Introduction à l'algorithmique et la complexité (et un peu de CAML)

Graphes, Arbres, Arbres Binaires

Nicolas Nisse

Université Côte d'Azur, Inria, CNRS, I3S, France

Cours dispensés en MPSI (option Info) au CIV, depuis 2011-

http://www-sop.inria.fr/members/Nicolas.Nisse/lectures/prepa/

Outline

1. Graphes et Arbres
2. Arbres et Arbres Binaires en CAML
3. Parcours d'Arbres en profondeur (DFS) et en largeur (BFS)
4. Exemples de programmation dynamique dans les arbres

Graphes

Définition d'un graphe

Un graphe $G = (V, E)$ est un objet mathématique qui est une paire constituée d'un ensemble V d'éléments appelés *sommets* (ou *vertex*, pluriel *vertices*, en anglais) et d'une relation binaire $E \subseteq V \times V$ sur V, appelée l'ensemble des *arêtes* (ou *edges* en anglais).

Ici, on ne considère que les graphes finis, i.e., V sera toujours un ensemble fini.

Par ailleurs, on ne considère que les graphes non-orientés, i.e., les éléments de E sont des ensembles de deux sommets (l'ordre n'est pas important).

Soit un graphe $G = (V, E)$: Terminologie et notations.

Deux sommets $u, v \in V$ en relation sont liés par l'arête $\{u, v\} \in E$. On dit que u et v sont voisins ou adjacents. On note $N(u) = \{v \in V \mid \{u, v\} \in E\}$ l'ensemble des voisins de u ("N" pour "neighbours" en anglais). Le nombre $|N(u)|$ de voisins de $u \in V$ est appelé son degré.

Représentation.

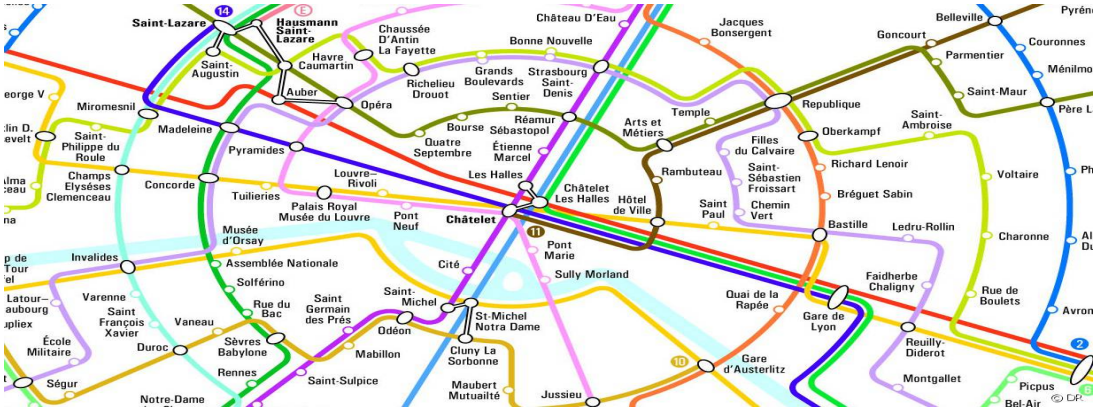
Il est généralement utile de dessiner un graphe en représentant ses sommets par des cercles (ou des points) et une arête entre deux sommets par une courbe liant les deux cercles correspondants.

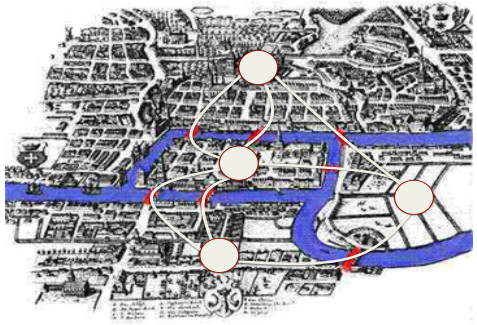
Notons que la terminologie (sommet, arête) est la même que pour celle décrivant, par exemple, un cube.

