

Algorithmes basés sur des parcours : algorithmes de Dijkstra et de Prim

- Plus courts chemins : algorithme de Dijkstra
- Arbre couvrant de coût minimum : algorithme de Prim

Chargés de cours : Fanny Pascual et Olivier Spanjaard.

Plus courts chemins
(chemins de coûts minimum)

Jussieu, Place Jussieu, Quartier Saint-Victor

Châtelet - Les Halles, Place Basse, Quartier

À pied (GraphHopper)

Aller

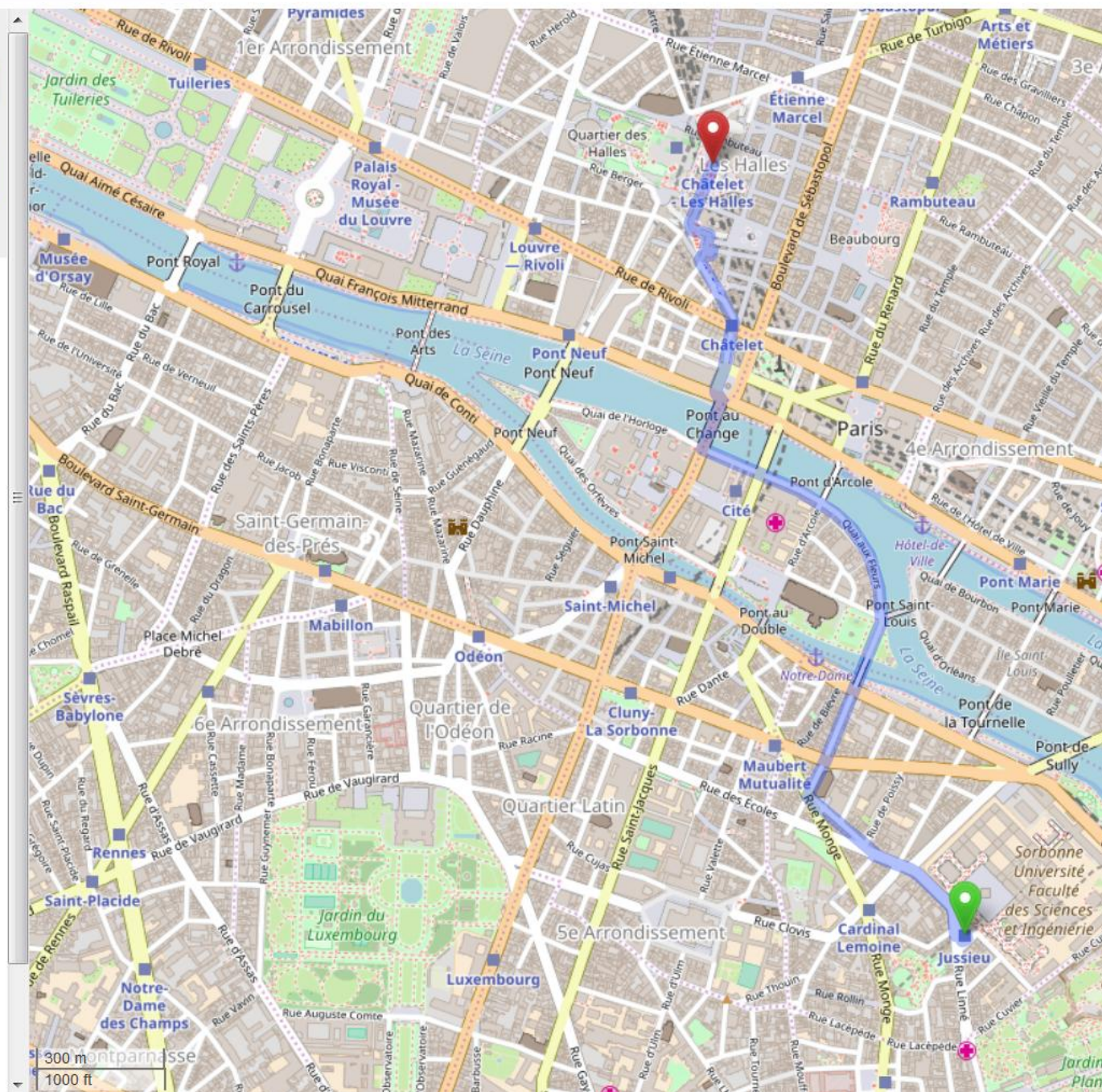
Inverser les directions

Itinéraire

Distance: 2.4km. Temps: 0:29.

Croissant: 40m. Décroissant: 43m.

1. Continuez 10m
2. Tournez à droite 20m
3. Tournez à gauche 10m
4. Tournez à droite sur Rue Linné 300m
5. Tournez à droite 200m
6. Tournez à droite sur Rue des Bernardins 1100m
7. Tournez à droite sur Boulevard du Palais 120m
8. Tournez à gauche sur Quai de la Mégisserie 10m
9. Restez sur la droite sur Place du Châtelet 200m
10. Restez sur la droite sur Rue des Halles 10m
11. Continuez sur Place Sainte-Opportune 30m
12. Continuez sur Rue des Halles 110m
13. Tournez à gauche sur Rue de la Ferronnerie 300m



Définition du problème

Soit $G=(S,A)$ un graphe orienté (n sommets, m arcs).

Soit $c : A \rightarrow \mathbb{R}$ une **fonction coût** sur les arcs.

Soient un sommet **origine** s et un sommet **destination** d .

Coût d'un chemin $l = (a_1, a_2, \dots, a_p)$, noté $c(l)$ de G : $\sum_{k=1..p} c(a_k)$.

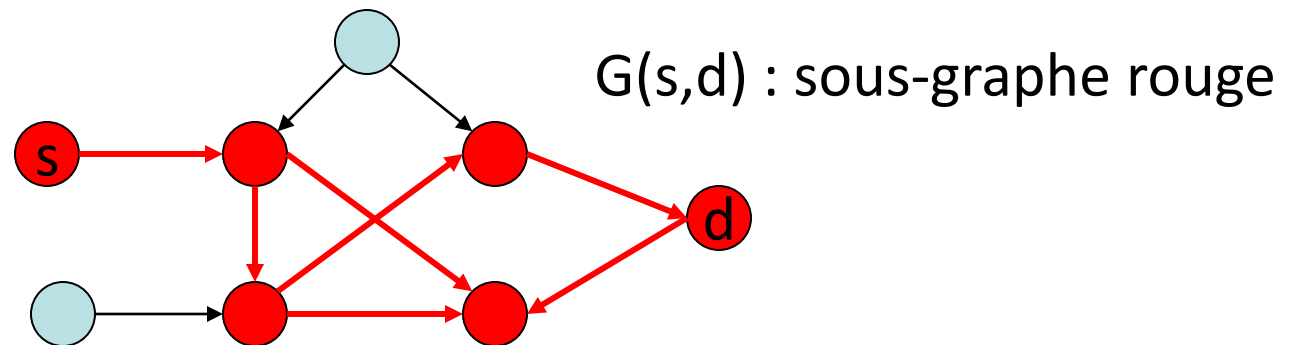
Etant donnés **2 sommets** s et d , on veut :

- a) savoir s'il **existe** un chemin de coût minimum de s à d ;
- b) si oui, **déterminer** un tel chemin.

Le sous-graphe $G(s,d)$

Les sommets x du graphe qui n'appartiennent pas à un chemin de s à d peuvent être supprimés de G .

$G(s,d)$ est le sous-graphe de G obtenu après suppression de ces sommets.

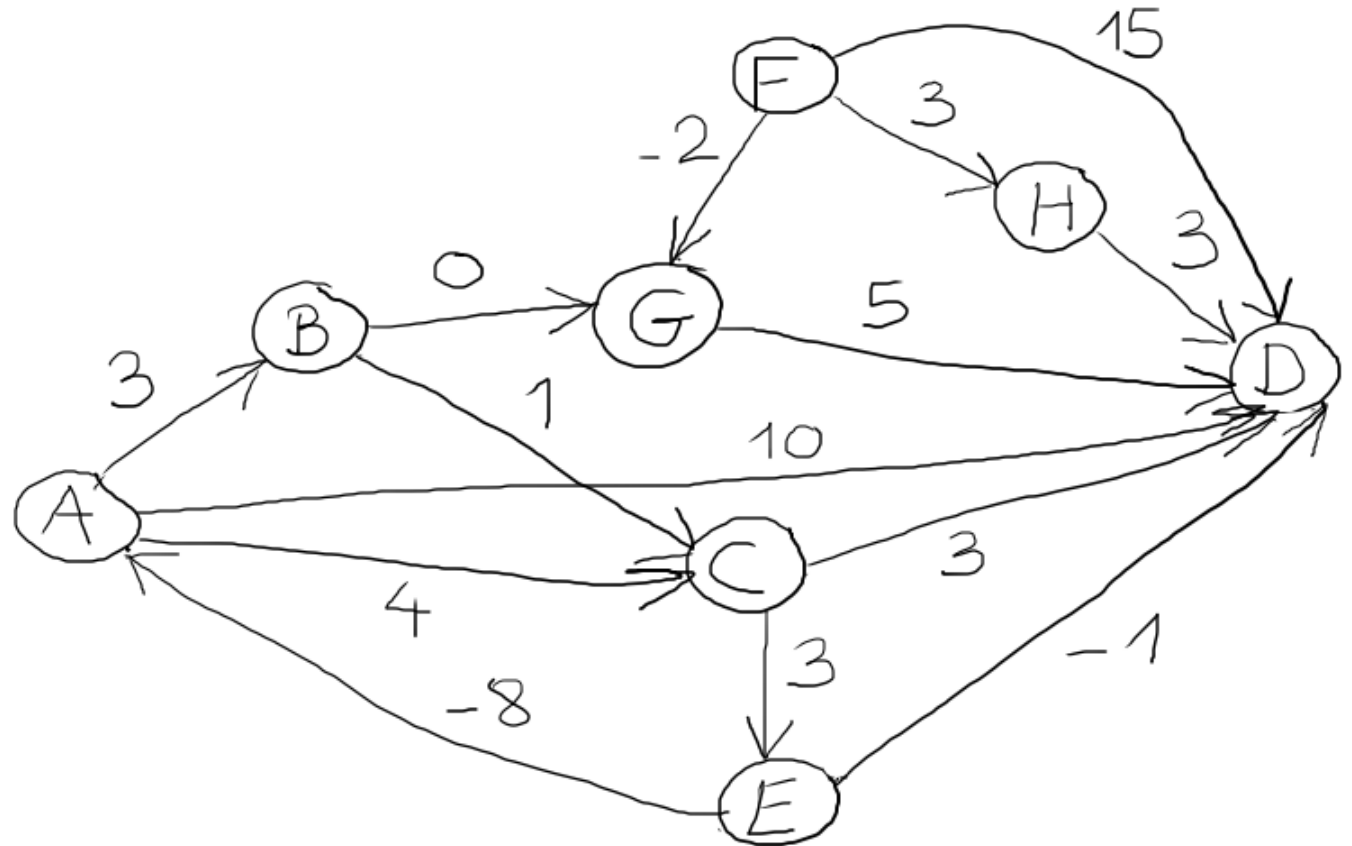


Propriété :

c chemin de s à d dans $G \Leftrightarrow c$ chemin de s à d dans $G(s,d)$

Quiz

Quel est le coût d'un chemin de coût minimum entre F et D dans G ?



