Arbres couvrants minimaux d'un graphe connexe valué

Algorithmes de Prim et de Kruskal

Vocabulaire

- graphe valué : (S, A, c) avec $A \subset \mathcal{P}_2(S)$ et $c : A \to \mathbb{R}$;
- chaîne, chaîne (ou cycle) élémentaire ;
- cocycle : si $X \subset S$, $\omega(X) = \{\{x, y\}, x \in X, y \notin X\} \subset A$;
- bordure : si $X \subset S$, $\beta(X) = \{y \notin X, \exists x \in X, \{x, y\} \in A\} \subset S$;
- arbre couvrant minimal

Dans toute la suite, n désigne le nombre de sommets, et p le nombre d'arêtes du graphe valué connexe considéré.

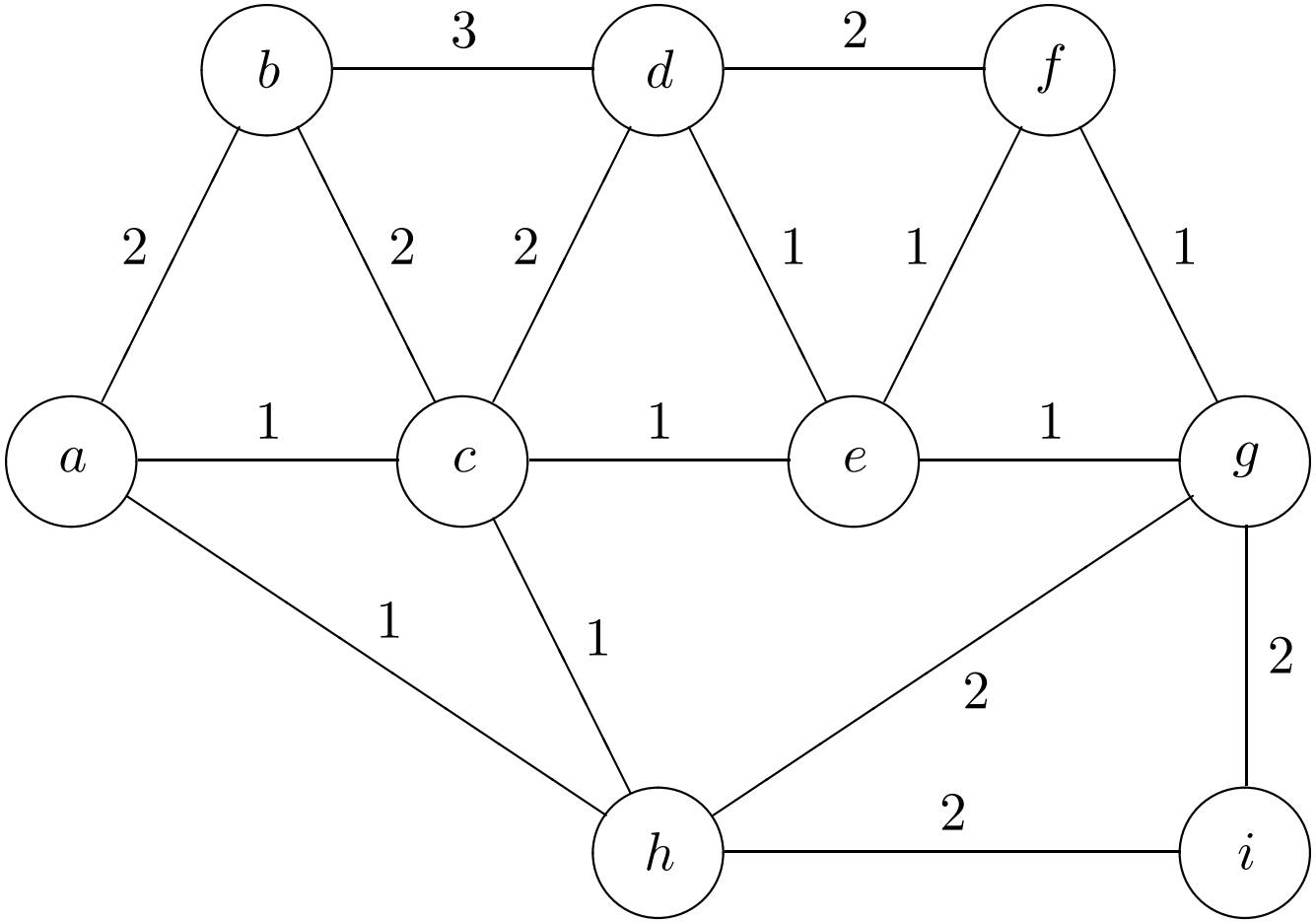


Fig. 1: un exemple de graphe valué

Partitions approximantes

Partition (O, R, L) de A telle qu'il existe un arbre couvrant minimal contenant toutes les arêtes de O mais aucune de R :

O est l'ensemble des arêtes **obligées** ;

R est l'ensemble des arêtes **refusées** ;

L est l'ensemble des arêtes **libres**

Au départ : $(\emptyset, \emptyset, A)$,

à l'arrivée : on veut $(O, R, \emptyset)$ où O est l'ensemble des $n-1$ arêtes d'un arbre couvrant minimal solution

Théorème des cycles candidats

Théorème 1

Soit (O, R, L) une partition approximante.