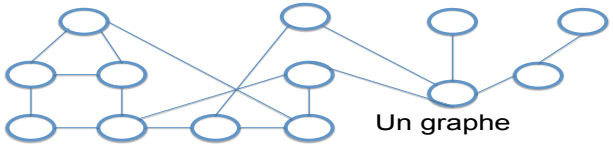
Tout (?) est graphe

Les graphes servent de modèle mathématique simple pour représenter et étudier une multitude d'applications, e.g., réseaux géographiques, de transport, biologiques, sociaux, Internet, Web...

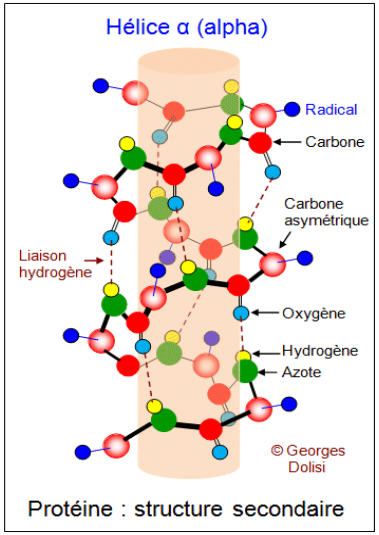
Les graphes permettent de décrire les interactions entre les éléments d'un ensemble.

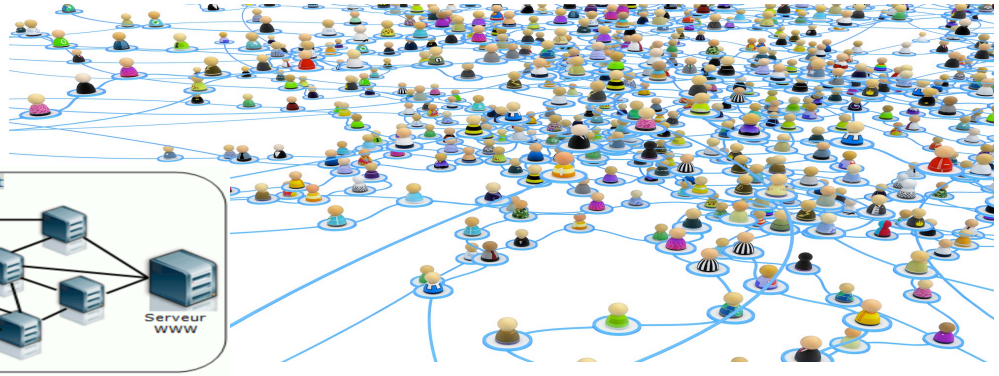


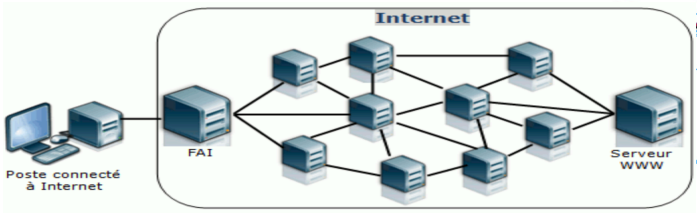




Un graphe







Un peu de terminologie (intuitive) dans les graphes

Soit un graphe $G = (V, E)$.

Un chemin dans G est une séquence $P = (v_1, \cdots, v_n)$ de sommets de G, deux-à-deux distincts, et tels que deux sommets consécutifs sont adjacents, i.e., $\{v_i, v_{i+1}\} \in E$ pour tout $1 \leq i < n$.
$P = (v_1, \cdots, v_n)$ est appelé un chemin entre v_1 et v_n *(P pour "path" en anglais).*
Si, de plus, $n > 2$ et $\{v_n, v_1\} \in E$, alors $(v_1, \cdots, v_n)$ est un cycle.

