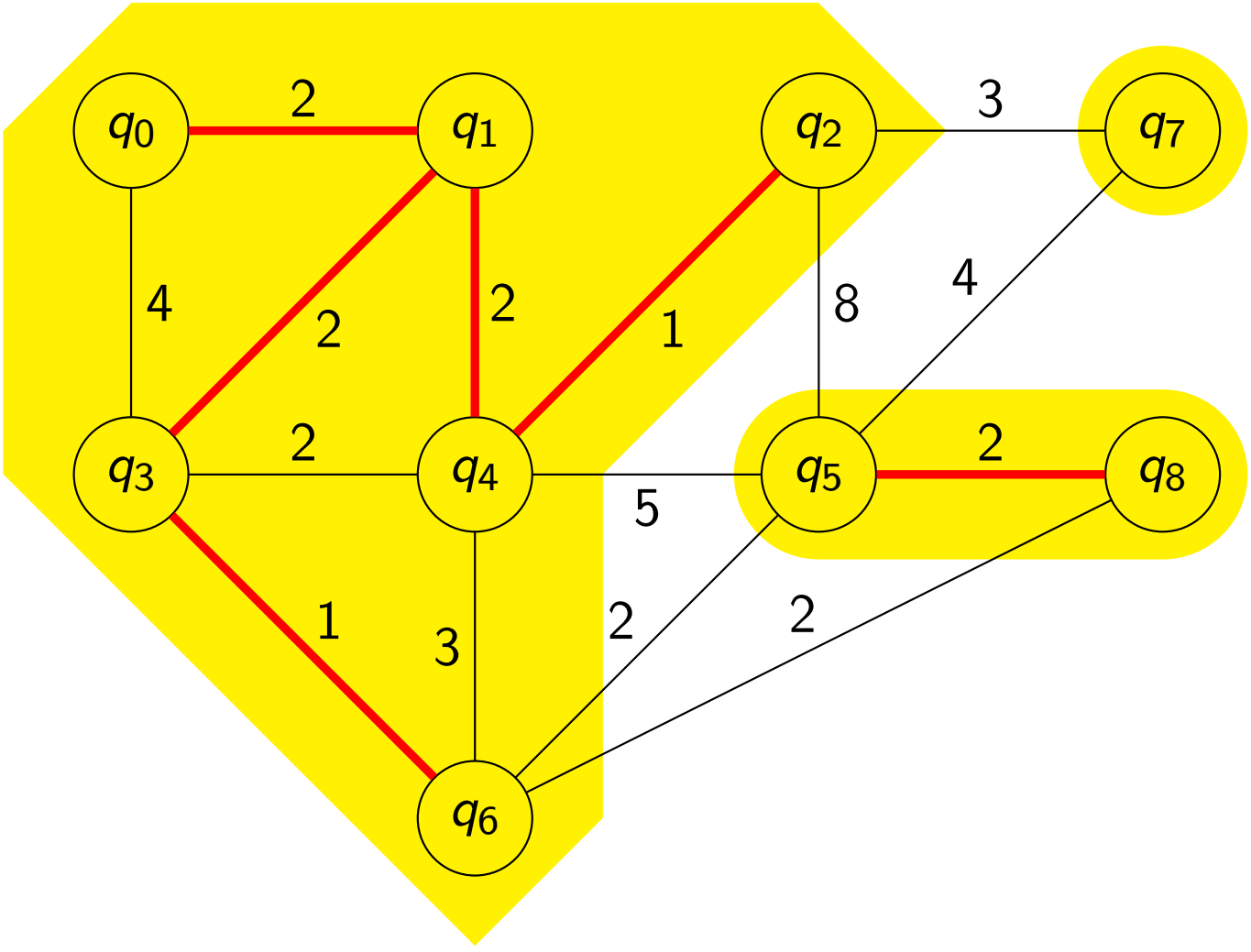
Algorithme de Kruskal



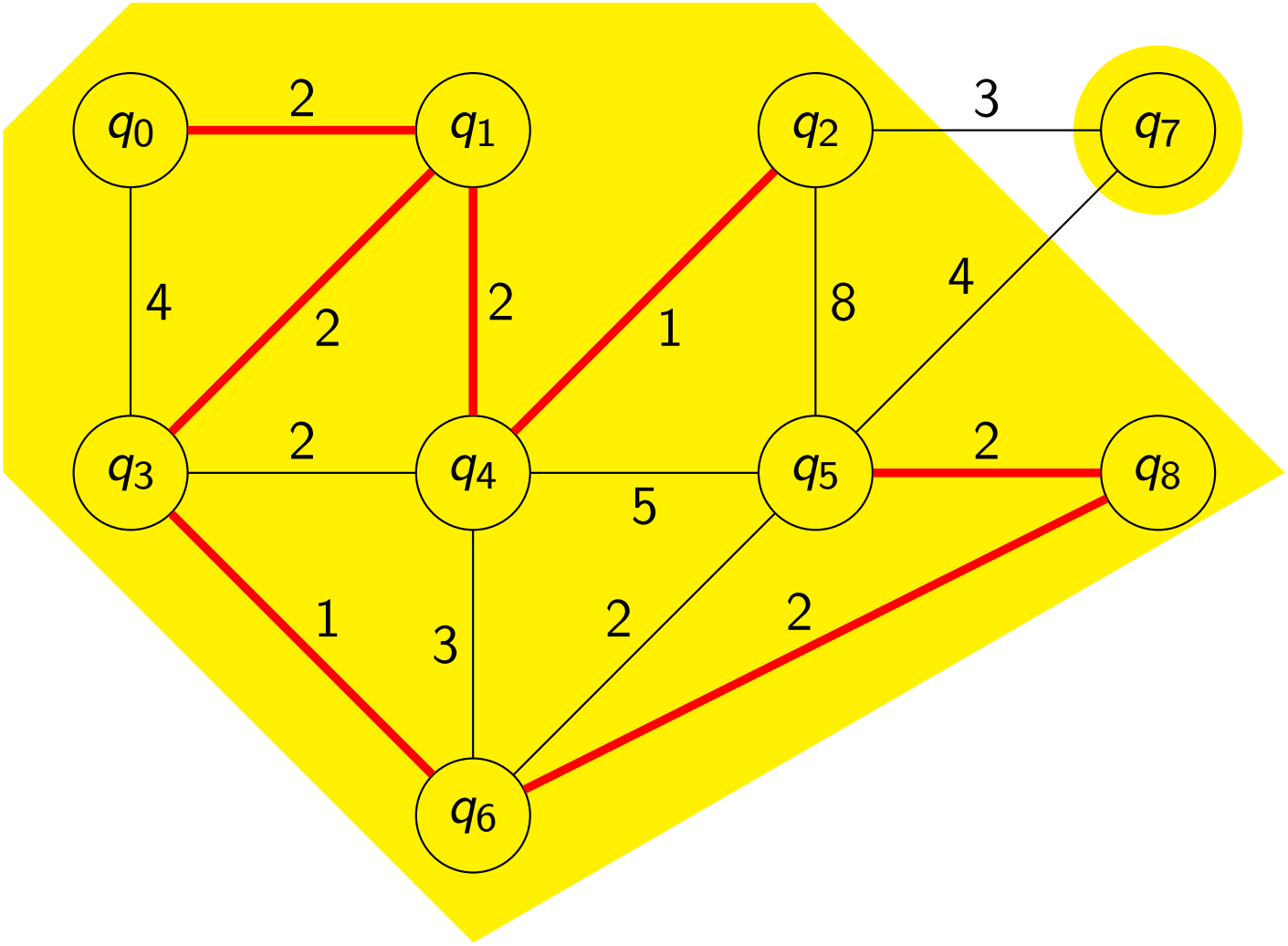
Algorithme de Kruskal



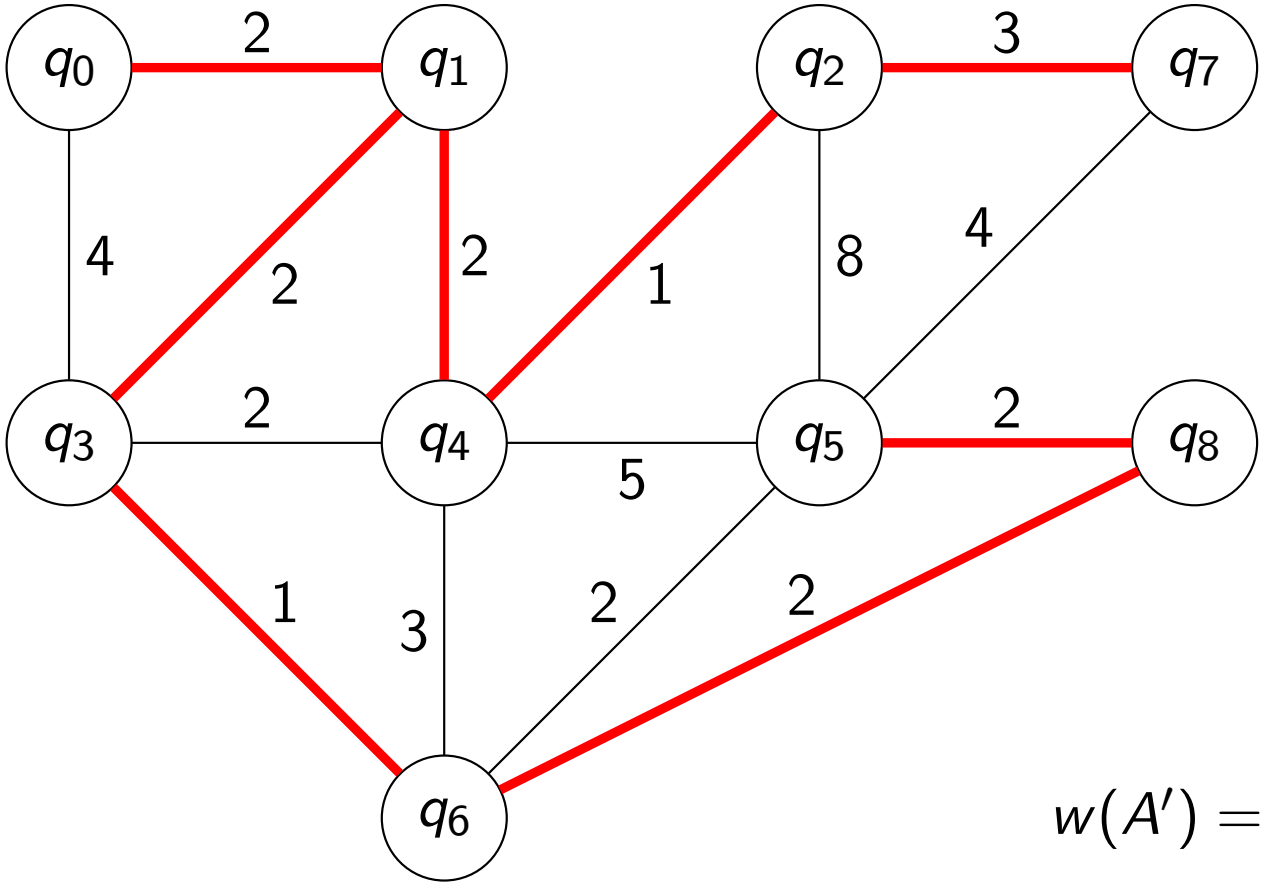
Algorithme de Kruskal



Algorithme de Kruskal



Algorithme de Kruskal



$w(A') = 15$

Comment tester si x et y sont dans le même arbre ?