A. 3

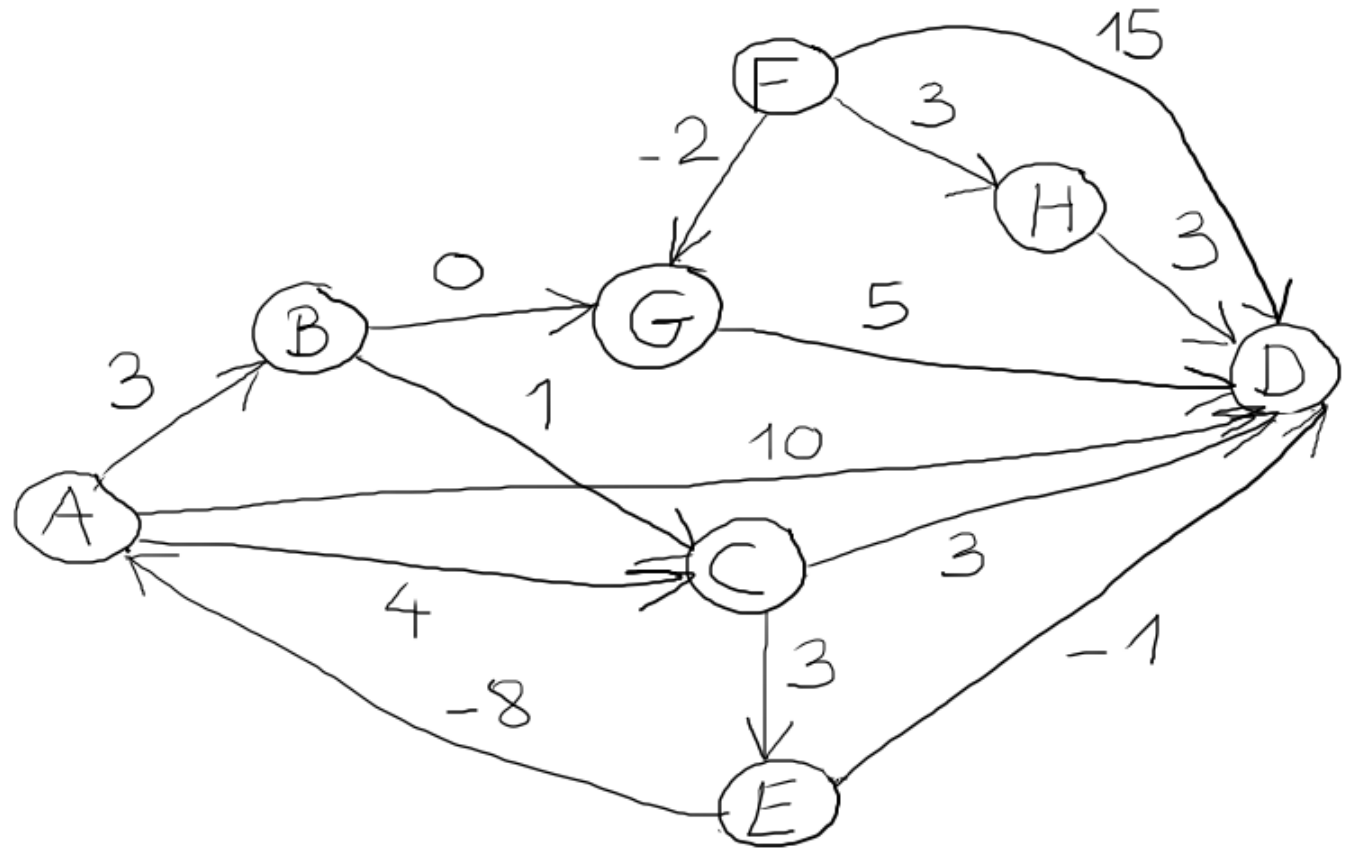
B. 6

C. 15

D. Il n'y a pas de chemin de coût minimum entre F et D.

Quiz

Quel est le coût d'un chemin de coût minimum entre A et D dans G ?



A. 6

B. 7

C. 10

D. Il n'y a pas de chemin de coût minimum entre A et D.

Définition :

Un **circuit** est dit **absorbant** si son coût est strictement négatif.

Propriété :

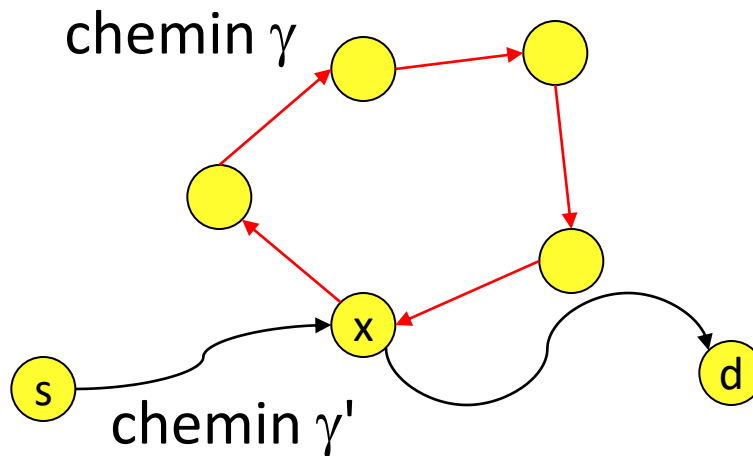
Il existe un **chemin de s à d de coût minimum** \Leftrightarrow

- il existe un **chemin** de s à d dans G.
- il n'existe **pas de circuit absorbant** dans $G(s,d)$.

Propriété :

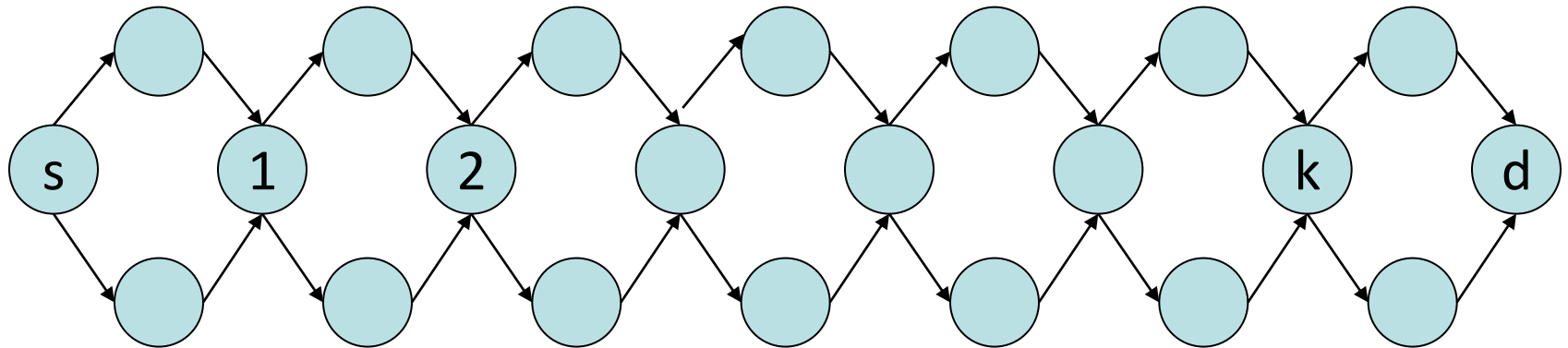
Un chemin de coût minimum de s à d est **élémentaire**.

Preuve : Si γ est un chemin de coût minimum de s à d ,
tout chemin élémentaire γ' extrait de γ satisfait $c(\gamma') \leq c(\gamma)$.



Un chemin élémentaire n'empruntant pas 2 fois le même arc,
le nombre de chemins de s à d à examiner est fini.

Remarque : il peut être très grand (exponentiel en n).



$$n = 3(k+1) + 1$$

Nombre de chemins de s à d : $2^{(k+1)} = 2^{((n-1)/3)}$.

Arborescence des chemins de coût minimum

Hypothèses :

- Le sommet **s** est une racine de G;
- G ne possède pas de circuit absorbant.



Propriété :

G possède une **arborescence couvrante H de racine s** telle que pour tout sommet x de G, le chemin de s à x dans H est un **chemin de coût minimum** de s à x dans G.

H est appelée **arborescence des chemins de coût minimum** d'origine s.

Arborescence des chemins de coût minimum

Soit H une **arborescence couvrante** de racine s dans G .

Soit $d(x)$ le coût du chemin de s à x dans H .

Propriété :

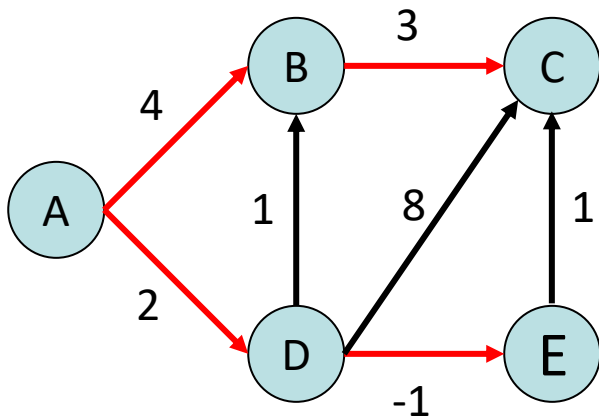
H est une **arborescence des chemins de coût minimum** pour G si et seulement si pour tout arc (x,y) de G on a :

$$d(y) \leq d(x) + c(x,y).$$

Cette propriété est à la **base de la plupart des algorithmes de calcul des chemins de coût minimum.**

Quiz

L'arborescence en rouge est une arborescence des chemins de coût minimum.

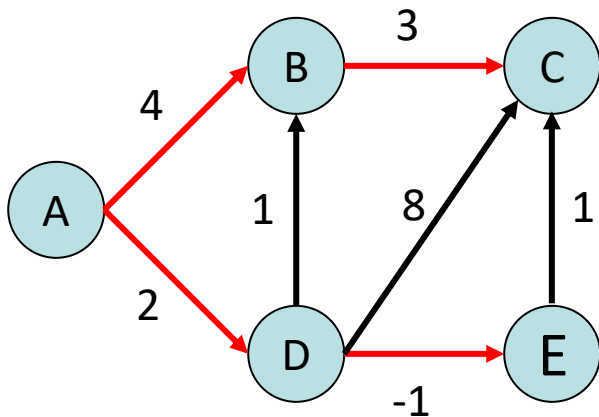


- A. Vrai
- B. Faux



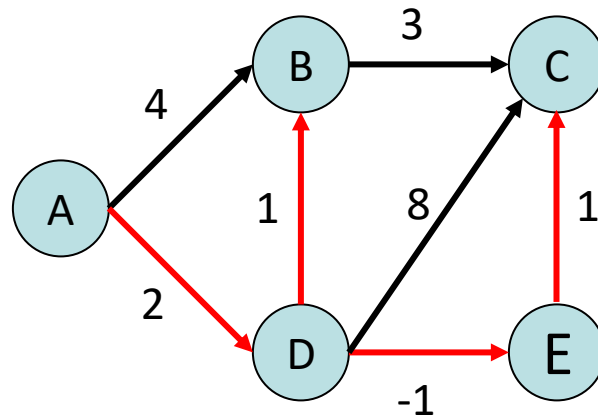
Quiz

L'arborescence en rouge est une arborescence des chemins de coût minimum.



A. Vrai

B. Faux :



Notations :

Soit $H=(X,U)$ l'arborescence « couvrante » (de racine s) en cours.

Si $x \in X$, le coût du chemin de s à x dans H est noté $d(x)$.

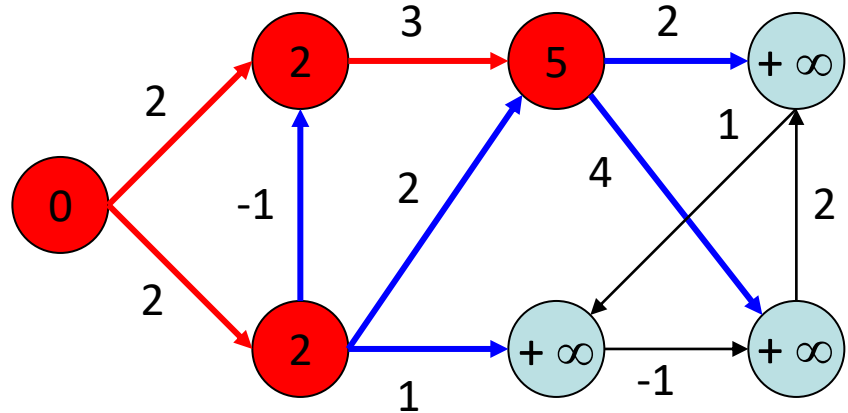
Si $x \notin X$, on fixe $d(x) = +\infty$

Un arc (x,y) de G , avec $x \in X$ est dit **incompatible** pour H si :
 $d(y) > d(x) + c(x,y)$

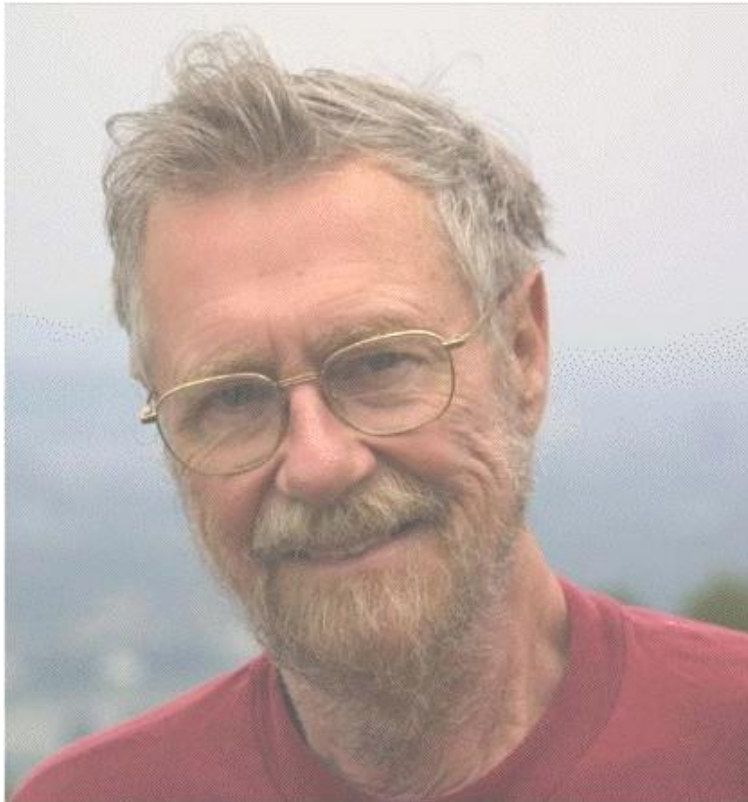
H : arborescence couvrante :

- sommets et arcs rouges
- valeurs $d(x)$ dans les sommets rouges

Arcs incompatibles en bleu



Coûts positifs ou nuls : algorithme de Dijkstra



Edsger Dijkstra (1930-2002)

Informaticien néerlandais.