Le graphe G est connexe si (et seulement si), pour tous sommets $u, v \in V$, il existe un chemin entre u et v.

La longueur $\ell(P)$ d'un chemin $P = (v_1, \cdots, v_n)$ entre v_1 et v_n est son nombre d'arêtes $(n-1)$.
La distance $dist_G(u, v)$ entre $u \in V$ et $v \in V$ est la longueur minimum d'un chemin entre u et v, i.e., $dist_G(u, v) = \min\{\ell(P) \mid P \texttt{ est un chemin entre } u \texttt{ et } v\}$.

Dessinez votre/vos propre(s) graphe(s) et testez y ces définitions.

Arbres

Le contenu de ce slide (et des précédents) ne fait (je crois) pas partie du programme de MPSI, mais il est intéressant de le connaître pour mieux comprendre la notion d'arbre (par ailleurs, ces notions sont souvent présentes dans les sujets de concours).

Un graphe est un arbre si et seulement si il est connexe et sans cycle.

Soit un graphe $G = (V, E)$.

Si G est un arbre, alors il a au moins 2 feuilles (sommets de degré 1).
Idée de preuve : considérer 2 sommets $u, v, \in V$ maximisant $dist_G(u, v)$.

G est un arbre si et seulement si, pour tout $u, v \in V$, il existe un unique chemin entre u et v.
Idée de preuve : $\Rightarrow$ le chemin existe par connexité. Il est unique, sinon il y aurait un cycle. $\Leftarrow$ l'existence d'un chemin entre toute paire de sommets implique la connexité, le fait qu'un tel chemin soit unique implique l'acyclicité.

Si G est un arbre, alors $|V| = |E| + 1$.
Idée de preuve : soit $v \in V$ une feuille, prouver que $G \setminus v$ est un arbre et induction sur $|V|$.

Si G est connexe et $|V| = |E| + 1$, alors G est un arbre.

Arbres Enracinés

Soient un arbre $T = (V, E)$ (T pour “tree" en anglais) et $r \in V$ un sommet arbitraire appelé racine. (T, r) est un appelé arbre enraciné en r.

Pour tout $v \in V$, les voisins w de v tels que $dist_G(r, w) = dist_G(r, v) + 1$ sont appelés les enfants de v. C'est-à-dire que les enfants w de v sont les voisins de v tels que v est sur l'unique chemin entre r et w.
Les sommets $w \in V$ tels que v appartient à l'unique chemin entre r et w sont les descendants de v. Le sous-arbre T_v enraciné en v est alors l'arbre $T_v = (V', E')$ avec V' constitué de l'ensemble des descendants de v et de v, et $E' = E \cap (V' \times V')$.
Pour tout $v \in V \setminus \{r\}$, l'unique voisin $p(v)$ de v qui appartient à l'unique chemin entre r et v est appelé le parent de v.

Ces notions suggèrent la définition récursive suivante pour un arbre enraciné.

Un arbre enraciné $(T = (V, E), r)$ est défini par une racine r et un ensemble $\{(T_1, r_1), \cdots, (T_d, r_d)\}$ tel que, pour tout $1 \leq i \leq d$, (T_i, r_i) est un arbre enraciné en r_i, et T est obtenu de $T_1, \cdots, T_d$ en ajoutant l'arête $\{r_i, r\} \in E$ pour tout $1 \leq i \leq d$.

Outline

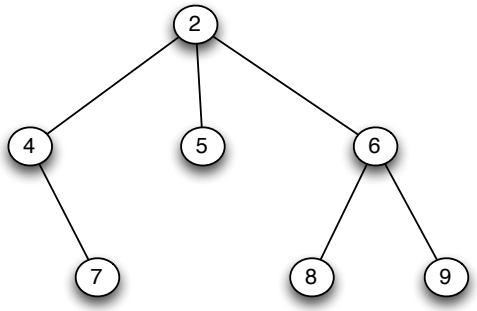
1. Graphes et Arbres
2. Arbres et Arbres Binaires en CAML
3. Parcours d'Arbres en profondeur (DFS) et en largeur (BFS)
4. Exemples de programmation dynamique dans les arbres