Pour tester efficacement si deux sommets sont dans le même arbre de A', on utilise des structures $\ll$ Union-Find $\gg$:

- tester si x et y sont dans le même sous-ensemble :
  1. Représentant-Ens(x) associe à x un représentant canonique de l'ensemble contenant s.
  2. Représentant-Ens(x) = Représentant-Ens(y) : renvoie VRAI ssi x et y sont dans le même arbre.

- Fusion(x, y) fusionne les deux ensembles contenant x et y.

Complexité : le coût total de m opérations dont n opérations CréerEnsemble est en $O(m \cdot \alpha(m, n))$.
$\alpha(m, n)$: fct réciproque de la fct d'Ackermann. ($<<$ log(m)).

Algorithme de Kruskal

Procédure `Recherche-ACM-Kruskal`(G)
begin
    $A' := \emptyset$
    pour chaque $s \in S$ **faire**
        CréerEnsemble(s)
    Trier A par poids $w(u, v)$ croissant
    pour chaque $(x, y) \in A$ **faire**
        si *Représentant-Ens*$(x) \neq$ *Représentant-Ens*(y) **alors**
            $A' := A' \cup \{(x, y)\}$
            Fusion(x, y)
    return A'
end

Complexité de l'algorithme de Kruskal

Tri : $O(|A| \cdot \log(|A|))$

il y aura $|S| - 1$ arêtes ajoutées dans A'.
Il y aura donc $|S| - 1$ appels à Fusion$(-, -)$.

Au pire, il y aura $2 \cdot |A|$ appels à Représentant-Ens$(-)$.

Complexité totale en $O(|A| \cdot \log(|A|))$ car $|A| \geq |S| - 1$.

Plan

1 Définitions

2 Algorithme de Prim

3 Algorithme de Kruskal

4 Application au ≪ voyageur de commerce ≫

Le voyageur de commerce

Une ville, des routes, des distances...

Question : comment passer une et une seule fois par chaque ville en parcourant une distance minimale ?

Le voyageur de commerce

Une ville, des routes, des distances...

Question : comment passer une et une seule fois par chaque ville en parcourant une distance minimale ?

Données : un graphe valué non-orienté, connexe : $G = (S, A, w)$
Question : trouver un chemin hamiltonien de poids minimal.

Le voyageur de commerce

Une ville, des routes, des distances...

Question : comment passer une et une seule fois par chaque ville en parcourant une distance minimale ?

Données : un graphe valué non-orienté, connexe : $G = (S, A, w)$
Question : trouver un chemin hamiltonien de poids minimal.

C'est problème très classique... et NP-complet.

Approximation du VdC avec les ACM

On fait les hypothèss suivantes :

- $G = (S, A, w)$ est complet.
  $(\forall x, y \in S, \ \exists (x, y) \in A)$
- $G = (S, A, w)$ vérifie l'inégalité triangulaire :
  $\forall x, y, z \in S$, on a : $w(x, y) + w(y, z) \geq w(x, z)$

