Introduction à l'algorithmique, structures de contrôle et de données

Informatique et sciences du numérique
9 juin 2010

François Laroussinie
LIAFA, Univ. Paris-Diderot – CNRS

Plan

Plan

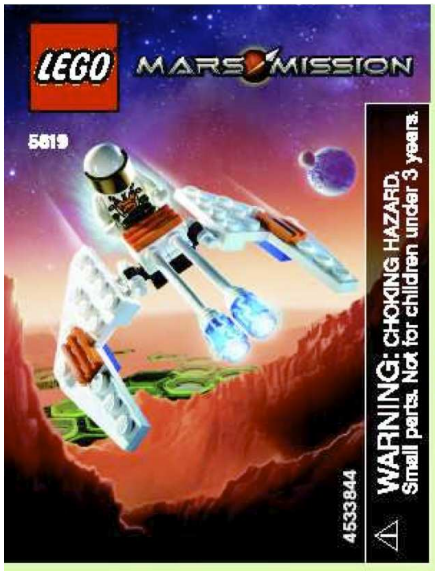
Vocabulaire

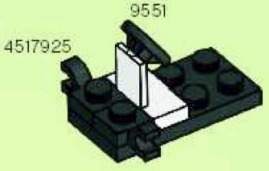
- un ordinateur = un objet, des ressources matérielles (le hardware).
- un algorithme = une recette, une méthode pour obtenir un résultat.
- un programme = un algorithme écrit dans un langage de programmation.
- un problème algorithmique = un problème à résoudre.

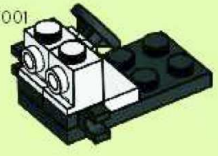
Vocabulaire

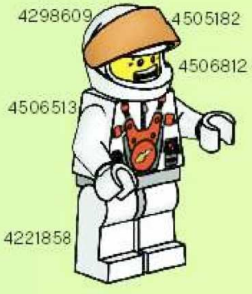
- un ordinateur = un objet, des ressources matérielles (le hardware).
- un algorithme = une recette, une méthode pour obtenir un résultat.
- un programme = un algorithme écrit dans un langage de programmation.
- un problème algorithmique = un problème à résoudre.

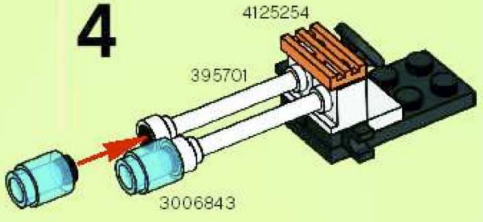
Des exemples !

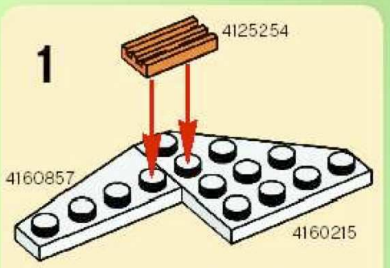


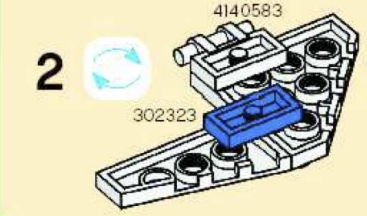


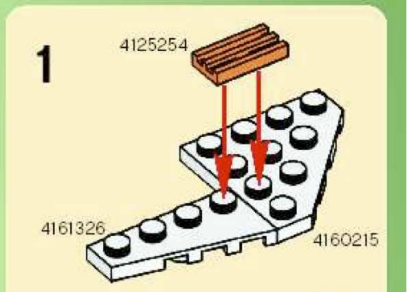


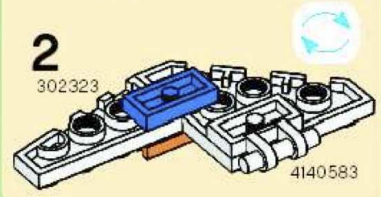


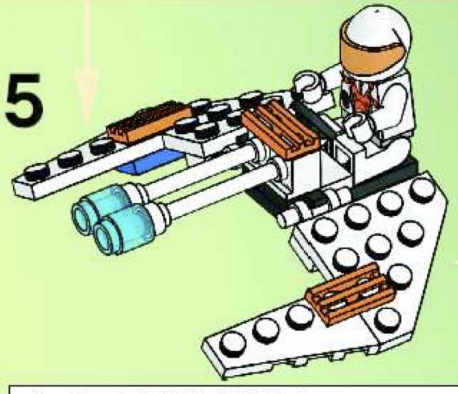












⚠ **AVERTISSEMENT :** RISQUE D'ÉTOUFFEMENT.
Contient des petites pièces. Ne convient pas aux enfants de moins de 3 ans.

⚠ **ADVERTENCIA:** PELIGRO DE ASFIXIA.
No recomendado para niños menores de 3 años.
Contiene piezas pequeñas.

SERVICE
WWW.LEGO.COM

484 Émincé de rouget au pistou

1 kg 500 rougets.
400 g pâtes.
500 g fenouil.
2 dl huile olive.
0 dl 5 vinaigre xérès.
Coulis de tomates.
1 citron.
Basilic.
Herbes de Provence.
Thym.
Vinaigrette.
Sel.
Poivre.

24 heures à l'avance
Préparation : 40 mn - Cuisson : 30 à 35 mn

Écailler les rougets, les laver, puis soulever délicatement les filets et enlever les arêtes qui restent avec une pince à épiler. Mettre les filets dans un plat creux ; arroser avec de l'huile d'olive ; saupoudrer d'herbes de Provence et laisser mariner au frais, pendant 24 heures, le tout couvert par un torchon.
Cuire le fenouil à l'eau bouillante salée, citronnée et parfumée à l'huile d'olive et avec une brindille de thym (15 à 20 mn). Égoutter.
Cuire les pâtes à l'eau bouillante salée, huilée, pendant 12 mn. Égoutter. Rafraîchir. Tenir au chaud.
Pendant ces cuissons préparer le coulis de tomates (41).
Assaisonner le fenouil coupé en tranches avec la vinaigrette. Tenir au chaud. Assaisonner les pâtes avec le basilic (à volonté) haché et un mélange d'huile d'olive et de vinaigre de xérès. Disposer les pâtes sur le plat, recouvrir avec la fondue de fenouil.
Rapidement, cuire à la poêle Téfal, à feu très vif, les filets marinés pour qu'ils deviennent dorés et croustillants. Saler. Poivrer et disposer en étoile les filets de rougets, sur le plat de légumes. Servir avec le coulis de tomates en saucière.

Algorithme

Un algorithme est la description univoque d'une méthode effective pour résoudre un problème, exprimée à l'aide d'une suite d'instructions élémentaires.

Problèmes algorithmiques

- Données : un entier a
  Résultat : la somme $1 + 2 + 3 + \ldots + a$
- Données : une liste de mots
  Résultat : la liste triée dans l'ordre alphabétique
- Données : une liste de mots
  Résultat : une liste de pages Web contenant ces mots
- Données : une carte, deux villes A et B
  Résultat : un plus court chemin entre A et B
- Données : un programme P, une donnée a
  Résultat : réponse à "est-il vrai que le résultat de P appliqué à a vaut 8a+5" ?
- ...

Un algorithme doit résoudre toutes les instances d'un problème.

Exemple

La recherche du minimum dans une liste d'entiers...

Exemple

La recherche du minimum dans une liste d'entiers...

1. Noter le premier entier de la liste
2. Pour tous les entiers suivants, faire :
   Si l'entier est inférieur à celui noté
   Alors remplacer celui-ci par le nouvel entier
3. Renvoyer le nombre noté.

Exemple

La recherche du minimum dans une liste d'entiers...
En pseudo-code :

Recherche du minimum

Données : liste non vide d'entiers L
Résultat : entier m le plus petit de L

$m \leftarrow L(1)$
pour chaque i *allant de* 2 *à* $|L|$ **faire**
  si $L(i) < m$ **alors** $m \leftarrow L(i)$
retourner m

Exemple

La recherche du minimum dans une liste d'entiers...
En pseudo-code :

Recherche du minimum

Données : liste non vide d'entiers L
Résultat : entier m le plus petit de L

$m \leftarrow L(1)$
pour chaque i *allant de* 2 *à* $|L|$ **faire**
  si $L(i) < m$ **alors** $m \leftarrow L(i)$
retourner m

Des instances :

- [3, 10, 5, 24, 2, 12]
- [10]

Un problème... des algorithmes ?

Pour un problème donné, il peut y avoir plusieurs algorithmes différents... ou aucun ! (cf. l'exposé de Paul Gastin du 2 juin).

Un problème. . . des algorithmes ?

Pour un problème donné, il peut y avoir plusieurs algorithmes différents. . . ou aucun ! (cf. l'exposé de Paul Gastin du 2 juin).

Lorsqu'il existe plusieurs algorithmes, on peut les comparer selon plusieurs critères :

- les idées sous-jacentes, leur structure (récursif / itératif, glouton, prog. dynamique, diviser pour régner,. . .),
- les structures de données utilisées,
- la complexité algorithmique (*i.e.* les ressources – temps, mémoire– nécessaires à son exécution).

Complexité d'un algorithme

Complexité en temps ou en espace mémoire.

Complexité d'un algorithme

Complexité en temps ou en espace mémoire.

Objectif : trouver un ordre de grandeur du nombre d'opérations élémentaires nécessaires à l'exécution de l'algorithme.

On ne veut pas mesurer le temps nécessaire en minutes ou microsecondes.

On veut une notion robuste : indépendante d'un ordinateur donné, d'un compilateur, d'un langage de programmation, *etc.* et exprimée en fonction de la taille de la donnée à traiter.

Complexité d'un algorithme

Complexité en temps ou en espace mémoire.

Objectif : trouver un ordre de grandeur du nombre d'opérations élémentaires nécessaires à l'exécution de l'algorithme.

On ne veut pas mesurer le temps nécessaire en minutes ou microsecondes.

On veut une notion robuste : indépendante d'un ordinateur donné, d'un compilateur, d'un langage de programmation, *etc.* et exprimée en fonction de la taille de la donnée à traiter.

opération élémentaire : opération qui prend un temps constant (ou presque).

$\rightarrow$ ce choix dépend du problème.

(Recherche du minimum : $n - 1$ comparaisons sont faites.)

Complexité d'un algorithme

Coût de $\mathcal{A}$ sur x : l'exécution de l'algorithme $\mathcal{A}$ sur la donnée x requiert $C_{\mathcal{A}}(x)$ opérations élémentaires.

Complexité d'un algorithme

Coût de $\mathcal{A}$ sur x : l'exécution de l'algorithme $\mathcal{A}$ sur la donnée x requiert $C_{\mathcal{A}}(x)$ opérations élémentaires.

Complexité dans le pire cas :

$$\mathcal{C}_{\mathcal{A}}(n) \stackrel{\text{def}}{=} \max_{x.|x|=n} C_{\mathcal{A}}(x)$$

Complexité d'un algorithme

Coût de $\mathcal{A}$ sur x : l'exécution de l'algorithme $\mathcal{A}$ sur la donnée x requiert $C_{\mathcal{A}}(x)$ opérations élémentaires.

Complexité dans le pire cas :

$$\mathcal{C}_{\mathcal{A}}(n) \stackrel{\text{def}}{=} \max_{x.|x|=n} C_{\mathcal{A}}(x)$$

Complexité en moyenne

$$\mathcal{C}^{moy}_{\mathcal{A}}(n) \stackrel{\text{def}}{=} \sum_{x.|x|=n} p(x) \cdot C_{\mathcal{A}}(x)$$

p : distribution de probabilités sur les données de taille n.

Algorithmes efficaces. . . et au delà !

Quelques familles d'algorithmes selon leur complexité :

- les algorithmes sous-linéaires : par exemple, en $O(\log(n))$
- les algorithmes linéaires : $O(n)$
  et quasi-linaires : $O(n \cdot \log(n))$
- les algorithmes polynomiaux : $O(n^k)$

Algorithmes efficaces. . . et au delà !

Quelques familles d'algorithmes selon leur complexité :

- les algorithmes sous-linéaires : par exemple, en $O(\log(n))$
- les algorithmes linéaires : $O(n)$
  et quasi-linaires : $O(n \cdot \log(n))$
- les algorithmes polynomiaux : $O(n^k)$
- les algorithmes exponentiels : $O(2^{p(n)})$
- . . . doublement exponentiels : $O(2^{2^{p(n)}})$
- . . .

NB : $O(g(n))$ contient l'ensemble des fonctions majorées par un $c \cdot g(n)$ avec $c > 0$ au delà d'un certain n_0. . .

Algorithmes efficaces. . . ou pas !

Considérons un algorithme de complexité $f(n)$ qui permette de résoudre en une heure les instances de taille X d'un problème sur un ordinateur aujourd'hui.
Alors un ordinateur 1000 fois plus rapide permettrait en une heure de résoudre les instances de taille. . .

- $1000 \cdot X$ si $f(n) = n$,

Algorithmes efficaces. . . ou pas !

Considérons un algorithme de complexité $f(n)$ qui permette de résoudre en une heure les instances de taille X d'un problème sur un ordinateur aujourd'hui.
Alors un ordinateur 1000 fois plus rapide permettrait en une heure de résoudre les instances de taille. . .

- $1000 \cdot X$ si $f(n) = n$,
- $31,6 \cdot X$ si $f(n) = n^2$,

Algorithmes efficaces. . . ou pas !

Considérons un algorithme de complexité $f(n)$ qui permette de résoudre en une heure les instances de taille X d'un problème sur un ordinateur aujourd'hui.
Alors un ordinateur 1000 fois plus rapide permettrait en une heure de résoudre les instances de taille. . .

- $1000 \cdot X$ si $f(n) = n$,
- $31,6 \cdot X$ si $f(n) = n^2$,
- $X + 9,97$ si $f(n) = 2^n$.

(voir "Algorithmics, the spirit of computing", D. Harel)

A la recherche du bon algorithme

Étant donné un problème, on cherche donc des algorithmes :

- corrects : qu'ils calculent bien ce que l'on veut,
- efficaces : nécessitant des ressources (temps, mémoire) raisonnables.

A la recherche du bon algorithme

Étant donné un problème, on cherche donc des algorithmes :

- corrects : qu'ils calculent bien ce que l'on veut,
- efficaces : nécessitant des ressources (temps, mémoire) raisonnables.

Peu de problèmes admettent des algorithmes utilisables en pratique.

Il est important de savoir concevoir, vérifier et analyser des algorithmes.

Et maintenant ?

Deux parties :

1) des algorithmes de tri.

2) des algorithmes dans les graphes.

- Plusieurs formes d'algorithmes (backtracking, diviser-pour-régner, glouton, programmation dynamique),
- des structures de données très différentes,
- des résultats de complexité.
- Et basé sur des exemples d'applications...

Plan

Le tri

Problème : étant donné un tableau d'entiers T, trier T dans l'ordre croissant.

- Problème connu
- Grand classique de l'enseignement en algorithmique
- Opération très fréquente en algorithmique
- Grande richesse conceptuelle

Plan

Le tri par **sélection**

- Trouver le plus petit élément et le mettre au début de la liste

Le tri par **sélection**

- Trouver le plus petit élément et le mettre au début de la liste
- Trouver le 2^e plus petit et le mettre en seconde position

Le tri par **sélection**

- Trouver le plus petit élément et le mettre au début de la liste
- Trouver le 2^e plus petit et le mettre en seconde position
- Trouver le 3^e plus petit élément et le mettre à la 3^e place,

Le tri par **sélection**

- Trouver le plus petit élément et le mettre au début de la liste
- Trouver le 2^e plus petit et le mettre en seconde position
- Trouver le 3^e plus petit élément et le mettre à la 3^e place,
- ...

Le tri par **sélection**

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

Le tri par **sélection**

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

3

Le tri par **sélection**

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

3, 5

Le tri par **sélection**

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

3, 5, 7

Le tri par **sélection**

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

3, 5, 7, 10

Le tri par **sélection**

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

3, 5, 7, 10, 12

Le tri par **sélection**

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

3, 5, 7, 10, 12, 13

Le tri par **sélection**

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

3, 5, 7, 10, 12, 13, 15

Le tri par **sélection**

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

3, 5, 7, 10, 12, 13, 15, 16

Le tri par **sélection**

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

3, 5, 7, 10, 12, 13, 15, 16, 19

Le tri par **sélection**

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

3, 5, 7, 10, 12, 13, 15, 16, 19, 20

Le tri par **sélection**

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

3, 5, 7, 10, 12, 13, 15, 16, 19, 20, 25

Le tri par **sélection**

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

3, 5, 7, 10, 12, 13, 15, 16, 19, 20, 25, 35

Le tri par **sélection**

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

3, 5, 7, 10, 12, 13, 15, 16, 19, 20, 25, 35, 38

Le tri par **sélection**

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

3, 5, 7, 10, 12, 13, 15, 16, 19, 20, 25, 35, 38, 40

Le tri par **sélection**

Tri par sélection

Données : Un tableau de n entiers T

Résultat : Le tableau T trié

pour chaque *i allant de* 1 *à* $n-1$ **faire**
    $ind \leftarrow$ `Indice-Min`(T, i, n)
    $T[i] \leftrightarrow T[ind]$

retourner T

`Indice-Min`(T, i, n) : retourne l'indice du plus petit élément de $\{T[i], T[i+1], \ldots, T[n]\}$.

Le tri par **sélection**

Tri par sélection

Données : Un tableau de n entiers T

Résultat : Le tableau T trié

pour chaque i *allant de* 1 *à* $n-1$ **faire**
    $ind \leftarrow$ `Indice-Min`(T, i, n)
    $T[i] \leftrightarrow T[ind]$

retourner T

`Indice-Min`(T, i, n) : retourne l'indice du plus petit élément de $\{T[i], T[i+1], \ldots, T[n]\}$.

Propriété : Après la i^e étape ($i = 1, \ldots, n-1$), les i premières cases sont occupées par les i plus petits entiers de T

Complexité du tri par sélection

Tri par sélection

Données : Un tableau de n entiers T
Résultat : Le tableau T trié

pour chaque i *allant de* 1 *à* $n-1$ **faire**
    $ind \leftarrow$ `Indice-Min`(T, i, n)
    $T[i] \leftrightarrow T[ind]$
retourner T

Dans le pire cas ou en moyenne, la complexité (ici : nombre de comparaisons) du tri par sélection est en $O(n^2)$.

Le tri par **insertion**

(le tri du joueur de cartes !)

- Ordonner les deux premiers éléments

Le tri par **insertion**

(le tri du joueur de cartes !)

- Ordonner les deux premiers éléments
- **Insérer** le 3^e élément de manière à ce que les 3 premiers éléments soient triés

Le tri par **insertion**

(le tri du joueur de cartes !)

- Ordonner les deux premiers éléments
- **Insérer** le 3^e élément de manière à ce que les 3 premiers éléments soient triés
- **Insérer** le 4^e élément à “sa” place pour que...

Le tri par **insertion**

(le tri du joueur de cartes !)

- Ordonner les deux premiers éléments
- **Insérer** le 3^e élément de manière à ce que les 3 premiers éléments soient triés
- **Insérer** le 4^e élément à “sa” place pour que...
- ...

Le tri par **insertion**

(le tri du joueur de cartes !)

- Ordonner les deux premiers éléments
- **Insérer** le 3^e élément de manière à ce que les 3 premiers éléments soient triés
- **Insérer** le 4^e élément à “sa” place pour que...
- ...
- **Insérer** le n^e élément à sa place.

Le tri par **insertion**

(le tri du joueur de cartes !)

- Ordonner les deux premiers éléments
- **Insérer** le 3^e élément de manière à ce que les 3 premiers éléments soient triés
- **Insérer** le 4^e élément à “sa” place pour que...
- ...
- **Insérer** le n^e élément à sa place.

Le tri par **insertion**

(le tri du joueur de cartes !)

- Ordonner les deux premiers éléments
- **Insérer** le 3^e élément de manière à ce que les 3 premiers éléments soient triés
- **Insérer** le 4^e élément à "sa" place pour que...
- ...
- **Insérer** le n^e élément à sa place.

A la fin de la i^e itération, les i premiers éléments de T sont triés et rangés au début du tableau T'.

Le tri par insertion

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

, 20

Le tri par insertion

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

15, 20

Le tri par insertion

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

10, 15, 20

Le tri par insertion

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

10, 15, 20, 35

Le tri par insertion

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

10, 15, 19, 20, 35

Le tri par insertion

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

10, 13, 15, 19, 20, 35

Le tri par insertion

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

5, 10, 13, 15, 19, 20, 35

Le tri par insertion

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

3, 5, 10, 13, 15, 19, 20, 35

Le tri par insertion

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

3, 5, 10, 12, 13, 15, 19, 20, 35

Le tri par insertion

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

3, 5, 7, 10, 12, 13, 15, 19, 20, 35

Le tri par insertion

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

3, 5, 7, 10, 12, 13, 15, 16, 19, 20, 35

Le tri par insertion

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

3, 5, 7, 10, 12, 13, 15, 16, 19, 20, 35, 40

Le tri par insertion

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

3, 5, 7, 10, 12, 13, 15, 16, 19, 20, 25, 35, 40

Le tri par insertion

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

3, 5, 7, 10, 12, 13, 15, 16, 19, 20, 25, 35, 38, 40

Le tri par insertion

Pour $i = 2 \ldots n$: Insérer(T, i)

Le tri par insertion

Pour $i = 2 \ldots n$: Insérer(T, i)

Insérer(T, k)

---

si $k > 1$ **alors**
  si $T[k-1] > T[k]$ **alors**
    $T[k] \leftrightarrow T[k-1]$
    Insérer(T,k-1)

Le tri par insertion

Pour $i = 2 \ldots n$: Insérer(T, i)

Insérer(T, k)

---

si $k > 1$ **alors**
  si $T[k-1] > T[k]$ **alors**
    $T[k] \leftrightarrow T[k-1]$
    Insérer(T,k-1)

Dans le pire cas ou en moyenne, la complexité du tri par sélection est en $O(n^2)$.

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad\qquad$ $3, 7, 12, 16, 25, 38, 40$

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $3, 7, 12, 16, 25, 38, 40$

$3,$

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

$3, 5,$

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

$3, 5, 7,$

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

$3, 5, 7, 10,$

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

$3, 5, 7, 10, 12,$

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

$3, 5, 7, 10, 12, 13,$

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

$3, 5, 7, 10, 12, 13, 15,$

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

$3, 5, 7, 10, 12, 13, 15, 16,$

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

$3, 5, 7, 10, 12, 13, 15, 16, 19,$

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

$3, 5, 7, 10, 12, 13, 15, 16, 19, 20,$

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

$3, 5, 7, 10, 12, 13, 15, 16, 19, 20, 25,$

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

$3, 5, 7, 10, 12, 13, 15, 16, 19, 20, 25, 35,$

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

$3, 5, 7, 10, 12, 13, 15, 16, 19, 20, 25, 35, 38,$

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

$3, 5, 7, 10, 12, 13, 15, 16, 19, 20, 25, 35, 38, 40$

Tri **fusion**

Étant donné un tableau (ou une liste) de $T[1, \ldots, n]$:

- si $n = 1$, retourner le tableau T !
- sinon :
  - Trier le sous-tableau $T[1 \ldots \frac{n}{2}]$
  - Trier le sous-tableau $T[\frac{n}{2} + 1 \ldots n]$
  - Fusionner ces deux sous-tableaux...
- Il s'agit d'un algorithme "diviser-pour-régner".
- $O(n \log n)$ opérations (au pire).

Plan

Le tri **rapide**

Un autre tri récursif... plus efficace en pratique.

Étant donné un tableau de $T[1, \ldots, n]$:

- si $n = 1$, retourner le tableau T.
- sinon :
  - Choisir un élément (le "pivot") p dans T
  - Placer les éléments inférieurs à p au début de T
  - Placer p à sa place dans T
  - Placer les éléments supérieurs à p à la fin de T
  - Trier la première partie de T puis la seconde...

(plus de fusion !)

Le tri **rapide**

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

Le tri **rapide**

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

$\underbrace{15,\ 10,\ 19,\ 13,\ 5,\ 3,\ 12,\ 7,\ 16,}_{\text{à trier !}}\ 20,\ \underbrace{35,\ 40,\ 25,\ 38}_{\text{à trier !}}$

Complexité du tri rapide

Dans le pire cas, la complexité du tri rapide est en $O(n^2)$.

Mais en moyenne, elle est en $O(n \cdot \log(n))$.

Plan

Un tri avec des arbres !

A partir d'une liste d'entiers, on va construire un arbre binaire où chaque noeud contiendra un entier de la liste en respectant la propriété suivante :

Tout noeud x doit contenir un entier...

- supérieur (ou égal) aux entiers de son sous-arbre gauche, et
- inférieur strictement aux entiers de son sous-arbre droit.

$\rightarrow$ un "arbre binaire de recherche".

Un tri avec des arbres !

A partir d'une liste d'entiers, on va construire un arbre binaire où chaque noeud contiendra un entier de la liste en respectant la propriété suivante :

Tout noeud x doit contenir un entier...

- supérieur (ou égal) aux entiers de son sous-arbre gauche, et
- inférieur strictement aux entiers de son sous-arbre droit.

$\rightarrow$ un "arbre binaire de recherche".

Comment faire ?

Un tri avec des arbres : exemple...

$20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38$

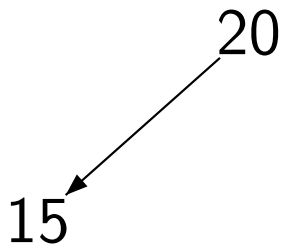
Un tri avec des arbres : exemple...

$20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38$

20

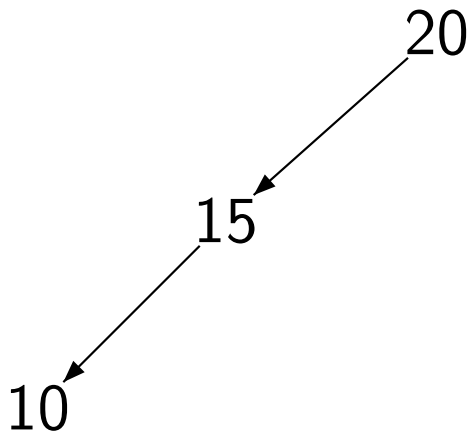
Un tri avec des arbres : exemple...

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38



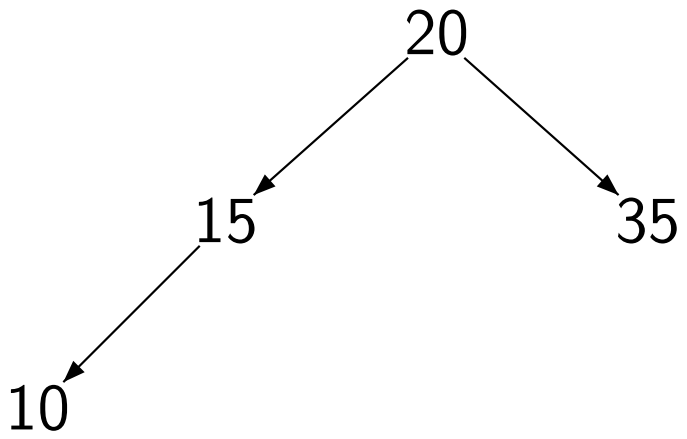
Un tri avec des arbres : exemple...

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38



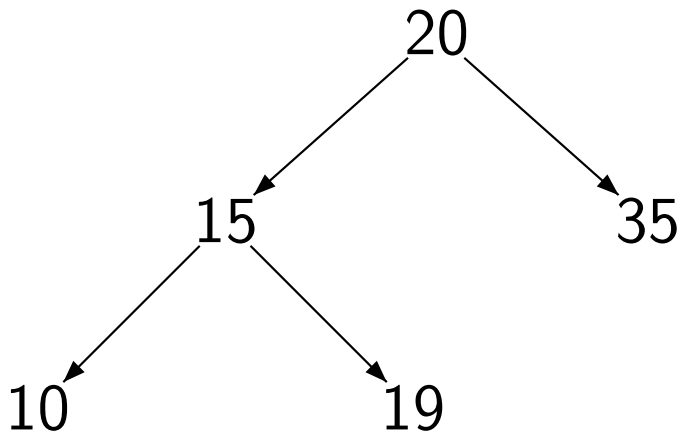
Un tri avec des arbres : exemple...

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38



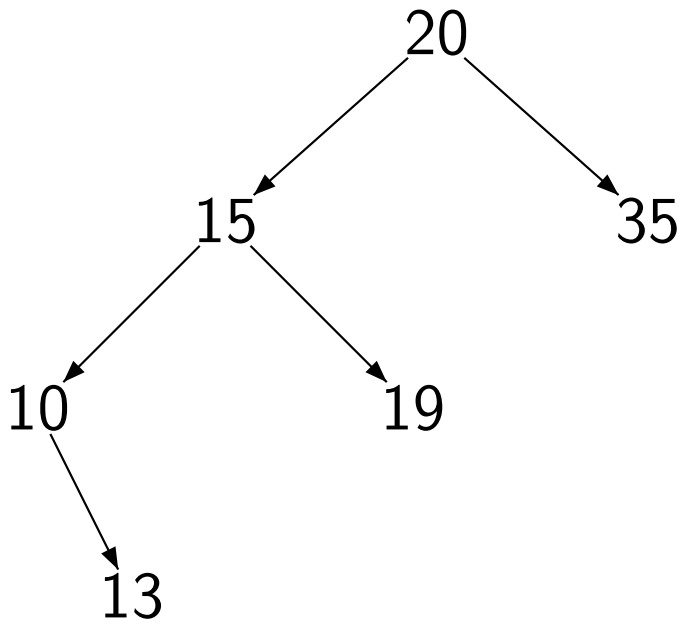
Un tri avec des arbres : exemple...