Arbres enracinés en CAML

Un arbre enraciné est donc défini par une racine et une liste de sous-arbres enracinés.
On peut définir le type Arbre grâce au type somme.

```
>       Caml Light version 0.80

#type 'a Arbre =
|Vide
|Sommet of 'a * ('a Arbre) list;;
Type Arbre defined.
#let t = Sommet(2,[Sommet(4,[Vide;Sommet(7,[Vide;Vide])]); Sommet(5,[Vide;Vide]
;Sommet(6,[Sommet(8,[Vide;Vide]);Sommet(9,[Vide;Vide])])]);;
t : int Arbre =
 Sommet
  (2,
   [Sommet (4, [Vide; Sommet (7, [Vide; Vide])]); Sommet (5, [Vide; Vide]);
    Sommet (6, [Sommet (8, [Vide; Vide]); Sommet (9, [Vide; Vide])])])
#
```

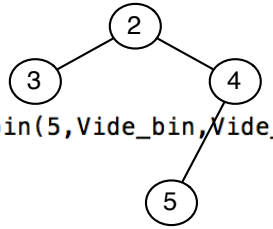


Ainsi, chaque sommet a une *étiquette* de type '*a* (qui peut représenter l'identifiant du sommet ou une valeur quelconque liée à ce sommet...).

Dans la suite, on restreint notre propos à une classe "plus simple" d'arbres.

Un arbre enraciné (T, r) est binaire si tout sommet a au plus 2 enfants.

```
#type 'a Arbre_Binaire =
|Vide_bin
|Sommet_bin of 'a * 'a Arbre_Binaire * 'a Arbre_Binaire ;;
Type Arbre_Binaire defined.
#let t2 = Sommet_bin(2,Sommet_bin(3,Vide_bin,Vide_bin),Sommet_bin(4,Sommet_bin(5,Vide_bin,Vide_bin),Vide_bin));;
t2 : int Arbre_Binaire =
 Sommet_bin
  (2, Sommet_bin (3, Vide_bin, Vide_bin),
  _ Sommet_bin (4, Sommet_bin (5, Vide_bin, Vide_bin), Vide_bin))
```



Arbres Binaires Complets

Soit $(T = (V, E), r)$ un arbre binaire enraciné.

La hauteur $h(T)$ de (T, r) est la distance maximum entre r et une feuille de T.

(T, r) est localement complet si tout sommet a exactement 0 ou 2 enfants.

(T, r) est complet s'il est loc. complet et que toutes les feuilles sont à même distance de r.

```
>       Caml Light version 0.80

#type 'a Arbre_Binaire =
|Vide
|Sommet of 'a * 'a Arbre_Binaire * 'a Arbre_Binaire ;;
Type Arbre_Binaire defined.
#let rec nbrSommet t = match t with
|Vide -> 0
|Sommet(v,tg,td) -> 1 + (nbrSommet td) + (nbrSommet tg) ;;
nbrSommet : 'a Arbre_Binaire -> int = <fun>
#let rec hauteur t = match t with
|Vide -> 0
|Sommet(v,Vide,Vide) -> 0
|Sommet(v,tg,td) -> 1 + (max (hauteur td) (hauteur tg)) ;;
hauteur : 'a Arbre_Binaire -> int = <fun>
#
```

```
#let rec loc_Complet t = match t with
|Vide -> true
|Sommet(v,Vide,Vide) -> true
|Sommet(v,Vide,_) -> false
|Sommet(v,_,Vide) -> false
|Sommet(v,tg,td) -> (loc_Complet tg) && (loc_Complet td) ;;
loc_Complet : 'a Arbre_Binaire -> bool = <fun>
```

```
#let rec Complet t = match t with
|Vide -> true
|Sommet(v,Vide,Vide) -> true
|Sommet(v,Vide,_) -> false
|Sommet(v,_,Vide) -> false
|Sommet(v,tg,td) -> (Complet td) && (Complet tg) && ((hauteur tg) == (hauteur td)) ;;
Complet : 'a Arbre_Binaire -> bool = <fun>
```

```
#let rec Complet2 t = match t with
|Vide -> (true,0)
|Sommet(v,Vide,Vide) -> (true,0)
|Sommet(v,Vide,_) -> (false,0)
|Sommet(v,_,Vide) -> (false,0)
|Sommet(v,tg,td) -> let (x,y) = Complet2 tg in
                    let (a,b) = Complet2 td in
                    (x && a && (y == b), (1+y)) ;;
Complet2 : 'a Arbre_Binaire -> bool * int = <fun>
```

