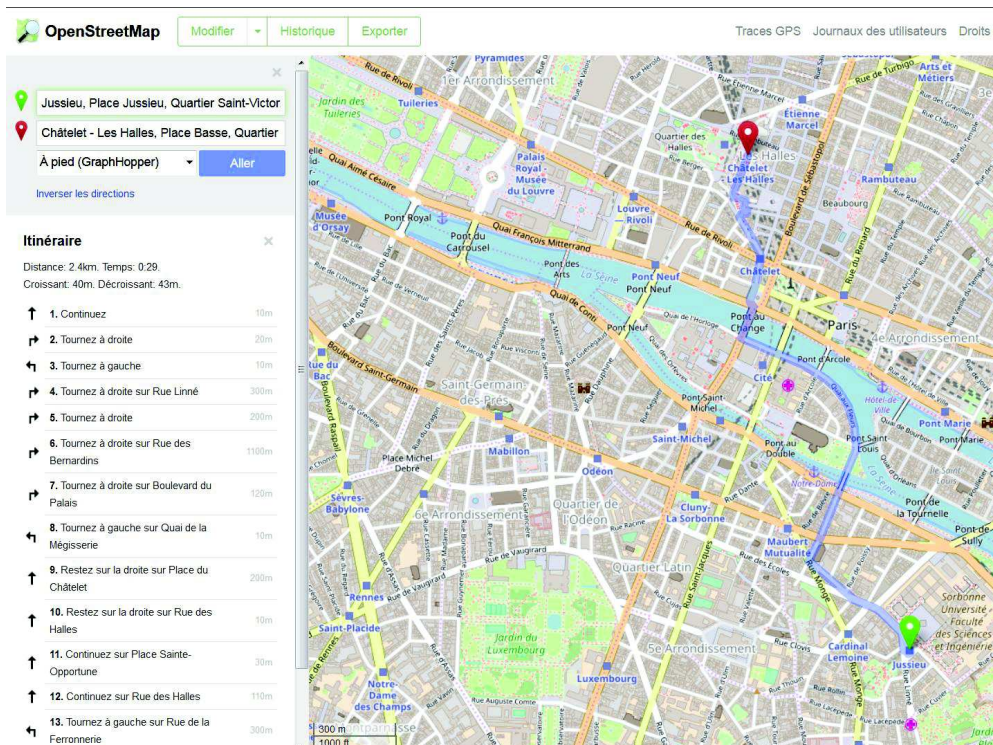


Algorithmes basés sur des parcours : algorithmes de Dijkstra et de Prim

Plus courts chemins (chemins de coûts minimum)

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

Chargés de cours : Fanny Pascual et Olivier Spanjaard.



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 p .

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

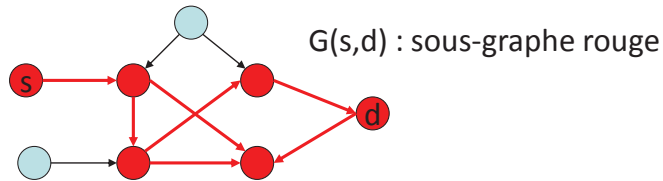
Etant donnés 2 sommets s et d , on veut :

- savoir s'il existe un chemin de coût minimum de s à d ;
- 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.



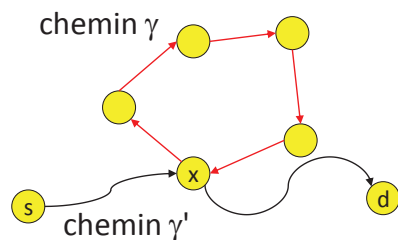
Propriété :

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

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 $c(\gamma') \leq c(\gamma)$ (lemme de König).