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38



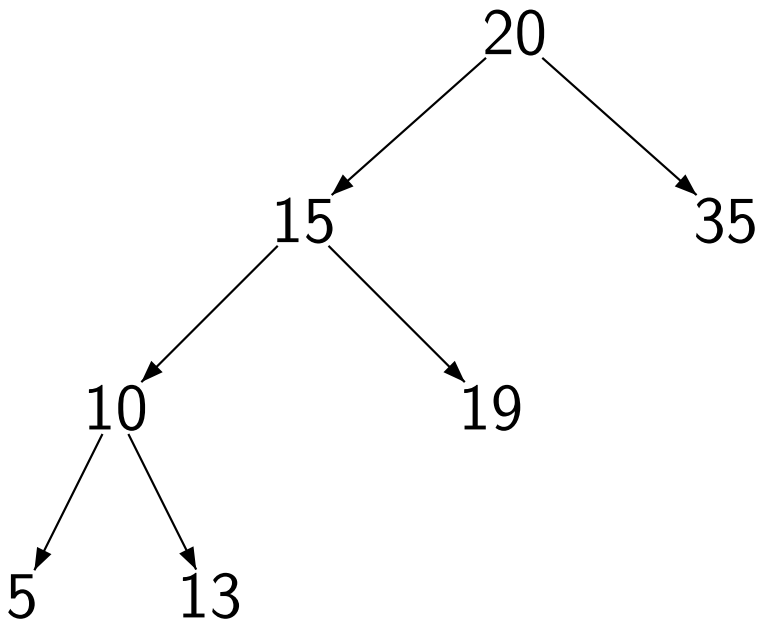
Un tri avec des arbres : exemple...

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38



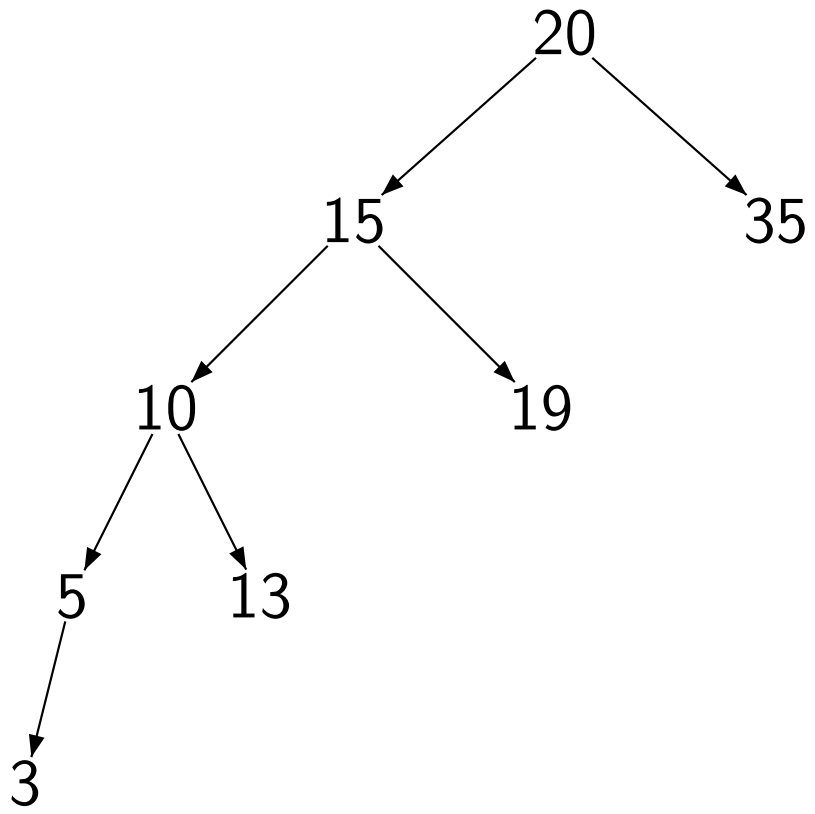
Un tri avec des arbres : exemple...

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38



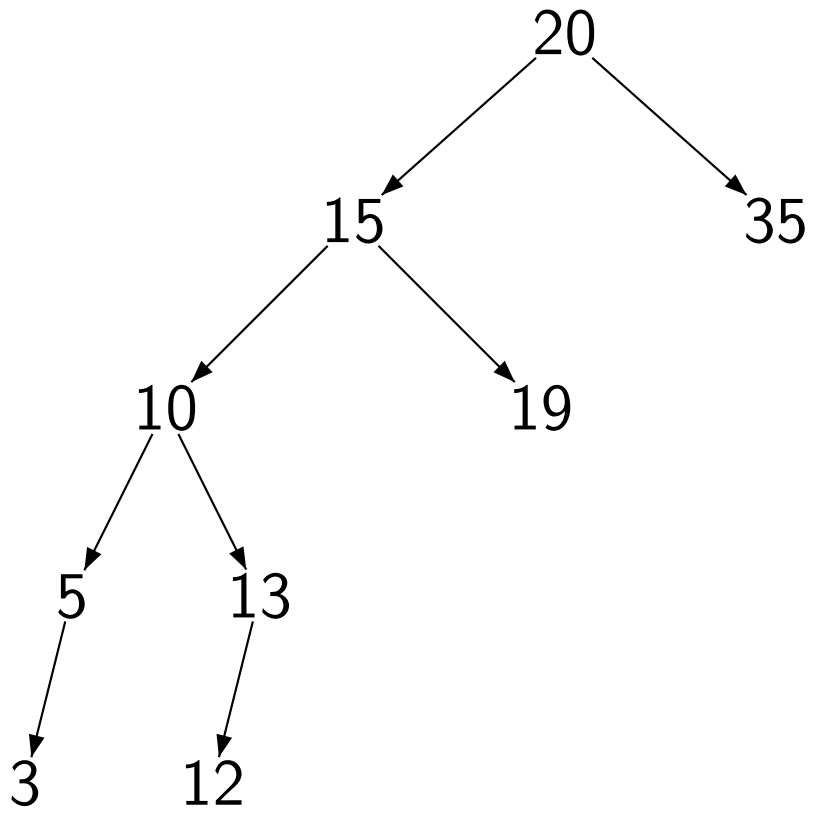
Un tri avec des arbres : exemple...

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38



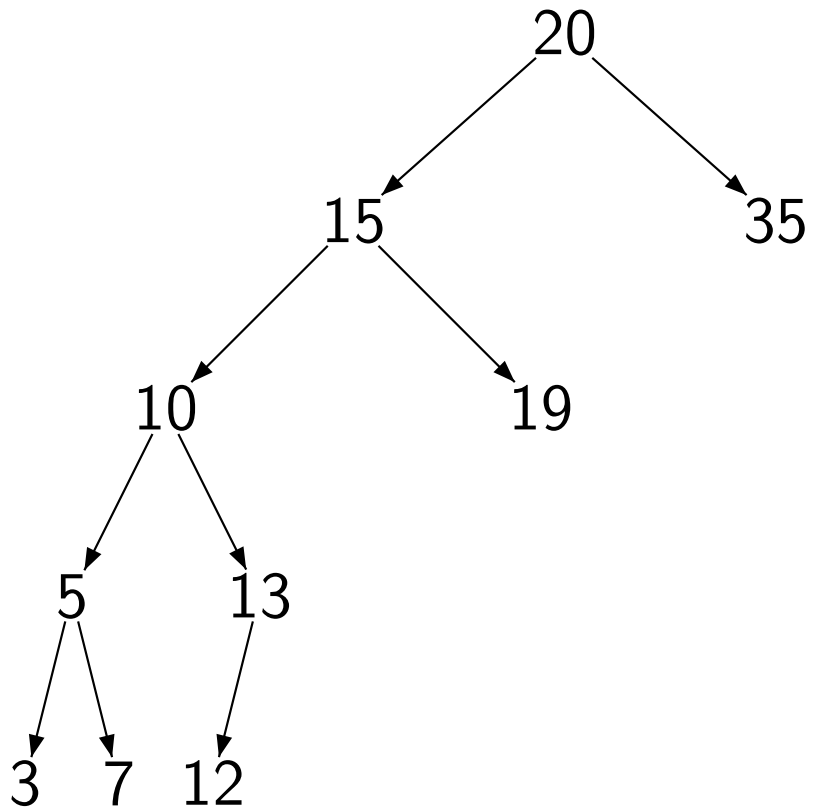
Un tri avec des arbres : exemple...

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38



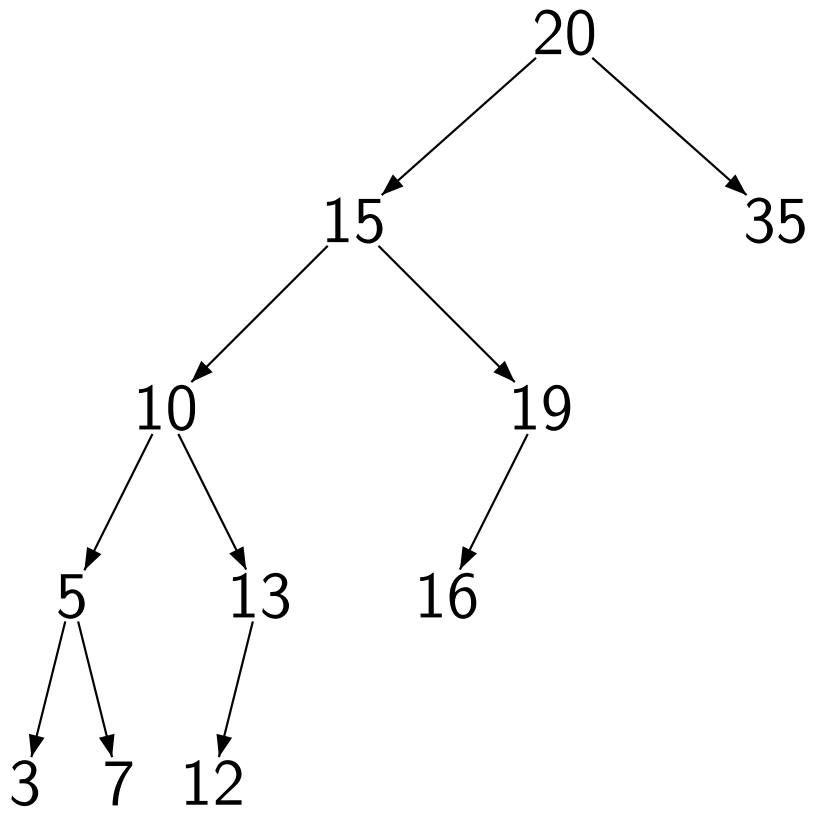
Un tri avec des arbres : exemple...

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38



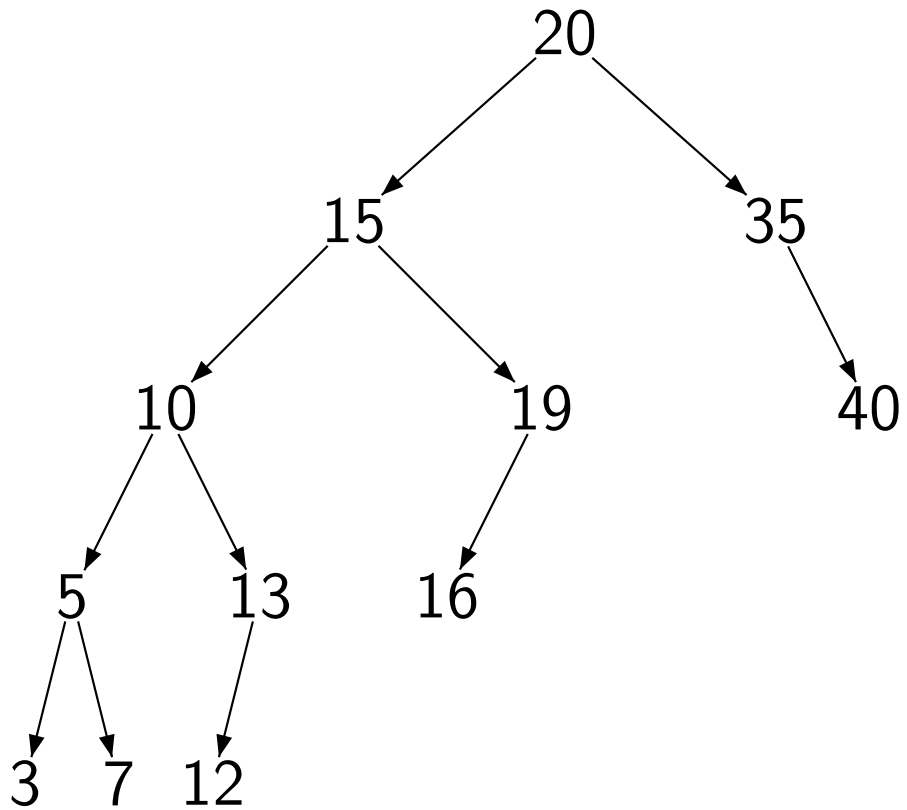
Un tri avec des arbres : exemple...

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38



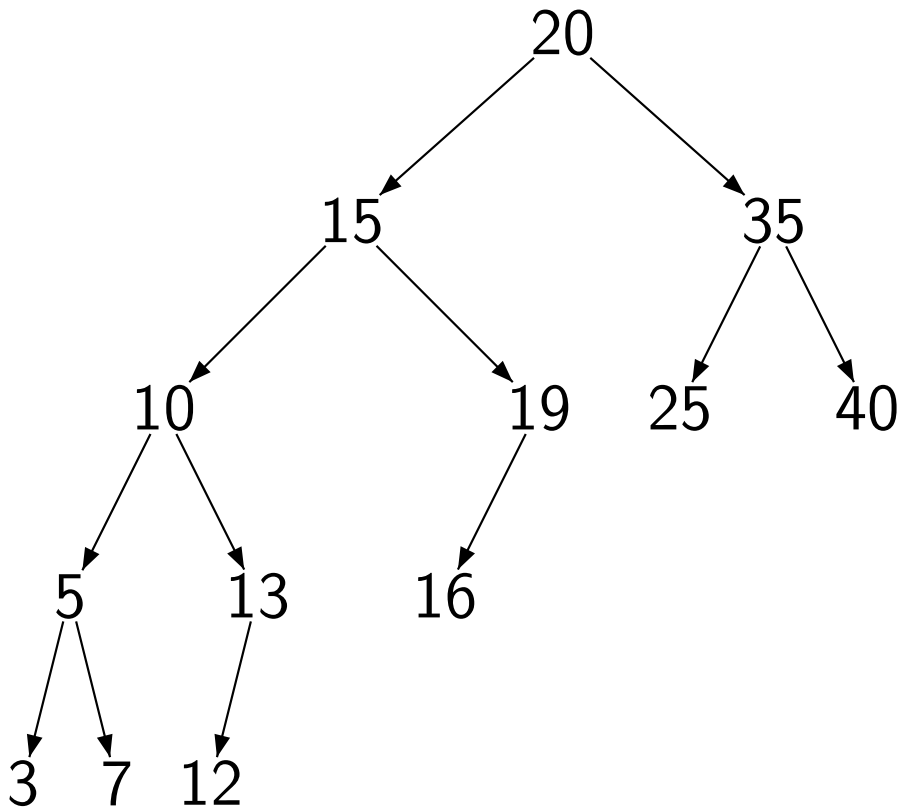
Un tri avec des arbres : exemple...

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38



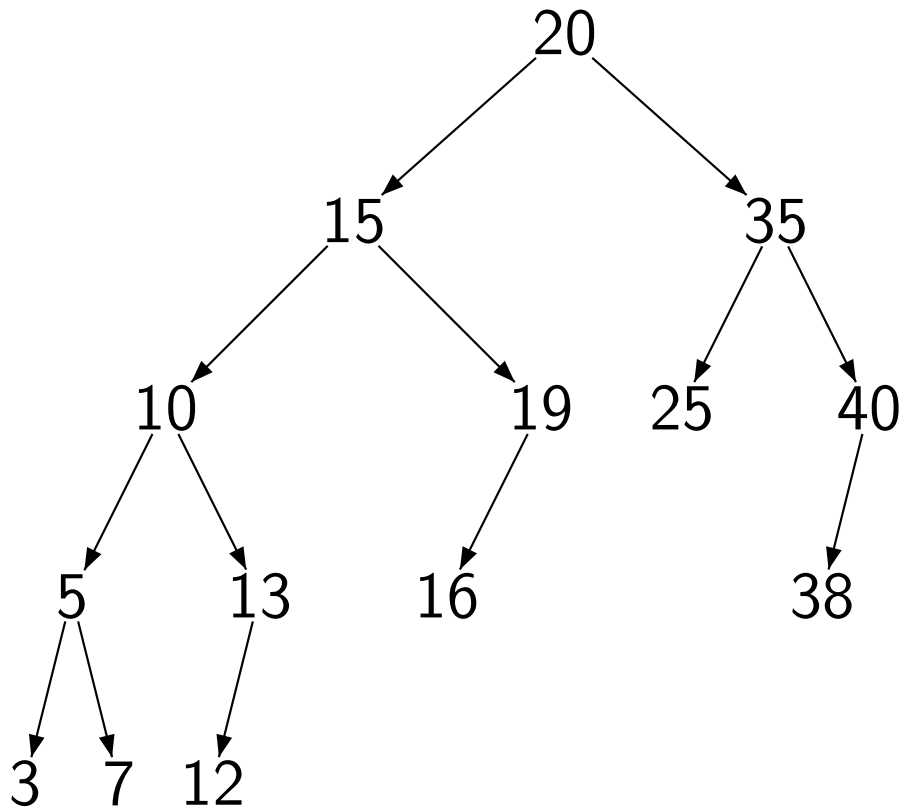
Un tri avec des arbres : exemple...

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38



Un tri avec des arbres : exemple...

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38



Construire l'arbre

Ajouter (entier x, ABR a)

```
begin
    si EstVide(a) alors
        a = Arbre(x, −, −)
    sinon
        si x ≤ valeur(a) alors
            Ajouter(x,G(a))
        sinon
            Ajouter(x,D(a))
end
```

Et ensuite...

Il reste à parcourir l'arbre construit et à afficher la valeur d'un noeud lorsqu'on le visite pour la deuxième fois (parcours infixe)...

Parcours (noeud a)

```
begin
    si ¬EstVide(a) alors
        [premier passage]
        Parcours(G(a))
        [second passage]
        Parcours(D(a))
        [troisième passage]
end
```

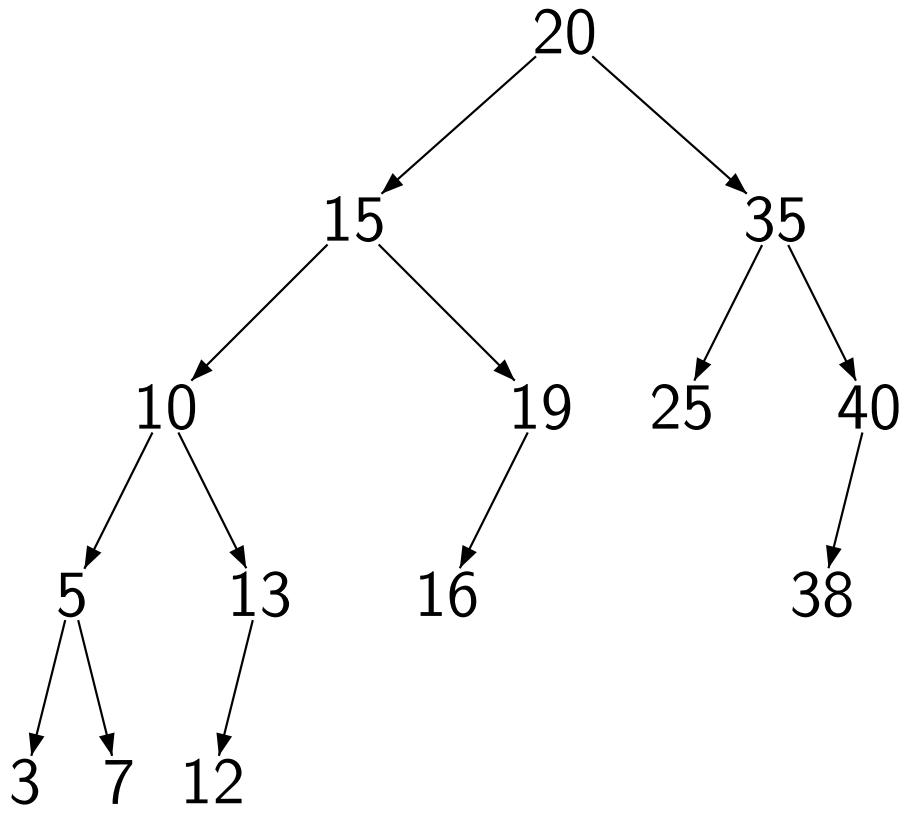
Et ensuite...

Il reste à parcourir l'arbre construit et à afficher la valeur d'un noeud lorsqu'on le visite pour la deuxième fois (parcours infixe)...

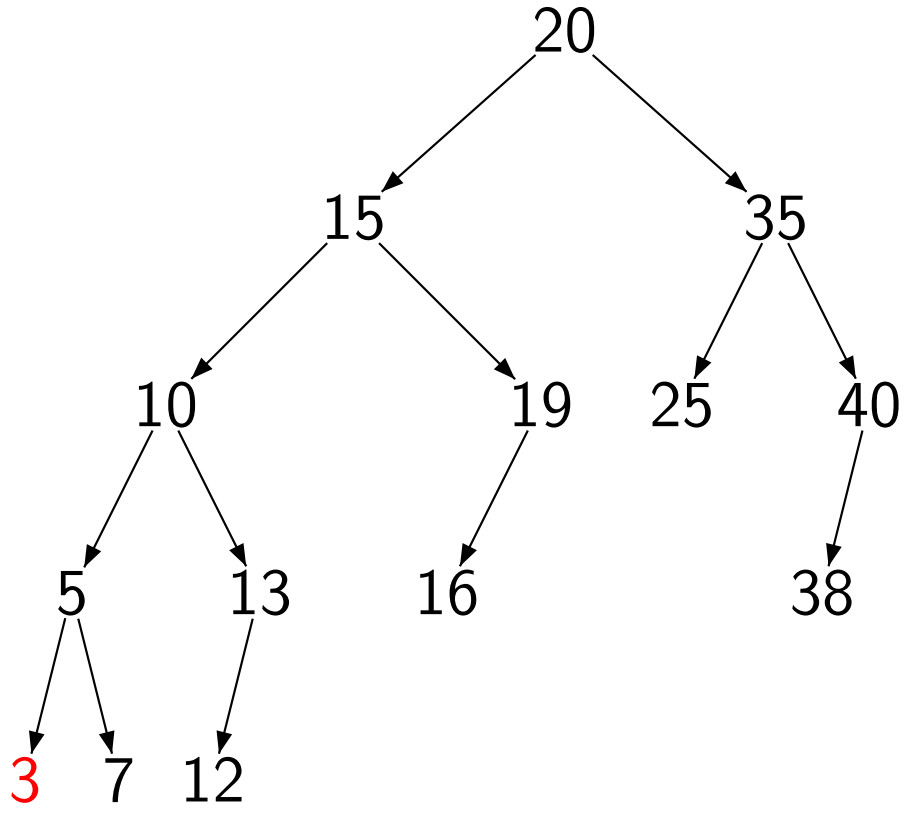
Parcours (noeud a)

```
begin
    si ¬EstVide(a) alors
        [premier passage]
        Parcours(G(a))
        [second passage] Afficher valeur(a)
        Parcours(D(a))
        [troisième passage]
end
```

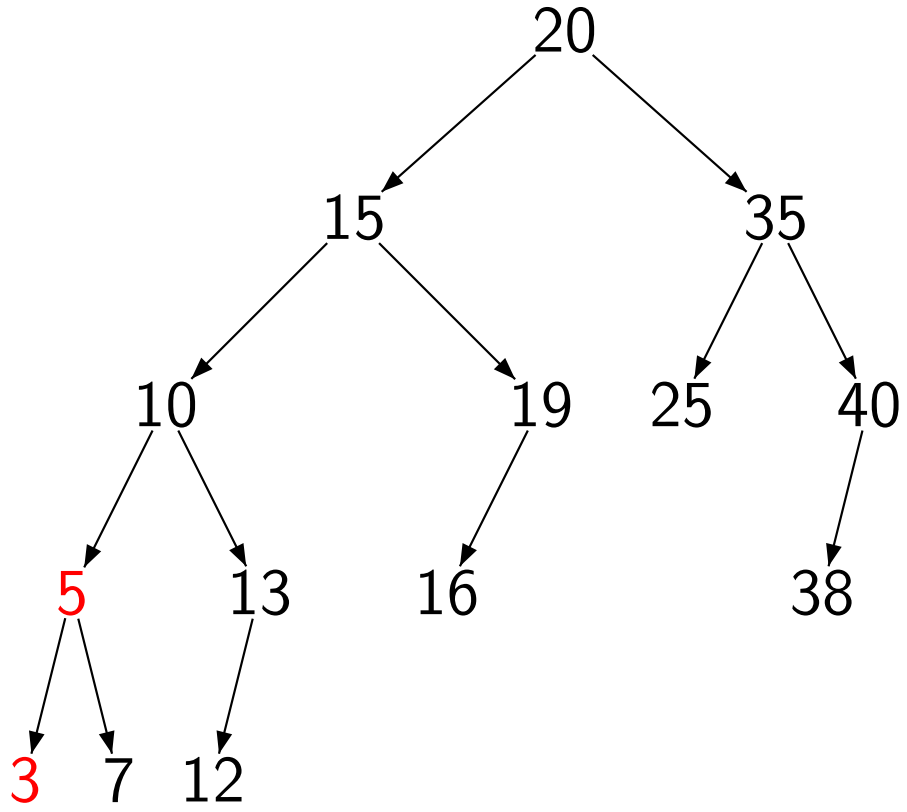
Exemple...



Exemple. . .

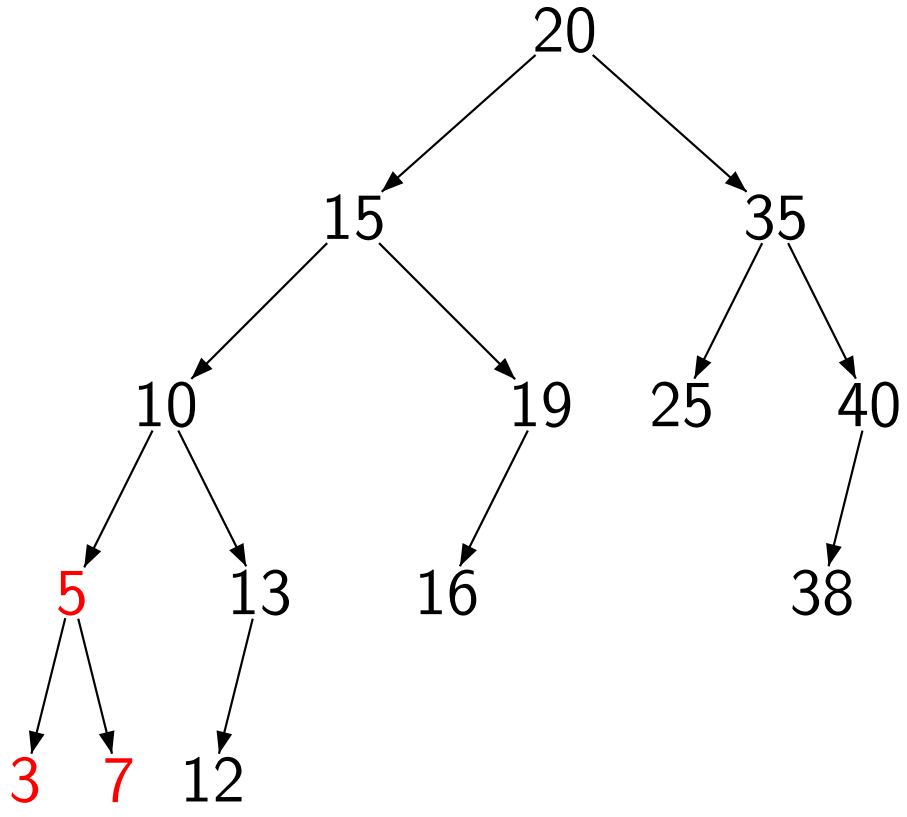


Exemple. . .



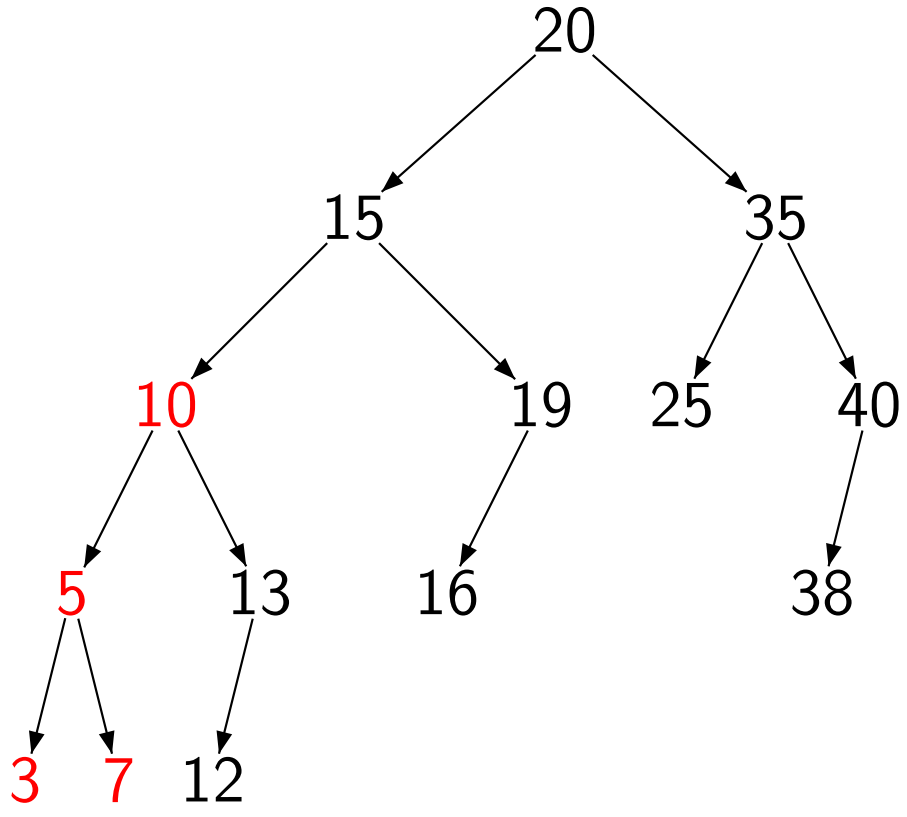
3 5

Exemple. . .