1959 : publication de
« l'algorithme de Dijkstra ».

1972 : Prix Turing (science et art
des langages de
programmation, Algol).

Coûts positifs ou nuls : algorithme de Dijkstra

Donnée :

Graphe orienté $G=(S,A)$,

Sommet s **racine** de G ,

Fonction coût c sur les arcs telle que pour tout arc (x,y) : $c(x,y) \geq 0$.

Algorithme de Dijkstra :

Initialisation;

Pour k de 1 à n faire

 Soit x un **sommet ouvert** tel que $d(x)$ est minimum ;

Examiner(x);

FinPour.

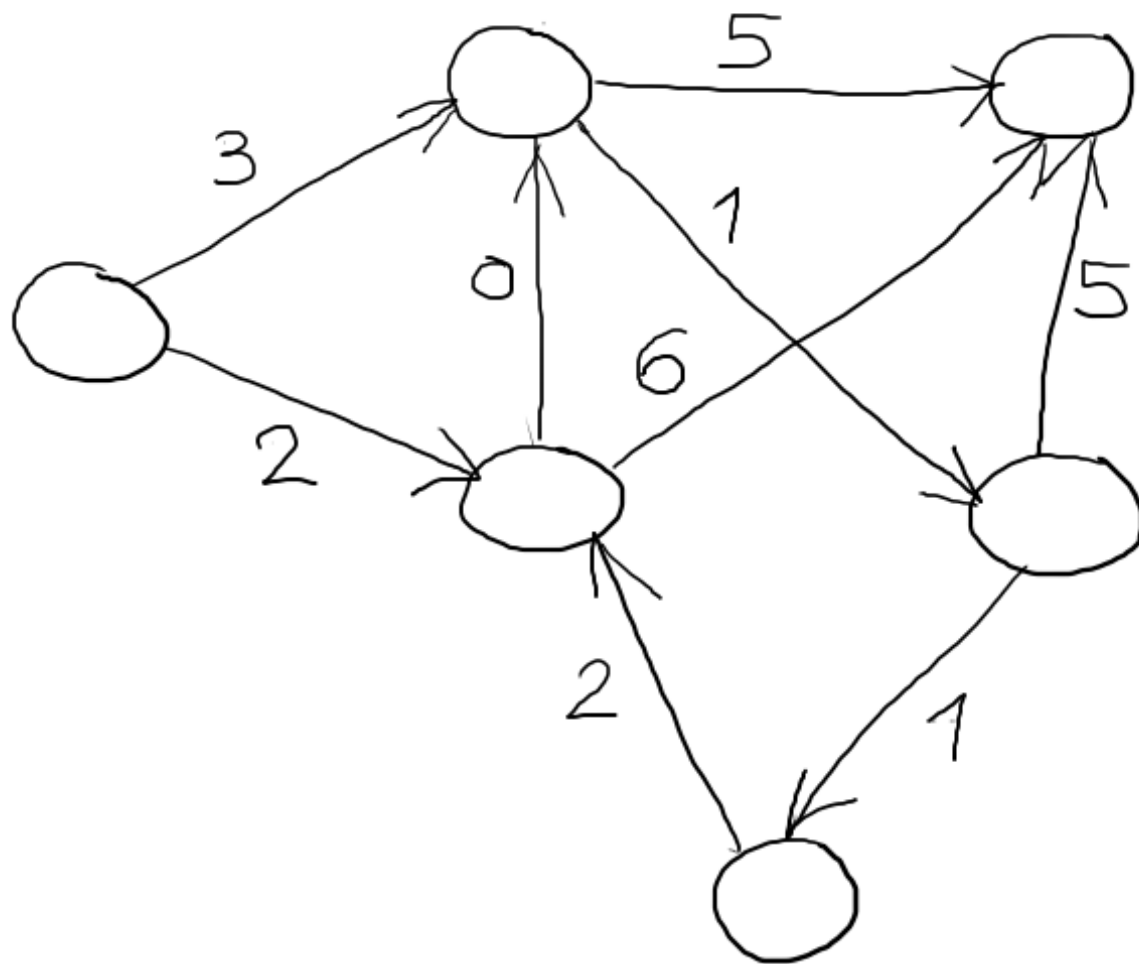
// $d(x)$ le coût du chemin de s à x dans H

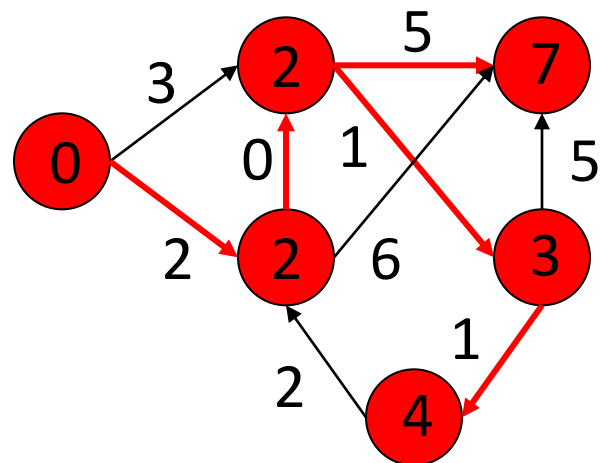
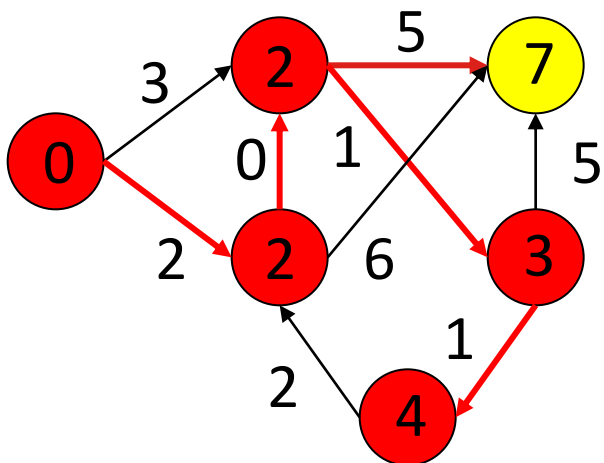
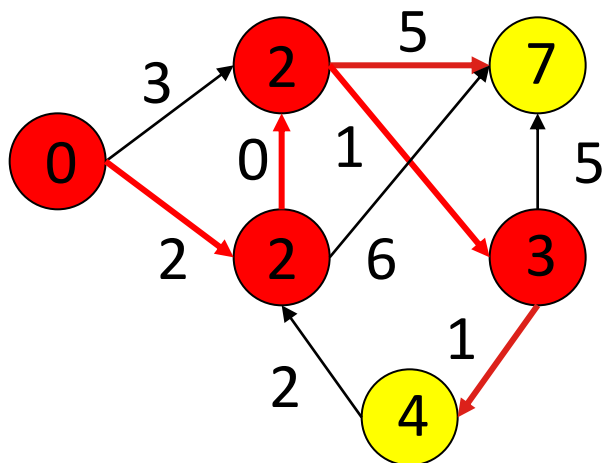
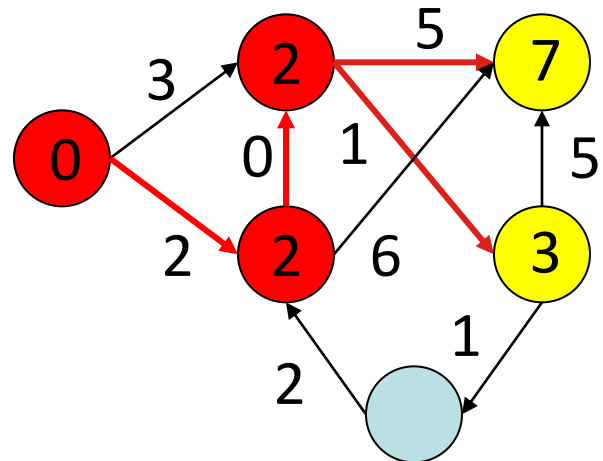
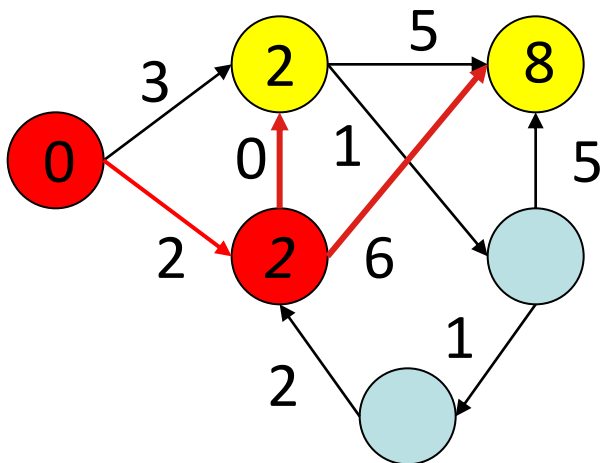
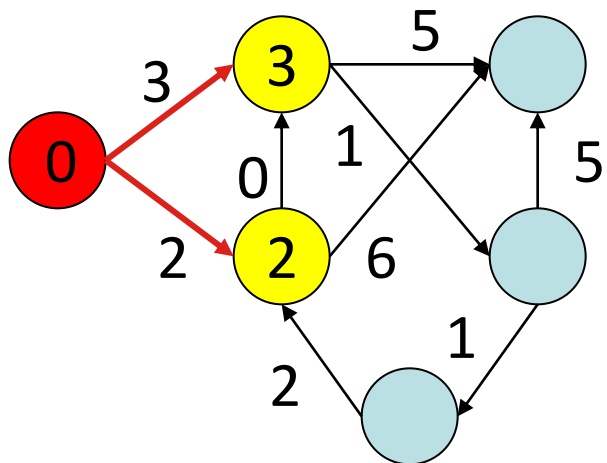
Initialisation

```
d(s1) = 0; ouvrir(s1);  
Pour tout k de 2 à n faire  
    d(sk) = + ∞ ;  
Fin Pour
```