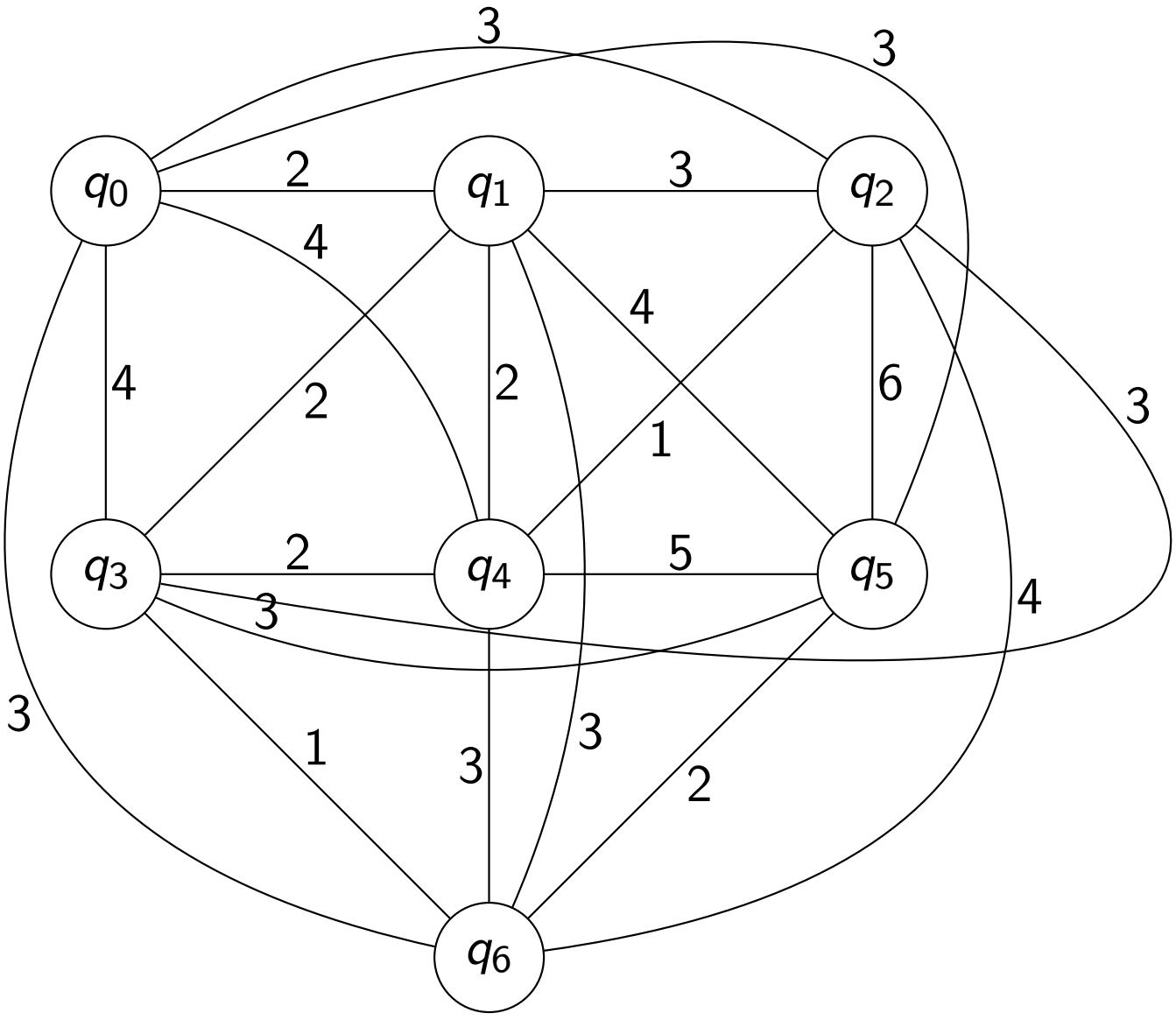
Et on va essayer d'approximer le circuit hamiltonien minimal...

Approximation du VdC

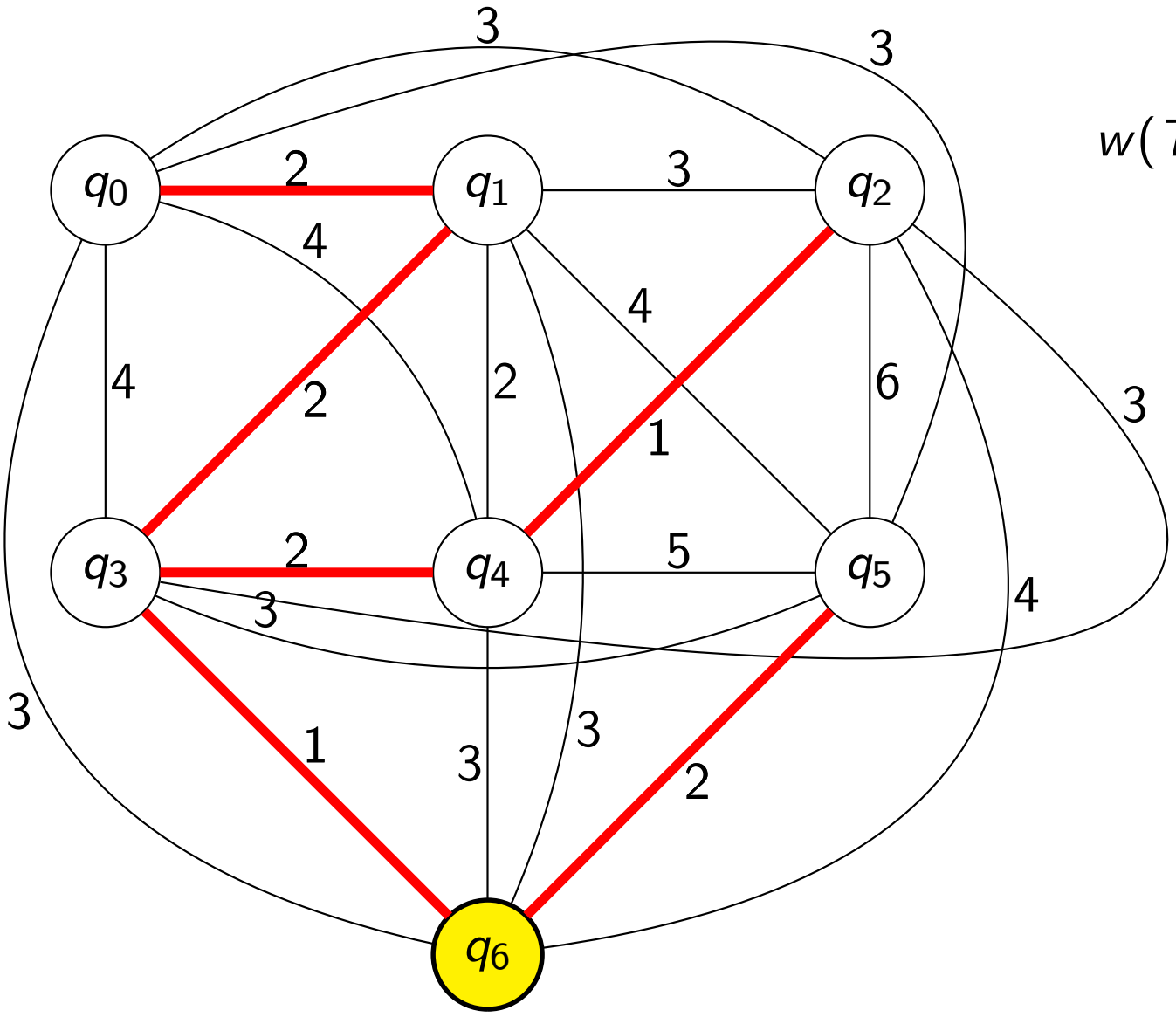
Fonction Approx-VdC(Graphe $G = (S, A, w)$)