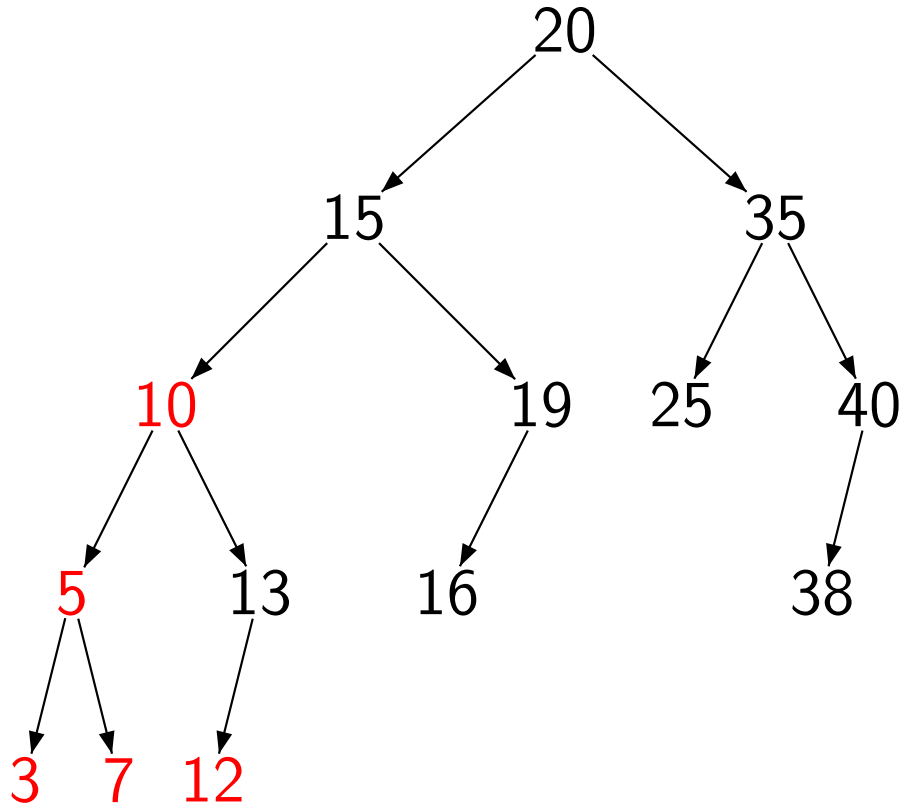
3 5 7

Exemple. . .



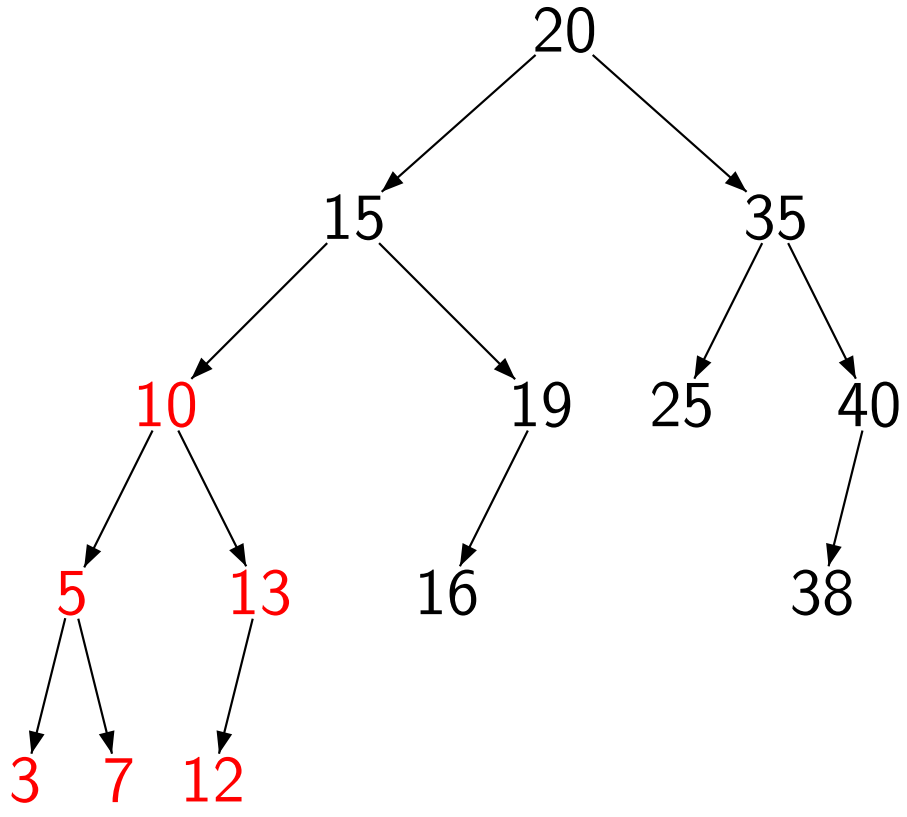
3 5 7 10

Exemple. . .



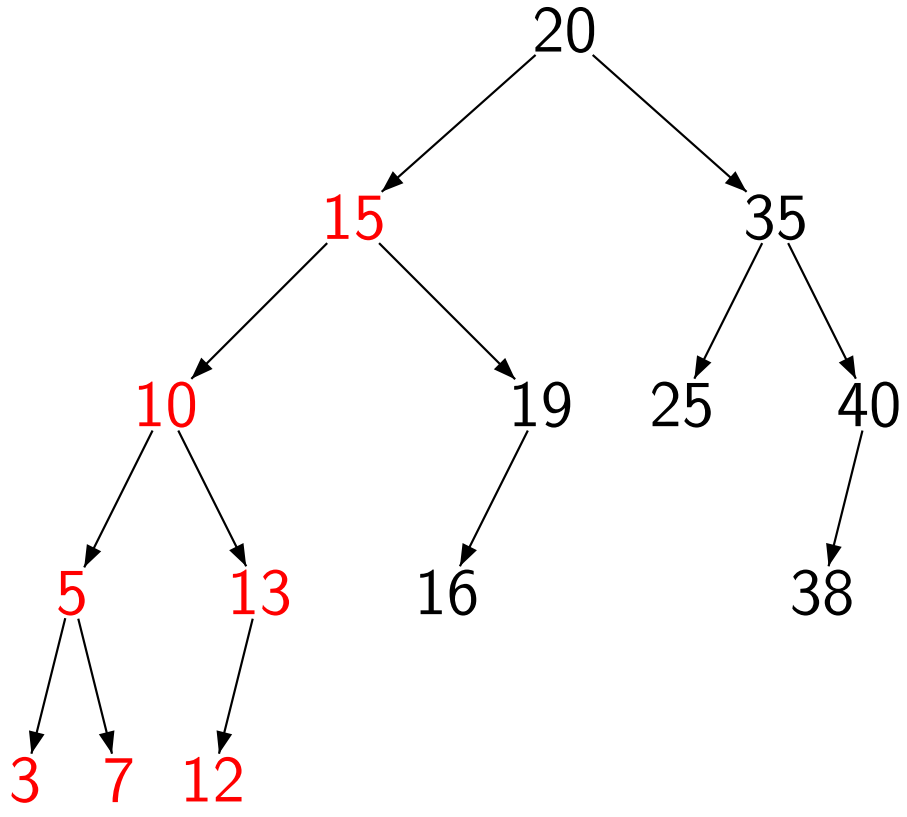
3 5 7 10 12

Exemple...



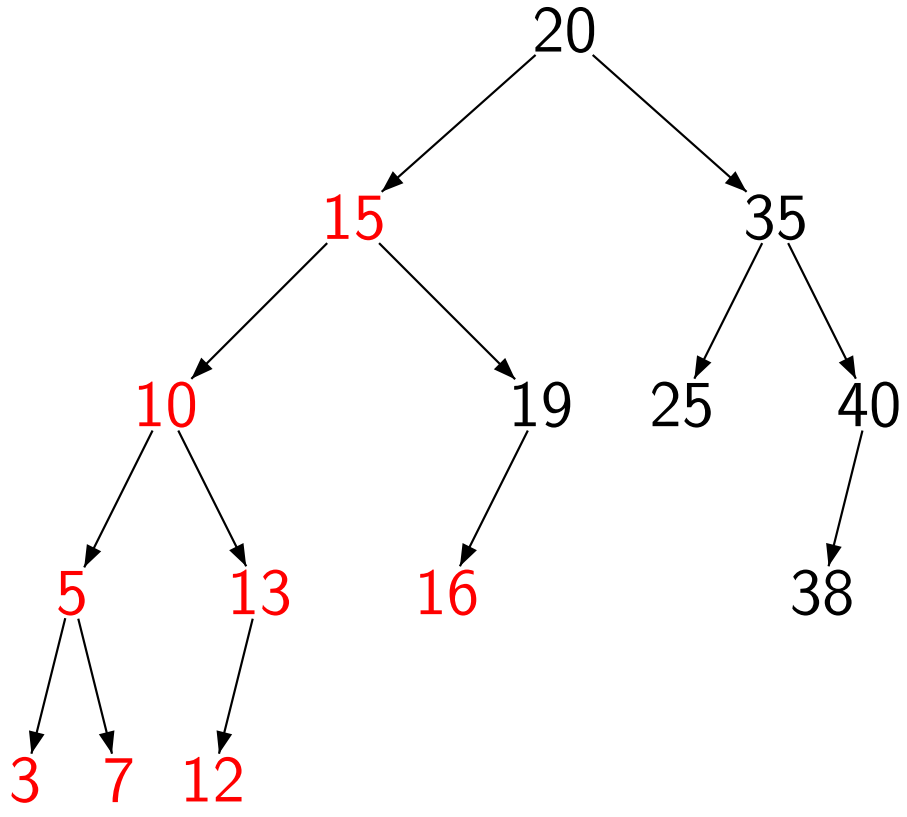
3 5 7 10 12 13

Exemple...



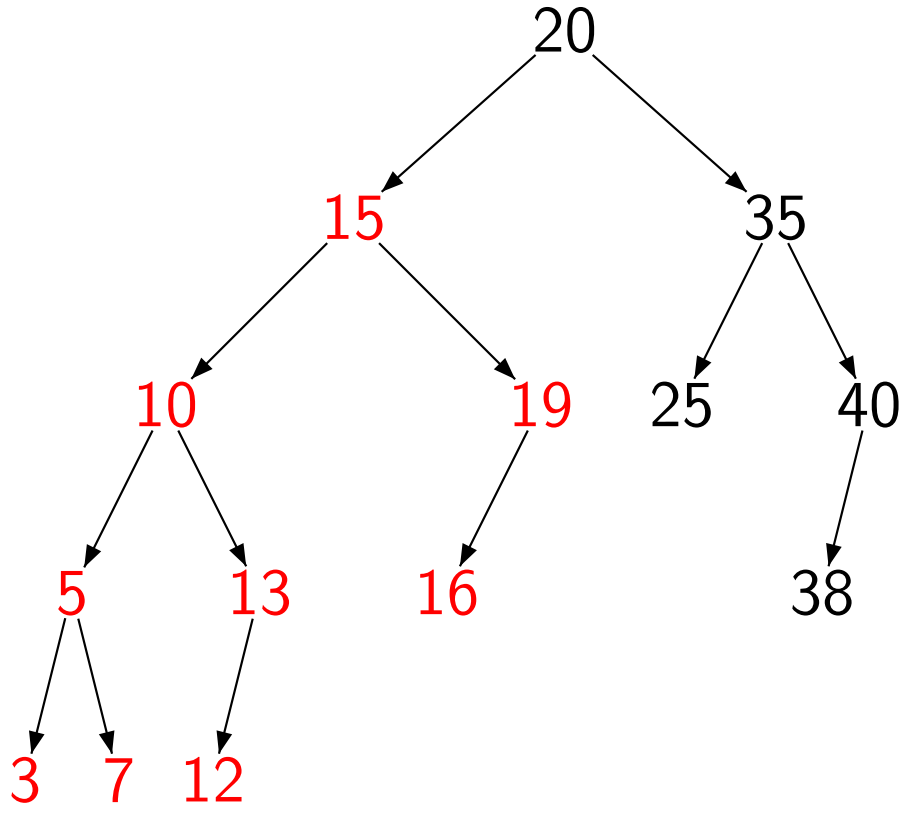
3 5 7 10 12 13 15

Exemple...



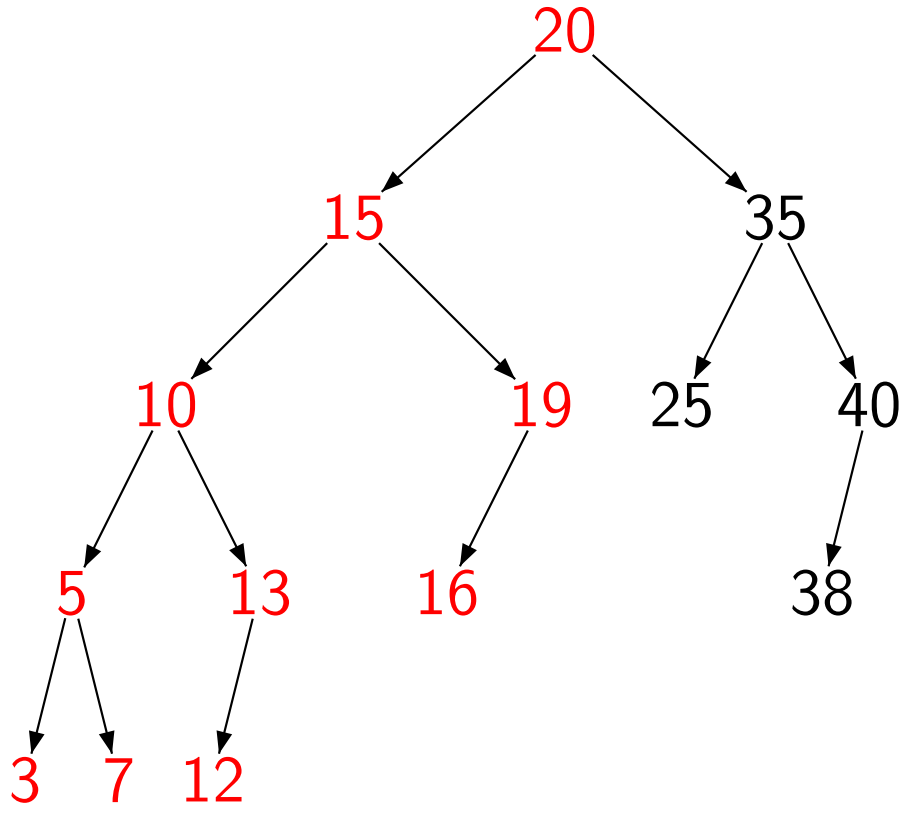
3 5 7 10 12 13 15 16

Exemple...



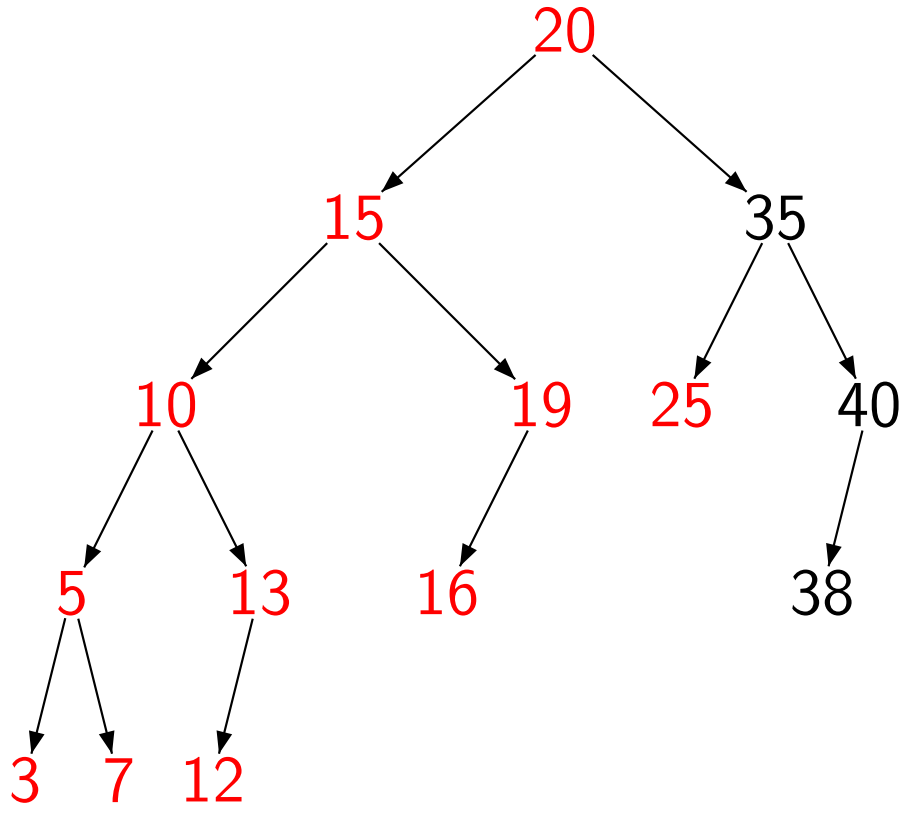
3 5 7 10 12 13 15 16 19

Exemple...



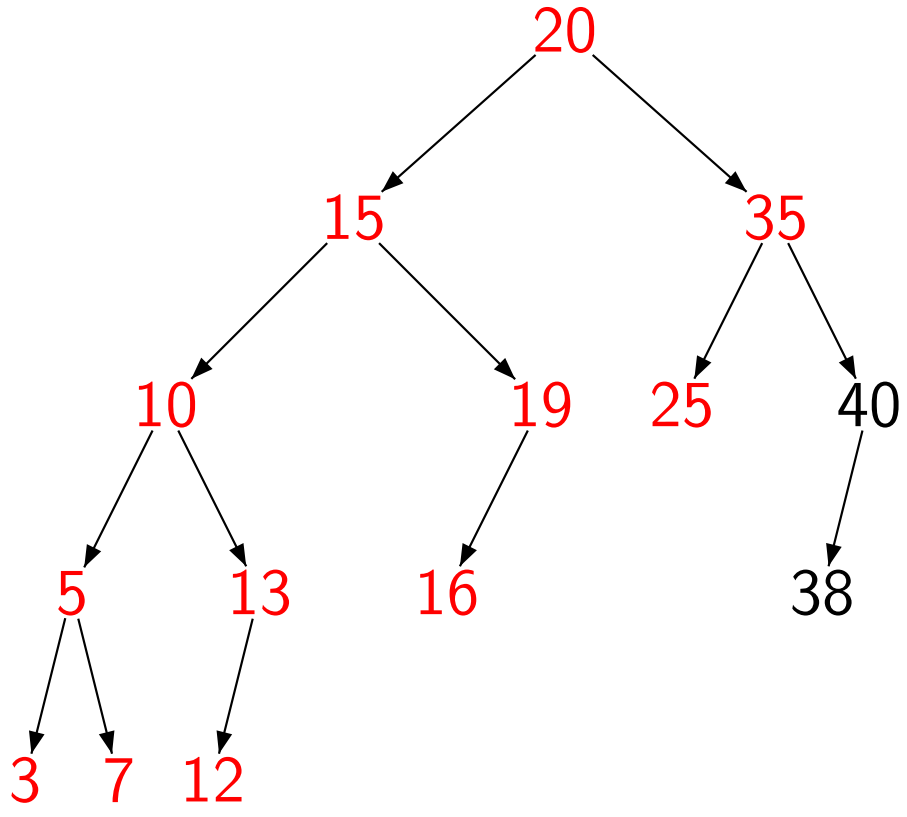
3 5 7 10 12 13 15 16 19 20

Exemple...



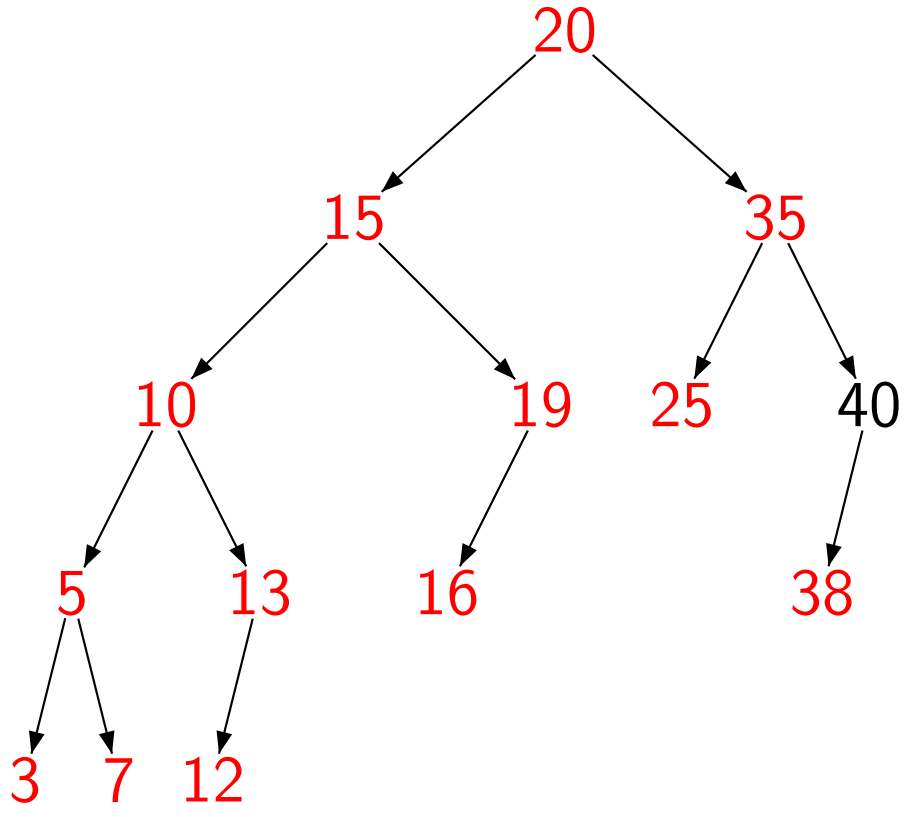
3 5 7 10 12 13 15 16 19 20 25

Exemple...



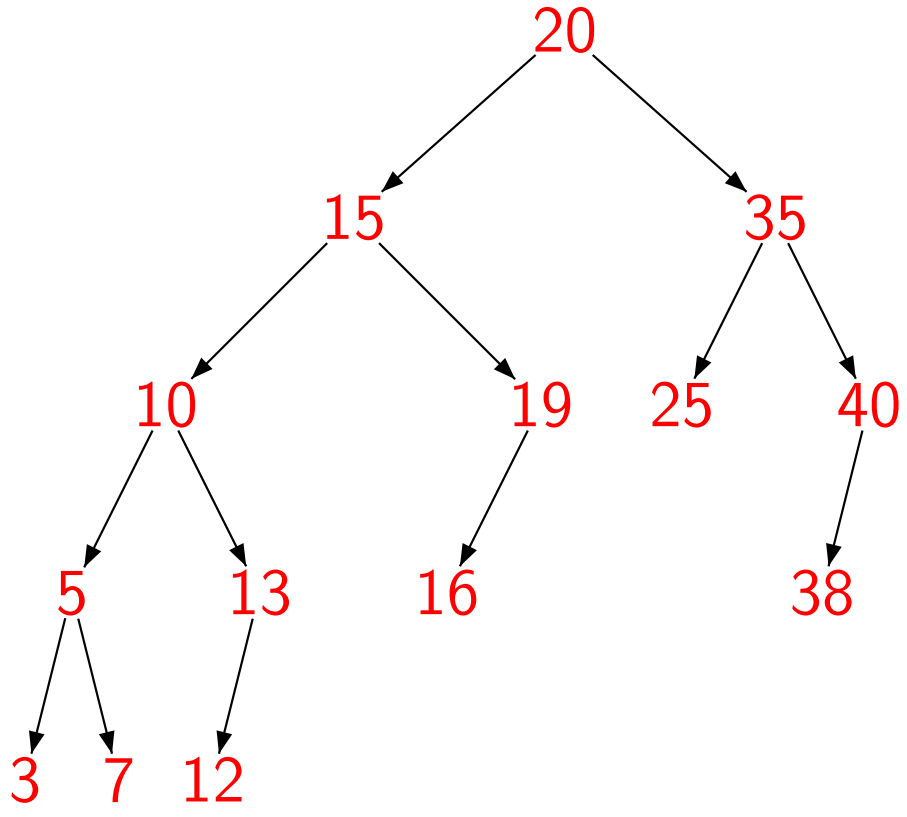
3 5 7 10 12 13 15 16 19 20 25 35

Exemple...



3 5 7 10 12 13 15 16 19 20 25 35 38

Exemple...



3 5 7 10 12 13 15 16 19 20 25 35 38 40

Complexité du tri par ABR

Dans le pire cas, la complexité est en $O(n^2)$.

En moyenne, la complexité est en $O(n \cdot \log(n))$:

Le nombre moyen de comparaisons de clés effectuées pour construire un ABR en insérant n clés distinctes dans un ordre aléatoire à partir d'un ABR vide est :

$$2(n+1)(H_{n+1} - 1) - 2n$$

Ce nombre est donc en $O(n \cdot \log(n))$.

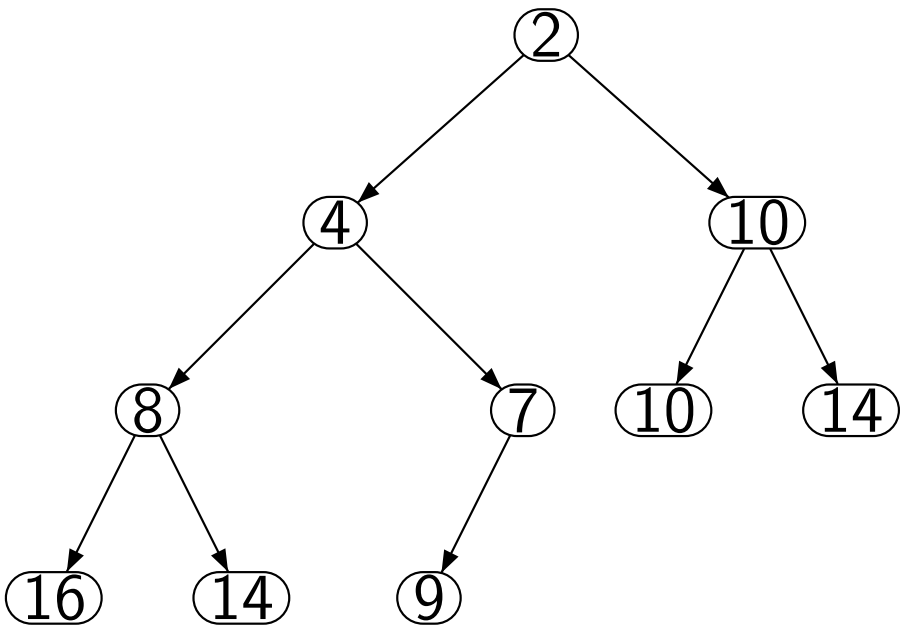
Plan

Tri par tas

Un tas est un arbre binaire...

- parfait (tous les niveaux sont remplis sauf éventuellement le dernier et les feuilles sont regroupées à gauche), et
- chaque noeud contient une valeur inférieure (ou égale) à celles stockées dans ses sous-arbres.

Exemple :



Représentation des tas

Un tas se représente facilement avec une paire (T, n) :

- un entier n donnant le nombre d'éléments du tas, et
- un tableau T (de taille $\geq n$).

$T[1]$ correspond à la racine du tas.
Tout noeud i a son père en $\frac{i}{2}$, et :

- son fils gauche en $2 \cdot i$ (si il existe, c.-à-d.$2i \leq n$), et
- son fils droit en $2 \cdot i + 1$ (si $2i + 1 \leq n$).

Algorithmes sur les tas

On dispose d'algorithmes efficaces pour :

- ajouter un élément au tas,
- extraire le minimum (et remettre l'arbre en tas), et

Efficace = linéaire dans la hauteur de l'arbre, donc en $O(\log(n))$.

Algorithmes sur les tas

On dispose d'algorithmes efficaces pour :

- ajouter un élément au tas,
- extraire le minimum (et remettre l'arbre en tas), et

Efficace = linéaire dans la hauteur de l'arbre, donc en $O(\log(n))$.

Le tri par tas consiste à :

- transformer T en un tas, $O(n)$
- puis extraire les éléments un à un... $O(n \log n)$

Opérations sur les tas : ajouter

Ajouter(T, n, x)

$n \leftarrow n + 1$

$i \leftarrow n$

tant que $i/2 > 0$ && $T[i/2] > x$ **faire**

  $T[i] \leftarrow T[i/2]$

  $i \leftarrow i/2$

$T[i] \leftarrow x$

Opérations sur les tas : extraire-min

1. retourner $T[1]$, $T[1] \leftarrow T[n]$, $n \leftarrow n-1$,
2. puis appeler Tasser$(T, n, 1)$

Tasser(T, n, i)

$g \leftarrow 2 \cdot i$,
$d \leftarrow 2 \cdot i + 1$
$win \leftarrow i$
si $g \leq n$ && $T[g] < T[i]$ **alors** $win \leftarrow g$
si $d \leq n$ && $T[d] < T[win]$ **alors** $win \leftarrow d$
si $win \neq i$ **alors**
  $T[i] \leftrightarrow T[win]$
  Tasser(T, n, win)

(NB : $T[2i]$ et $T[2i+1]$ sont des racines de tas)

ABR vs Tas

Les ABR et les tas (ou Files de priorité) servent à stocker des éléments en fonction d'une clé.

Pour les ABR, les opérations suivantes sont facilement implémentables :

- Ajouter, Supprimer et Rechercher un élément,
- Parcourir dans l'ordre.

Leur complexité sont en $O(h)$.

Pour les tas, on peut faire :

- Ajouter un élément,
- Extraire le plus petit élément.

Leur complexité sont en $O(\log(n))$.

Plan

Optimalité des algorithmes de tri

Question : Est-il possible de trier un tableau de n éléments en moins de $n \cdot \log(n)$ opérations dans le pire cas ?

Optimalité des algorithmes de tri

Question : Est-il possible de trier un tableau de n éléments en moins de $n \cdot \log(n)$ opérations dans le pire cas ?

Non si on n'utilise que des comparaisons 2 à 2 et sans hypothèses sur le contenu du tableau...