Définition :

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

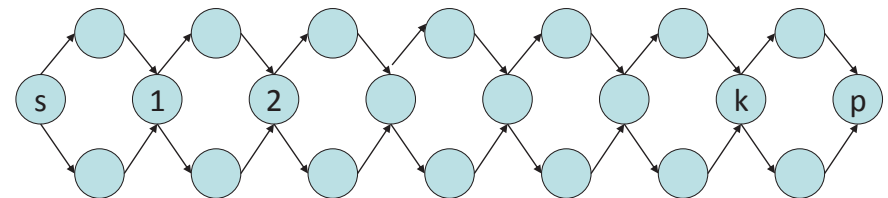
Propriété :

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 $G(s,d)$.

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).



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

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

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.

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)**.

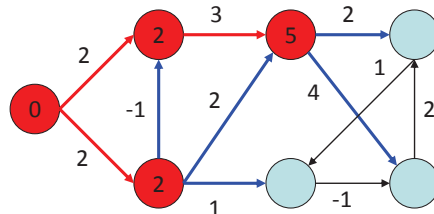
Si $x \notin X$, on fixe $d(x) = +\infty$

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



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.

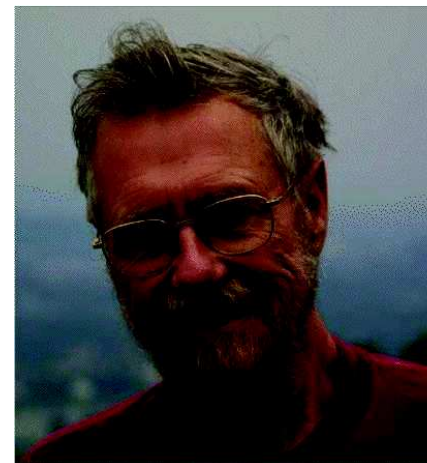
Propriété :

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.**

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 langage 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

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

Pour tout k de 2 à n faire

$d(s_k) = +\infty$;

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 ;

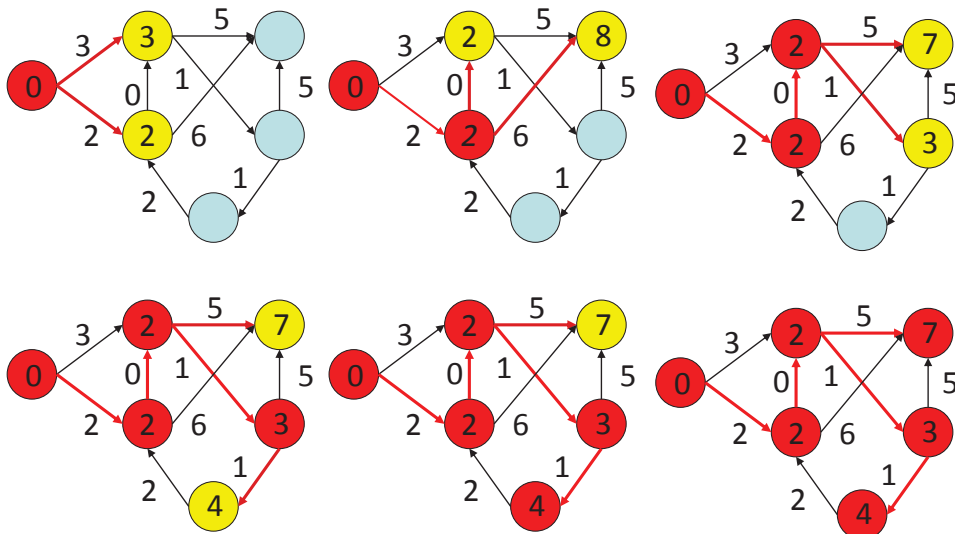
 Si y n'est pas ouvert, ouvrir(y)