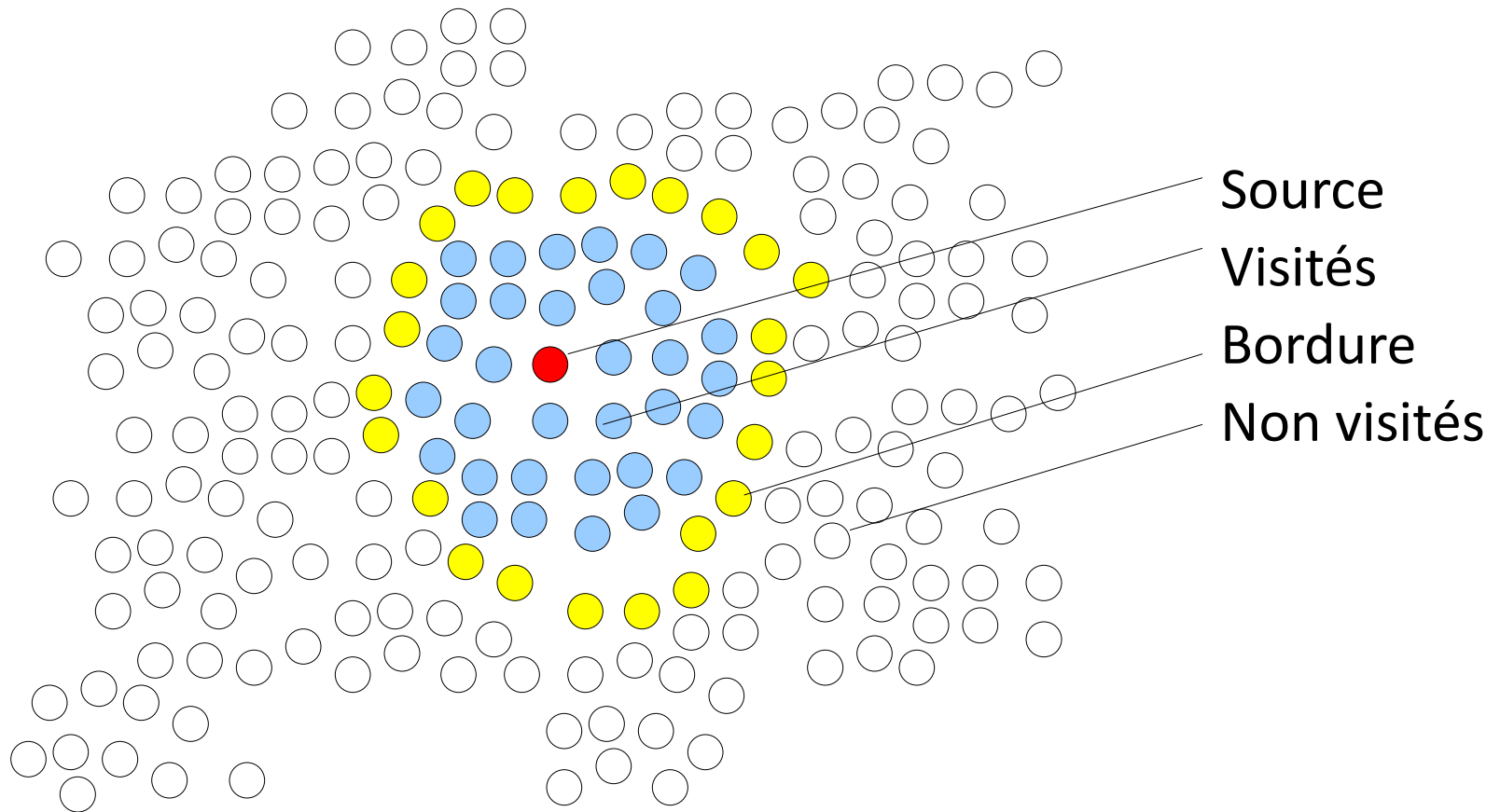
Examiner(x)

```
Pour tout successeur y de x faire  
    Si d(y) > d(x)+c(x,y) alors  
        d(y) = d(x)+c(x,y);  
        pred(y) = x;  
        Si y n'est pas ouvert, ouvrir(y)  
    FinSi  
FinSi  
FinPour  
fermer(x);
```



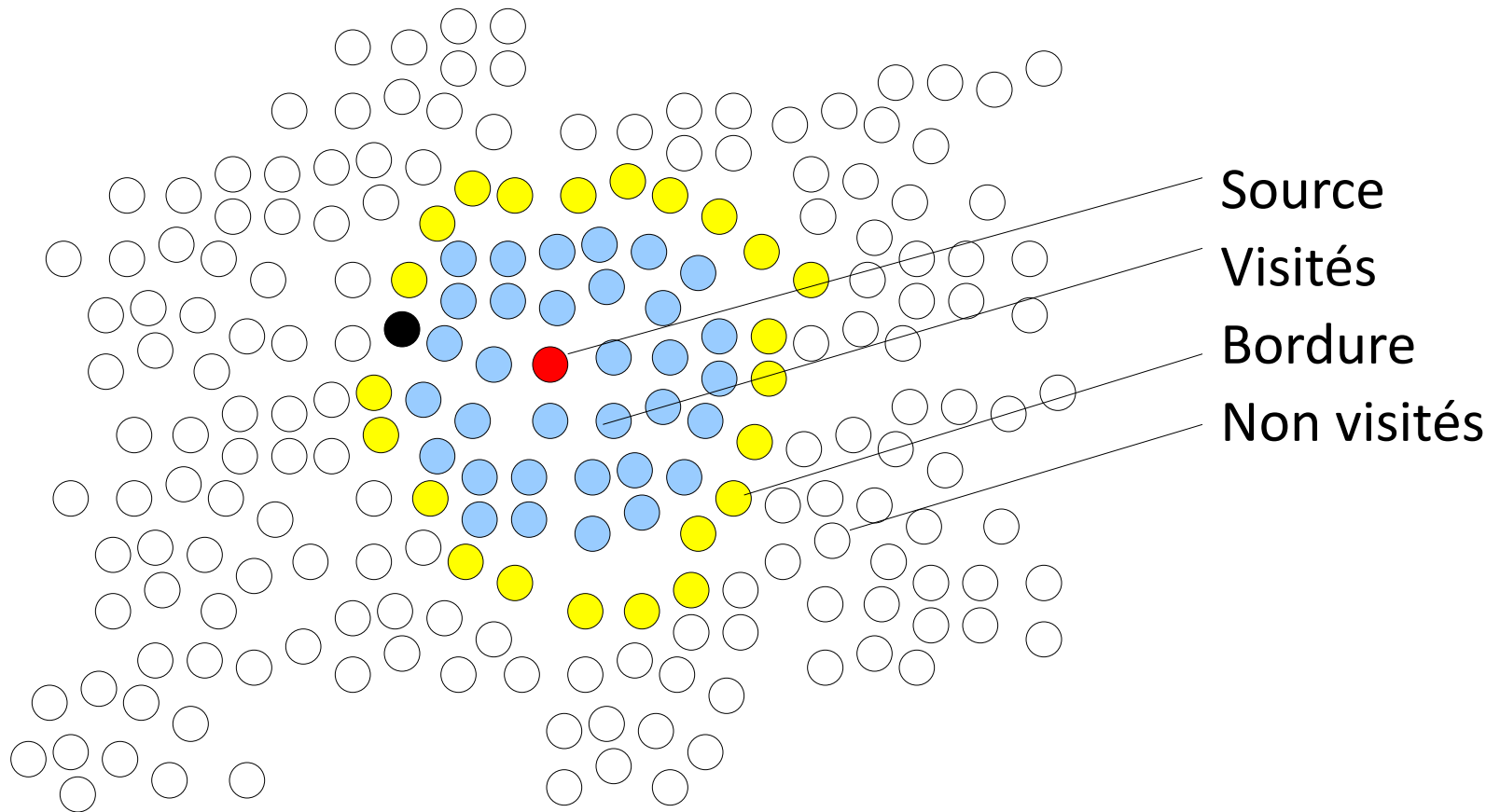


Interprétation physique



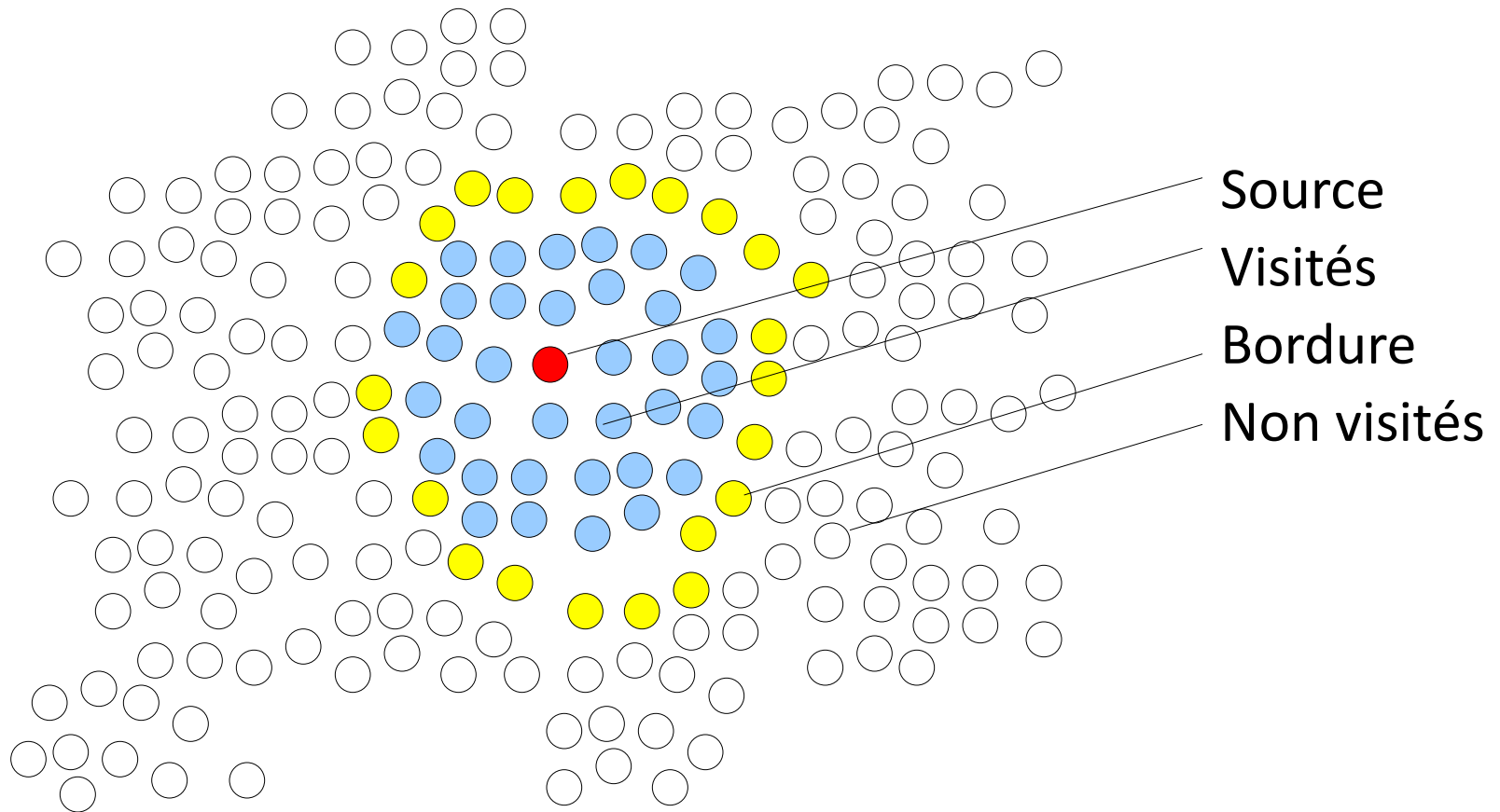
Front de propagation autour de la source = ronds dans l'eau

Interprétation physique



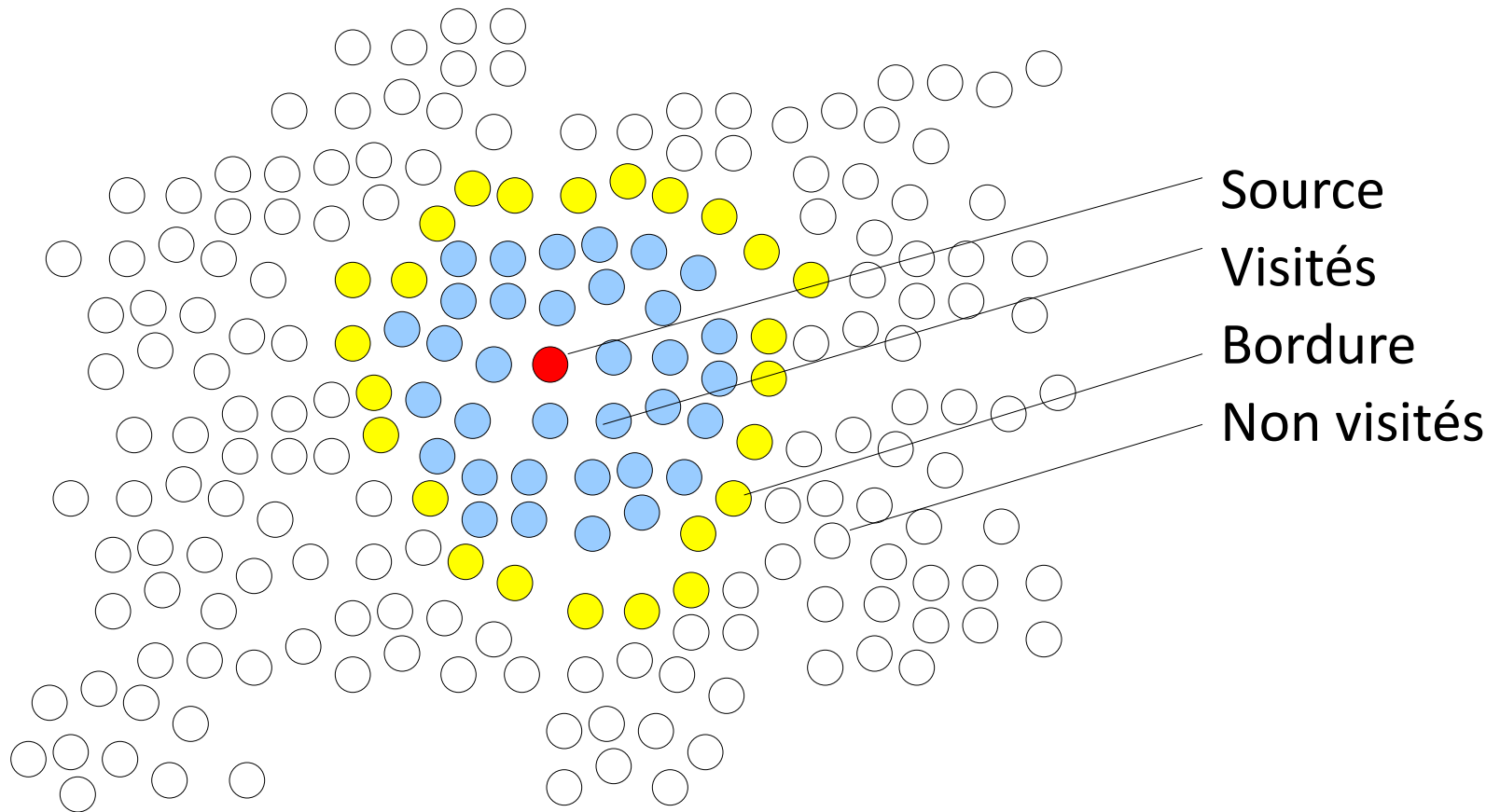
Front de propagation autour de la source = ronds dans l'eau