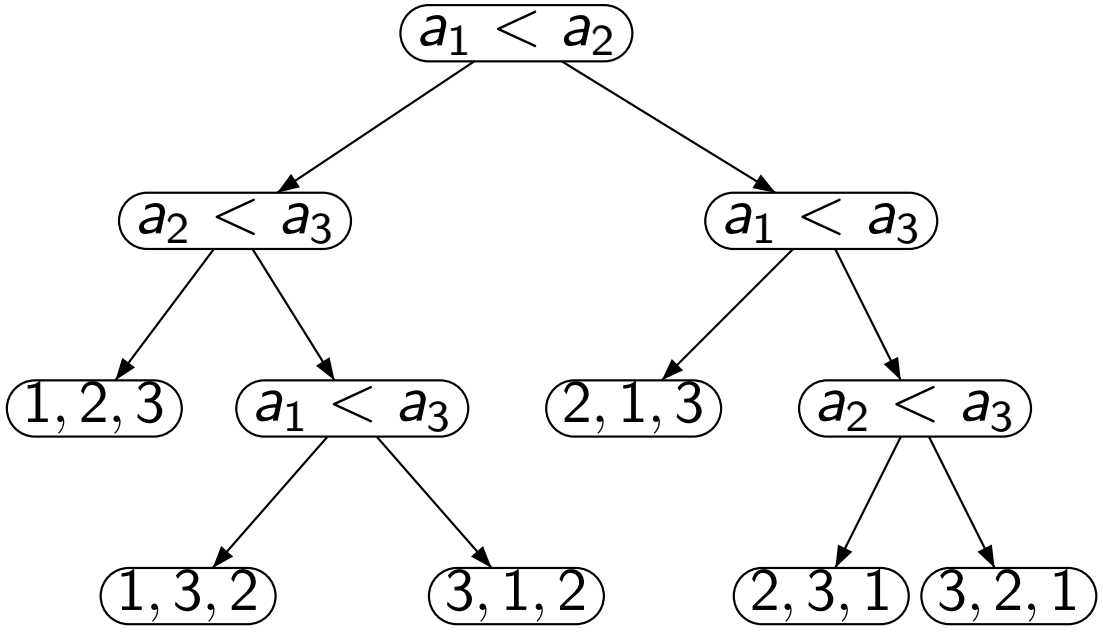
Optimalité des algorithmes de tri

Question : Est-il possible de trier un tableau de n éléments en moins de $n \cdot \log(n)$ opérations dans le pire cas ?

Non si on n'utilise que des comparaisons 2 à 2 et sans hypothèses sur le contenu du tableau...

Dans ce cadre, tout algorithme de tri peut être représenté par un arbre de décision où chaque noeud correspond à un test de deux éléments (le fils gauche correspond à la réponse négative et le droit la réponse positive).

Optimalité des algorithmes de tri



Chaque feuille correspond à la permutation à effectuer par l'algorithme. Il y a donc $n!$ feuilles dans tout arbre de décision pour trier n éléments.

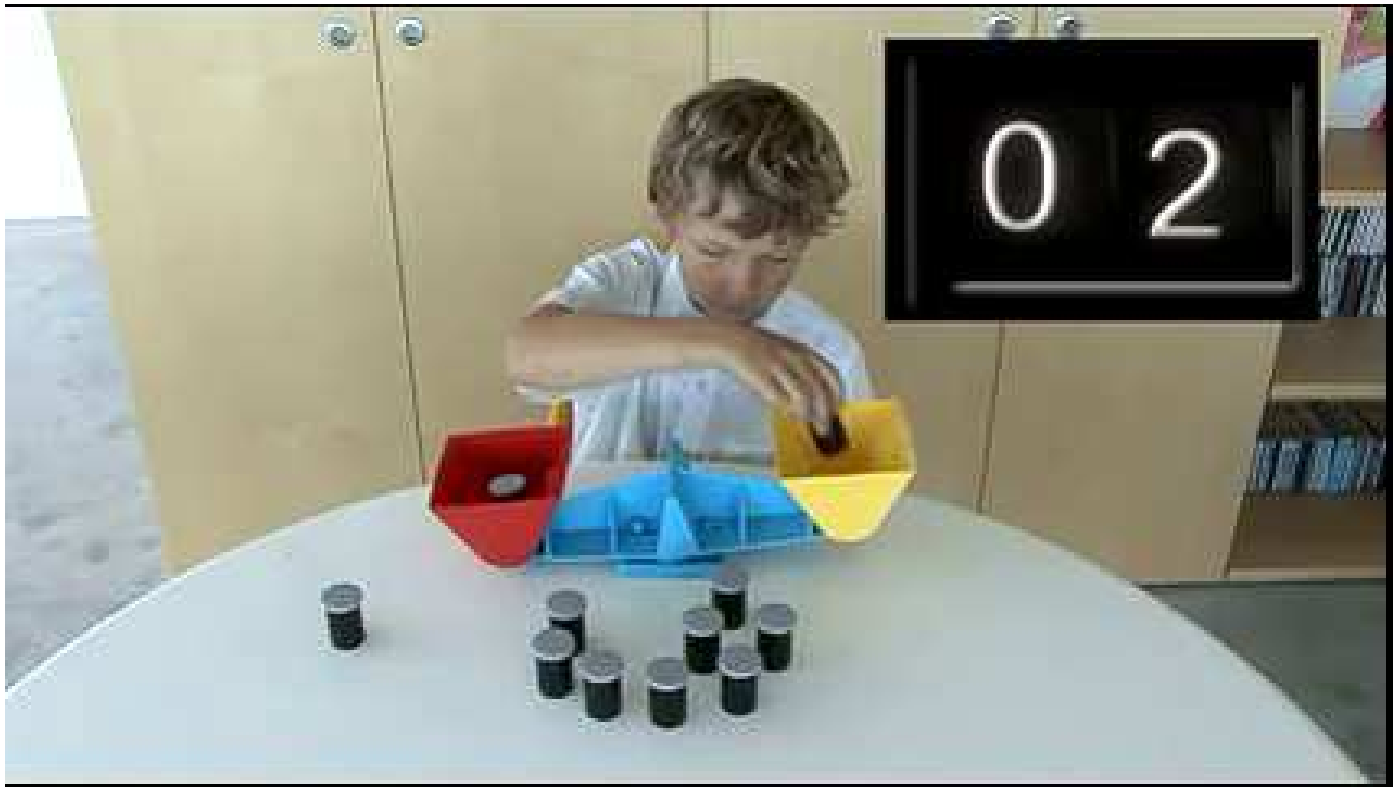
La complexité dans le pire correspond à la hauteur de l'arbre.

Tout arbre binaire équilibré a une hauteur log(nb feuilles).

Le tri fusion (et le tri par tas) sont optimaux (asymptotiquement).

Plan

Une idée d'activité en classe



- Activité de découverte des algorithmes de tri (conçue pour le primaire)
- Description en Français téléchargeable gratuitement sur http://www.csunplugged.org

Plan

Les graphes

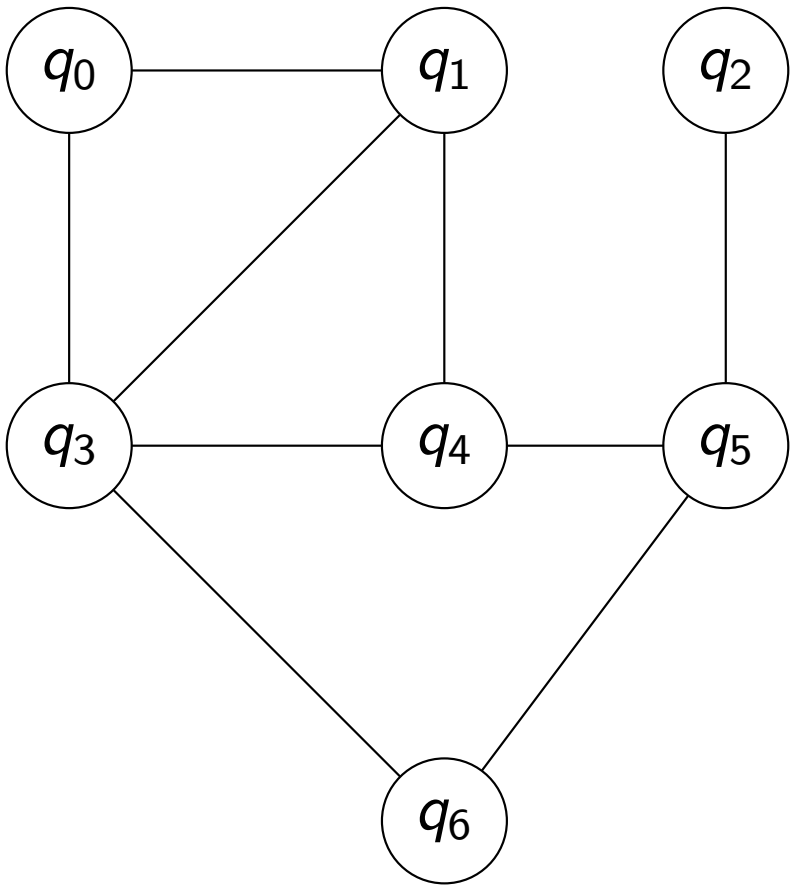
$G = (S, A)$
S est un ensemble fini de sommets.
On va considérer des graphes orientés ou non-orientés :

Les graphes

$G = (S, A)$

S est un ensemble fini de sommets.

On va considérer des graphes orientés ou non-orientés :

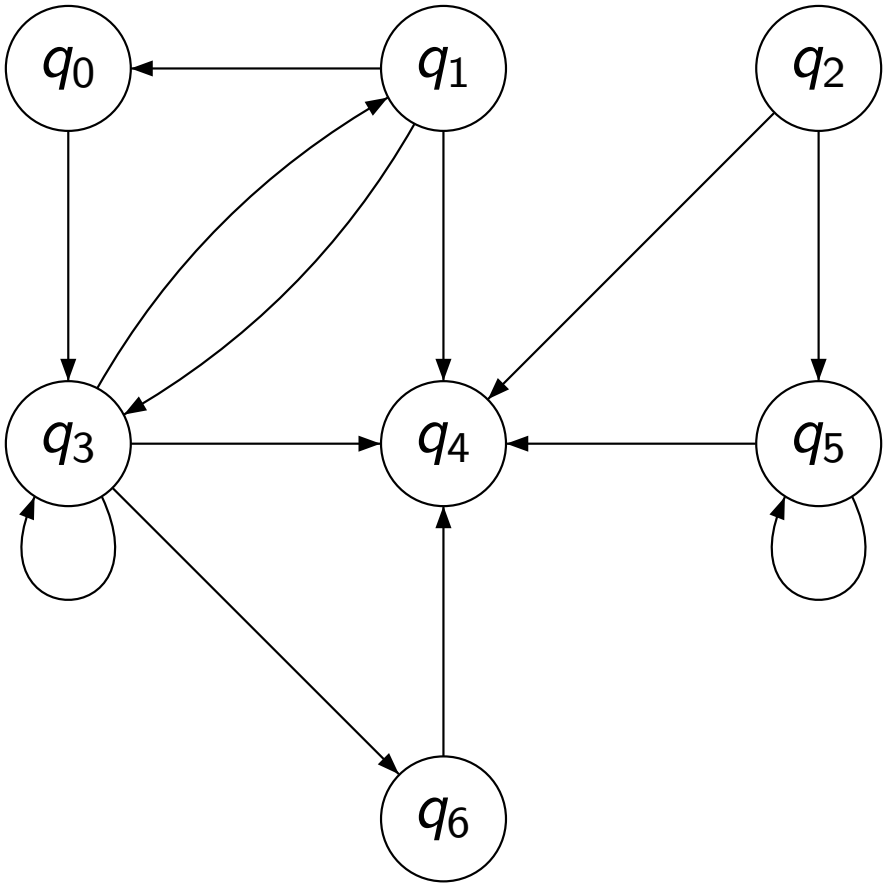


Les graphes

$G = (S, A)$

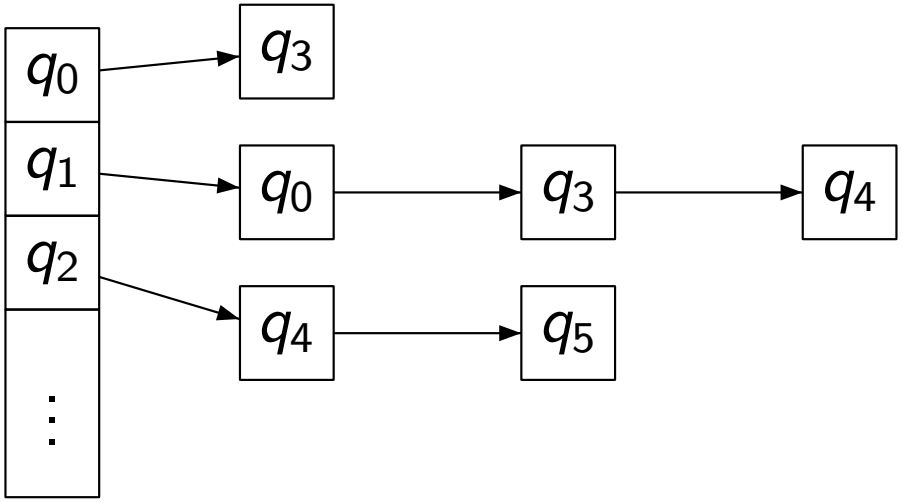
S est un ensemble fini de sommets.

On va considérer des graphes orientés ou non-orientés :



Représentation des graphes - 1

Par liste d'adjacence :



Représentation des graphes - 1

Par matrice d'adjacence :

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Graphes valués

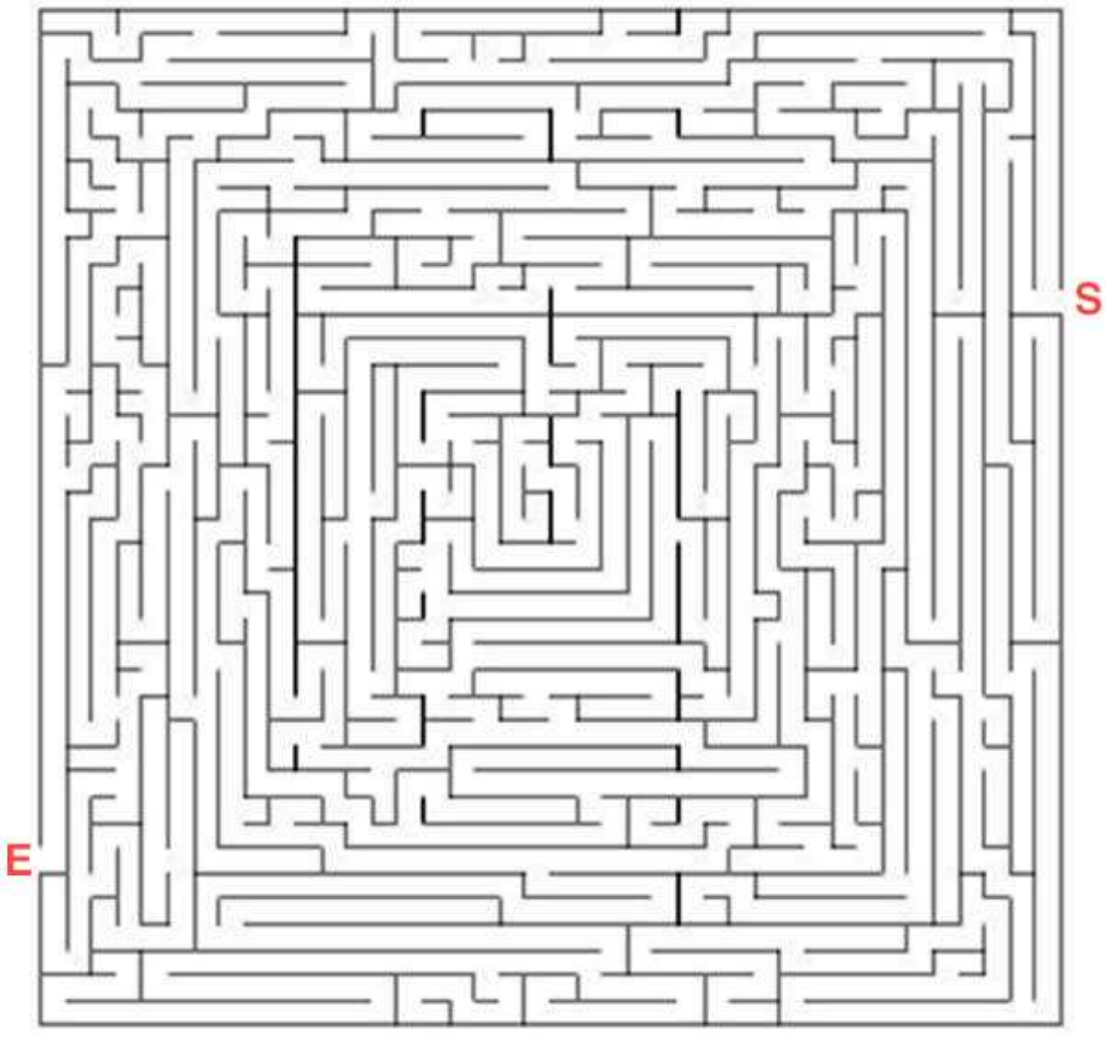
$G = (S, A)$ + $w : A \to \mathbb{R}$.

La fonction w associe un poids, une distance, etc. à chaque arc ou arête de G.
(La fonction w s'étend naturellement à tout chemin (fini) de G en sommant le poids de chaque arc/arête.)

Les deux représentations précédentes s'étendent facilement aux graphes valués.

Plan

Comment sortir ?



Parcours

Initialisation : Couleur$[u] \leftarrow$ blanc et $\Pi[u] \leftarrow$ nil $\forall u \in S$.
Puis appel de PP-Visiter(G, e) :

PP-Visiter(G, q)

```
begin
    si q =Sortie alors
        retourner Π ;
    Couleur[q] ← noir;
    pour chaque (q, u) ∈ A faire
        si Couleur[u] = blanc alors
            Π[u] ← q;
            PP-Visiter(G, u);
end
```

(variante simplifiée du parcours en profondeur)

Parcours en profondeur

Algorithme important !
A la base des algorithmes de recherche des composantes fortement connexes (par ex. algorithme de Tarjan) et de tri topologique.

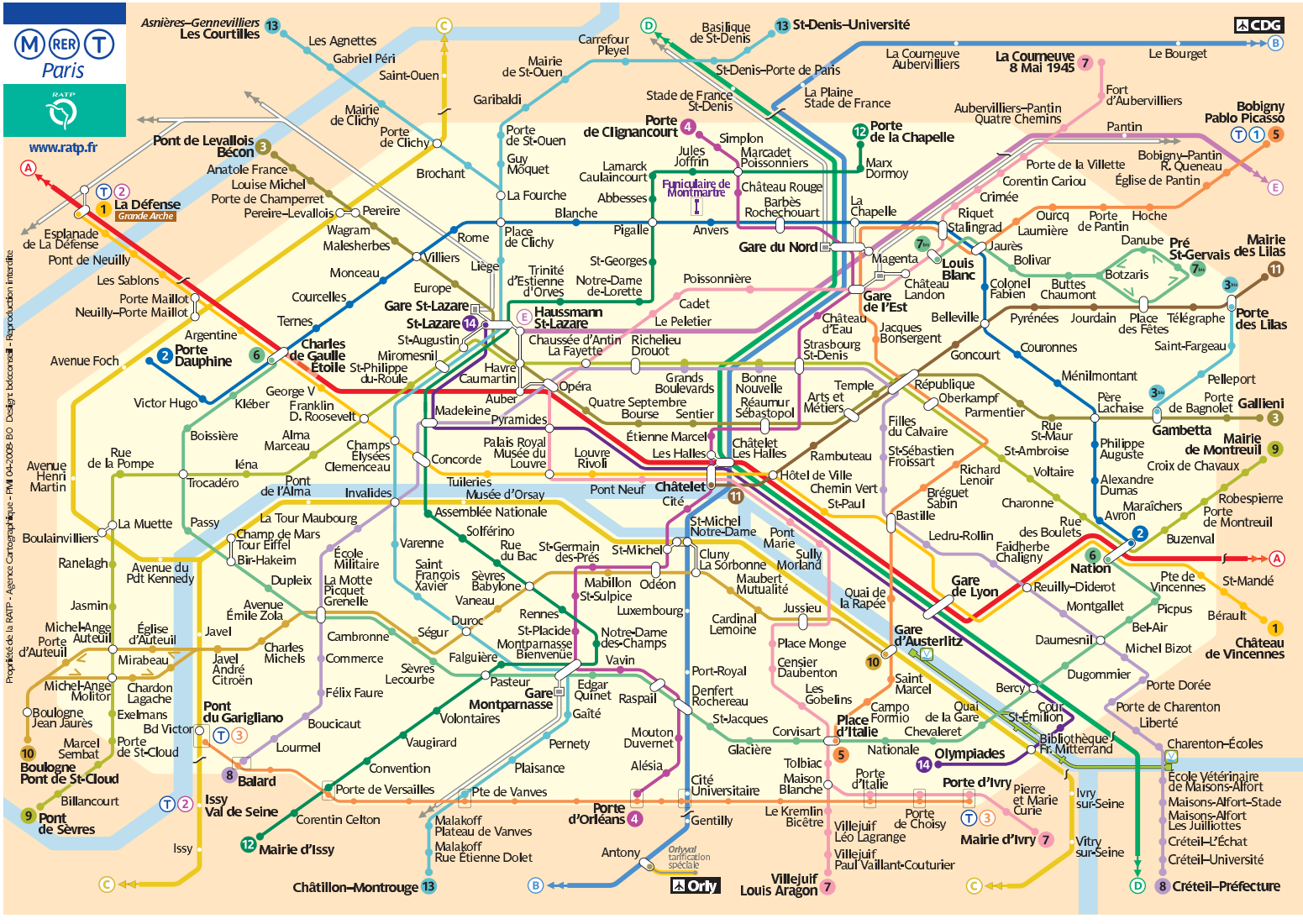
Algorithme de type “backtracking”.

Un autre parcours classique est le parcours en largeur.

Plan

Trouver son chemin

Comment aller de Chevaleret à Porte de Bagnolet ?



Trouver son chemin

Comment aller de Paris à Morlaix ?

- en minimisant la distance ?
- en minimisant la durée ?
- en minimisant le coût ?
- ...

Calculs de plus courts chemins dans un graphe !

Plus courts chemins

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

Pour $\rho \stackrel{\text{def}}{=} v_0 \to v_1 \to \ldots \to v_k$, on note $w(\rho)$ sa longueur ou sa distance :

$$w(\rho) \stackrel{\text{def}}{=} \sum_{i=1,\ldots,k} w(v_{i-1}, v_i)$$

Plus courts chemins

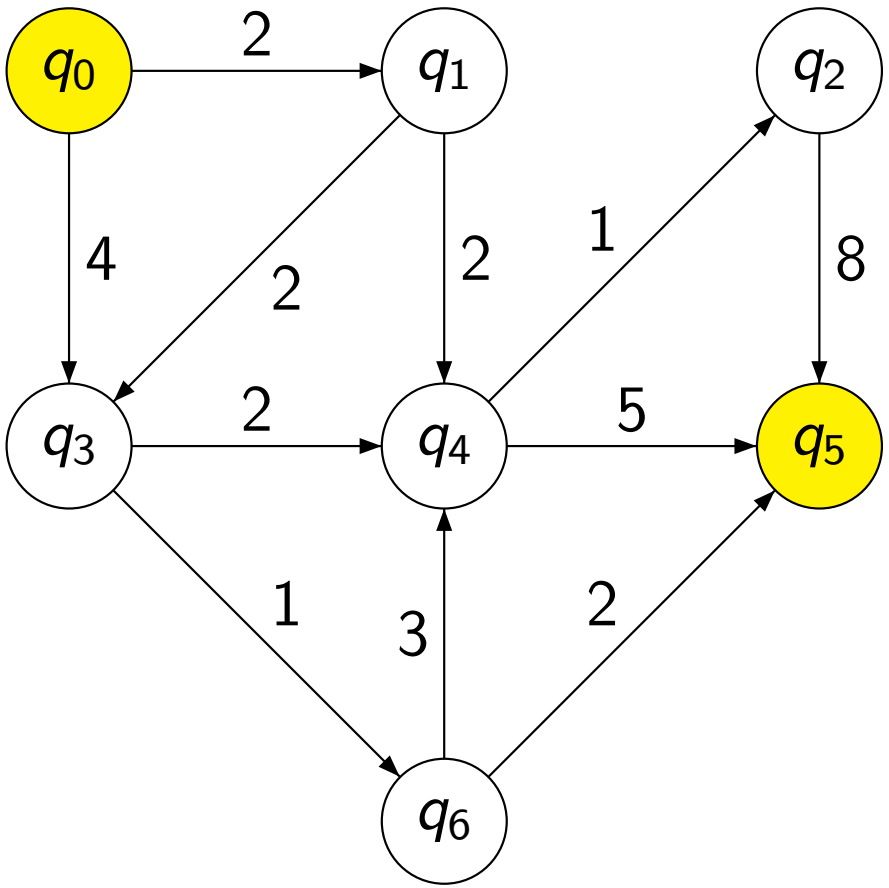
$G = (S, A, w)$: orienté et valué ($w : A \to \mathbb{R}$).

Pour $\rho \stackrel{\text{def}}{=} v_0 \to v_1 \to \ldots \to v_k$, on note $w(\rho)$ sa longueur ou sa distance :

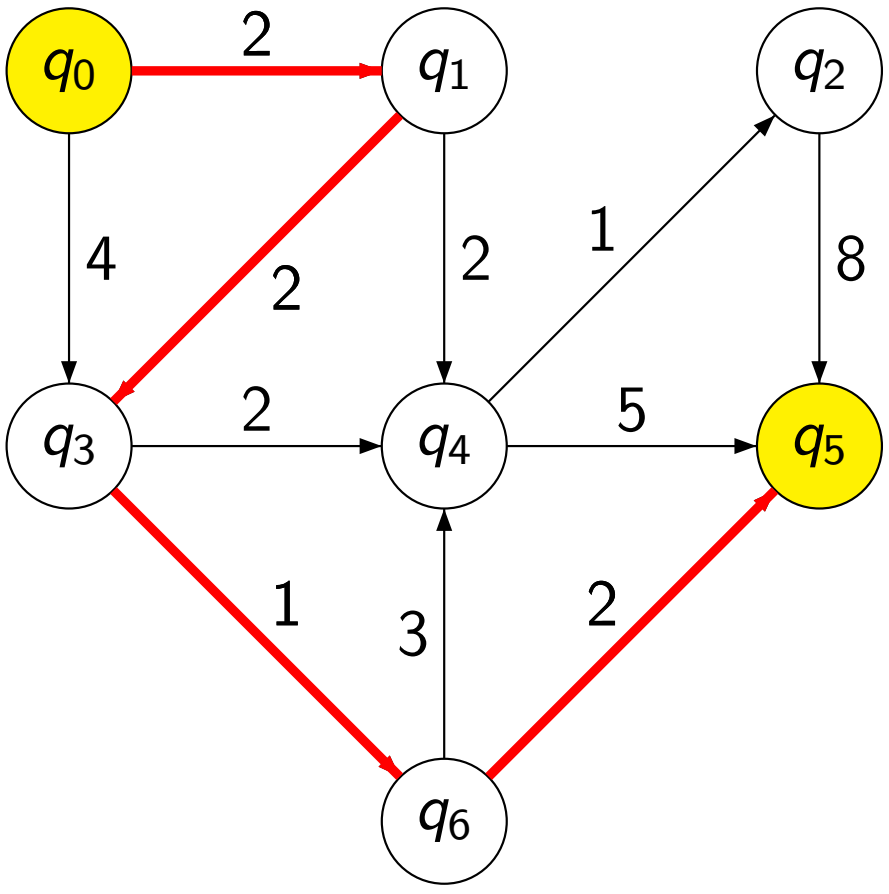
$$w(\rho) \stackrel{\text{def}}{=} \sum_{i=1,\ldots,k} w(v_{i-1}, v_i)$$

Définitions : Un chemin ρ de u à v est un plus court chemin (PCC) de u à v ssi, pour tout chemin π de u à v, on a $w(\pi) \geq w(\rho)$.