 FinSi

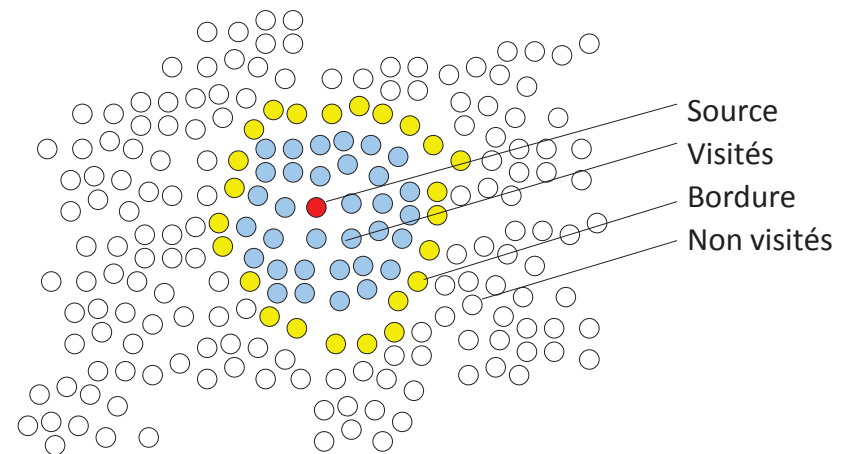
FinSi

FinPour

fermer(x);

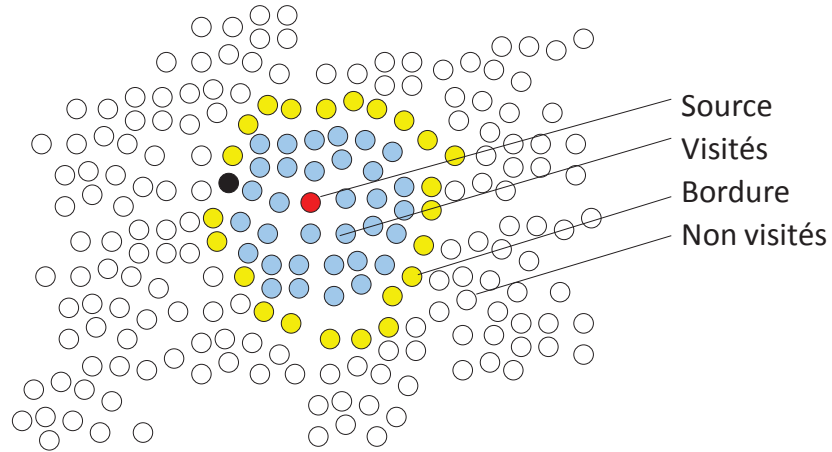


Interprétation physique



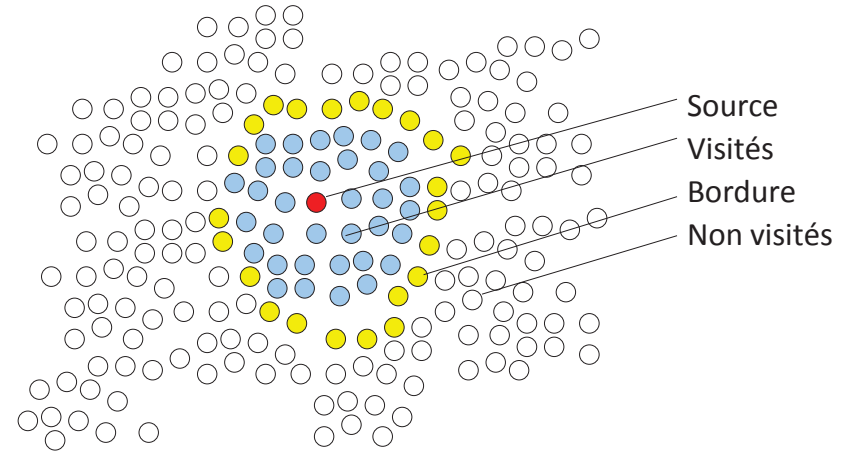
Front de propagation autour de la source = ronds dans l'eau