Interprétation physique



Front de propagation autour de la source = ronds dans l'eau

Interprétation physique



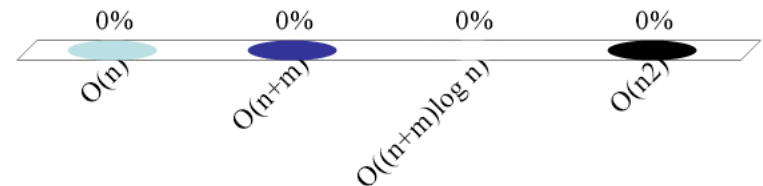
Front de propagation autour de la source = ronds dans l'eau

Quiz

Quelle est la complexité de l'algorithme de Dijkstra ?

```
Initialisation;  
Pour k de 1 à n faire  
    Soit x un sommet ouvert  
        tel que  $d(x)$  est minimum ;  
    Examiner(x)  
FinPour
```

- A. $O(n)$
- B. $O(n+m)$
- C. $O((n+m)\log n)$
- D. $O(n^2)$



Complexité de l'algorithme de Dijkstra.

Initialisation;

Pour k de 1 à n faire

 Soit x un sommet ouvert tel que $d(x)$ est minimum ;

 Examiner(x)

FinPour

- 1) Initialisation : $O(n)$
- 2) Recherche d'un sommet ouvert / d minimum : $O(n)$
- 3) Examiner(x) : $O(d^+(x))$

Complexité de l'algorithme : $O(n^2)$.

L'ensemble dynamique O des **sommets ouverts** peut être géré en utilisant une structure de données implémentant le **type de données abstrait TAS**.

On peut alors implémenter l'algorithme de Dijkstra avec une complexité de **$O((n+m)\log(n))$** .

Tas

Soit E un ensemble dynamique où chaque élément e est affecté d'une **priorité** $\text{prio}(e)$.

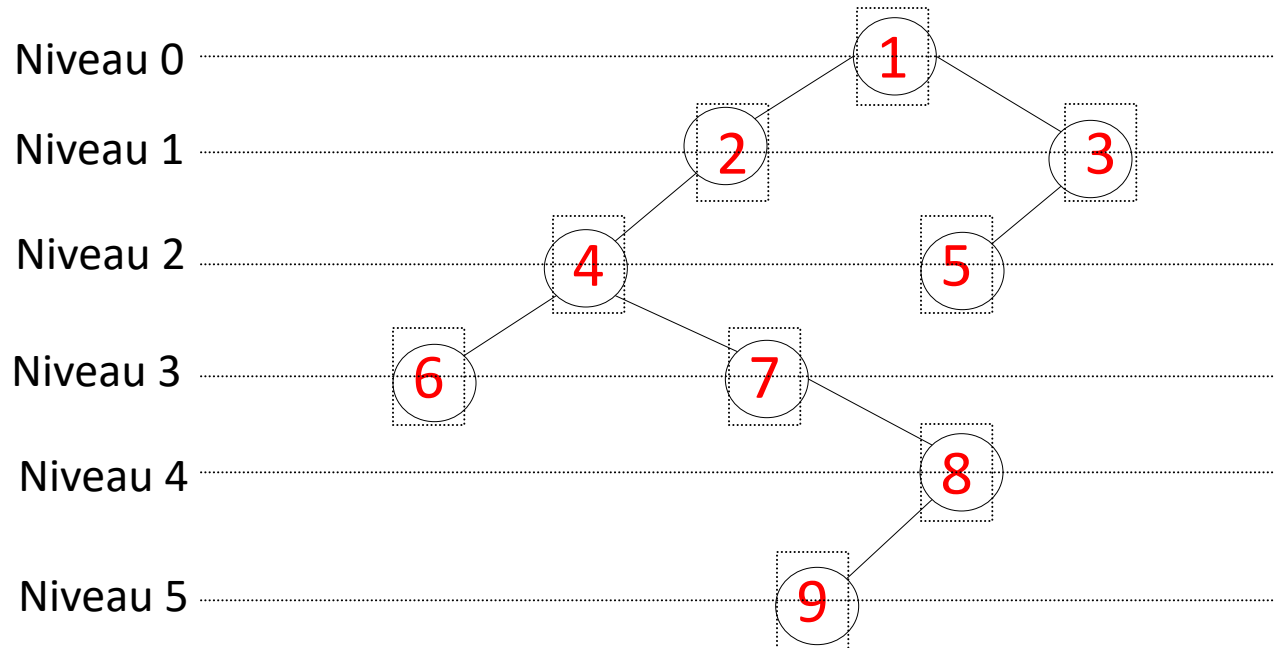
Opérations d'un tas

- **CréerTas(E)** crée un ensemble E vide
- **Insérer(e, p, E)** insère l'élément e de priorité p dans E
- **BaisserPriorité(e, p, E)** met la priorité de e à p dans E
- **SupprimerMin(E)** supprime dans E un élément de priorité minimum
- **Min(E)** renvoie un élément de priorité minimum de E

Numérotation hiérarchique des sommets d'un arbre binaire.

Ordre **croissant** des numéros :

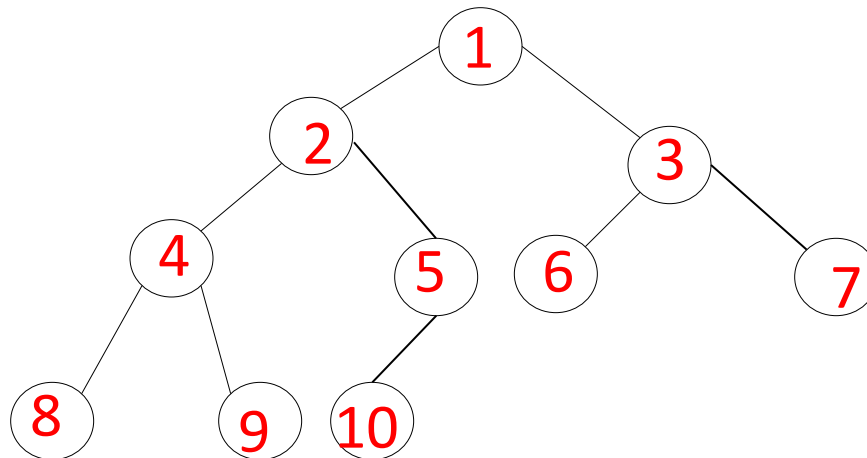
- 1) de **haut en bas** (par niveaux croissants)
- 2) de **gauche à droite**.



Arbre binaire parfait

Arbre tel que tous les niveaux sauf éventuellement le dernier sont remplis, et dans ce cas les feuilles du dernier niveau sont regroupées à gauche.

Exemple: L'arbre parfait à 10 sommets.



Arbre binaire parfait A de hauteur h :

- 1) les niveaux $0, 1, \dots, h-1$ de A sont complets (i.e : le niveau j contient 2^j sommets);
- 2) les j sommets du niveau h sont constitués:
 - si $j=2q$, des 2 fils des q premiers sommets du niveau $h-1$
 - si $j=2q+1$, des 2 fils des q premiers sommets du niveau $h-1$ du fils gauche du $(q+1)^{\text{ième}}$ sommet du niveau $h-1$.

Propriétés :

- 1) Il existe un seul arbre binaire parfait à n sommets (P_n)
- 2) La hauteur de (P_n) est $\lfloor \log_2 n \rfloor$

Tournoi

Un **tournoi** T pour (E, prio) est un arbre binaire sur E tel que :
pour tout sommet x distinct de la racine, $\text{prio}(\text{père}(x)) \leq \text{prio}(x)$.

Tas : **tournoi parfait**.

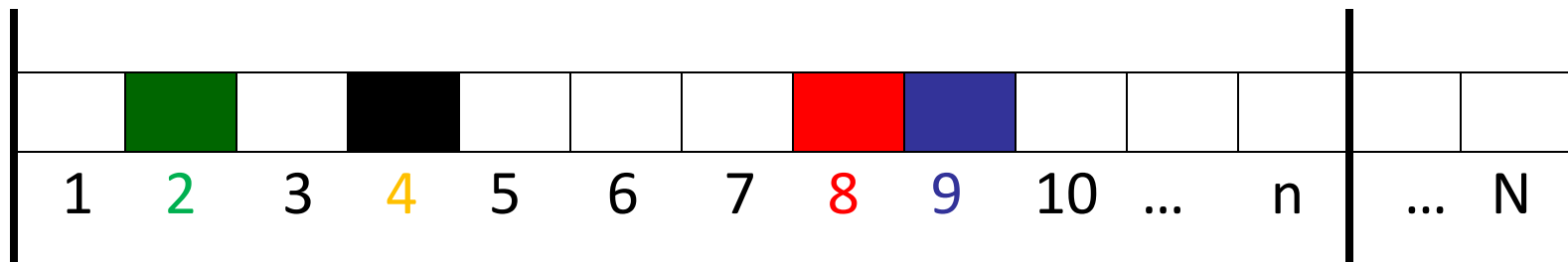
Propriétés

- 1) les **sous-arbres** $T(x)$ de T sont des **tas**;
- 2) **racine**(T) est un **sommet de priorité minimum**.
- 3) **Propriétés liées à la numérotation num**
 - $\text{num}(\text{fg}(x)) = 2 * \text{num}(x)$, $\text{num}(\text{fd}(x)) = 2 * \text{num}(x) + 1$,
 - $\text{num}(\text{père}(x)) = \text{num}(x) / 2$ (pour $x \neq \text{rac}(T)$)
 - $\text{EST_FEUILLE}(x) \Leftrightarrow 2 * \text{num}(x) > n$
 - $\text{num}(\text{DERNIERE_FEUILLE}(T)) = n$

Représentation d'un tas par un couple $(TAB[1..N], n)$

Valide si la constante N majore le nombre maximal d'éléments dans E

Les indices $\{1..n\}$ de TAB correspondent aux numéros des sommets de T .



sommet x

(indice 4 dans TAB):

fils gauche de x

(indice 8 dans TAB)

fils droit de x

(indice 9 dans TAB)

père de x

(indice 2 dans TAB)

Opérations sur un tas

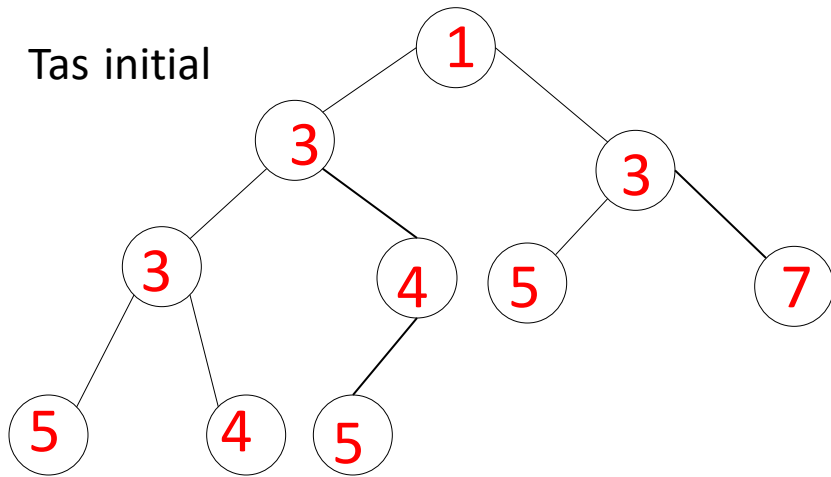
Insertion d'un élément de priorité k .
On utilise un tas T pour gérer (E, prio) .

```
Procédure Insérer( $e, k, T$ ) ;  
   $f := \text{Créer\_Dernière\_Feuille}(e, k, T)$ ;  
   $x := f$ ;  
  Tantque  $x \neq \text{rac}(T)$  et  $x.\text{prio} < \text{père}(x).\text{prio}$  faire  
    Echanger( $x, \text{père}(x)$ );  
     $x := \text{père}(x)$   
  FinTantque.
```