1. choisir un sommet $s \in S$
2. construire un arbre couvrant minimal T de G à partir de s (avec l'algorithme de Prim, ou Kruskal,...)
3. soit L la liste des sommets visités lors d'un parcours préfixe de T
4. retourner le cycle correspondant à L

Exemple

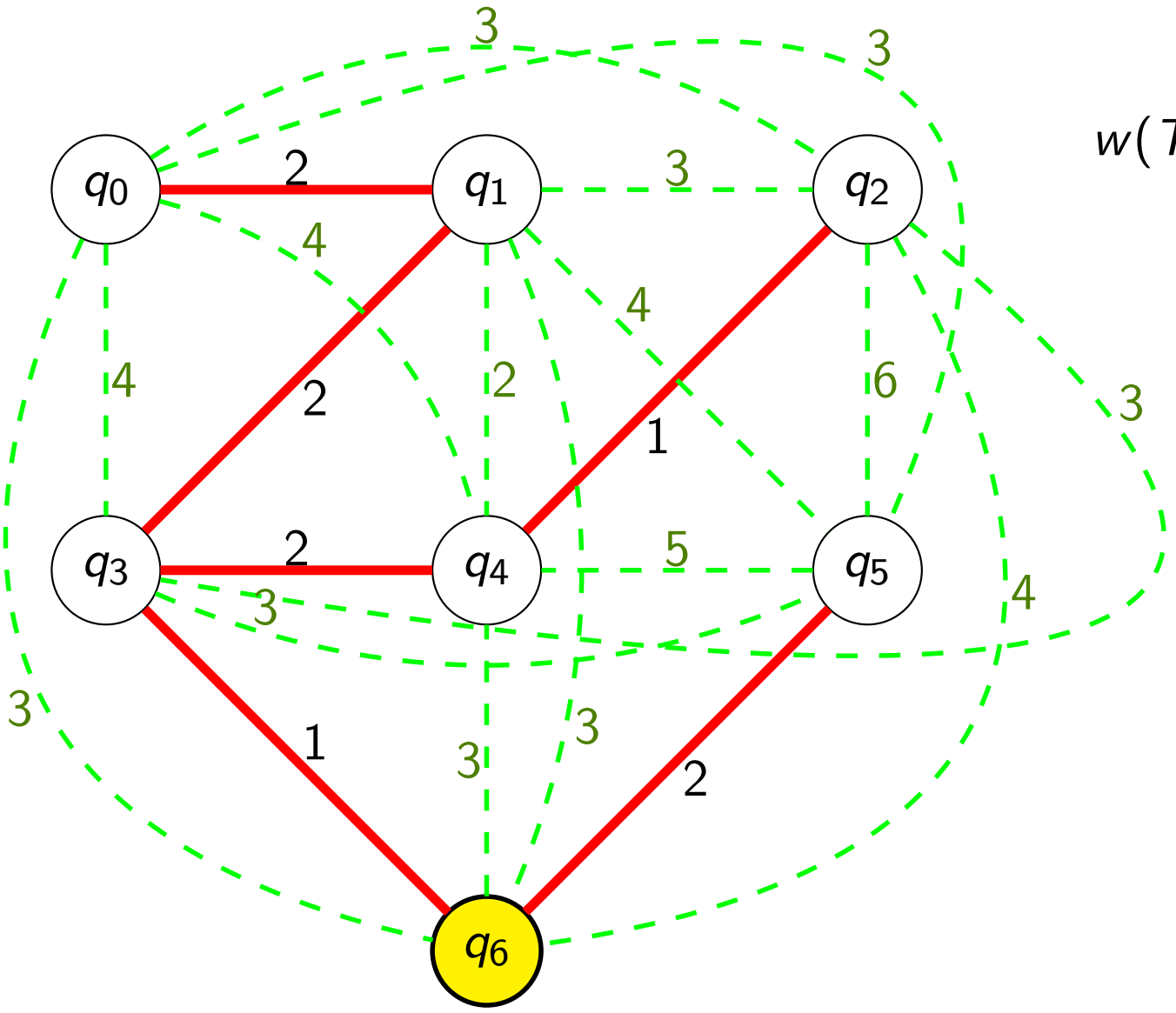


Exemple