Prouvez la correction de chacune des 5 fonctions ci-dessus et donnez leur complexité.
En particulier, expliquez pourquoi *Complet2* est plus efficace que *Complet*.
Adaptez les fonctions *nbrSommet* et *hauteur* aux cas des arbres généraux.

Outline

1. Graphes et Arbres
2. Arbres et Arbres Binaires en CAML
3. Parcours d'Arbres en profondeur (DFS) et en largeur (BFS)
4. Exemples de programmation dynamique dans les arbres

Parcours en profondeur (DFS) (1/2)

Les arbres sont généralement utilisés pour organiser des données à stocker. Il faut donc être capable de parcourir cette structure pour retrouver les données recherchées.

Parcours en profondeur d'abord (Depth First Search (DFS))

Étant donné un arbre enraciné $(T = (V, E), r)$, un parcours en profondeur d'abord consiste à partir de la racine r, et, à chaque fois que l'on se trouve sur un sommet $v \in V$, s'il existe un enfant de v non visité, on y va, sinon, on retourne sur le parent de v.
Ce processus se poursuit jusqu'à atteindre la racine alors que tous ses enfants ont été visités.

Remarque : dans un DFS, tout sommet est visité autant de fois que son degré.

Comme dit précédemment, chaque sommet dispose d'une étiquette (ou un *message*). Parcourir un arbre enraciné (T, r) permet de lire ces messages dans différents ordres possibles : lire l'étiquette d'un sommet lors de la première (resp., la dernière) rencontre de ce sommet correspond à un parcours préfixe, resp., parcours postfixe.

Dans le cas d'un arbre binaire, on peut aussi envisager un parcours infixe où les sommets internes sont "lus" lorsqu'on les atteint pour la deuxième fois (les feuilles autres que la racine sont "lues" dès leur première, et unique, visite).

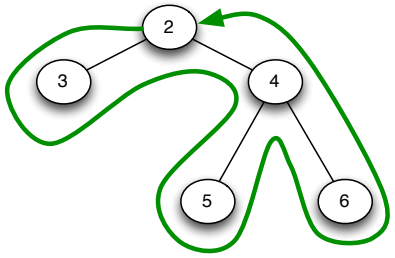
Parcours en profondeur (DFS) (2/2)