On suppose qu'il existe un cycle élémentaire γ ne contenant pas d'arête de R.

Soit alors α une arête de $\gamma \cap L$ de coût maximal.

$(O, R \cup \{\alpha\}, L \setminus \{\alpha\})$ est encore une partition approximante.

Théorème des cycles : démonstration

✧ Notons (S, T) un arbre couvrant minimal associé à notre partition approximante initiale, c'est-à-dire tel que $O \subset T$ et $T \cap R = \emptyset$.

Notons que γ contient au moins une arête libre (c'est-à-dire une arête de L). En effet, dans le cas contraire, les arêtes de γ seraient toutes dans O et donc dans T, et notre arbre contiendrait un cycle, ce qui est contradictoire avec la définition d'un arbre.

De deux choses l'une maintenant : ou bien notre arbre (S, T) ne contenait pas l'arête α choisie, et dans ce cas il est encore associé à $(O, R \cup \{\alpha\}, L \setminus \{\alpha\})$ et on a terminé.

Ou bien $\alpha \in T$. Mais alors $(S, T \setminus \{\alpha\})$ contient deux composantes connexes, qui sont des arbres disjoints. Mais parmi les arêtes de γ distinctes de α il existera une arête β (qui ne sera heureusement pas dans R) et qui reliera ces deux composantes connexes. Alors $(S, (T \cup \{\beta\}) \setminus \{\alpha\})$ est un arbre couvrant de coût plus petit ou égal que le coût de (S, T) qui était déjà minimal, donc lui aussi minimal. Et, bien sûr, $(S, T \cup \{\beta\} \setminus \{\alpha\})$ est, quant à lui, associé à la nouvelle partition $(O, R \cup \{\alpha\}, L \setminus \{\alpha\})$.❖

Théorème des cocycles candidats

Théorème 2

Soit (O, R, L) une partition approximante.

On suppose qu'il existe un cocyle $\omega(X)$ d'une partie X de S ne contenant pas d'arête de O.

Soit alors α une arête de $\omega(X) \cap L$ de coût minimal.

$(O \cup \{\alpha\}, R, L \setminus \{\alpha\})$ est encore une partition approximante.

Théorème des cocycles : démonstration

✧ Notons (S, T) un arbre couvrant minimal associé à notre partition approximante initiale, c'est-à-dire tel que $O \subset T$ et $T \cap R = \emptyset$.

Notons que $\omega(X)$ contient au moins une arête libre (c'est-à-dire une arête de L) : en effet, (S, T) étant couvrant, T rencontre $\omega(X)$. On conlut en rappelant que $\omega(X) \cap O = \emptyset$ et $T \cup R = \emptyset$.

Si $\alpha \in T$, l'arbre couvrant minimal (S, T) est encore associé à notre nouvelle partition approximante $(O \cup \{\alpha\}, R, L \setminus \{\alpha\})$ et on a terminé.

Sinon, notons $\alpha = \{x, y\}$ avec $x \in X$ et $y \notin X$. Il existe dans (S, T) une unique chaîne élémentaire qui lie x à y. Elle devra quitter X à un moment ou à un autre : c'est dire que dans cette chaîne se trouve une arête (d'ailleurs unique) $\beta \in \omega(X)$. Soit alors $(S, (T \cup \{\alpha\}) \setminus \{\beta\})$ l'arbre obtenu en substituant α à l'arête β. Bien sûr, il est de coût moindre que (S, T), et il est toujours couvrant. $(O \cup \{\alpha\}, R, L \setminus \{\alpha\})$ est donc bien encore une partition approximante. ✦

Algorithme générique

Algorithme :

(i) on commence par la partition $(\emptyset, \emptyset, A)$;

(ii) tant qu'il existe un cycle ou un cocycle candidat, on remplace la partition approximante courante par celle obtenue en appliquant le théorème 1 ou le théorème 2.

Théorème 3

L'algorithme générique ainsi défini termine avec une partition courante $(O, R, \emptyset)$. Alors (S, O) est bien un arbre couvrant minimal.

Algorithme de Prim

Algorithme :

Initialisation On choisit arbitrairement un sommet $a \in S$, et on pose $P = \{a\}$ et $(O, R, L) = (\emptyset, \emptyset, A)$.

Boucle principale Pour i de 1 à $n-1$ faire ce qui suit :

(i) on choisit une arête $\alpha = \{x, y\} \in \omega(P)$ de coût minimal, avec $x \in P$ et $y \notin P$;

(ii) on itère avec $P \leftarrow P \cup \{y\}$ et $(O, R, L) \leftarrow (O \cup \{\alpha\}, R, L \setminus \{\alpha\})$.