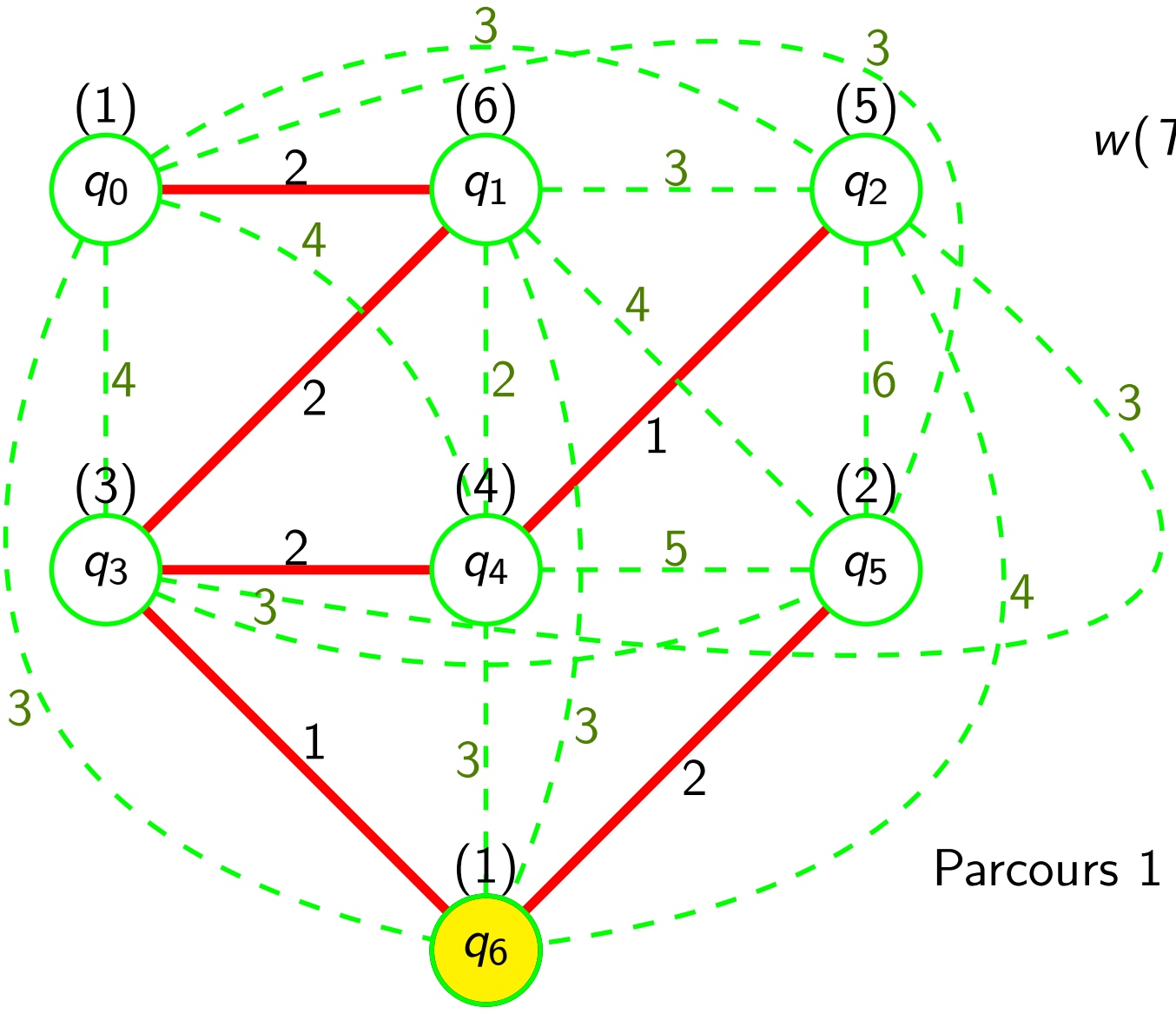
$w(T_{acm}) = 10$

Exemple



$w(T_{acm}) = 10$

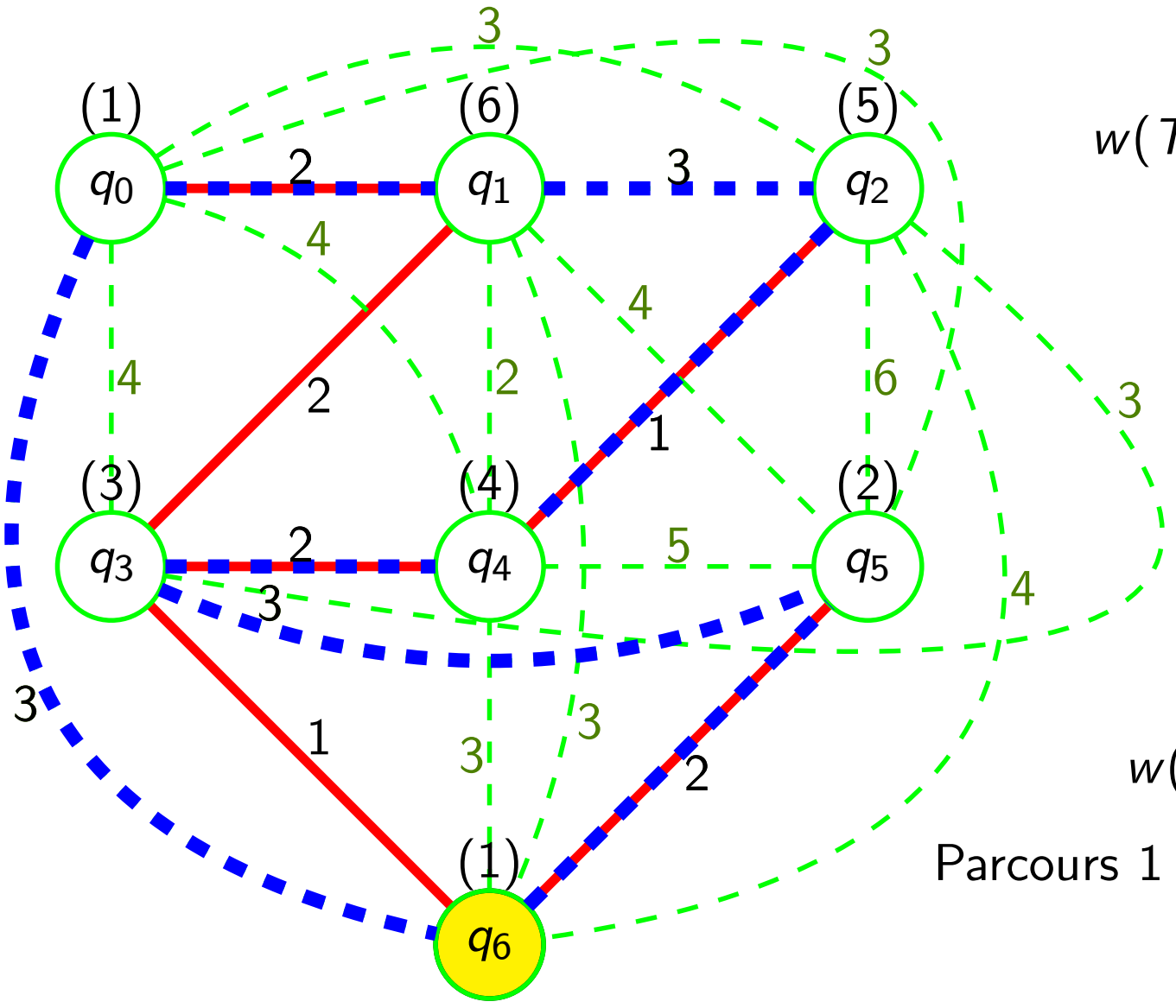
Exemple



$w(T_{acm}) = 10$

Parcours 1 : $q_6 q_5 q_3 q_4 q_2 q_1 q_0$

Exemple

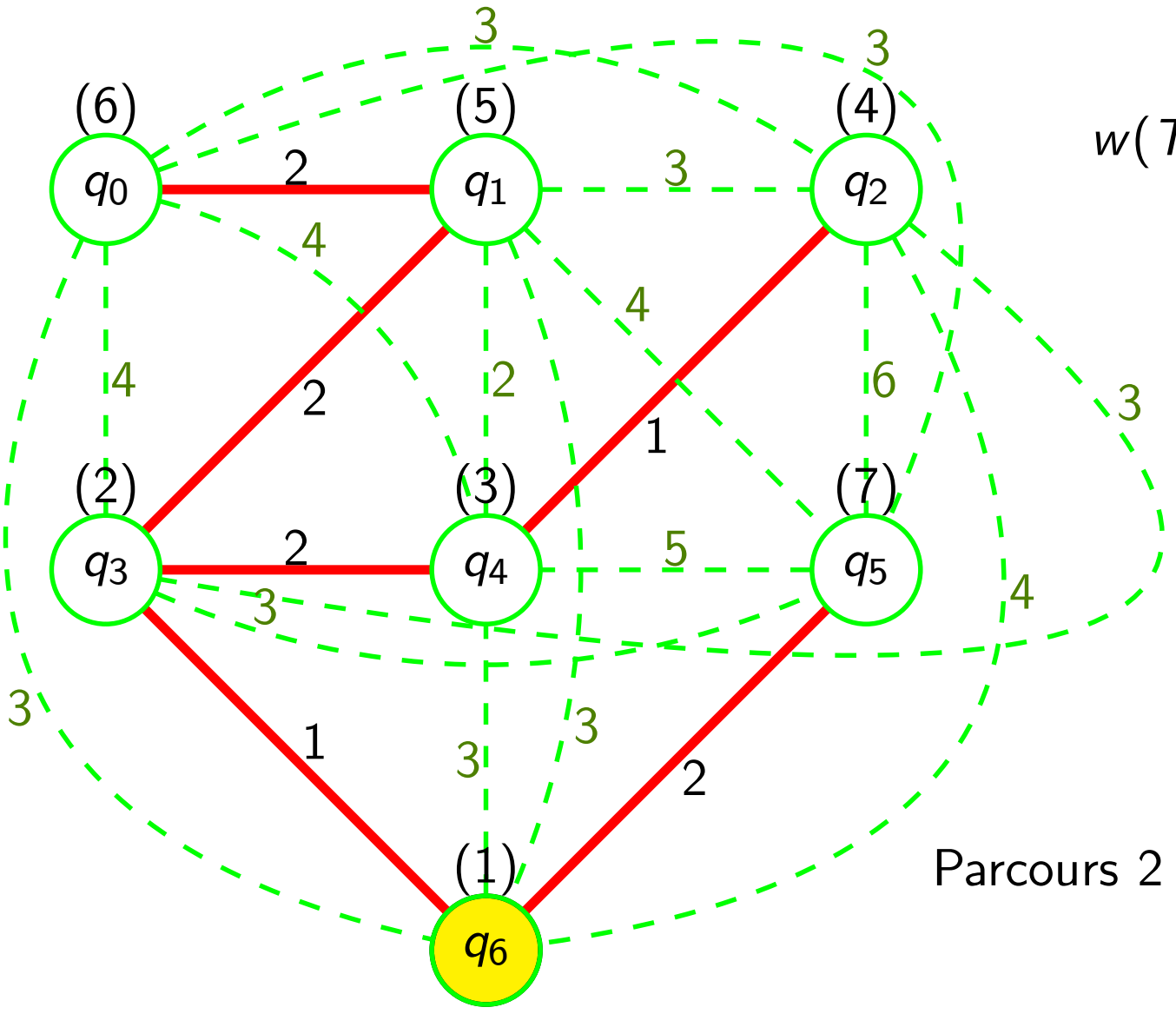


$w(T_{acm}) = 10$

$w(C) = 16$

Parcours 1 : $q_6 q_5 q_3 q_4 q_2 q_1 q_0$