Exemples de parcours en profondeur (DFS) dans le cas d'un arbre binaire.

```
>       Caml Light version 0.80

#type 'a Arbre_Binaire =
|Vide
|Sommet of 'a * 'a Arbre_Binaire * 'a Arbre_Binaire ;;
Type Arbre_Binaire defined.
#let parcoursPrefixe t=
        let rec parcoursAux t l =
                match t with
                |Vide -> l
                |Sommet(v,tg,td) -> v::(parcoursAux tg (parcoursAux td l)) in
        parcoursAux t [] ;;
parcoursPrefixe : 'a Arbre_Binaire -> 'a list = <fun>
#let parcoursPostfixe t =
        let rec parcoursAux t l =
                match t with
                |Vide -> l
                |Sommet(v,tg,td) -> parcoursAux tg (parcoursAux td (v::l)) in
        parcoursAux t [] ;;
parcoursPostfixe : 'a Arbre_Binaire -> 'a list = <fun>
#let parcoursInfixe t =
        let rec parcoursAux t l =
                match t with
                |Vide -> l
                |Sommet(v,tg,td) -> parcoursAux tg (v::(parcoursAux td l)) in
        parcoursAux t [] ;;
parcoursInfixe : 'a Arbre_Binaire -> 'a list = <fun>
```

```
#let t = Sommet(2,Sommet(3,Vide,Vide),Sommet(4,Sommet(5,Vide,Vide),Sommet(6,Vide,Vide))) ;;
t : int Arbre_Binaire =
 Sommet
  (2, Sommet (3, Vide, Vide),
   Sommet (4, Sommet (5, Vide, Vide), Sommet (6, Vide, Vide)))
#parcoursPrefixe t ;;
- : int list = [2; 3; 4; 5; 6]
#parcoursPostfixe t ;;
- : int list = [3; 5; 6; 4; 2]
#parcoursInfixe t ;;
- : int list = [3; 2; 5; 4; 6]
```



Prouvez la correction des fonctions ci-dessus, et donnez leur complexité.
Adaptez les fonctions *parcoursPrefixe* et *parcoursPostfixe* aux cas des arbres généraux.

Notez qu'un arbre enraciné peut admettre plusieurs ordres préfixes/postfixes/infixes. Donnez les 3 ordres préfixes (resp., 3 ordres postfixes et 3 ordres infixes) pour l'arbre en exemple (enraciné en le sommet 2).

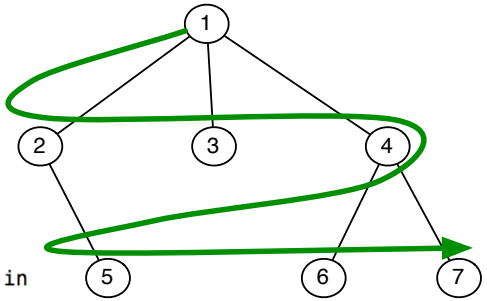
Parcours en largeur

Parcours en largeur d'abord (Breadth First Search (BFS))

Étant donné un arbre enraciné $(T = (V, E), r)$, un parcours en largeur d'abord consiste à partir de la racine r, et, à visiter chaque sommet dans l'ordre croissant des distances à r.
C'est-à-dire, un parcours BFS visite les sommets $(v_1, \cdots, v_n)$ de T dans un ordre tel que $dist_T(v_i, r) \leq dist_T(v_j, r)$ pour tout $i \leq j$.

```
>       Caml Light version 0.80

#type 'a Arbre =
|Vide
|Sommet of 'a * ('a Arbre) list ;;
Type Arbre defined.
#let ParcoursLargeur t =
        let rec ParcoursLargeurAux list_subtrees =
        match list_subtrees with
        |[] -> []
        |h::q -> match h with
                |Vide -> ParcoursLargeurAux q
                |Sommet(v,l) -> v::(ParcoursLargeurAux (q@l)) in
        ParcoursLargeurAux [t];;
ParcoursLargeur : 'a Arbre -> 'a list = <fun>
#let t = Sommet(1,[Sommet(2,[Vide;Sommet(5,[])]);Sommet(3,[]);Sommet(4,[Sommet(6,[]);Sommet(7,[])])]);;
t : int Arbre =
 Sommet
  (1,
   [Sommet (2, [Vide; Sommet (5, [])]); Sommet (3, []);
    Sommet (4, [Sommet (6, []); Sommet (7, [])])])
#ParcoursLargeur t;;
-_: int list = [1; 2; 3; 4; 5; 6; 7]
```



Prouvez la correction de la fonction *ParcoursLargeur* et donnez sa complexité.
Adaptez la fonction *ParcoursLargeur* au cas des arbres binaires.