Exemple de PCC



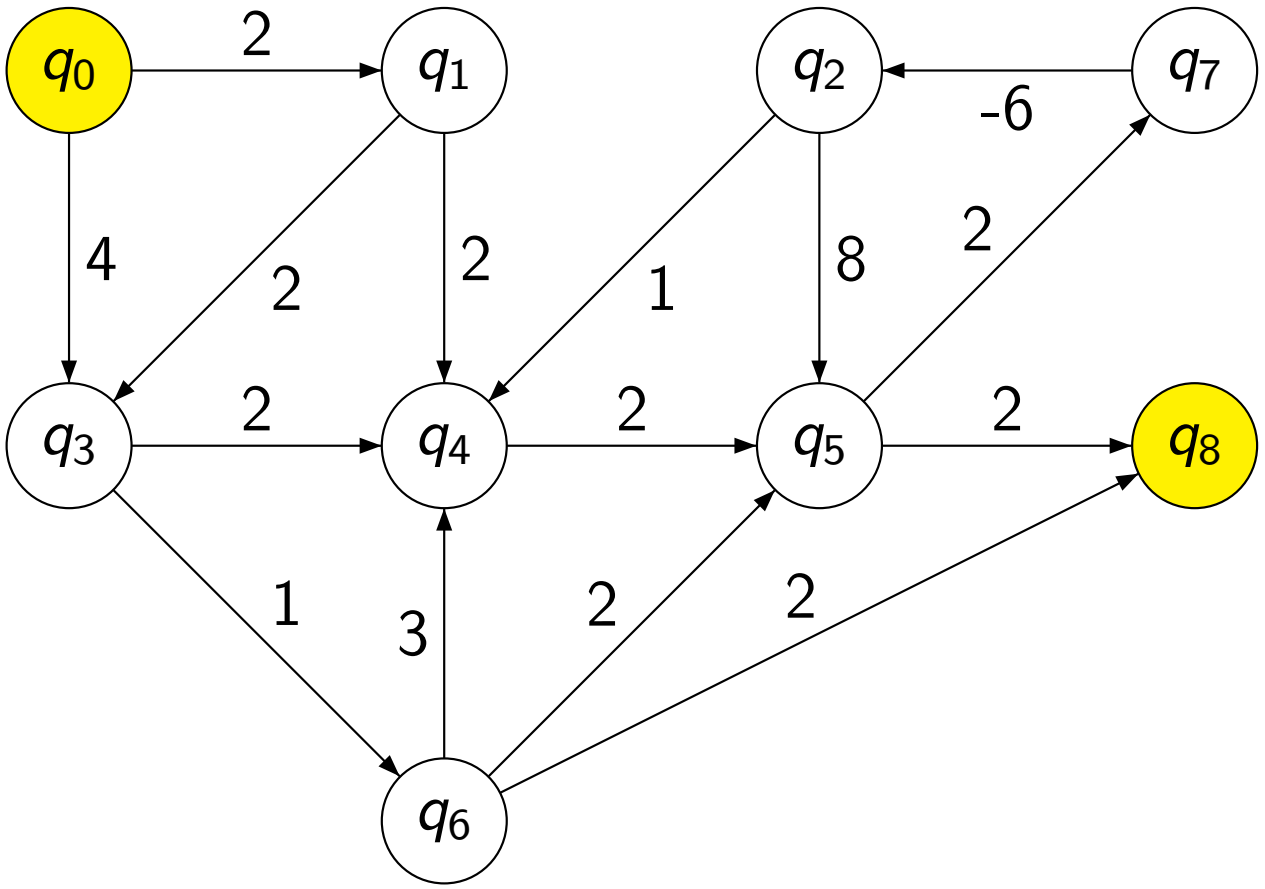
Exemple de PCC



distance $= 7$

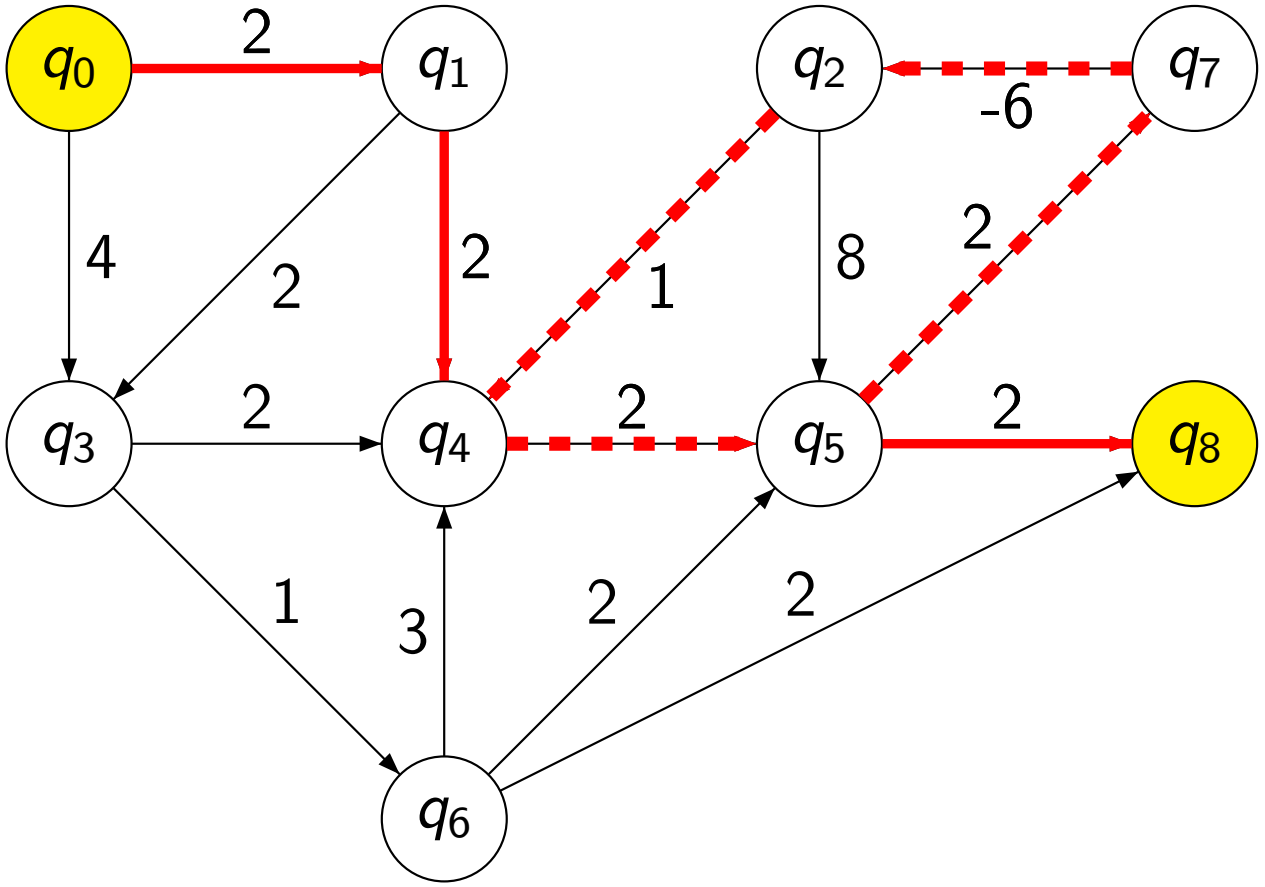
Exemple de PCC

Un PCC de q_0 à q_8 ?



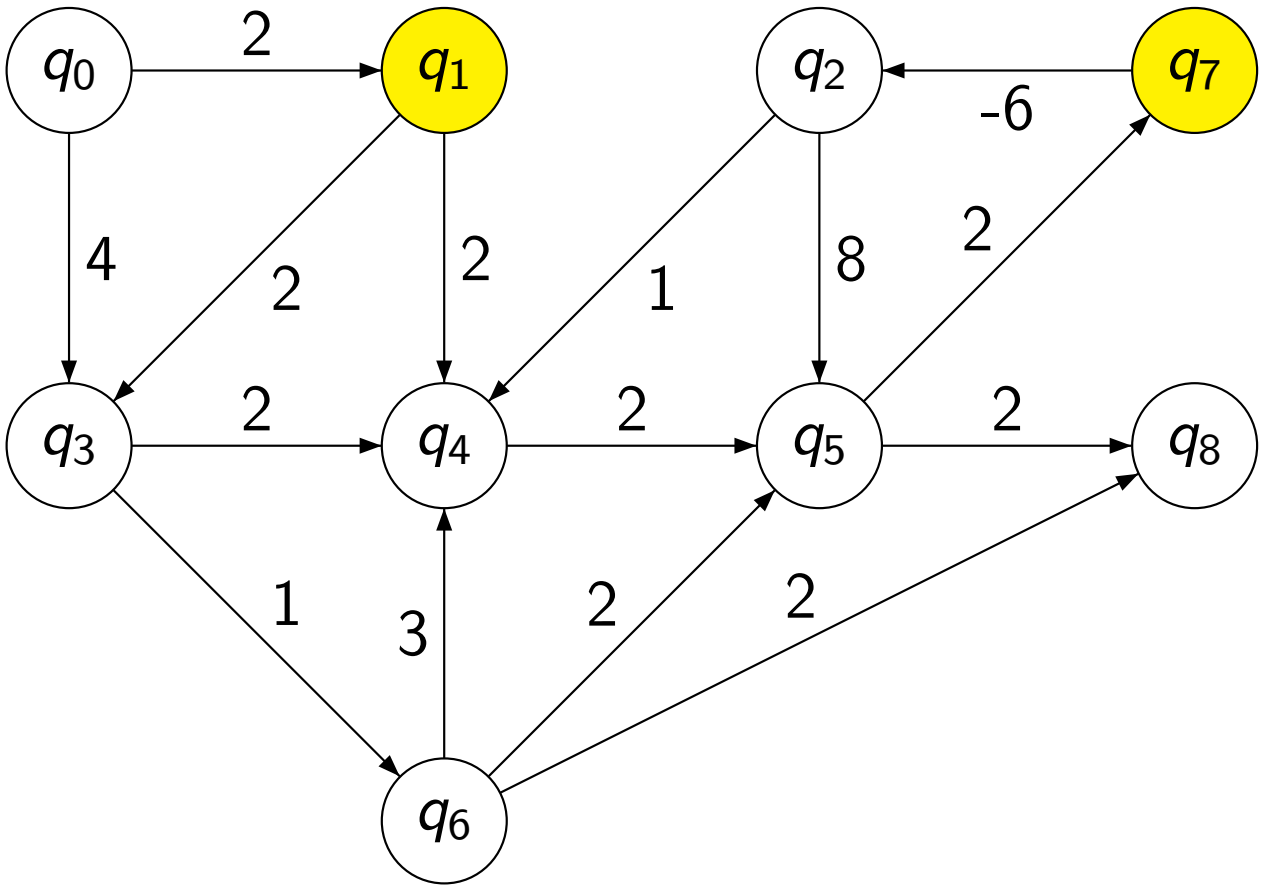
Exemple de PCC

Un PCC de q_0 à q_8 ? $-\infty$! il n'en existe pas ! !



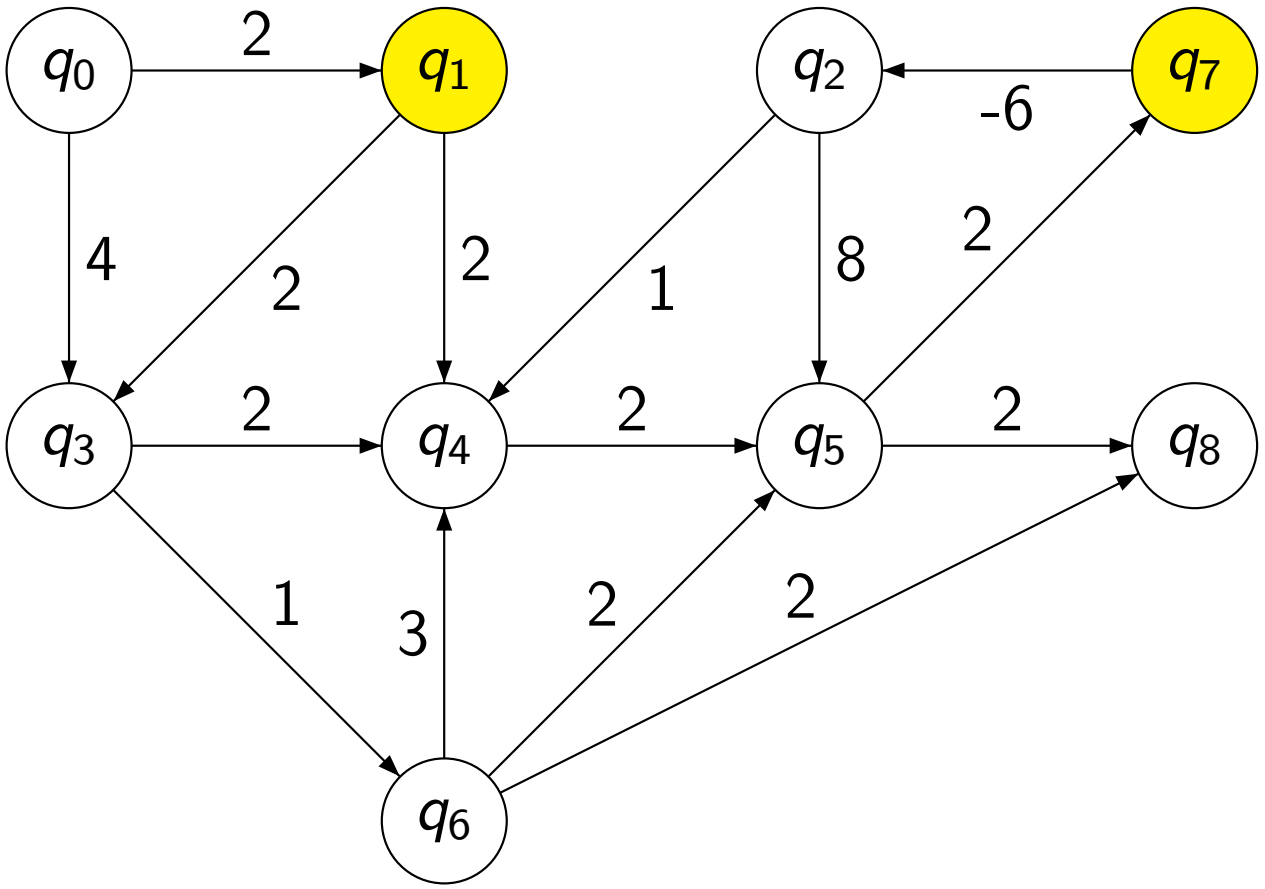
Exemple de PCC

Un PCC de q_7 à q_1 ?



Exemple de PCC

Un PCC de q_7 à q_1 ? ∞ ! il n'en existe pas ! !



Existence de PCC

Propriété :Existence d'un PCC

Il existe un PCC entre u et v ssi

(a) v est atteignable depuis u (*i.e.* $\exists u \rightarrow^* v$), et

(b) il n'existe pas de cycle strictement négatif $c : z \rightarrow^* z$ et un chemin $u \rightarrow^* z \rightarrow^* v$.

Dans la suite, on ne va considérer que des poids positifs ou nuls (mais l'algorithme de Bellman-Ford traite le cas des graphes avec poids négatifs)

Existence de PCC

Propriété :Existence d'un PCC

Il existe un PCC entre u et v ssi

(a) v est atteignable depuis u (*i.e.* $\exists u \to^* v$), et

(b) il n'existe pas de cycle strictement négatif $c : z \to^* z$ et un chemin $u \to^* z \to^* v$.

Dans la suite, on ne va considérer que des poids positifs ou nuls (mais l'algorithme de Bellman-Ford traite le cas des graphes avec poids négatifs)

On définit $\delta(s, u)$ la distance d'un PCC de s à u :

$$\delta(s,u) \stackrel{\text{def}}{=} \begin{cases} \min\{w(\rho) \mid s \to_\rho u\} & \text{si } \exists\, s \to^* u \\ \infty & \text{sinon} \end{cases}$$

Propriété fondamentale des PCC

Propriété

Si $\rho : v_0 \to v_1 \to \ldots \to v_k$ est un PCC entre v_0 et v_k, alors tout sous-chemin $v_i \to \ldots \to v_j$ (avec $0 \leq i < j \leq k$) de ρ est un PCC de v_i à v_j.

Les problèmes de PCC

- les PCC à origine unique : On cherche tous les PCC depuis un sommet de départ s ;
- les PCC à destination unique : On cherche tous les PCC menant à un sommet d'arrivée s ; et
- les PCC pour toutes les paires de sommets de G.

Les problèmes de PCC

- les PCC à origine unique : On cherche tous les PCC depuis un sommet de départ s ;
- les PCC à destination unique : On cherche tous les PCC menant à un sommet d'arrivée s ; et
- les PCC pour toutes les paires de sommets de G.

Algo. de Dijkstra = pour les PCC à origine unique.

Algorithme de Dijkstra

Le cadre : les graphes orientés $G = (S, A, w)$ et valués avec $w : A \to \mathbb{R}_+$

Le problème :

Étant donné un sommet s, trouver la distance d'un PCC entre s et tout autre sommet $u \in S$...

Algorithme de Dijkstra

Le cadre : les graphes orientés $G = (S, A, w)$ et valués avec $w : A \to \mathbb{R}_+$

Le problème :

Étant donné un sommet s, trouver la distance d'un PCC entre s et tout autre sommet $u \in S$... Et construire ces PCC !

Algorithme de Dijkstra

Le cadre : les graphes orientés $G = (S, A, w)$ et valués avec $w : A \to \mathbb{R}_+$

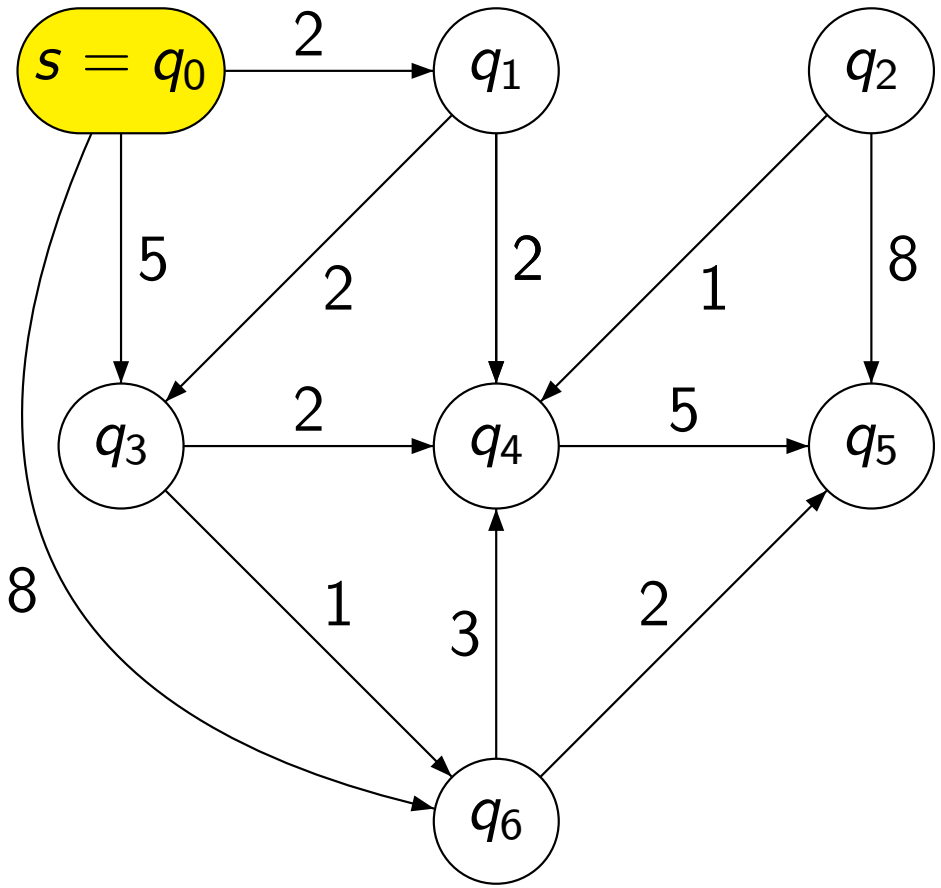
Le problème :

Étant donné un sommet s, trouver la distance d'un PCC entre s et tout autre sommet $u \in S$... Et construire ces PCC !

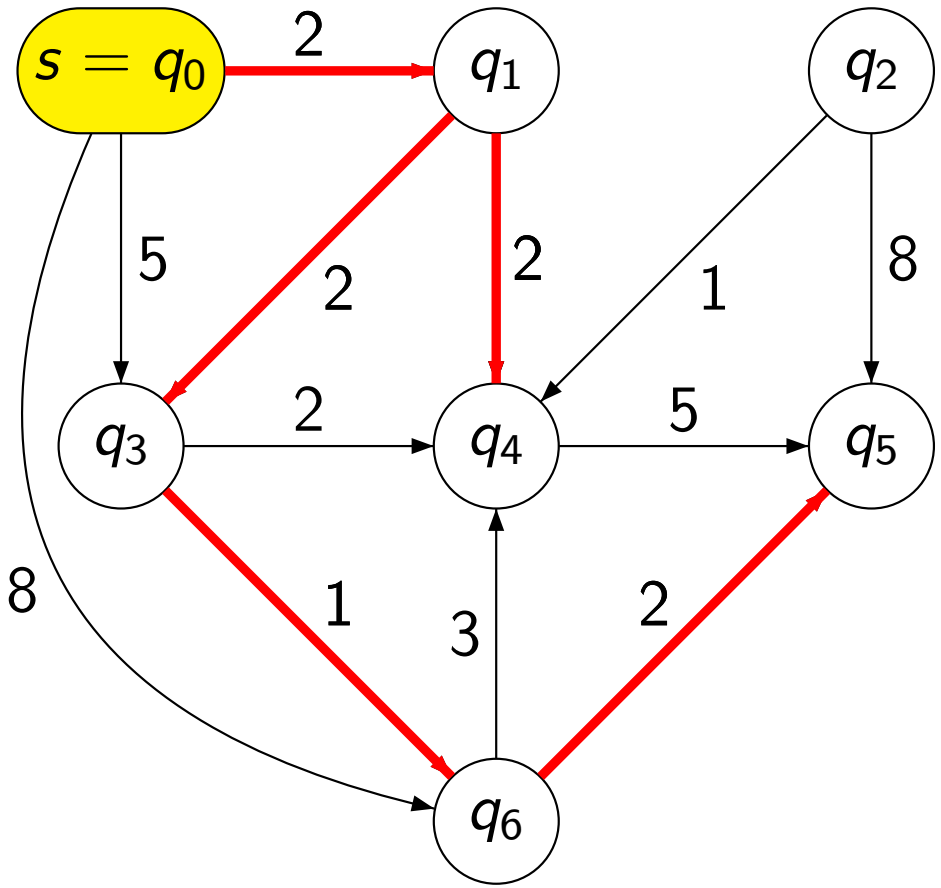
On va construire une arborescence des PCC $T = (S, A')$:

T est un arbre de racine s et contenant tous les sommets x accessibles depuis s et tels que : tout chemin de s à x dans T est un PCC de G.

Arborescence des PCC



Arborescence des PCC



Arborescence des PCC

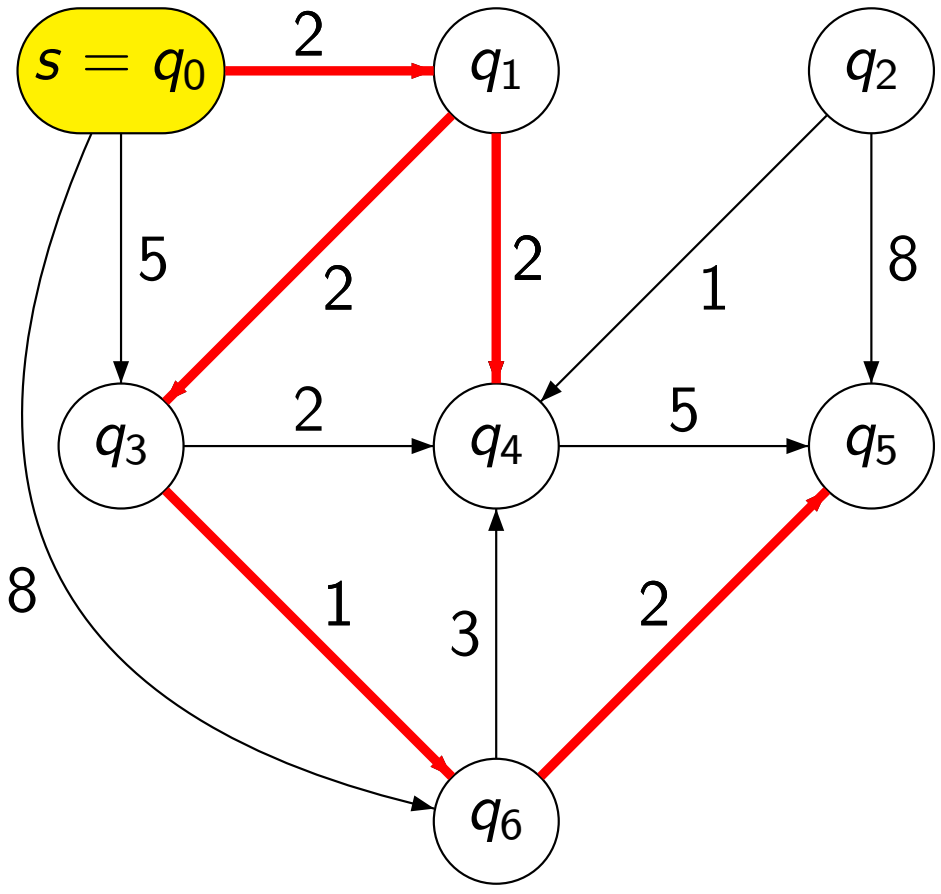


Table des prédécesseurs

$$\Pi[q_0] = \text{nil}$$
$$\Pi[q_1] = q_0$$
$$\Pi[q_2] = \text{nil}$$
$$\Pi[q_3] = q_1$$
$$\Pi[q_4] = q_1$$
$$\Pi[q_5] = q_6$$
$$\Pi[q_6] = q_3$$

Algorithme de Dijkstra

On utilise une file de priorité F pour stocker les sommets :

la clé d[x] de x est sa distance minimale à s en ne passant que par des sommets dont on a déjà trouvé un PCC depuis s et qui ont été déjà extraits de F.

$\Rightarrow$ d[x] est une sur-approximation de $\delta(s, x)$: d[x] $\geq \delta(s, x)$

Algorithme de Dijkstra

PCC-Dijkstra(G, s)

pour chaque $u \in S$ **faire**

  $\Pi[u] := \text{nil}, \quad \text{d}[u] := \begin{cases} 0 & \text{si } u = s \\ \infty & \text{sinon} \end{cases}$

$F := \text{File}(S, d, \text{IndiceDansF})$;

tant que $F \neq \emptyset$ **faire**

  $u := \text{Extraire-Min}(F)$;

  pour chaque $(u, v) \in A$ **faire**

    si $d[v] > d[u] + w(u, v)$ **alors**

      $\text{d}[v] := \text{d}[u] + w(u, v)$;

      $\Pi[v] := u$;

      $\text{MaJ-F-Dijkstra}(F, d, v, \text{IndiceDansF})$

retourner d, Π

Algorithme de Dijkstra

MaJ-F-Dijkstra(F, d, v,IndiceDansF)

begin

  $i :=$ IndiceDansF$[v]$;

  tant que $(i/2 \geq 1) \wedge (d[F[i/2]] > d[F[i]])$

  faire

    $F[i] \leftrightarrow F[i/2]$;

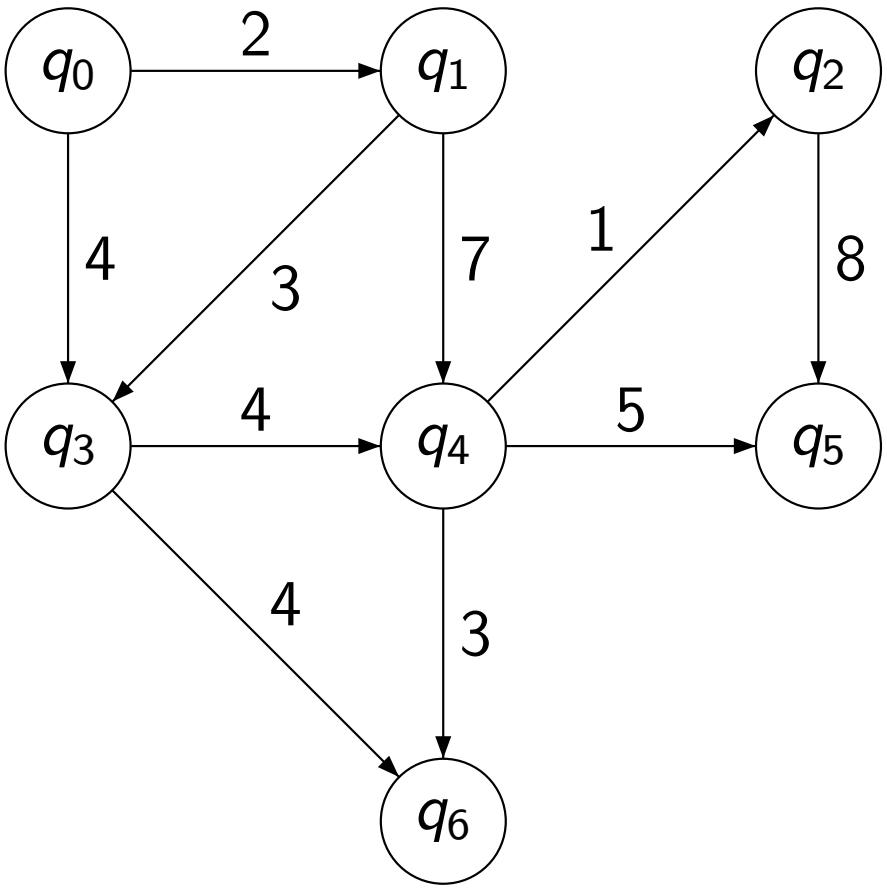
    IndiceDansF$[F[i]] := i$;

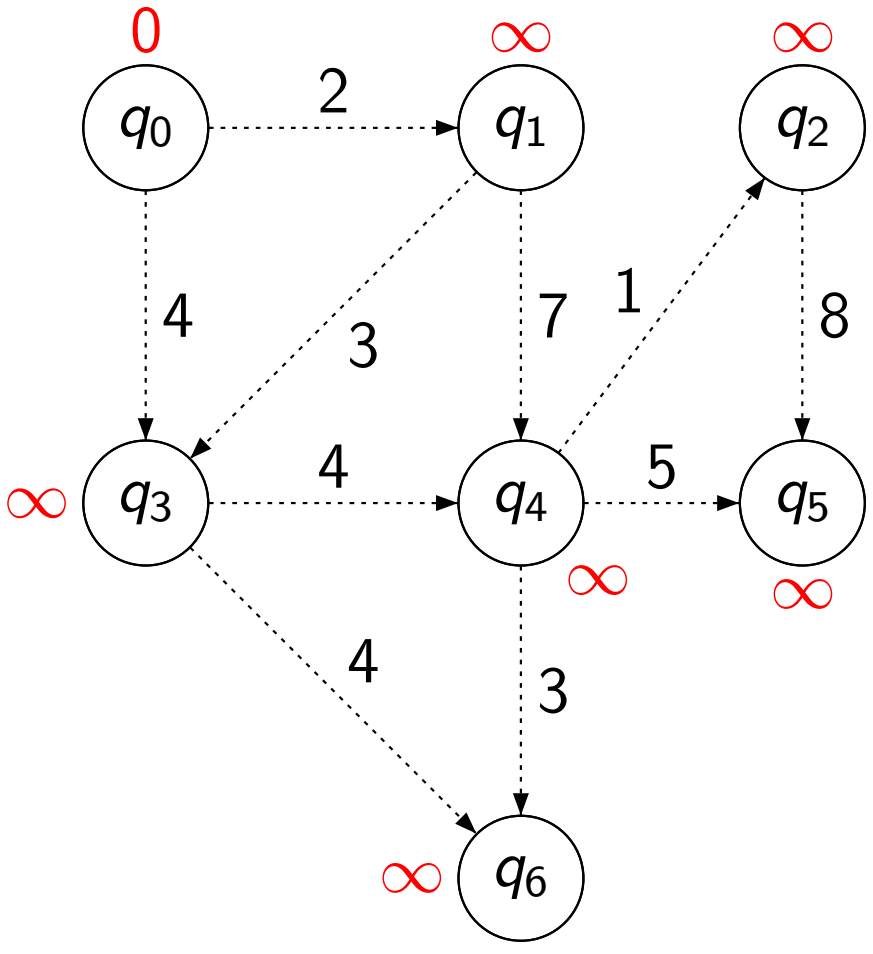
    IndiceDansF$[F[i/2]] := i/2$;

    $i := i/2$;