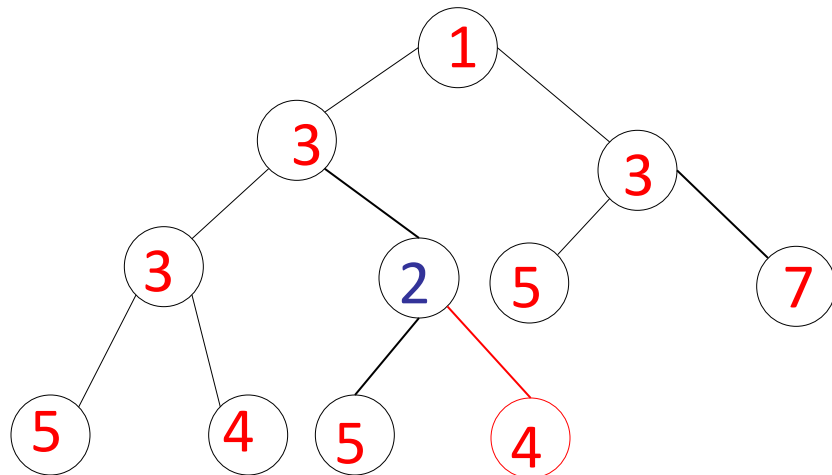
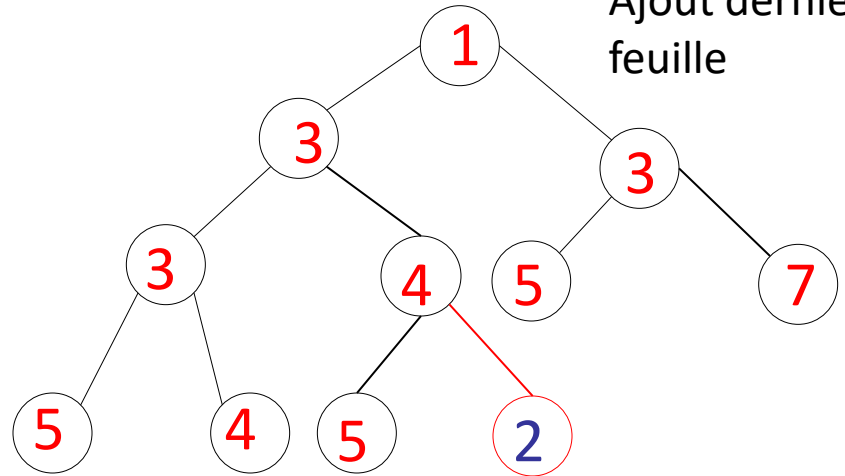
Complexité: $O(\log_2 n)$ (si la complexité de Echanger est $O(1)$).

Exemple: Insérer(e,2,T)

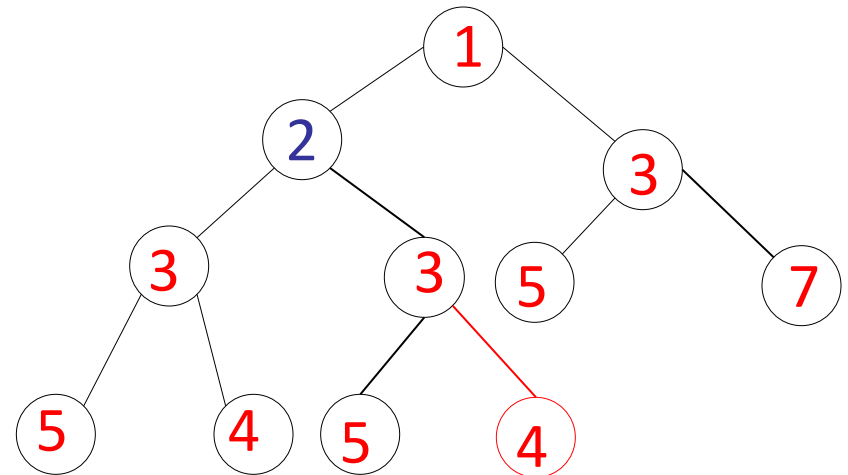
Tas initial



Ajout dernière
feuille



Remontée dernière feuille



Tas final

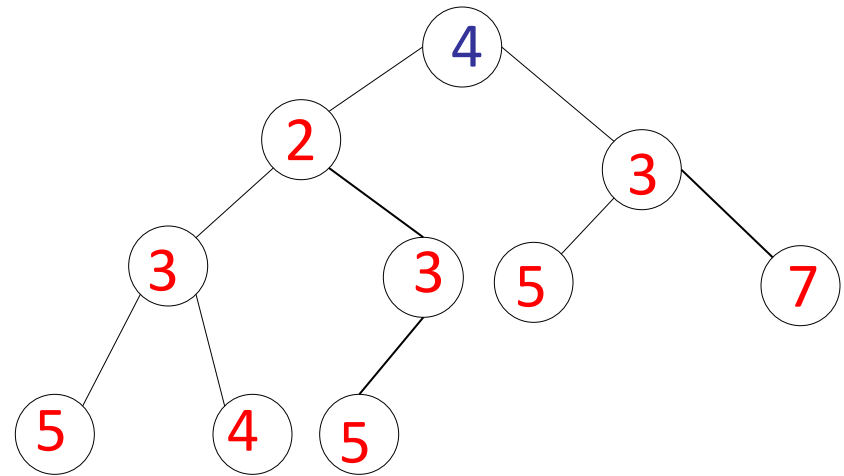
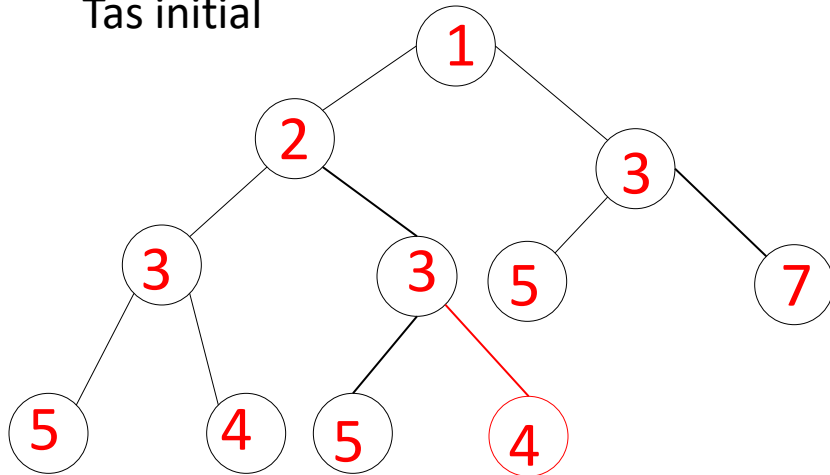
Suppression d'un élément de priorité minimale.

```
Procédure Supprimer_Min(T);  
Echanger(rac(T), Dernière_Feuille(T));  
Couper_Dernière_Feuille(T);  
x:=rac(T);  
Tantque (non Est_Feuille(x,T) et  
          x.prio>Filsmin(x).prio) faire  
    Echanger(x, Filsmin(x));  
    x:=Filsmin(x)  
Fintantque.
```

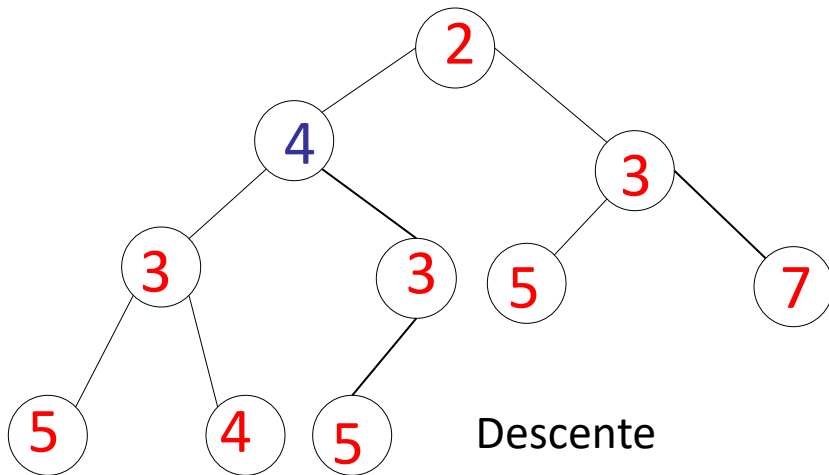
Complexité : $O(\log_2 (n))$

Exemple: Supprimer_Min(T)

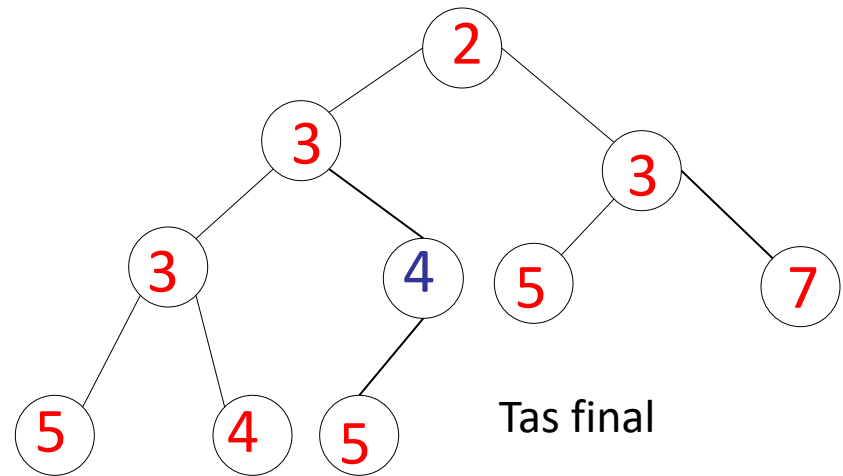
Tas initial



Echange racine-dernière feuille
Couper dernière feuille



Descente



Tas final

Baisser la priorité d'un élément
(en ayant un pointeur sur cet élément)

La priorité de l'élément e passe à p :

```
Procédure Baisser_Priorité (T,e,p);  
e.Prio :=p;  
x:=e ;  
Tantque x≠rac(T) et x.prio<père(x).prio faire  
    Echanger(x,père(x));  
    x:=père(x)  
Fintantque.
```

Complexité : $O(\log_2 (n))$

Gestion de l'ensemble des sommets ouverts dans l'algorithme de Dijkstra.

L'ensemble dynamique est O ;

La **priorité** d'un sommet ouvert x est $d(x)$: coût du chemin de s à x dans l'arborescence H courante.

Soient à la fin de l'itération k :

F l'ensemble des sommets fermés,

O l'ensemble des sommets ouverts,

Un sommet ouvert z de **priorité minimale** est à la **racine du tas**.
Donc, pour calculer z , complexité $O(1)$

Gestion de l'ensemble des sommets ouverts dans l'algorithme de Dijkstra.

Algorithme de Dijkstra :

$d(s_1) = 0$; ouvrir(s_1);

Pour tout k de 2 à n faire

$d(s_k) = +\infty$;

Fin Pour

Pour k de 1 à n faire

Soit x un sommet ouvert tel que $d(x)$ est minimum ;

Examiner(x);

FinPour.

Examiner(x)

Pour tout successeur y de x faire

Si $d(y) > d(x) + c(x, y)$ alors

$d(y) = d(x) + c(x, y);$

$\text{pred}(y) = x;$

Si y n'est pas ouvert, ouvrir(y)

FinSi

FinSi

FinPour

fermer(x);

Il faut **insérer** dans O chaque **successeur y de z non couvert** (c'est-à-dire ni dans O ni dans F) avec la priorité $d(z) + c(z,y)$.

Complexité : $O(\log(n))$ par successeur

Pour chaque successeur y de z tel que **y est ouvert et (z,y) est incompatible**, il faut **remplacer la priorité** de y par la nouvelle évaluation de y : c'est-à-dire $d(z) + c(z,y)$.

Complexité : $O(\log(n))$ par successeur

Il faut **supprimer z** du tas :

Complexité : $O(\log(n))$ par sommet supprimé.