Notez qu'un arbre enraciné peut admettre plusieurs ordres de parcours en largeur.
Donnez les 12 ordres possibles pour l'arbre en exemple (enraciné en le sommet 1).

Outline

1. Graphes et Arbres
2. Arbres et Arbres Binaires en CAML
3. Parcours d'Arbres en profondeur (DFS) et en largeur (BFS)
4. Exemples de programmation dynamique dans les arbres

Couverture sommet dans les graphes

Pour conclure, nous présentons deux problèmes classique de graphes et donnons des solutions efficaces pour les résoudre dans les arbres.

Couverture sommet (*Vertex Cover* en anglais)

Étant donné un graphe $G = (V, E)$, un ensemble K de sommets est une couverture sommet (attention au nom contre-intuitif) si toute arête de G a au moins une de ses extrémités dans K. C'est-à-dire, $K \subseteq V$ est une couverture sommet si, $\forall e \in E$, $e \cap K \neq \emptyset$.

Par exemple, si G représente une ville (chaque sommet est un carrefour et chaque arête est une rue), alors on peut voir une couverture sommet comme un ensemble de carrefours où placer un agent tel que chaque rue soit surveillée.

Problème de la couverture sommet minimum

Étant donnés un graphe $G = (V, E)$ et une fonction $w : V \to \mathbb{R}^+$, le poids d'un ensemble $X \subseteq V$ est $\sum_{v \in X} w(v)$.
Le problème considéré ici est celui de trouver une couverture sommet de poids minimum.

Remarque : Ce problème est NP-COMPLET. Sans rentrer dans les détails, cela implique qu'on ne sait pas s'il existe un algorithme polynomial (en $|V|$) pour le résoudre.
Nous allons voir que, si l'on se restreint aux cas des arbres, alors on sait le faire.

Couverture sommet dans les arbres non pondérés

Considérons d'abord le cas non pondéré (i.e., $w : V \to \mathbb{R}^+$ est telle que $w(v) = 1$ pour tout $v \in V$). Dans ce cas, on cherche une couverture sommet de taille minimum.

Lemme : Soit un graphe $G = (V, E)$ et $K \subseteq V$ une couverture sommet de G.

Soit $u \in V$ un sommet de degré 1 (s'il en existe) et v l'unique voisin de u.

- $\{u, v\} \cap K \neq \emptyset$;
- $K' = (K \setminus \{u\}) \cup \{v\}$ est une couverture sommet de G et $|K'| \leq |K|$.

Le lemme précédent (preuve laissée au lecteur) suggère, dans un arbre T, l'algorithme (récursif) glouton suivant :

- **Si** T n'a pas d'arête, **Retourner** $\emptyset$. **Sinon** :
- **Si** T a une unique arête $\{u, v\}$, **Retourner** $\{v\}$. **Sinon** :
- Soit $v \in V(T)$ un sommet (non feuille) de T adjacent à au plus un sommet non-feuille u (prouver qu'un tel sommet existe).
- **Retourner** $\{v\} \cup K$ avec K le résultat de l'algorithme appliqué récursivement sur l'arbre obtenu de T en lui supprimant v et $N(v) \setminus \{u\}$.

Théorème : l'algorithme glouton calcule une couverture sommet de taille minimum dans les arbres, en temps linéaire en le nombre de sommets.

Preuve : la correction se prouve par induction sur $|V|$ et par le lemme.
Pour la complexité (non détaillée ici), le point clé est de trouver v rapidement, ce qui peut reposer sur un calcul préalable d'un ordre topographique des sommets.

Couverture sommet dans les arbres

Dans le cas pondéré, nous allons procéder par programmation dynamique. Pour cela, nous devons définir les sous-problèmes suivants.
Soit $(T = (V, E), r, w)$ un arbre enraciné en r et avec une fonction w de pondération.

- Notons $K_{OPT}(T)$ une couverture sommet de T de poids minimum.
- Notons $K^r_{OPT}(T)$ une couverture sommet de T, contenant r, optimale, i.e., parmi toutes les couvertures sommets contenant r, $K^r_{OPT}(T)$ en est une de poids minimum.