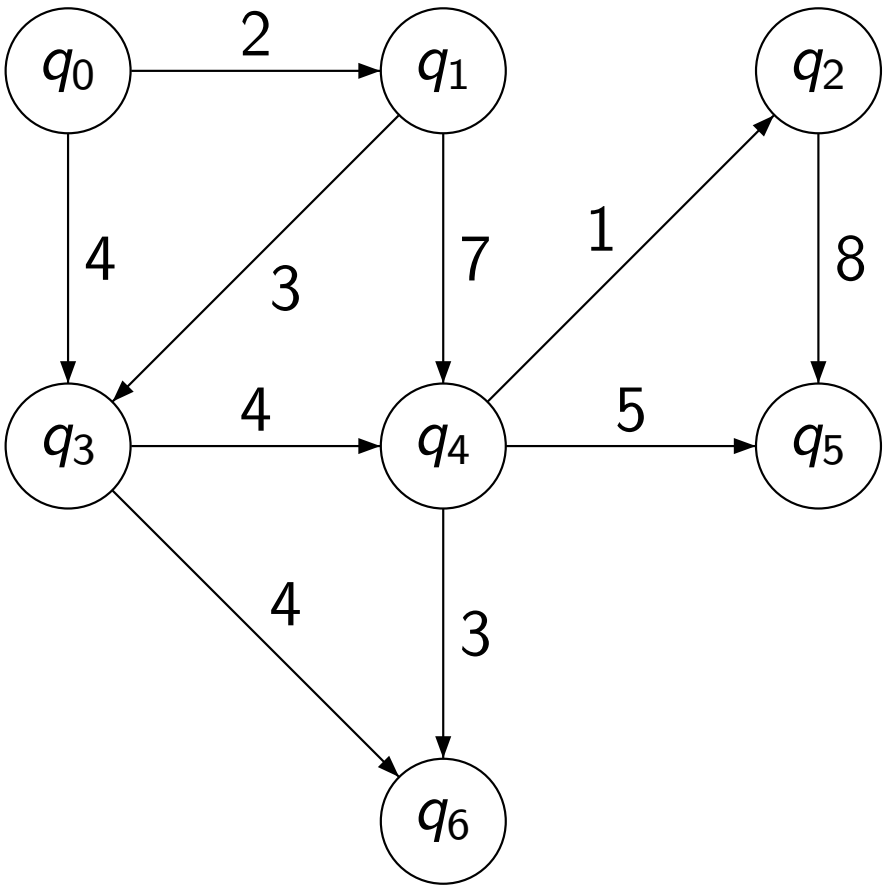
end

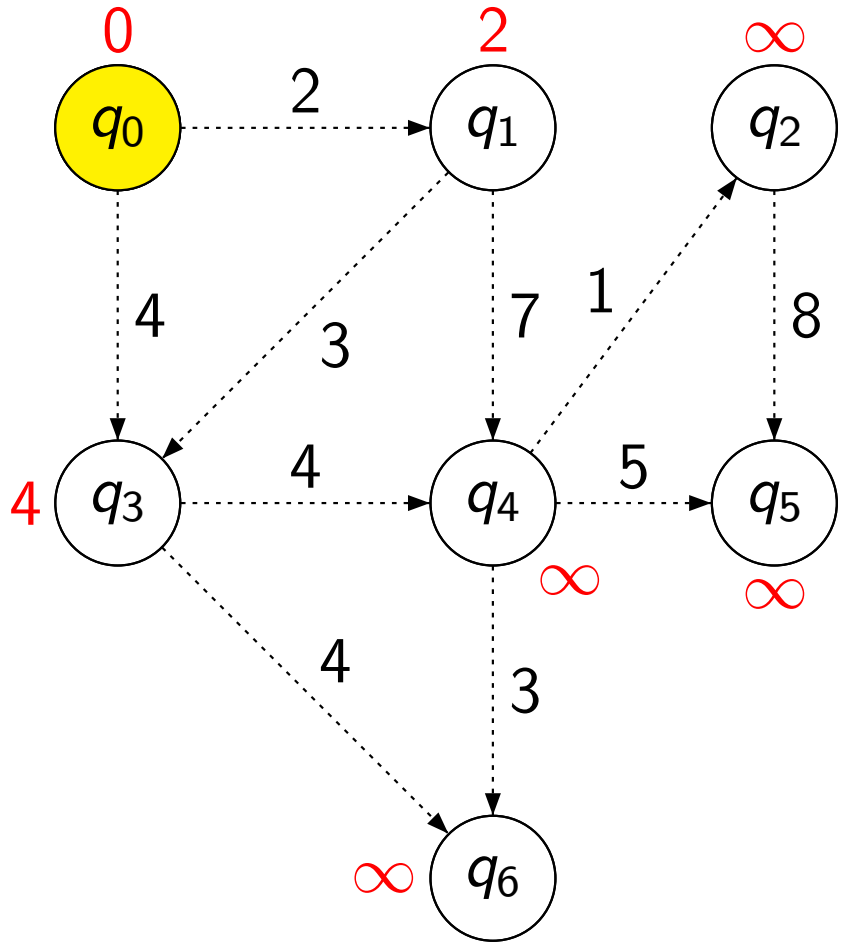
Exemple



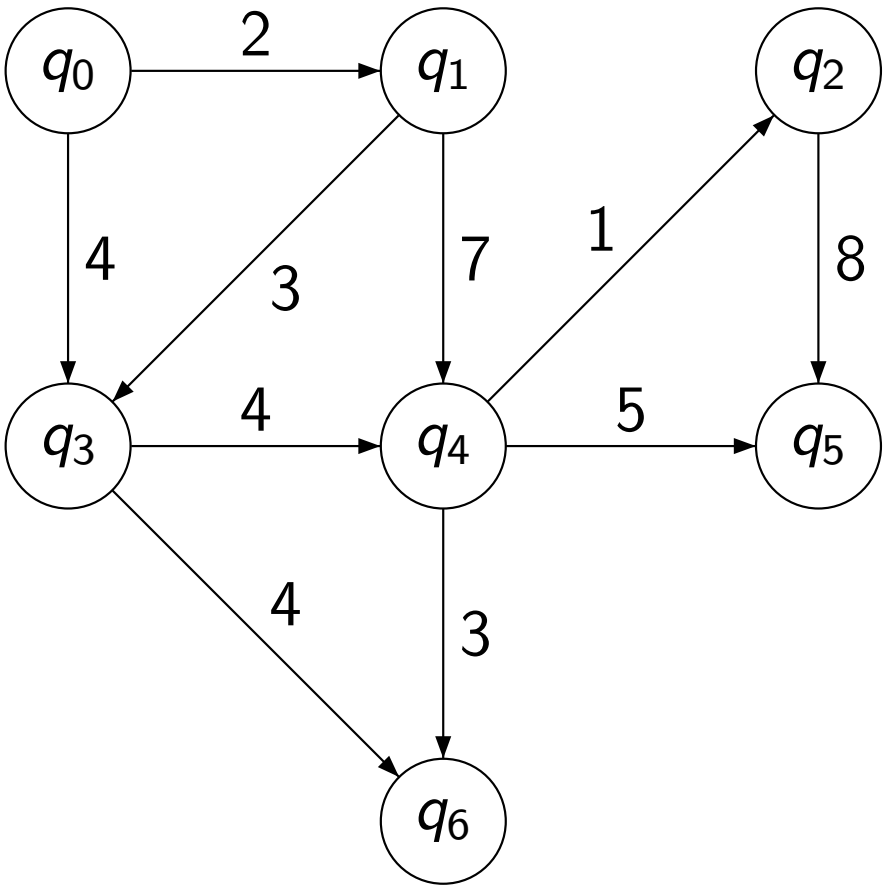


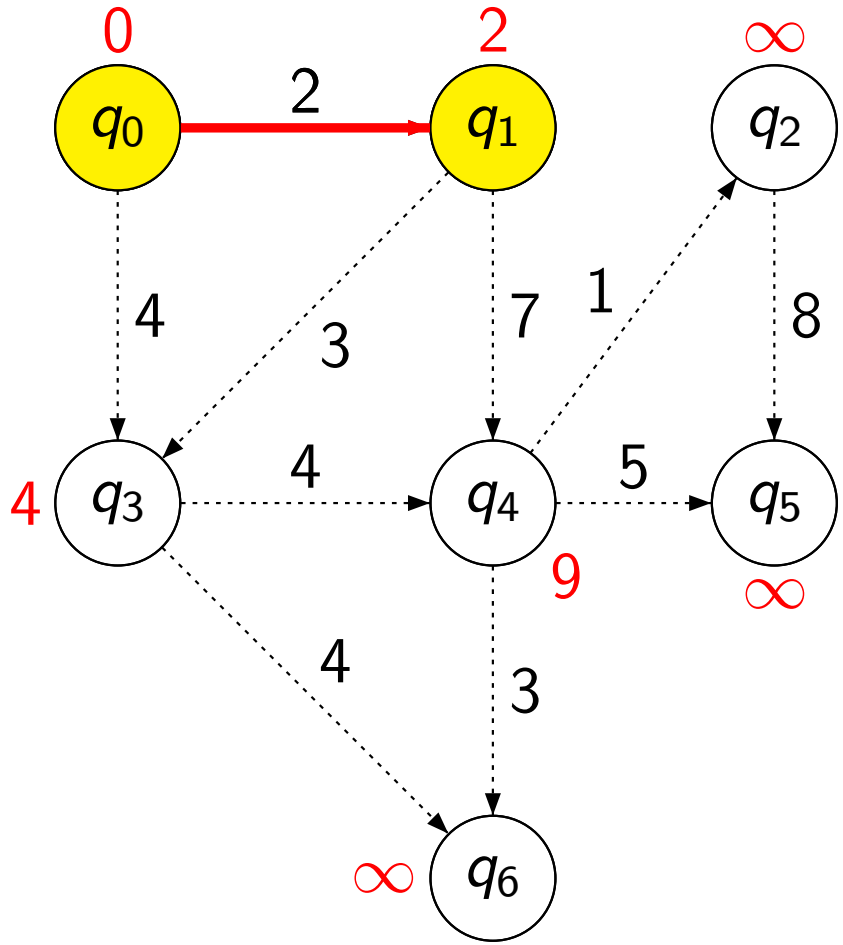
Exemple


q_0
q_1
q_2
q_3
q_4
q_5
q_6
2
4
3
7
1
8
4
5
4
3

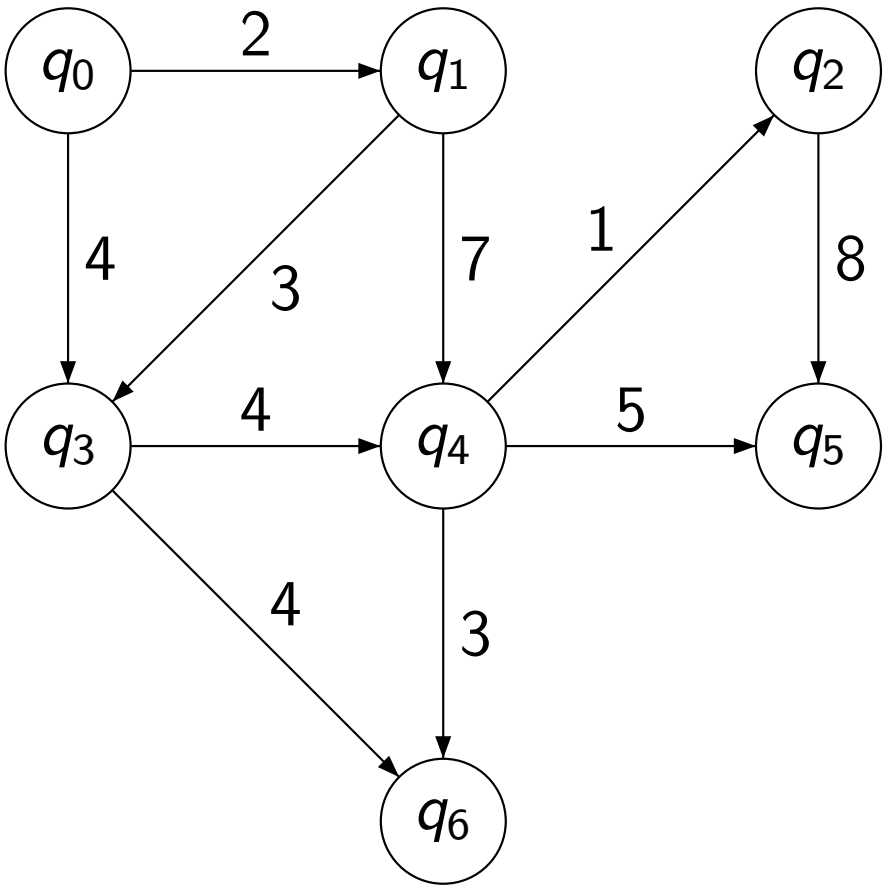

0
2
∞
q_0
q_1
q_2
4
3
7
1
8
4
q_3
4
q_4
5
q_5
∞
∞
4
3
∞
q_6

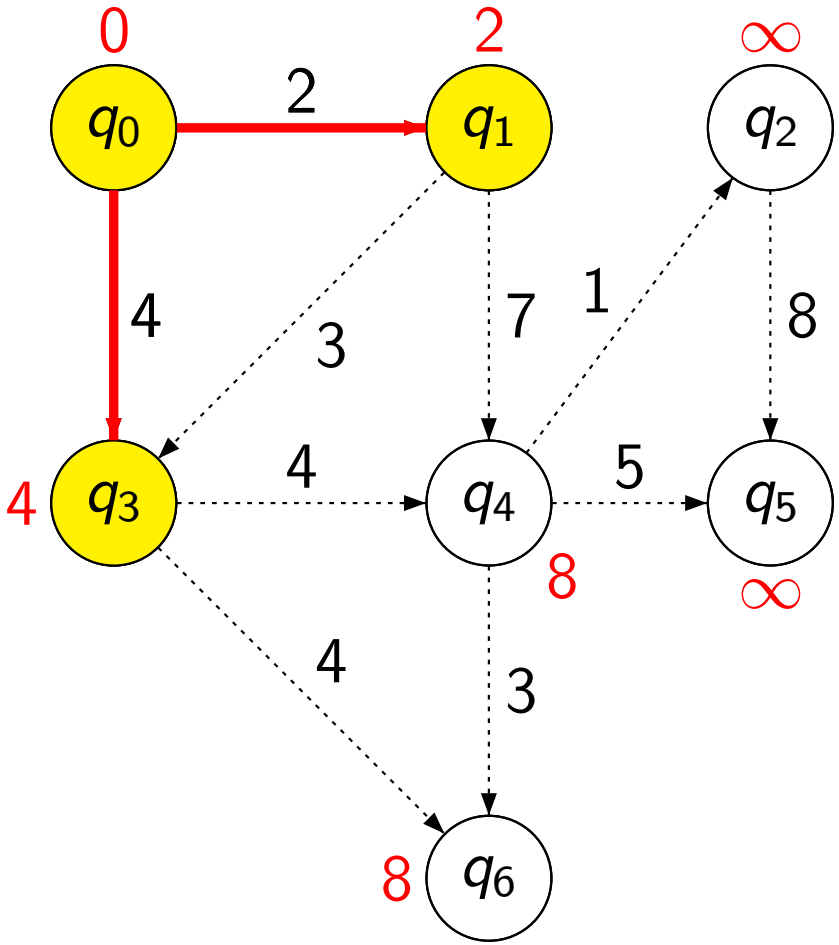
Exemple


q_0
q_1
q_2
q_3
q_4
q_5
q_6
2
4
3
7
1
8
4
5
4
3

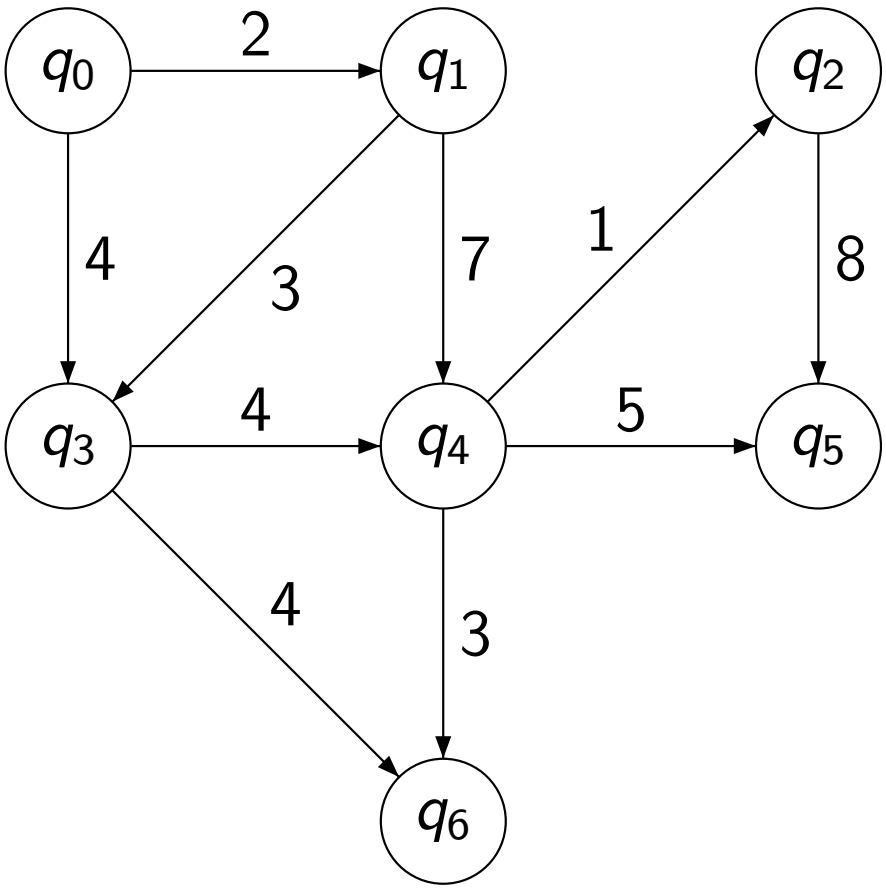

0
2
∞
q_0
q_1
q_2
2
4
3
7
1
8
4
q_3
4
q_4
5
q_5
9
∞
4
3
∞
q_6

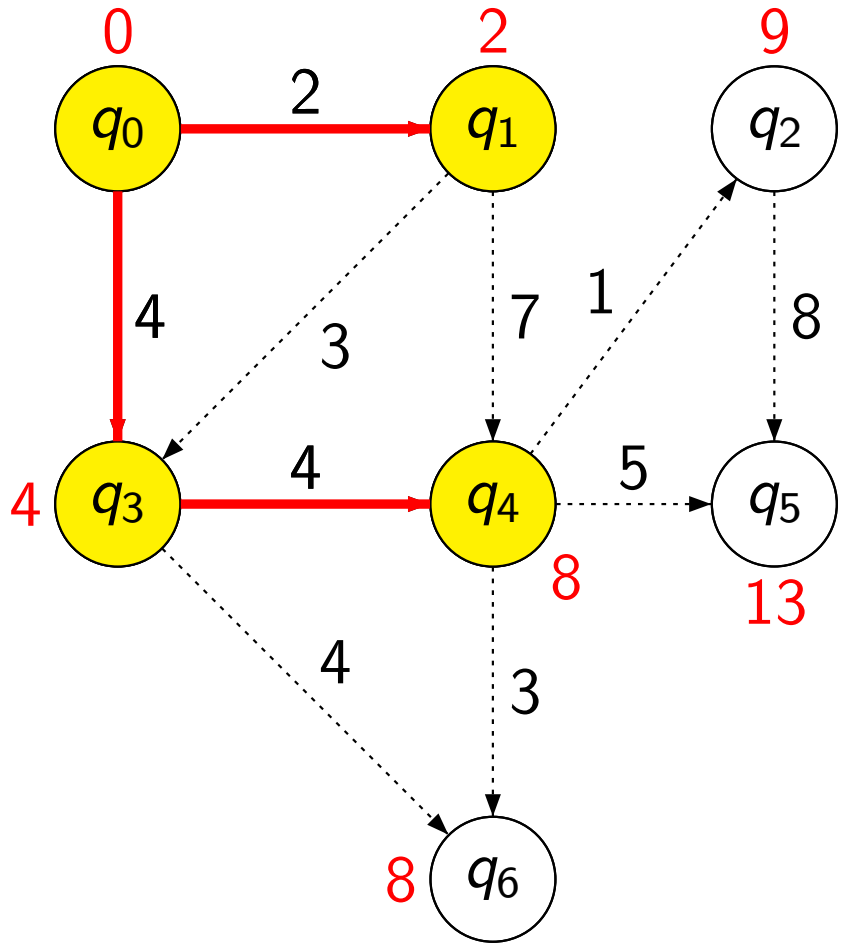
Exemple


q_0
q_1
q_2
q_3
q_4
q_5
q_6
2
4
3
7
1
8
4
5
4
3

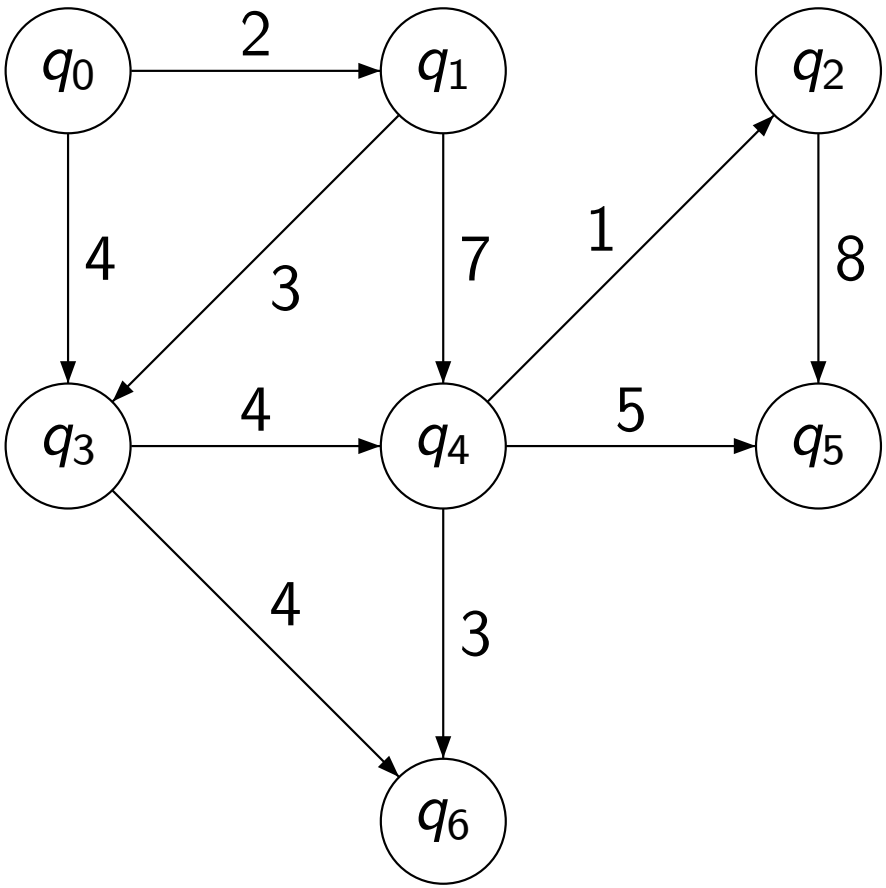

0
2
∞
q_0
q_1
q_2
2
4
3
7
1
8
4
q_3
4
q_4
5
q_5
8
∞
4
3
8
q_6

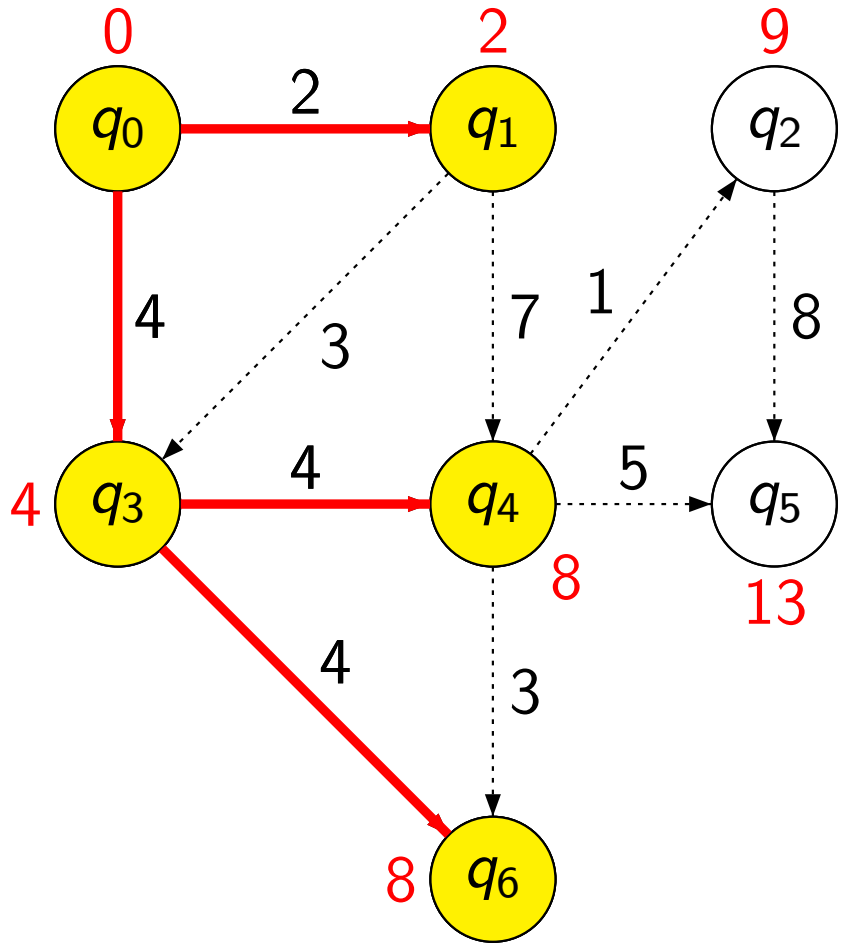
Exemple


q_0
q_1
q_2
q_3
q_4
q_5
q_6
2
4
3
7
1
8
4
5
4
3

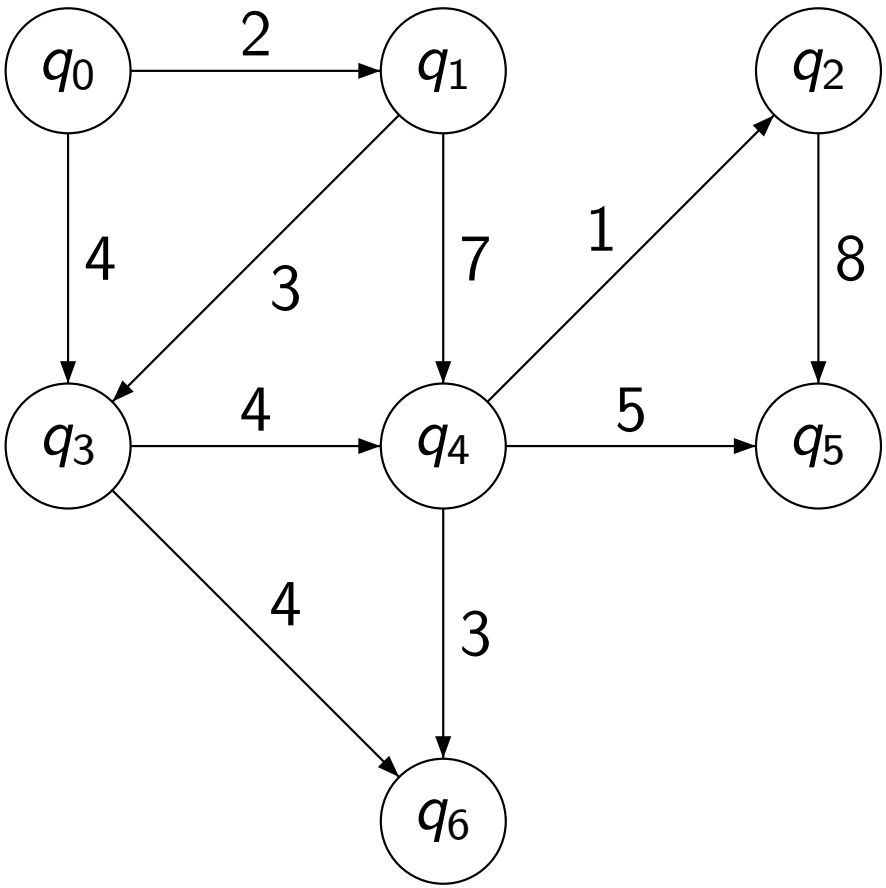

0
2
9
q_0
q_1
q_2
2
4
3
7
1
8
4
q_3
4
q_4
5
q_5
8
13
4
3
8
q_6

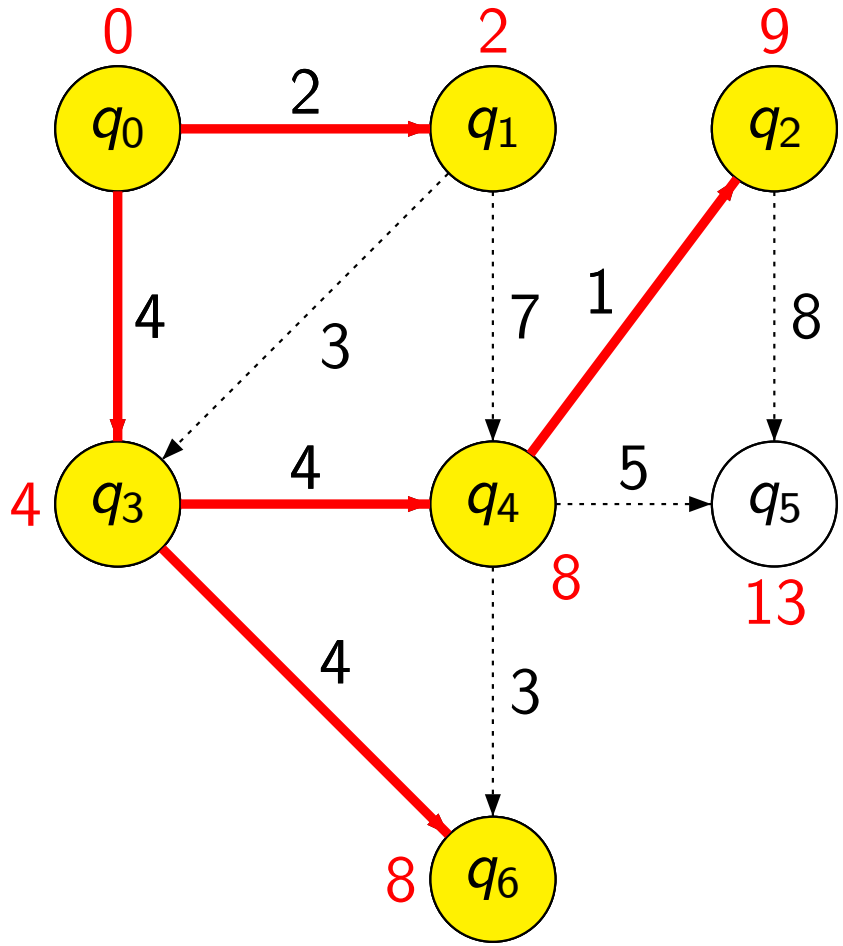
Exemple


q_0
q_1
q_2
q_3
q_4
q_5
q_6
2
4
3
7
1
8
4
5
4
3

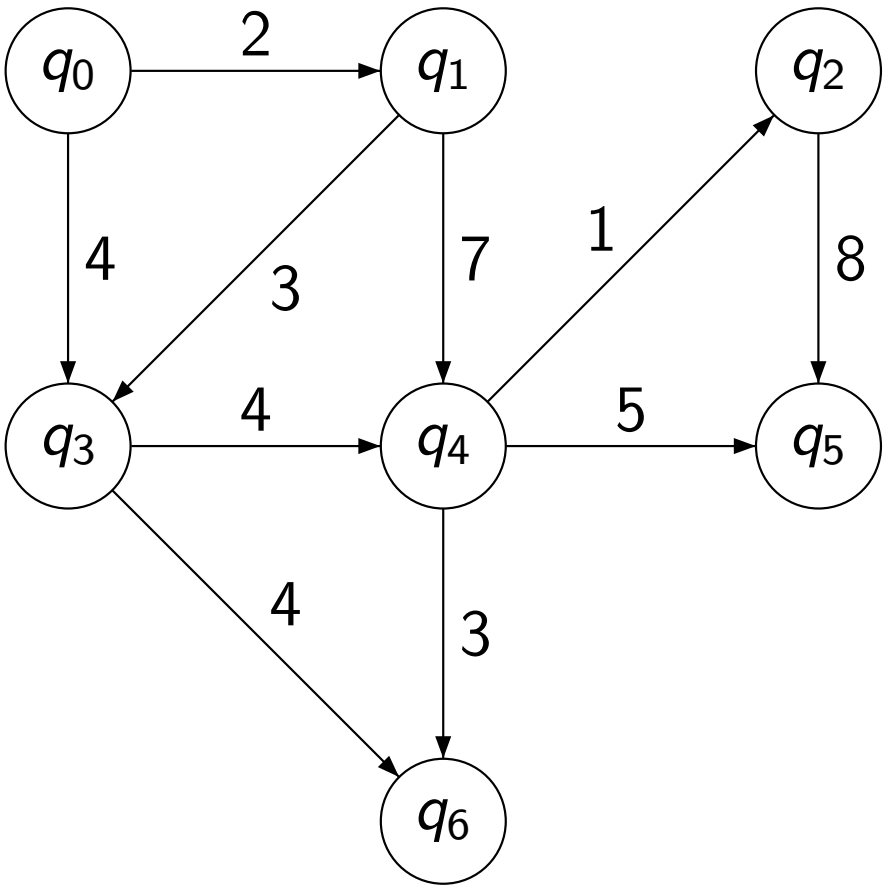

0
2
9
q_0
q_1
q_2
4
3
7
1
8
4
q_3
4
q_4
5
q_5
8
13
4
3
8
q_6

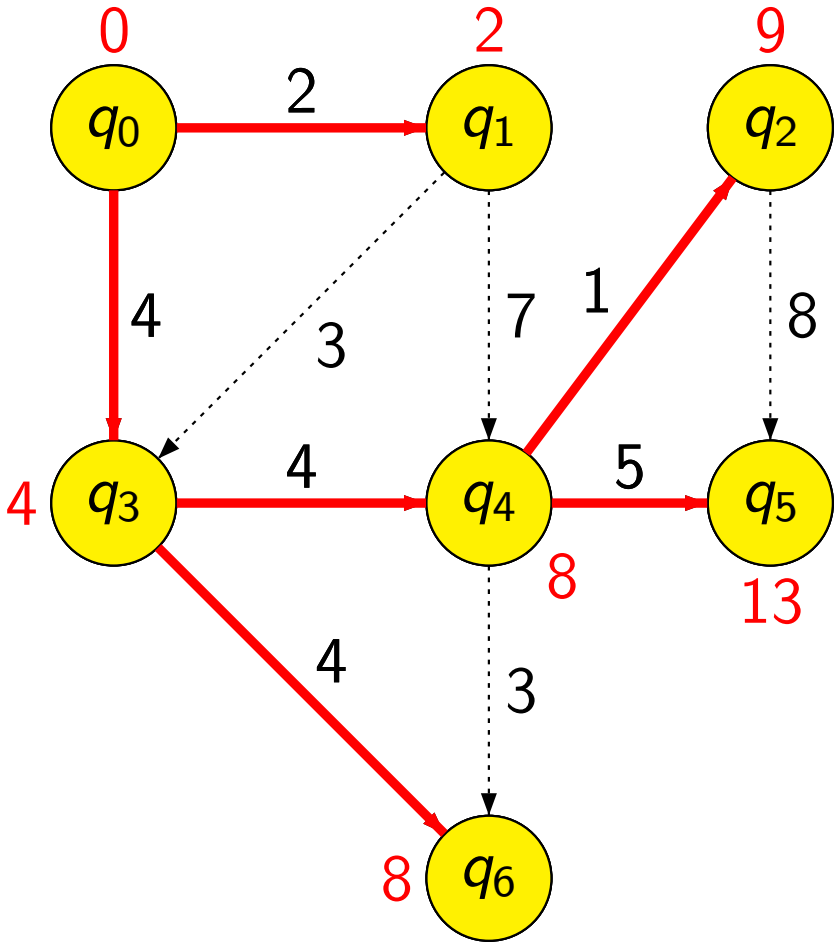
Exemple


q_0
q_1
q_2
q_3
q_4
q_5
q_6
2
4
3
7
1
8
4
5
4
3


0
2
9
q_0
q_1
q_2
2
4
3
7
1
8
4
q_3
4
q_4
5
q_5
8
13
4
3
8
q_6

Exemple


q_0
q_1
q_2
q_3
q_4
q_5
q_6
2
4
3
7
1
8
4
5
4
3


0
2
9
q_0
q_1
q_2
2
4
3
7
1
8
4
q_3
4
q_4
5
q_5
8
13
4
3
8
q_6

Propriété de l'algorithme de Dijkstra

Théorème :correction de l'algorithme de Dijkstra

$G = (S, A, w)$ orienté et valué tel que $w : A \to \mathbb{R}_+$.
L'algorithme de Dijkstra