Il en résulte que si les sommets sont examinés dans l'ordre (z_1, z_2, \dots, z_n) , le nombre total d'opérations de mise à jour du tas est majoré par :

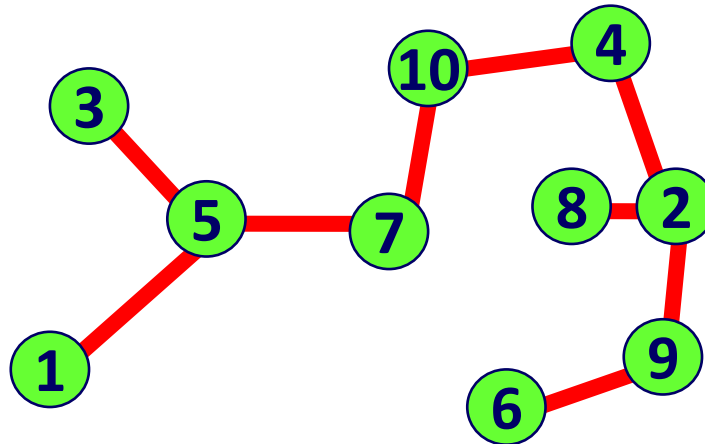
$$C \log(n)(n + d^+(z_1) + \dots + d^+(z_n)) = C(n+m)\log(n).$$

Complexité globale : **$O((n+m)\log(n))$**

Arbre couvrant de coût minimum

Application : conception de réseau

Une entreprise de télécommunication doit **relier un ensemble de clients**. Le coût de connection de deux clients est connu. But : relier les clients à **un coût minimal**.



Nombreuses applications :

- conception de réseau (hydraulique, électrique, communication...)
- utilisé pour d'autres problèmes : clustering, imagerie, relaxation de problèmes difficiles (arbre de Steiner, voyageur de commerce), etc.

Définition du problème

Donnée :

Un graphe non orienté **connexe** $G=(S,A)$ (n sommets, m arêtes)
Une fonction **coût** $c : A \rightarrow \mathbb{R}$ (valuation des arêtes)

Rappel :

Un arbre couvrant de G est un **graphe partiel** de G qui est un arbre.
(On identifiera un arbre couvrant à son ensemble d'arêtes)

Définition :

Soit H un arbre couvrant, le coût de H , noté aussi $c(H)$ est la somme $\sum_{e \in H} c(e)$ des coûts de ses arêtes.

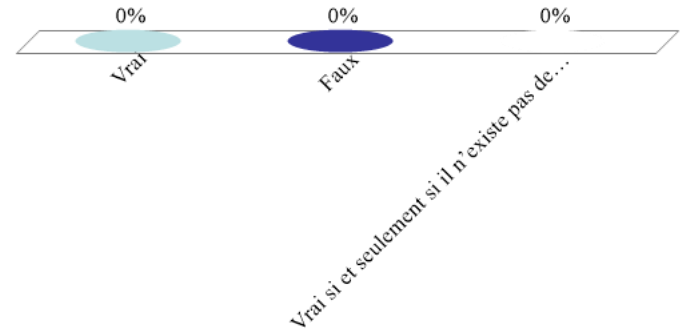
Problème :

Déterminer un **arbre couvrant** H^* de G de **coût minimum**.

Quiz

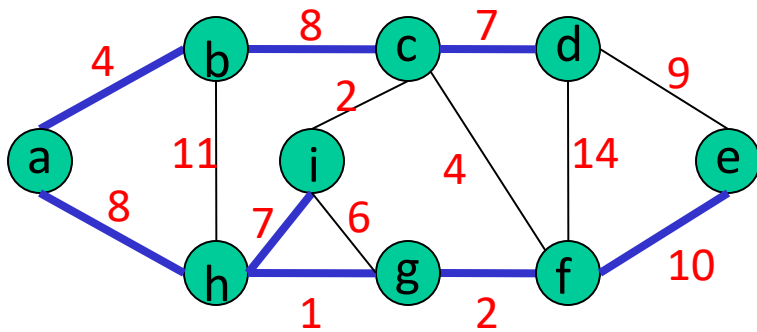
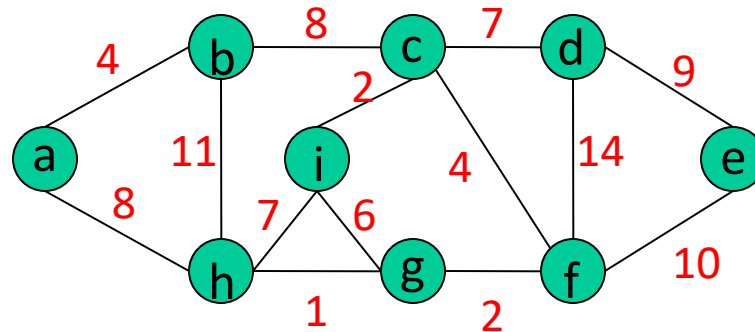
Soit G un graphe. Il existe toujours un arbre couvrant de coût minimum de G .

- A. Vrai
- B. Faux
- C. Vrai si et seulement si il n'existe pas de circuit absorbant dans G .

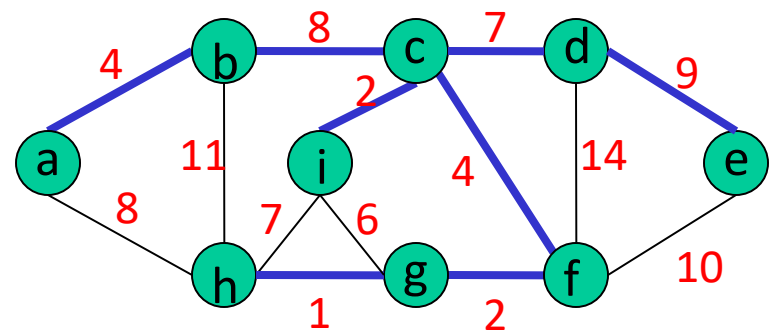


Exemple

Graphe G :



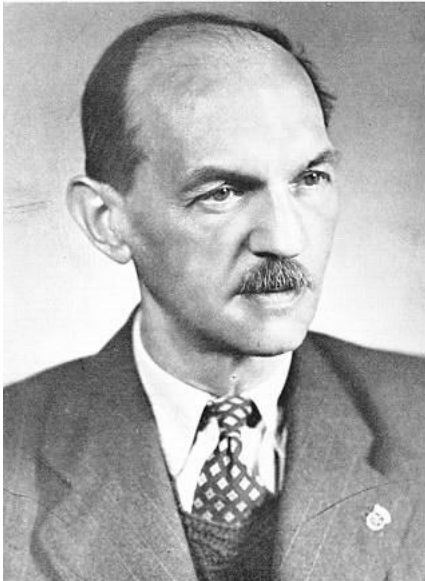
Arbre couvrant de coût = 47



Arbre couvrant de coût = 37

Algorithme de Prim

Algorithme découvert par Jarník en 1930, puis par Prim en 1957.
algorithme aussi dit : de Jarník-Prim, ou de Jarník-Prim-Dijkstra.



Vojtech Jarník (1897-1970)
mathématicien tchèque.



Robert C. Prim (1921)
mathématicien et informaticien
américain (laboratoires Bell).

Algorithme de Prim

Algorithme de Prim :

$d(s_1) = 0$; ouvrir(s1);

Pour tout k de 2 à n faire $d(s_k) = +\infty$;

FinPour

Pour k de 1 à n faire

Soit x un sommet ouvert tel que $d(x)$ est minimum ;

Examiner(x);

FinPour.

Examiner(x)

Pour tout voisin y de x faire

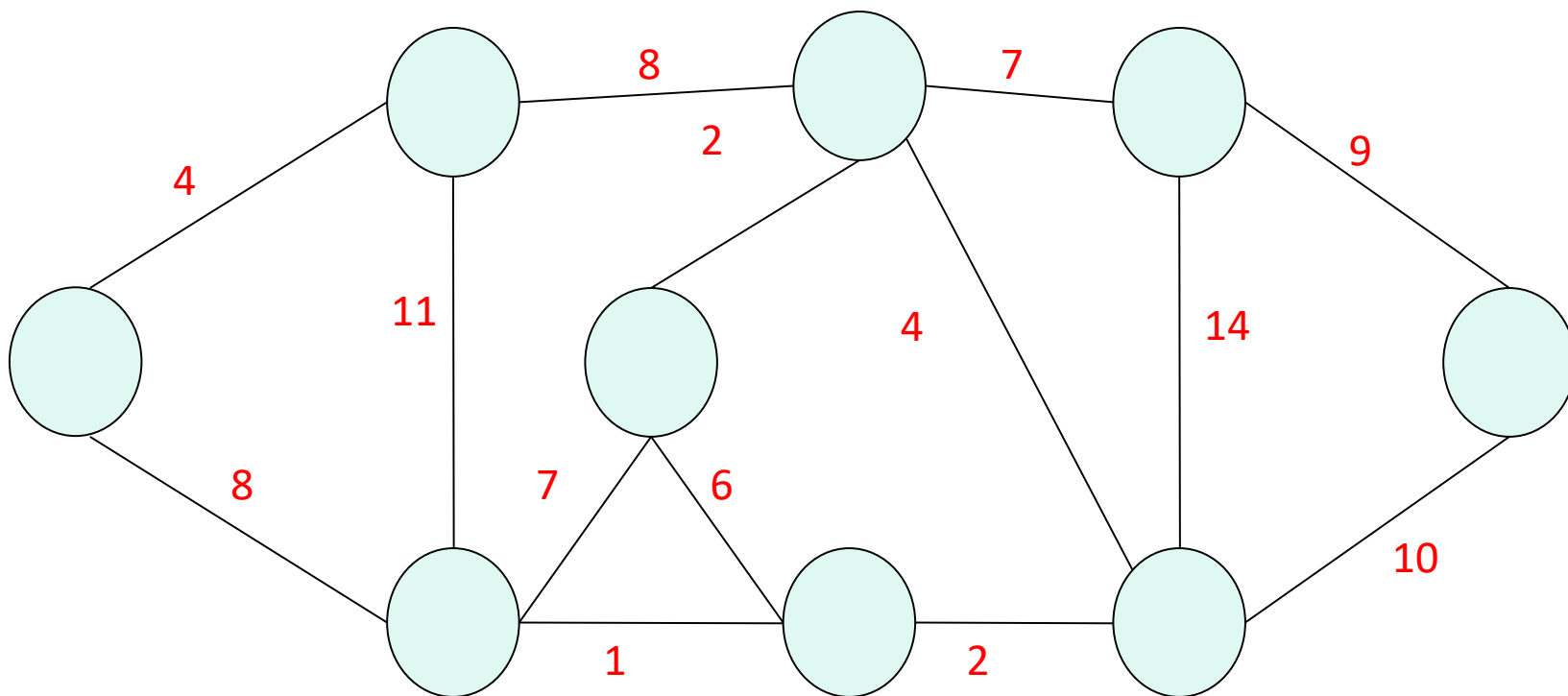
Si $d(y) > c(x,y)$ alors

$d(y) = c(x,y)$; père(y)=x; Si y n'est pas ouvert ouvrir(y);