Lemme (Cas de base): Si T réduit à un unique sommet r, alors

$K_{OPT}(T) = \emptyset$ (de poids nul) et $K^r_{OPT}(T) = \{r\}$ (de poids $w(r)$).

Lemme : Soit $r_1, \cdots, r_d$ les enfants de r dans T (existent, sinon lemme précédent) et, pour $1 \leq i \leq d$, soit T_i le sous-arbre de T enraciné en r_i.

Soient $K = \bigcup_{1 \leq i \leq d} K^r_{OPT}(T_i)$ et $K^r = \{r\} \bigcup_{1 \leq i \leq d} K_{OPT}(T_i)$.
Alors $K^r_{OPT}(T) = K^r$ et $K_{OPT}(T)$ est un ensemble de poids minimum entre K et K^r.

Prouver les lemmes précédents.
En déduire un algorithme récursif qui calcule une couverture sommet minimum dans tout arbre T en temps linéaire. Le coder en CAML.

Ensemble dominant dans les graphes

Pour conclure, nous présentons un problème classique de graphe et donnons une solution efficace pour le résoudre dans les arbres.

Ensemble dominant

Étant donné un graphe $G = (V, E)$, un ensemble D de sommets est dominant si tout sommet de G est soit dans l'ensemble D soit adjacent à au moins un sommet de D. C'est-à-dire, $D \subseteq V$ est dominant si, $\forall v \in V \setminus D$, $\exists u \in D$ tel que $\{u, v\} \in E$.

Par exemple, si G représente une ville (chaque sommet est un quartier et chaque arête est une rue), alors on peut voir un ensemble dominant D comme un ensemble d'endroits où placer des écoles de telle sorte que chaque quartier est à distance au plus une rue d'au moins une école.

Problème de l'ensemble dominant minimum

Étant donnés un graphe $G = (V, E)$ et une fonction $w : V \to \mathbb{R}$, le poids d'un ensemble $X \subseteq V$ est $\sum_{v \in X} w(v)$.
Le problème considéré ici est celui de trouver un ensemble dominant de poids minimum.

Remarque : Ce problème est NP-COMPLET. Sans rentrer dans les détails, cela implique qu'on ne sait pas s'il existe un algorithme polynomial (en $|V|$) pour le résoudre.
Nous allons voir que, si l'on se restreint aux cas des arbres, alors on sait le faire.

Ensemble dominant minimum dans les arbres

Dans le cas pondéré, nous allons procéder par programmation dynamique. Pour cela, nous devons définir les sous-problèmes suivants.
Soit $(T = (V, E), r, w)$ un arbre enraciné en r et avec une fonction w de pondération.

- Notons $d_{OPT}(T)$ le poids minimum d'un ensemble dominant de T.
- Notons $d^r_{OPT}(T)$ le poids minimum d'un ensemble dominant contenant r dans T.

Lemme (Cas de base): Si T réduit à un unique sommet r, alors

$d_{OPT}(T) = d^r_{OPT}(T) = w(r)$ (il existe un unique ensemble dominant $\{r\}$).

Lemme : Soit $r_1, \cdots, r_d$ les enfants de r dans T (existent, sinon lemme précédent) et, pour $1 \leq i \leq d$, soit T_i le sous-arbre de T enraciné en r_i.

$d^r_{OPT}(T) = w(r) + \sum_{1 \leq i \leq d} d_{OPT}(T_i)$ et

$d_{OPT}(T) = \min\{d^r_{OPT}(T); \min_{1 \leq i \leq d}(d^r_{OPT}(T_i) + \sum_{1 \leq j \leq d, j \neq i} d_{OPT}(T_j))\}$.

Prouver les lemmes précédents.
En déduire un algorithme récursif qui calcule un ensemble dominant minimum dans tout arbre T en temps linéaire. Le coder en CAML.