1. termine,
2. à la fin, on a $\mathrm{d}[u] = \delta(s, u)$ pour tout sommet $u \in S$, et
3. $\forall u \in S \backslash \{s\}$, si $\mathrm{d}[u] < \infty$, alors il existe un PCC de s à u dont le dernier arc est $(\Pi[u], u)$.

L'algorithme PCC-Dijkstra prend un temps en $O\Big((|S| + |A|) \cdot \log(|S|)\Big)\ldots$

c'est à dire en $O(|A| \cdot \log(|S|))$ lorsqu'on suppose $|S| \leq |A|$.

Plan

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 :

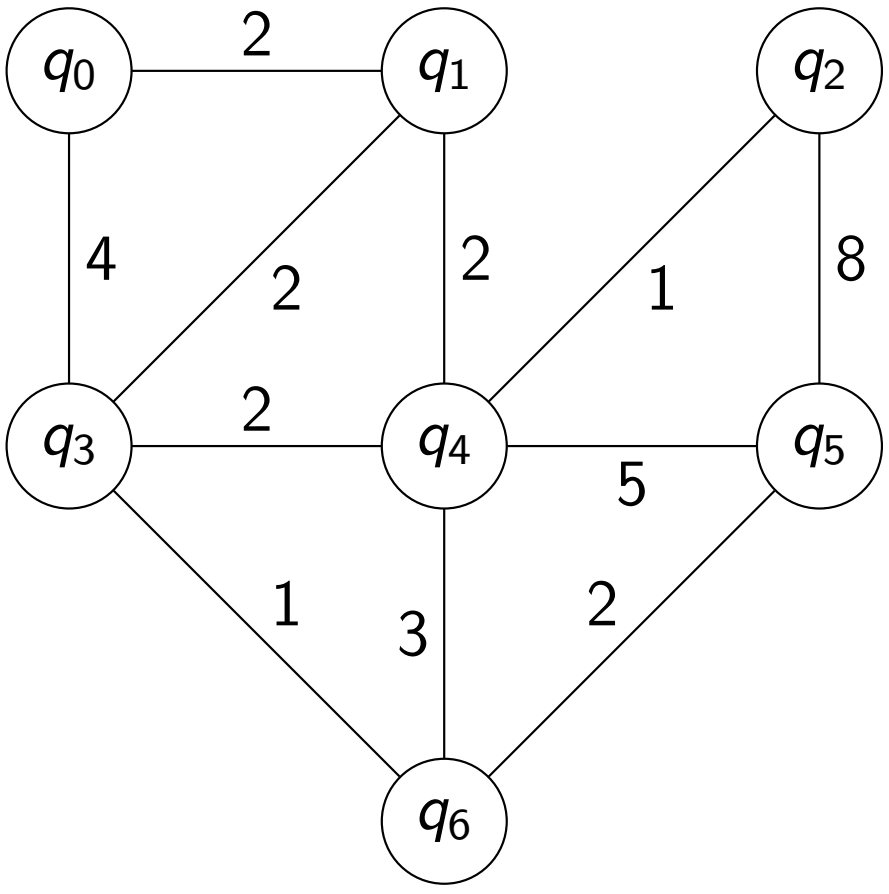
$$w(A') = \min\{w(A'') \mid T' = (G, A'') \text{ est un AC de } G\}$$

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

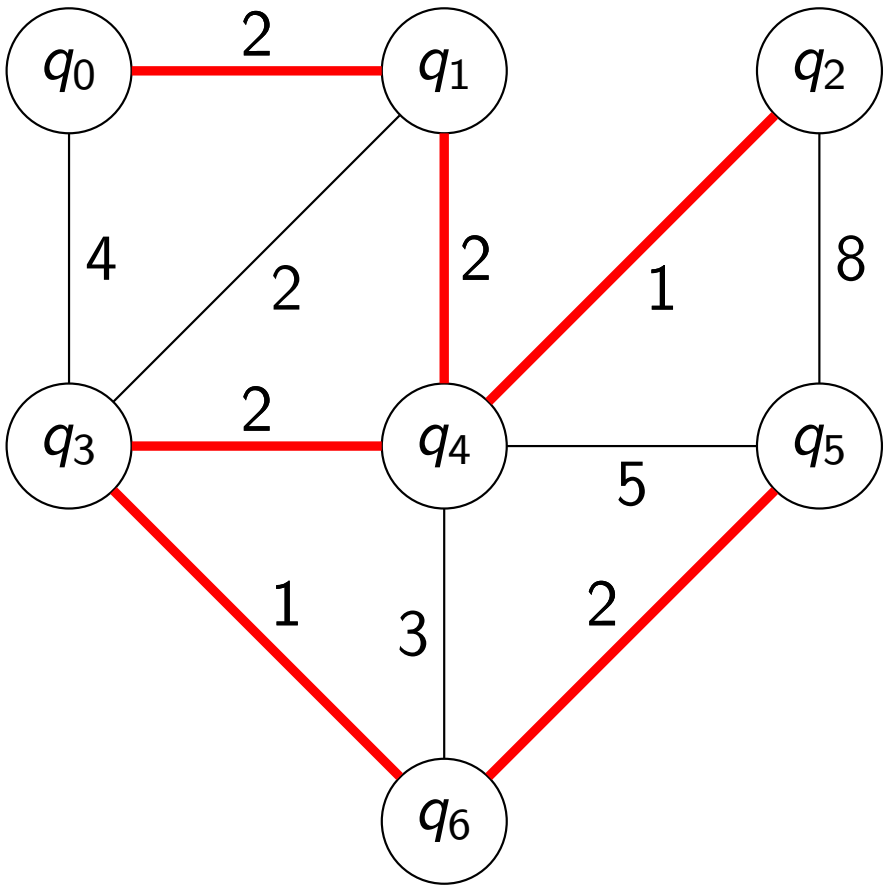
Propriété :Existence d'un ACM

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

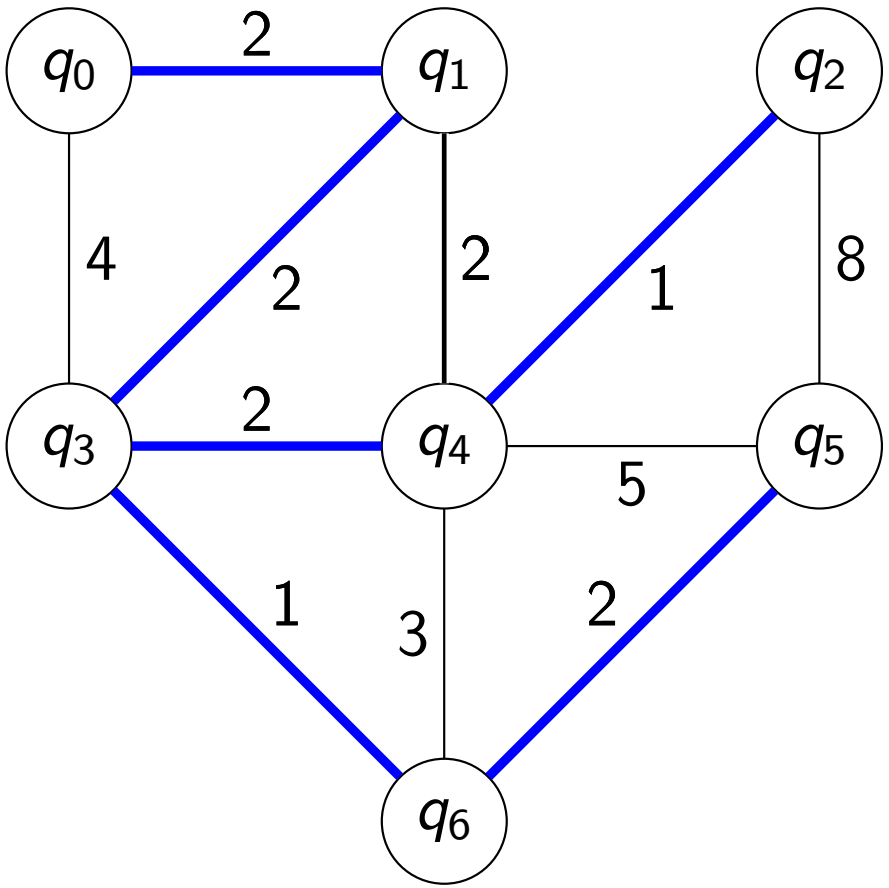
Exemple d'ACM



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'...

Deux algorithmes (glouton) : Prim et Kruskal.

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

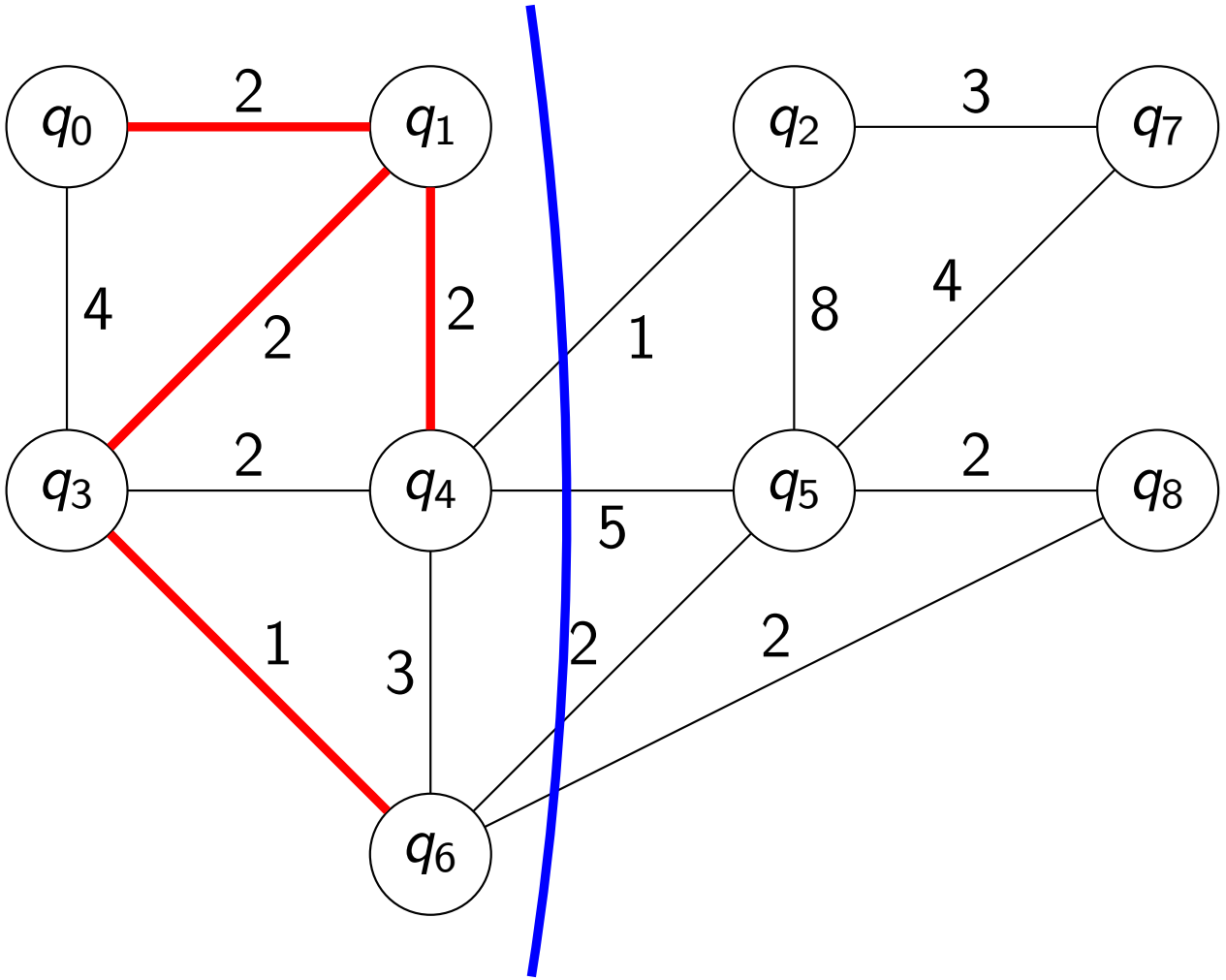
A chaque étape de la construction de A' :

- 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.

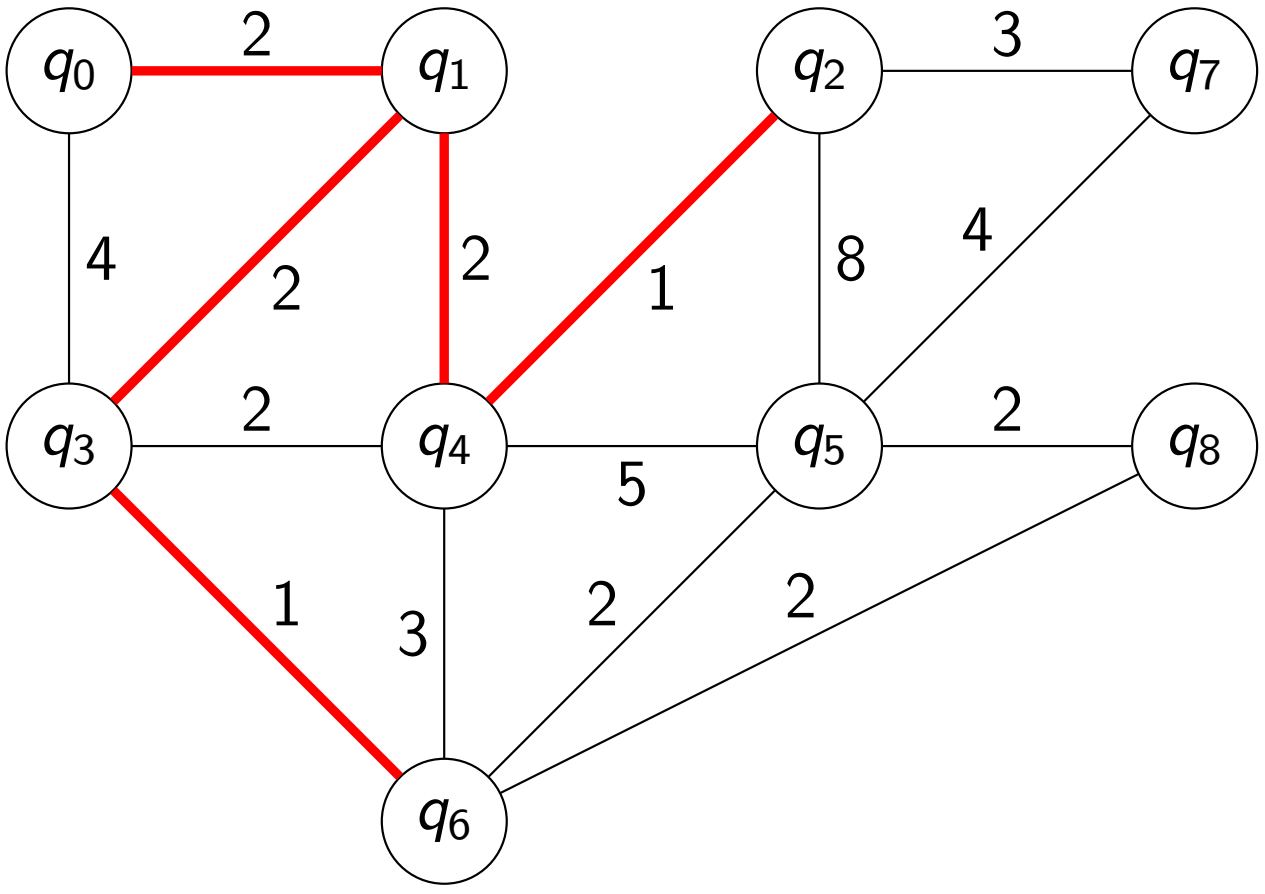
Propriété [correction de l'algorithme de Prim]

A chaque étape, l'arête choisie est compatible.

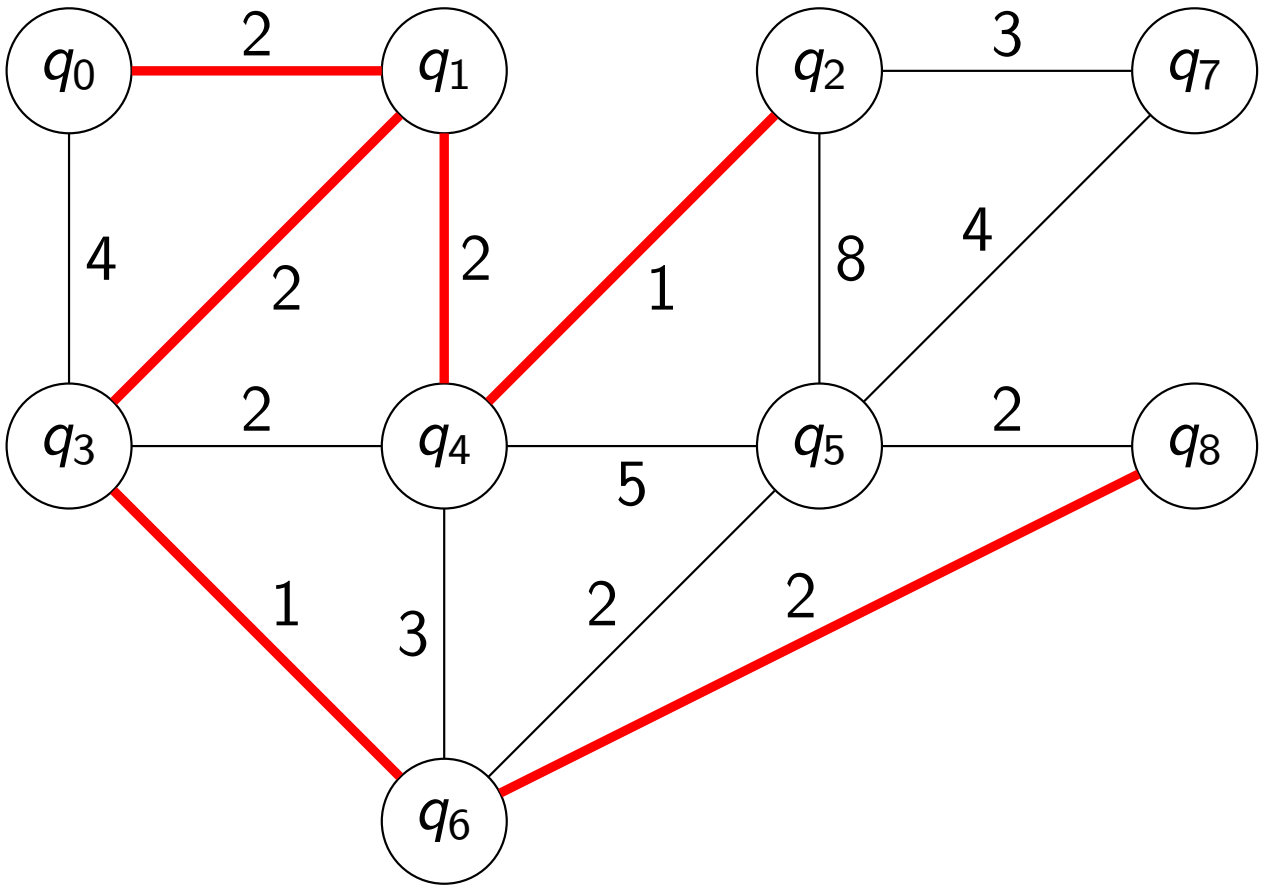
Exemple



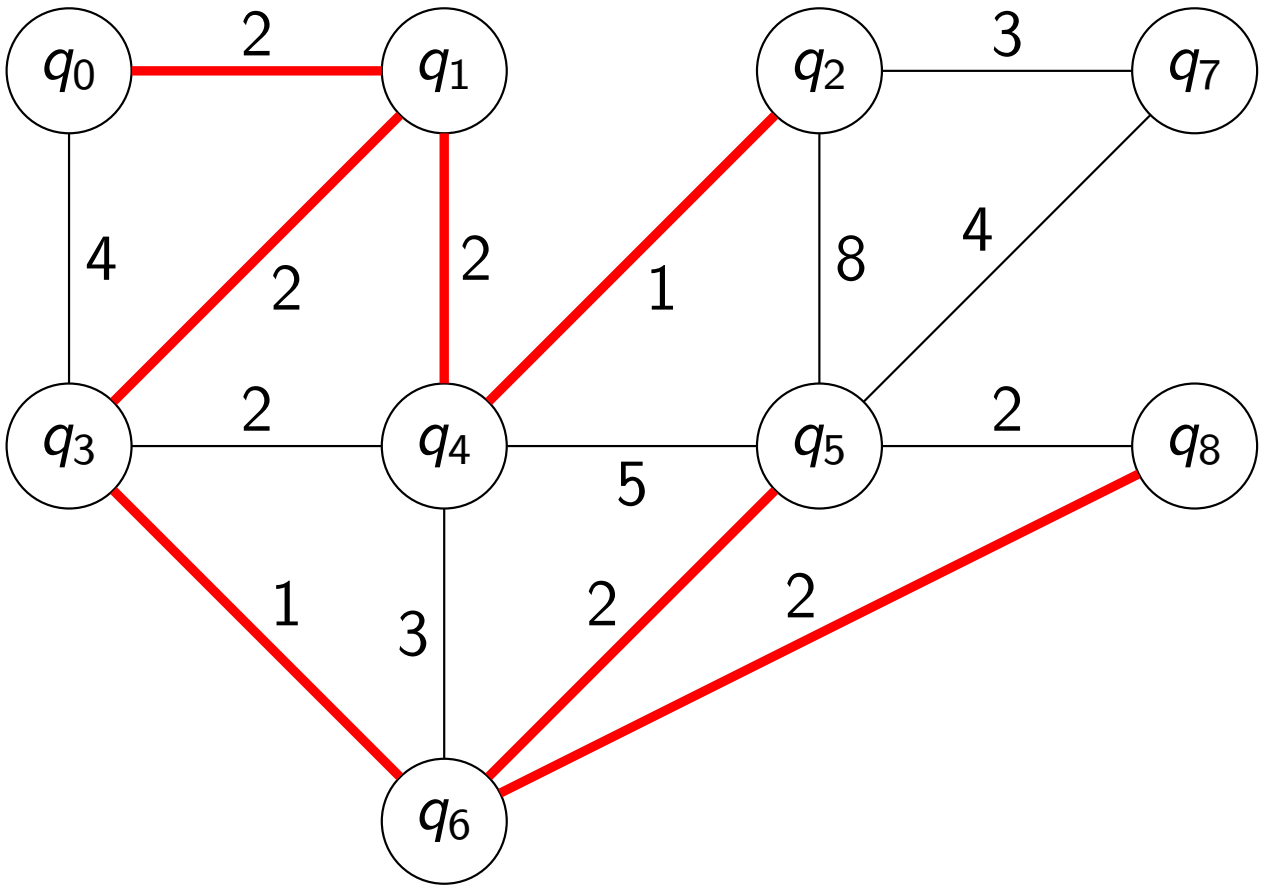
Exemple



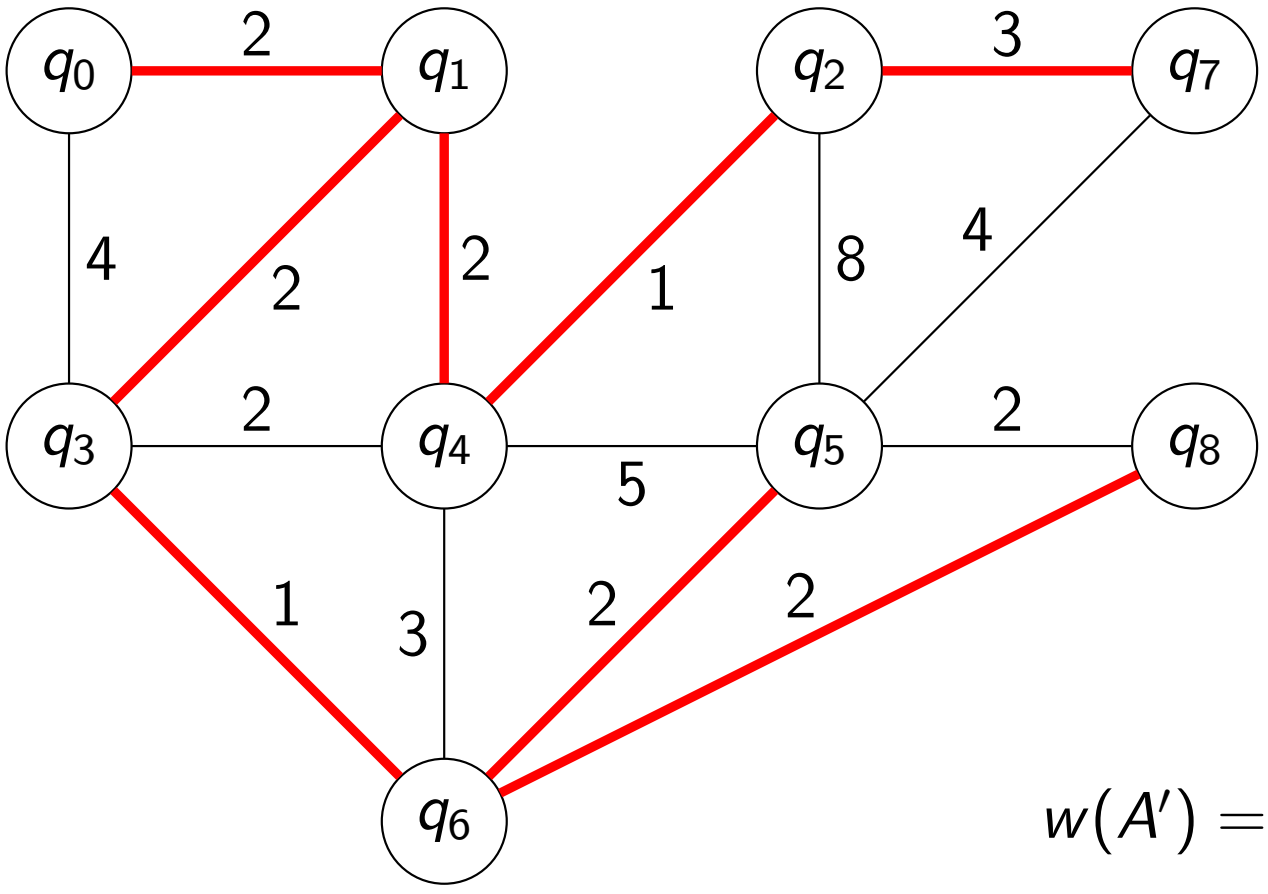
Exemple



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 :

- `IndiceF`$[x]$: 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$(F, \mathrm{d}, x,$ `IndiceF`$)$: met à jour F après l'ajout de x dans A' (cf. Dijkstra).