Interprétation physique



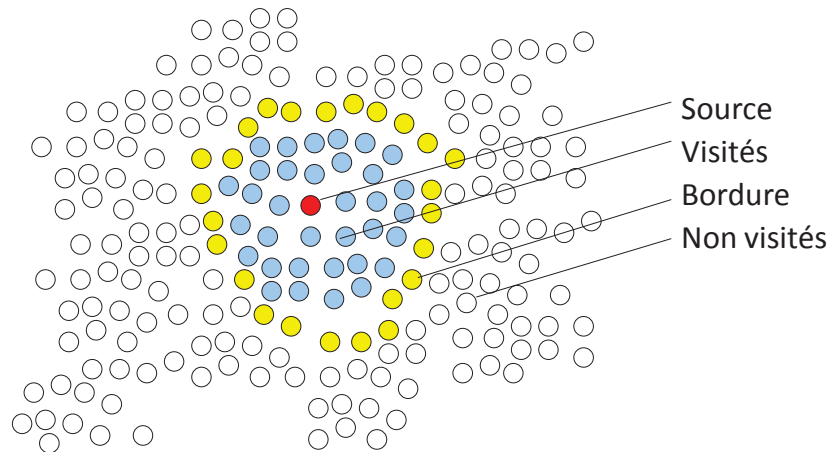
Front de propagation autour de la source = ronds dans l'eau

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

Complexité de l'algorithme de Dijkstra.

Rappel de l'algorithme :

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 : $O(n^2)$.

Tas

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 $O((n+m)\log(n))$.

Soit E un ensemble dynamique où chaque élément e est affecté d'une **priorité** $prio(e)$.
L'ensemble des priorités est muni d'un ordre total $<$.

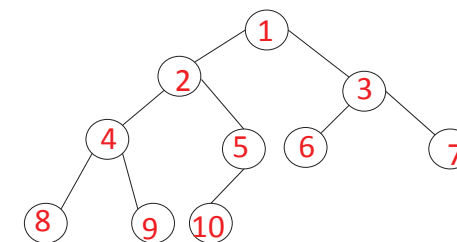
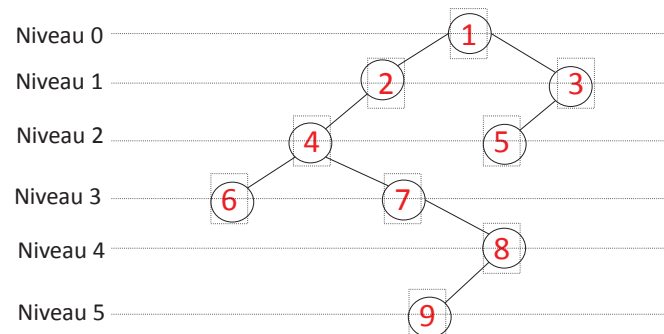
Opérations d'un tas

- **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

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.

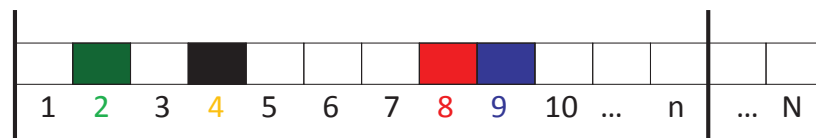
Propriétés :

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

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)

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) $\text{racine}(T)$ 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 $x \neq \text{rac}(T)$)
 - $\text{EST_FEUILLE}(x) \Leftrightarrow 2 * \text{num}(x) > n$
 - $\text{num}(\text{DERNIERE_FEUILLE}(T)) = n$