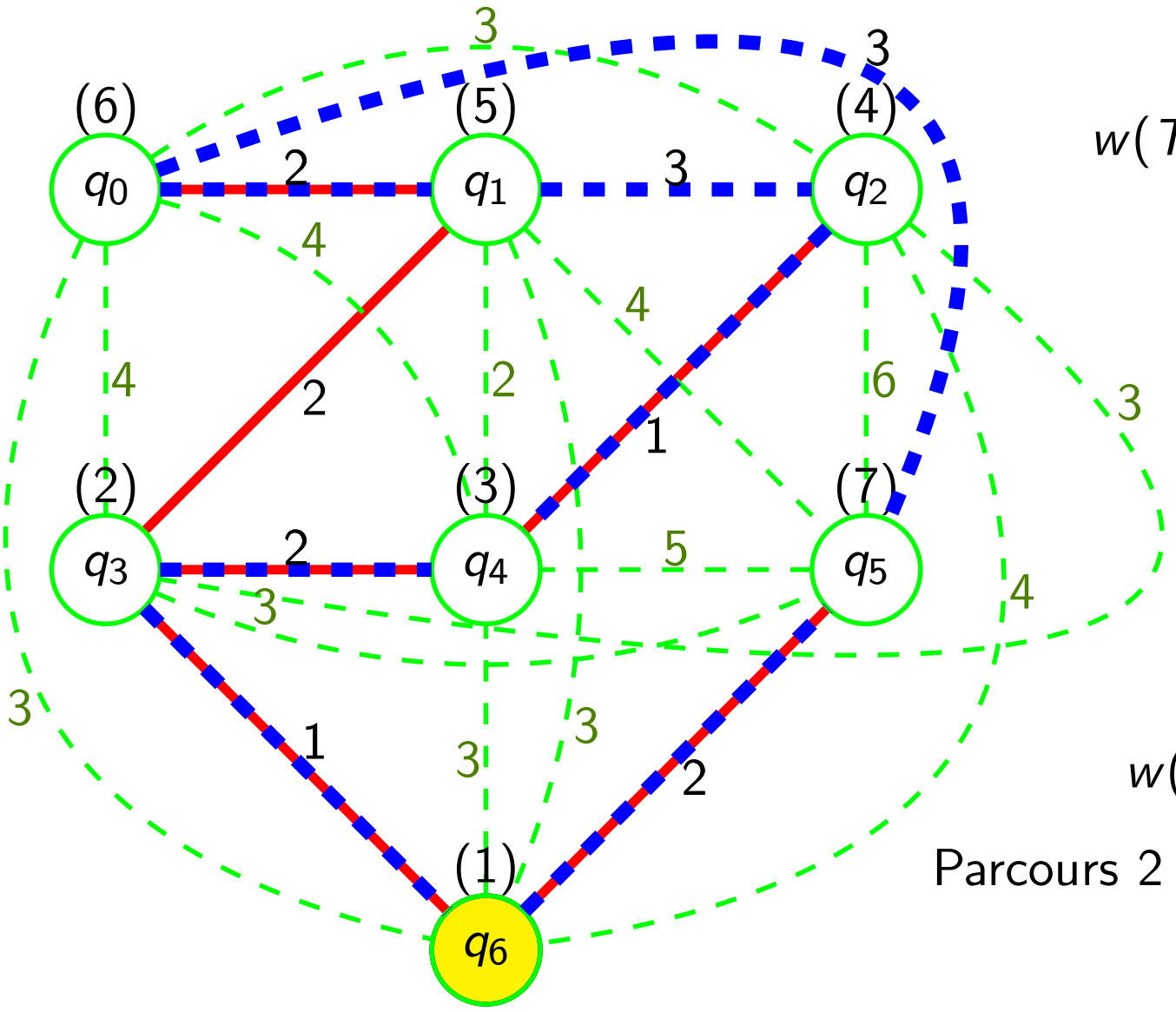
Exemple



$w(T_{acm}) = 10$

Parcours 2 : $q_6 q_3 q_4 q_2 q_1 q_0 q_5$

Exemple



$w(T_{acm}) = 10$

$w(C) = 14$

Parcours 2 : $q_6 q_3 q_4 q_2 q_1 q_0 q_5$

Borner l'approximation

Théorème **Approx-VdC** *donne un cycle C dont le coût est inférieur à $(2 \cdot w(S_{opt}))$ où S_{opt} est une solution optimale du problème.*

Pourquoi ?

Borner l'approximation

Théorème **Approx-VdC** *donne un cycle C dont le coût est inférieur à $(2 \cdot w(S_{opt}))$ où S_{opt} est une solution optimale du problème.*

Pourquoi ?

Complexité : le coût de la recherche de l'ACM...