Algorithme de Prim

Recherche-ACM-Prim(G, s)

pour chaque $u \in S$ **faire**

  $\Pi[u] := \text{nil}, \quad \text{d}[u] := \begin{cases} 0 & \text{si } u = s \\ \infty & \text{sinon} \end{cases}$

$A' := \emptyset, \quad F := \text{File}(S, \text{d}, \text{IndiceF});$

tant que $F \neq \emptyset$ **faire**

  $u := \text{Extraire-Min}(F), \ \text{IndiceF}[u] := -1;$

  si $u \neq s$ **alors** $A' := A' \cup \{(\Pi(u), u)\};$

  pour chaque $(u, v) \in A$ *tq*

  $(\mathit{IndiceF}[v] \neq -1) \wedge (w(u, v) < d[v])$ **faire**

    $\Pi[v] \leftarrow u, \quad \text{d}[v] \leftarrow w(u, v);$

    $\text{MaJ-F-Prim}(F, \text{d}, v, \text{IndiceF})$

return A'

MaJ-F-Prim = MaJ-F-Dijkstra

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-Prim$(F, \mathrm{d}, v, \mathrm{IndiceF})$ 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

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 hypothèses 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 : $w(x, y) + w(y, z) \geq w(x, z)$

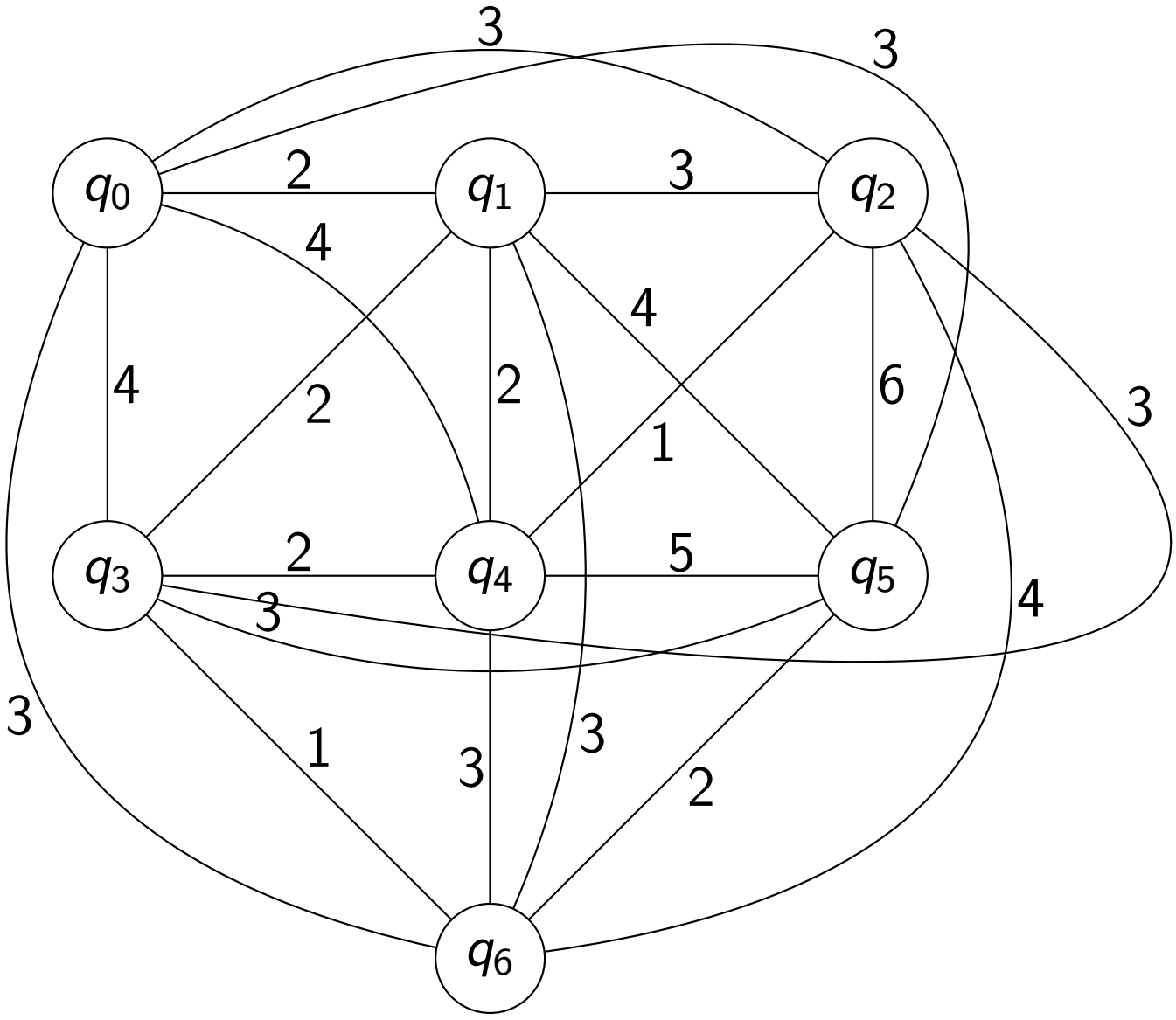
Et on va essayer d'approximer le circuit hamiltonien minimal...

Approximation du VdC

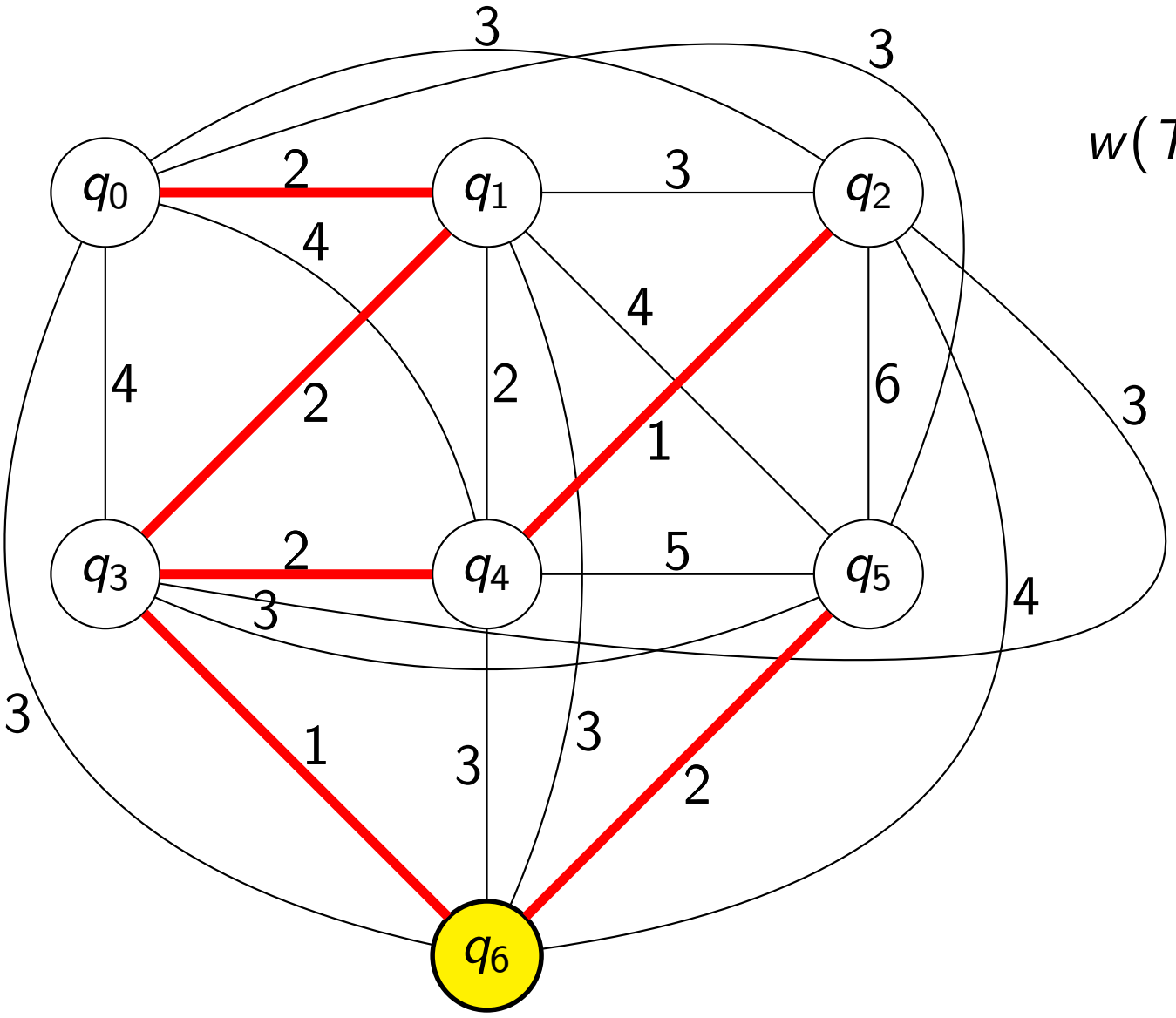
Fonction Approx-VdC(Graphe $G = (S, A, w)$)

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
4. retourner le cycle correspondant à L

Exemple

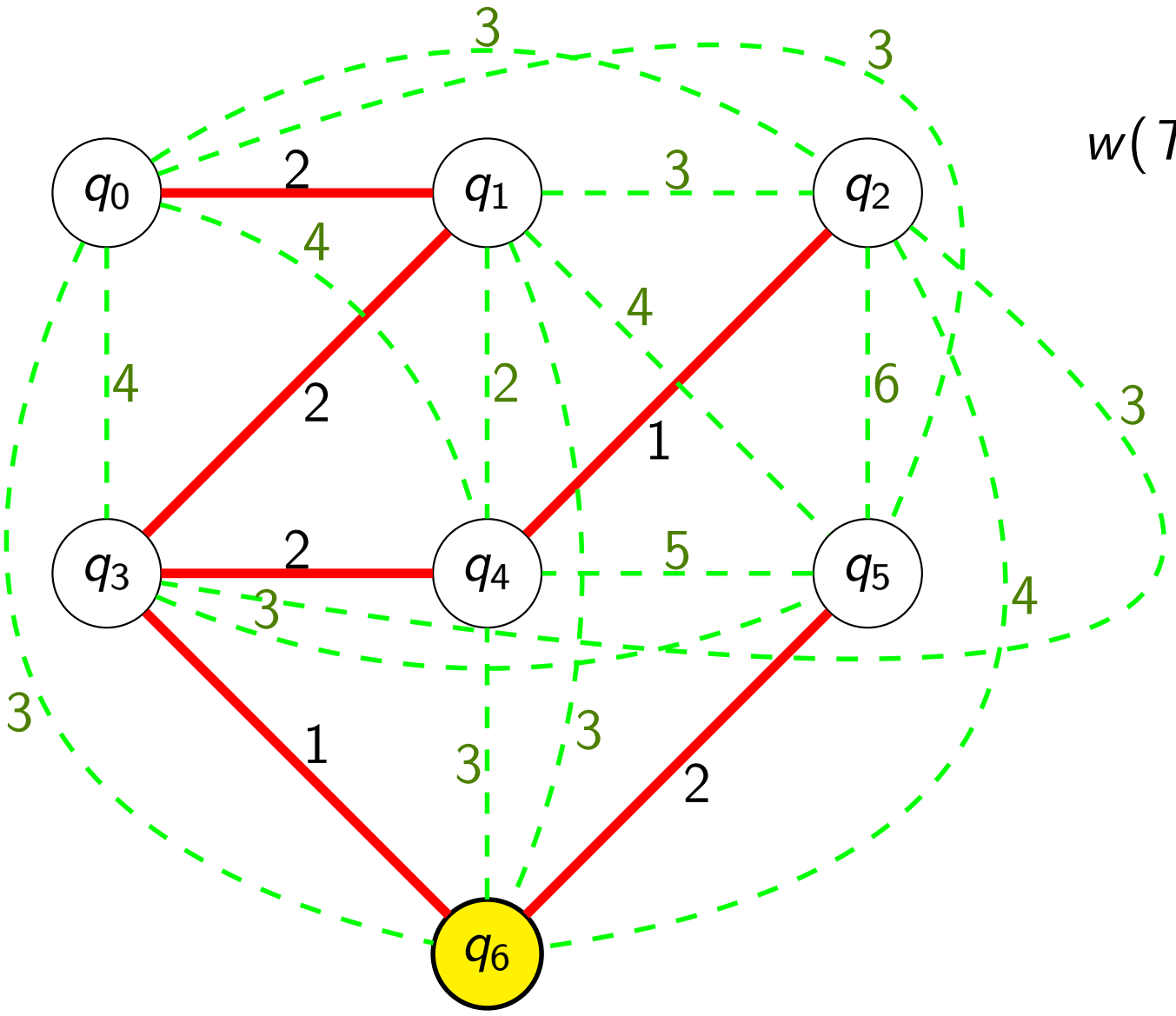


Exemple



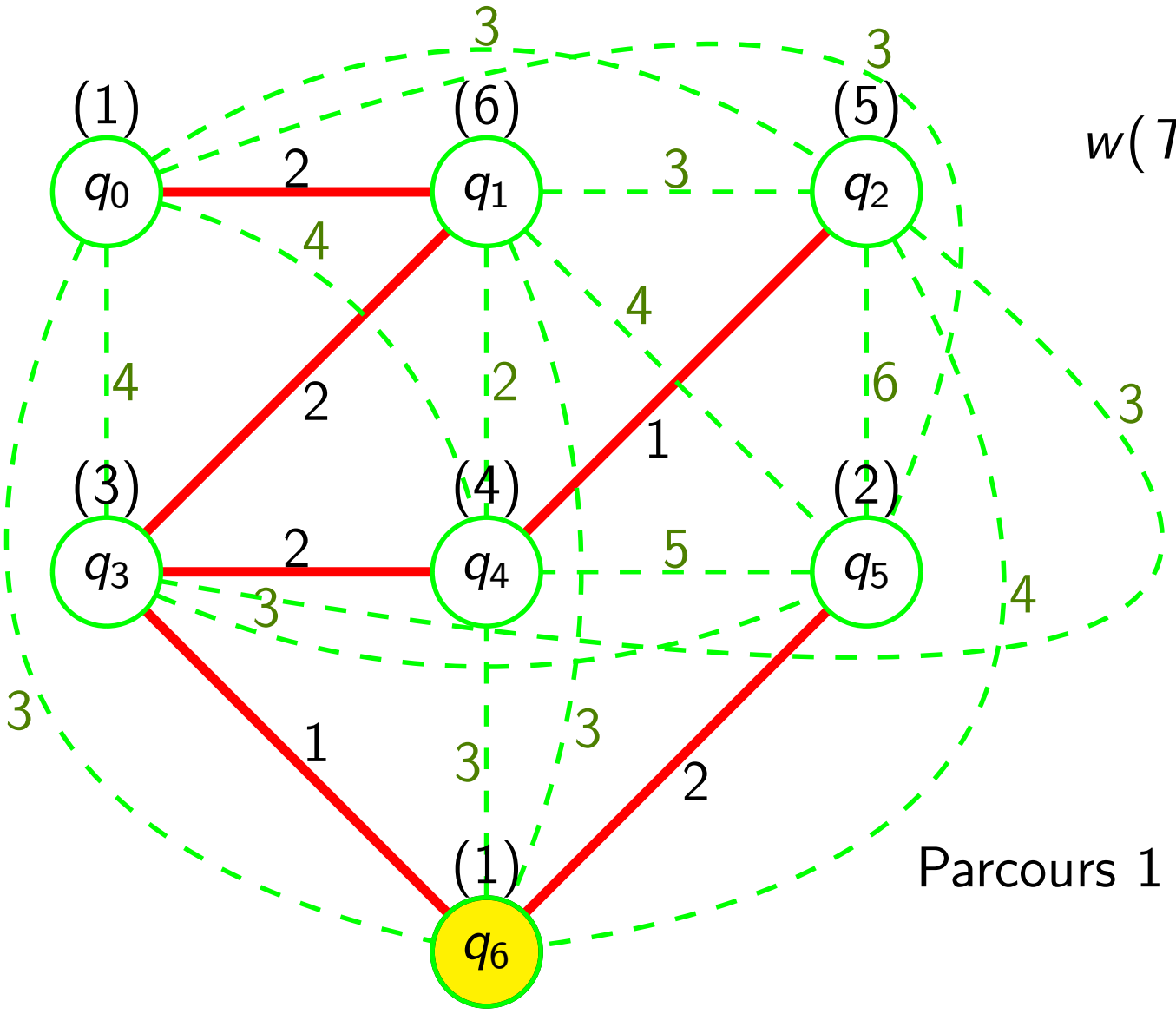
$w(T_{acm}) = 10$

Exemple



$w(T_{acm}) = 10$

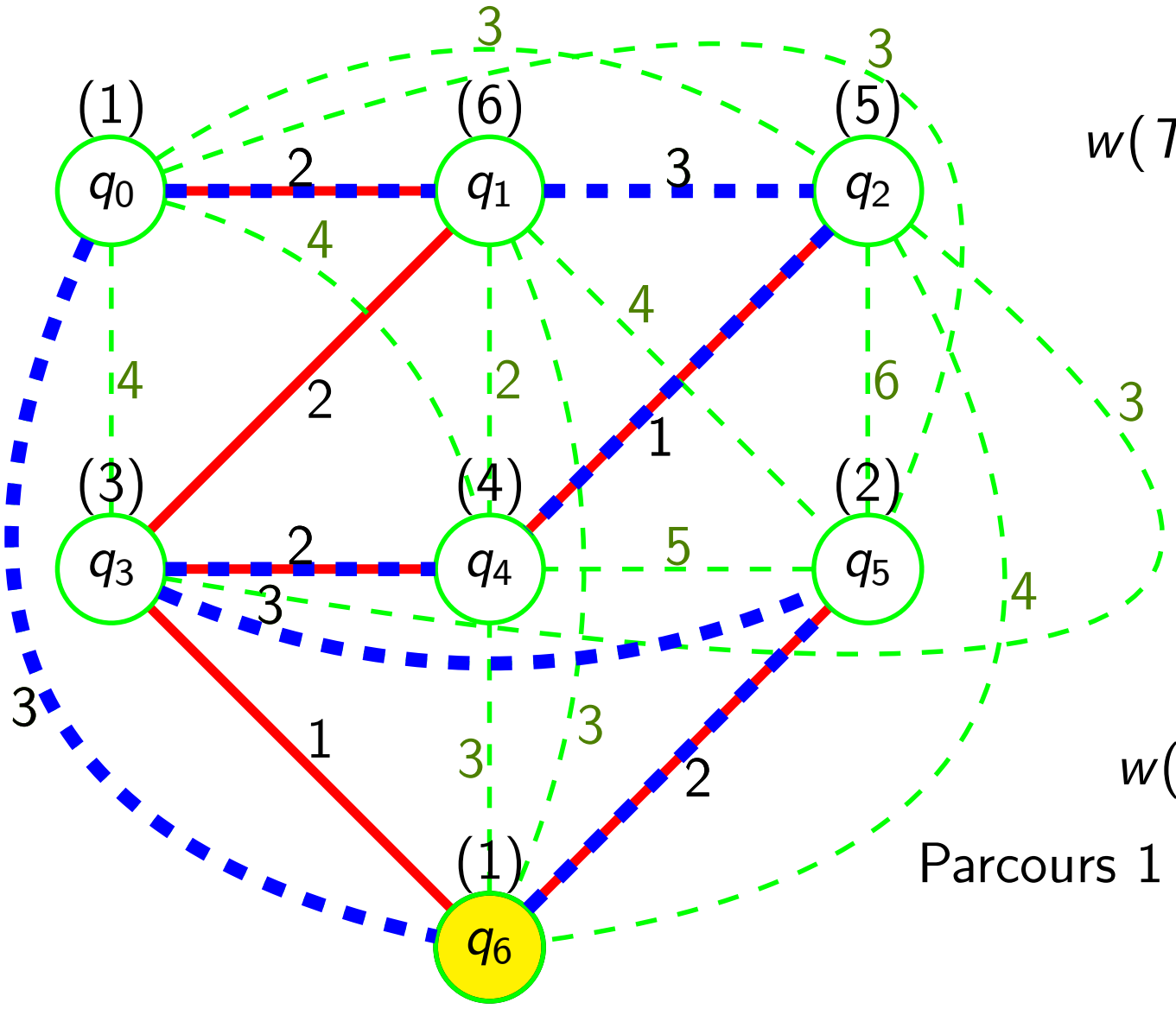
Exemple



$w(T_{acm}) = 10$

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

Exemple

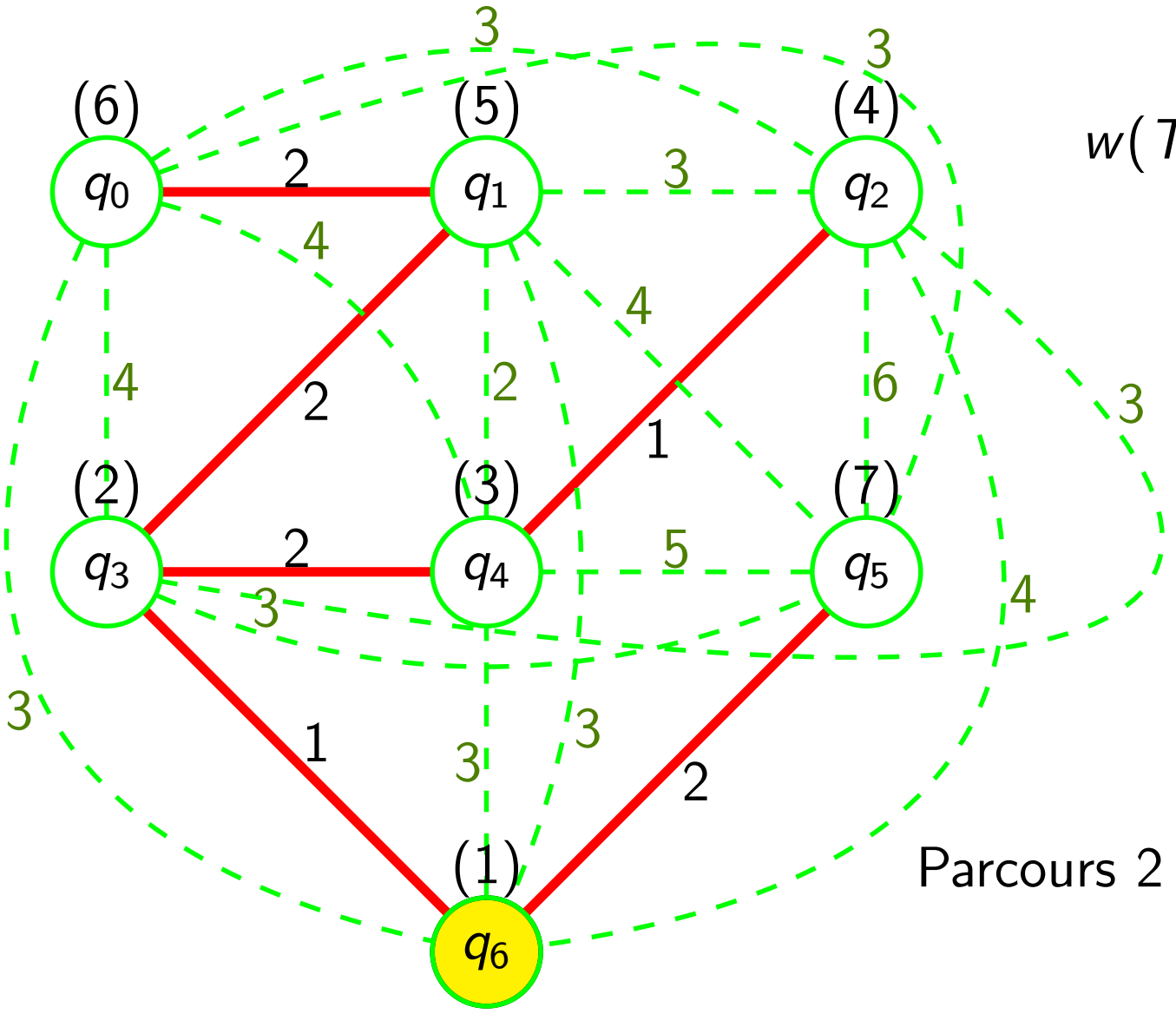


$w(T_{acm}) = 10$

$w(C) = 16$

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

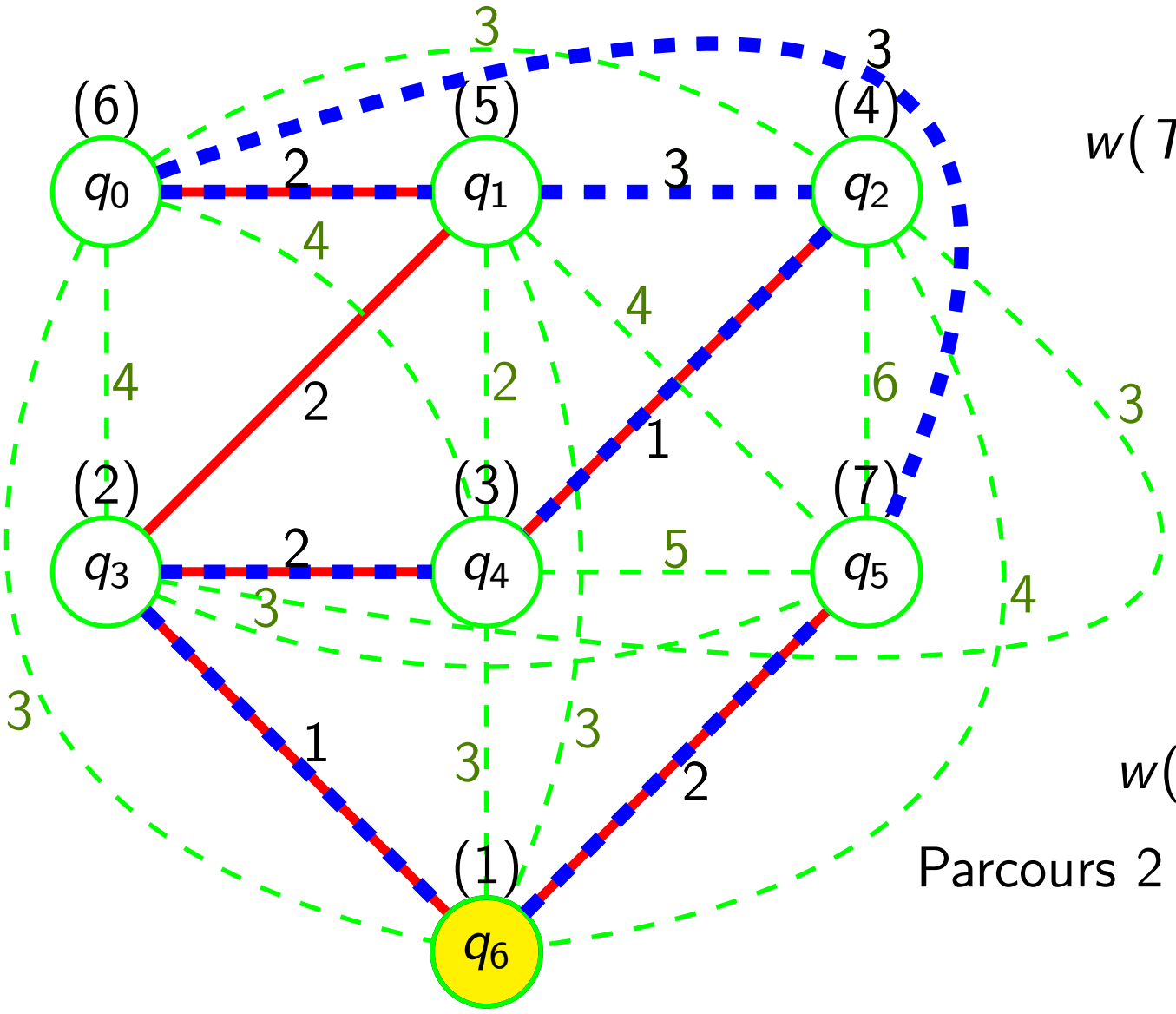
Exemple



$w(T_{acm}) = 10$

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.*

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

Plan

Conclusion

L'algorithmique est un domaine très vaste et au coeur de l'informatique...