Opérations sur un tas

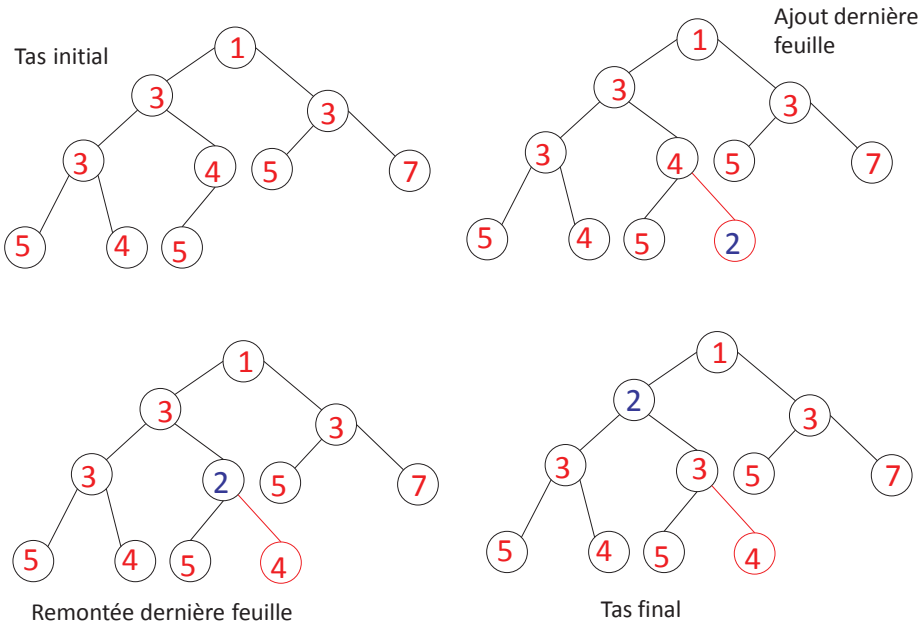
Insertion d'un élément de priorité k.

On utilise un tas T pour gérer (E, prio) .

```
Procédure Insérer(e,k,T) ;  
f:=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,père(x));  
    x:=père(x)  
FinTantque.
```

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

Exemple: Insérer(e,2,T)