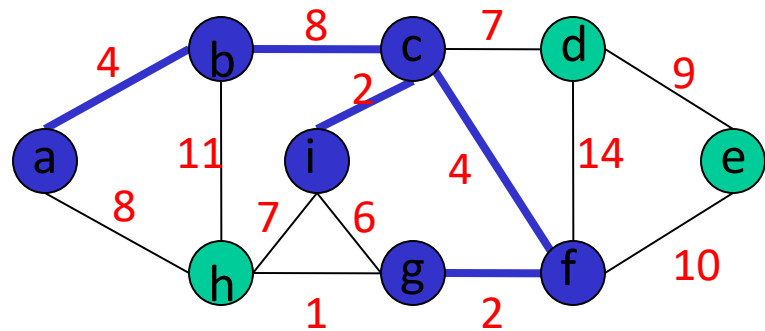
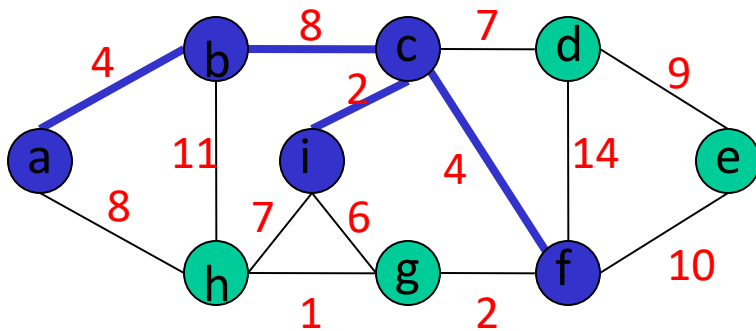
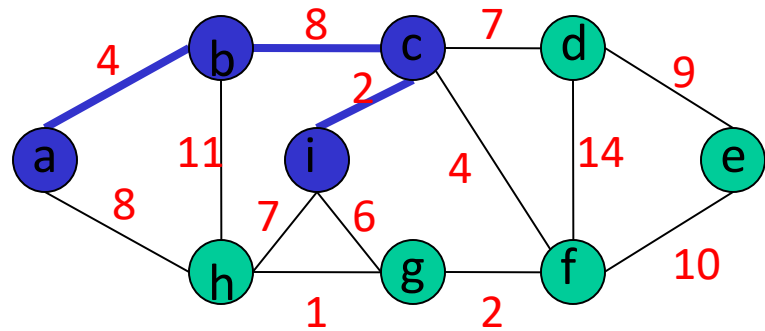
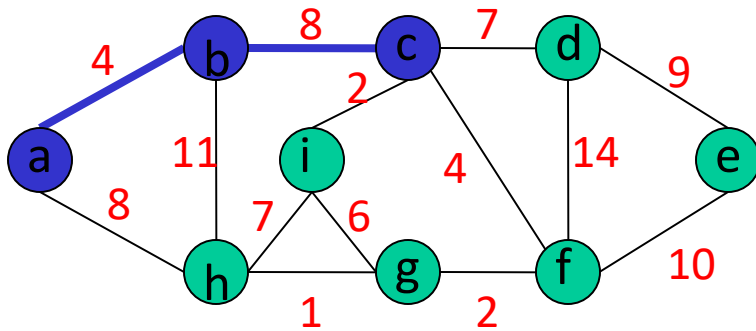
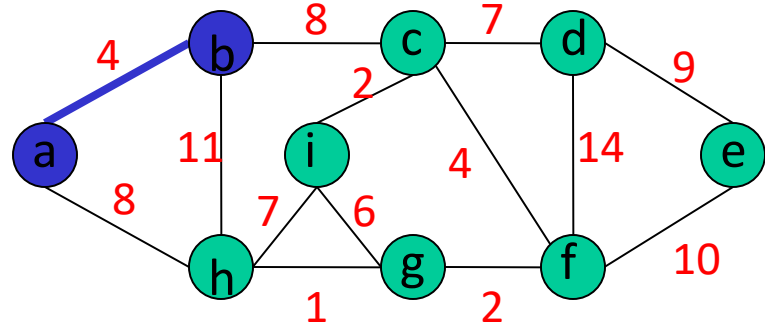
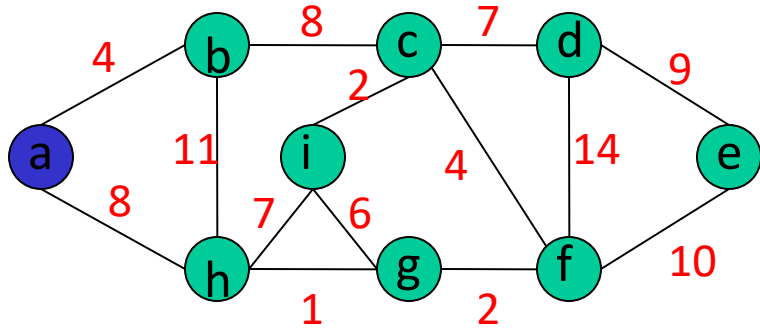
FinSi

FinPour

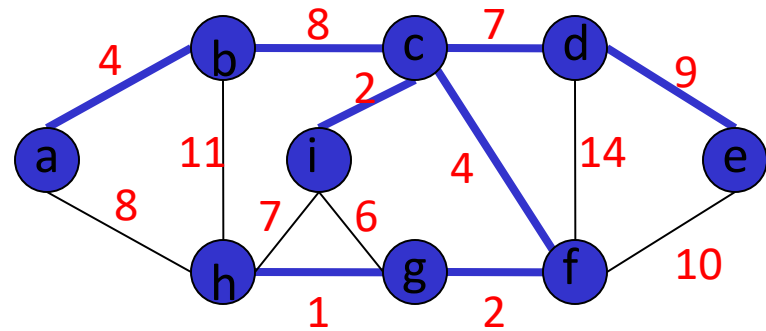
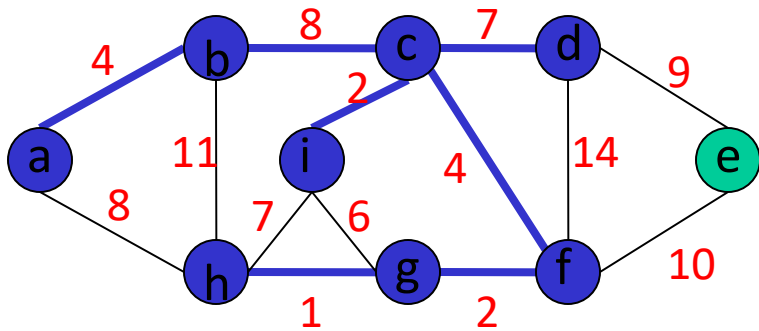
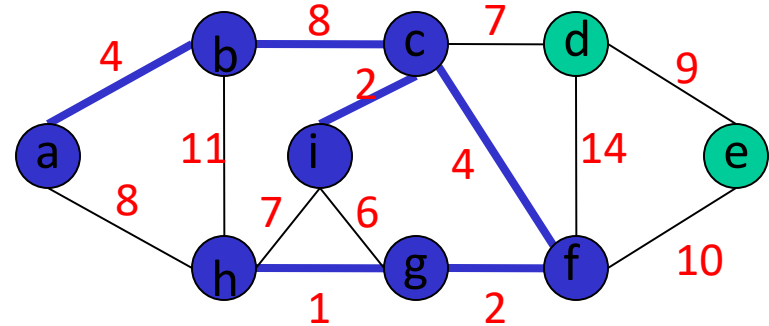
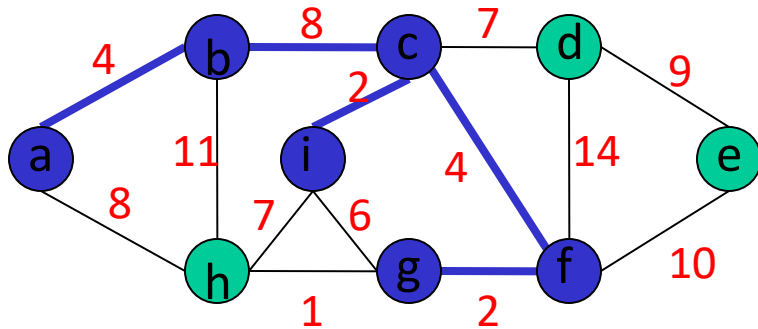
fermer(x);



Une exécution de l'algorithme de Prim



Une exécution de l'algorithme de Prim (fin)



Algorithme très proche de celui de Dijkstra :

$d(s_1) = 0$; ouvrir(s_1);

Pour tout k de 2 à n faire

$d(s_k) = +\infty$;

FinPour

Pour k de 1 à n faire

Soit x un **sommet ouvert** tel que $d(x)$ est minimum ;

Examiner(x);

FinPour.

Examiner(x) dans Dijkstra :

Pour tout successeur y de x faire

Si $d(y) > d(x) + c(x, y)$

alors $d(y) = d(x) + c(x, y)$;

père(y) = x ;

Si y n'est pas ouvert ouvrir(y);

FinSi

FinPour

fermer(x);

Examiner(x) dans Prim :

Pour tout successeur y de x faire

Si $d(y) > c(x, y)$

alors $d(y) = c(x, y)$;

père(y) = x ;

Si y n'est pas ouvert ouvrir(y);

FinSi

FinPour

fermer(x);

Complexité de l'algorithme de Prim

Même complexité que l'algorithme de Dijkstra : dépend de l'implémentation choisie.

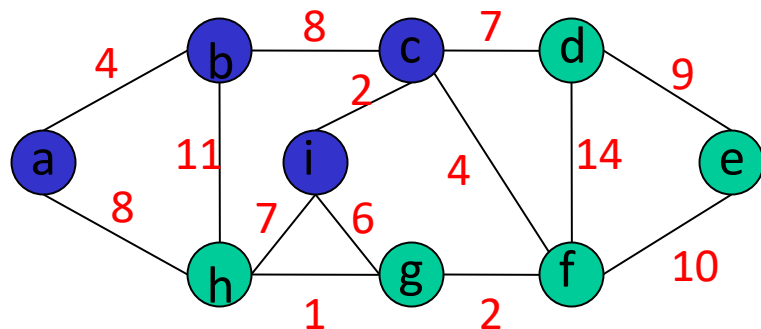
En utilisant un tas : complexité en $O(m \log n)$

Validité de l'algorithme de Prim : définition préliminaire

Définition d'un **co-cycle** :

Soit $G=(S,A)$. Soit U un ensemble de sommets de S .

Le co-cycle de U est l'ensemble des arêtes de A ayant une extrémité dans U et une extrémité dans $S \setminus U$.



L'algorithme de Prim retourne un arbre couvrant de coût minimum.

Preuve : Soit $G=(S,A)$

- Soit T l'arbre couvrant retourné par l'algo. de Prim.
et T_i l'arbre à la fin de l'itération i de l'algo.
- Soit T^* un arbre couvrant de coût minimum (un ACCM).

Si $T = T^* \rightarrow T$ est un ACCM.

L'algorithme de Prim retourne un arbre couvrant de coût minimum.

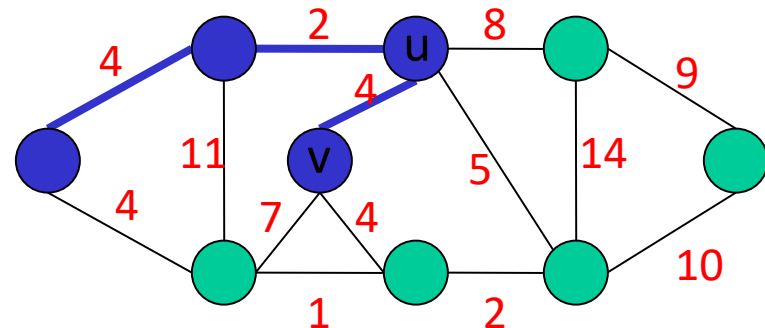
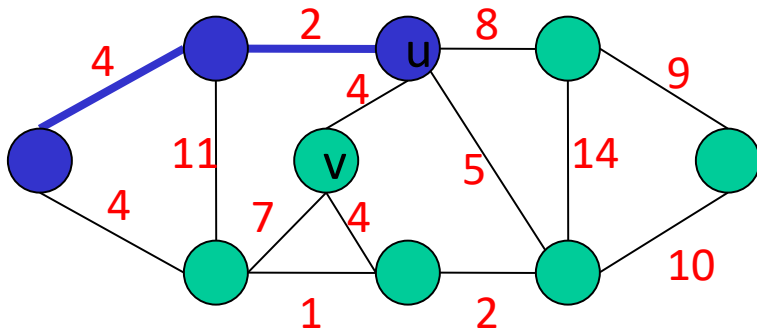
Preuve : Soit $G=(S,A)$

- Soit T l'arbre couvrant retourné par l'algo. de Prim.
et T_i l'arbre à la fin de l'itération i de l'algo.
- Soit T^* un arbre couvrant de coût minimum (un ACCM).

Si $T = T^* \rightarrow T$ est un ACCM.

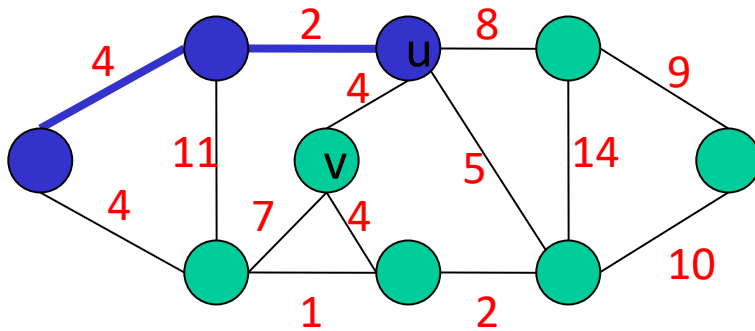
Sinon : Soit $e=\{u,v\}$ la première arête de $A \setminus T^*$ choisie par l'algo.

On suppose que cette arête a été choisie à l'itération k et que u a été ajoutée avant v (i.e. $u \in T_{k-1}$ et $T_{k-1} \cup \{e\} = T_k$) :



L'algorithme de Prim retourne un arbre couvrant de coût minimum.

Soit C la chaîne de u à v dans T^* .



$C \cup \{e\}$ forme un cycle.

Soit e^* une arête de C différente de e et qui est dans le co-cycle de T_{k-1}

Par construction $\text{coût}(e^*) \geq \text{coût}(e)$.

En remplaçant e^* par e dans T^* on obtient un arbre couvrant T_1^* tel que $\text{coût}(T_1^*) \leq \text{coût}(T^*)$

$\rightarrow T_1^*$ est un ACCM qui a une arête en commun de plus avec T .

On continue ce processus jusqu'à ce que $T_k^* = T \rightarrow T$ est un ACCM.