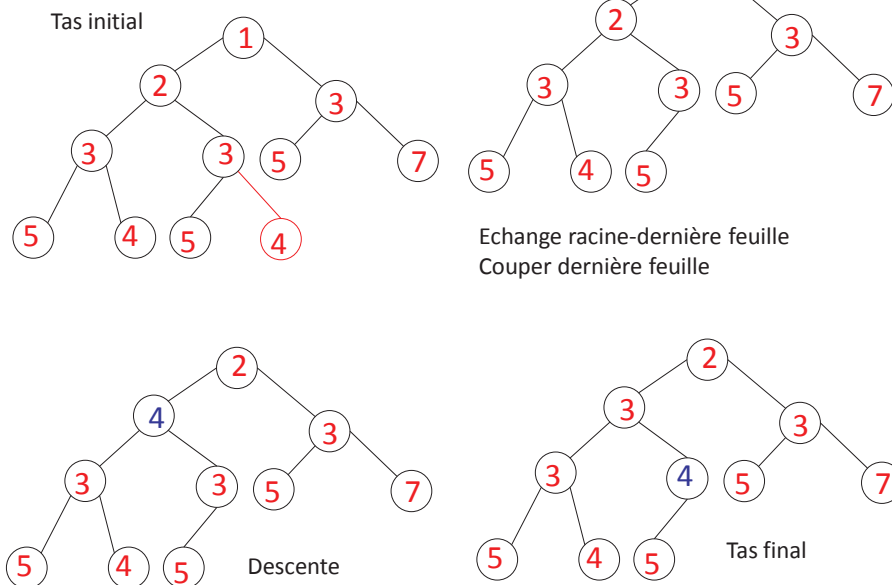
Suppression d'un élément de priorité minimale.

```

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)



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** $O(1)$

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 :

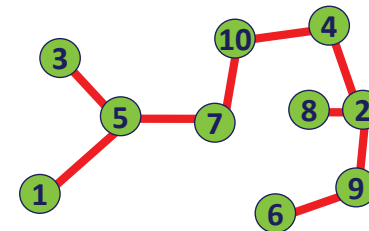
$$C \log(n)(n + d^+(z_1) + \dots + d^+(z_n)) = C(n+m)\log(n).$$

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

Application : conception de réseau

Arbre couvrant de coût minimum

Une entreprise de télécommunication doit **relier un ensemble de clients**. Le coût de connexion 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.

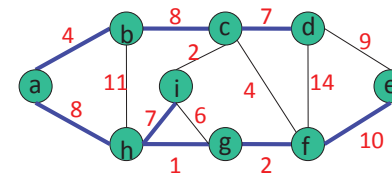
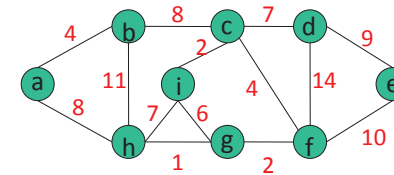
Problème :

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

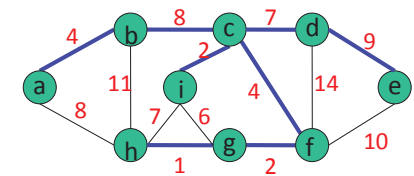
Remarque : H^* existe puisque le nombre d'arbres couvrants de G est fini.

Exemple

Graphe G :



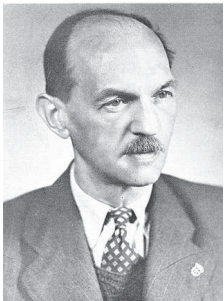
Arbre couvrant de coût = 47



Arbre couvrant de coût = 37

Algorithme de Prim

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



Vojtech Jarnik (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(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)

 Pour tout successeur y de x faire

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

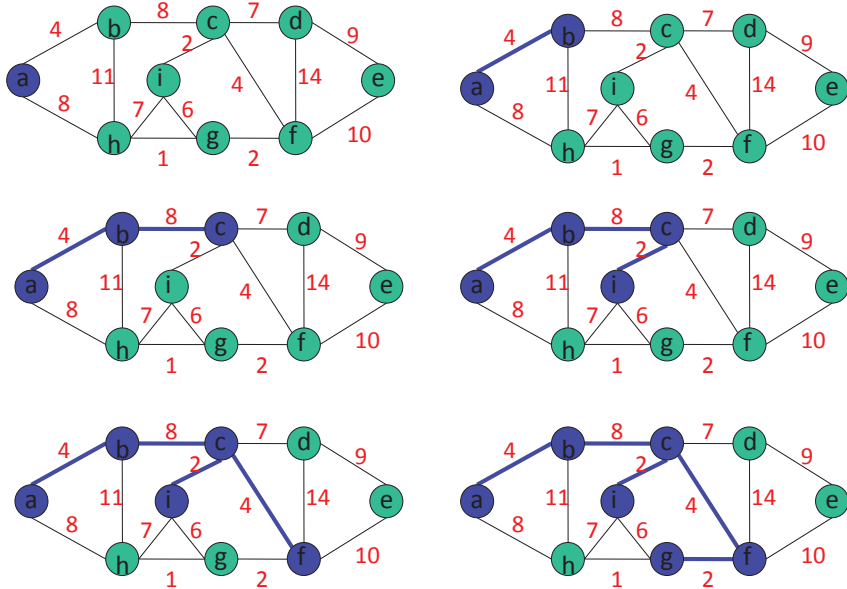
$d(y) = c(x,y)$; Si y n'est pas ouvert ouvrir(y);

 FinSi

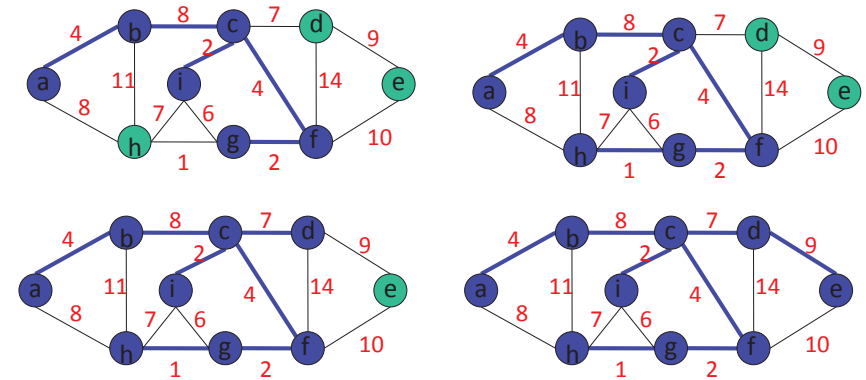
 FinPour

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

FinPour
Pour k de 1 à n-1 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

(preuve faite au tableau)