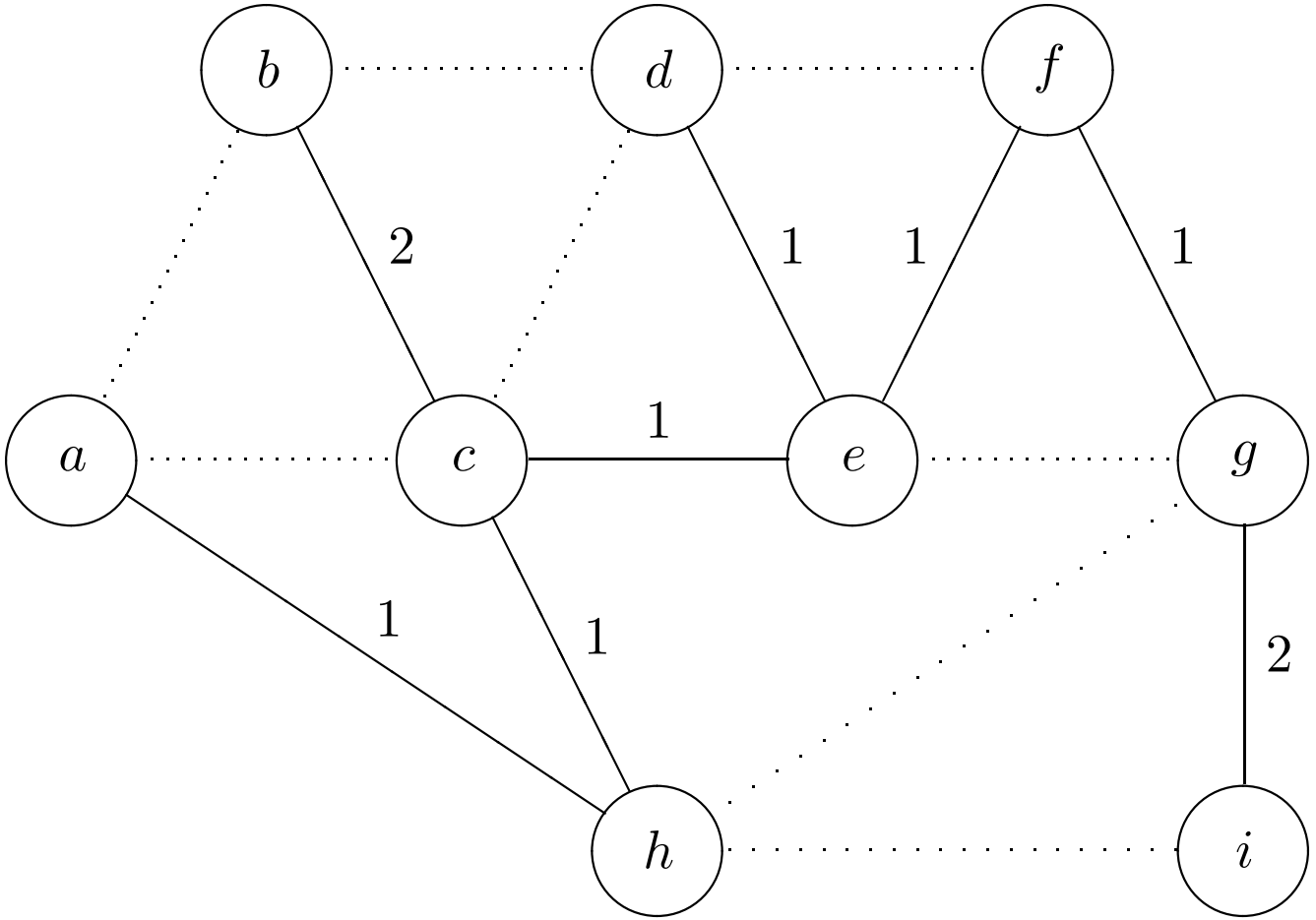


Fig. 2: Prim, en partant du sommet c

Implémentation de l'algorithme de Prim

On utilise trois tableaux :

`dans_p`, qui dit d'un sommet x s'il est dans P ;

`ancêtre`, qui donne pour chaque $y \notin P$ le (ou *un des*) sommet(s) $x_y \in P$ qui réalise $\min_{x \in P} c(x, y)$;

`coût_atteint`, qui donne pour chaque $y \notin P$ la valeur de $c(x_y, y)$

Il s'agit de mettre à jour ces tableaux à chaque fois qu'on *promeut* un nouvel élément y dans P.

L'algorithme ainsi écrit tourne en $O(n^2)$.

Algorithme de Kruskal

Algorithme :

Initialisation On trie par ordre de coût croissant les arêtes, obtenant une liste $(\alpha_i)_{0 \leqslant i < p}$; on pose $(O, R, L) \leftarrow (\emptyset, \emptyset, A)$.

Boucle principale Pour i de 0 à $p - 1$ faire ce qui suit :

(i) si α_i relie deux composantes connexes de O, itérer avec $(O, R, L) \leftarrow (O \cup \{\alpha_i\}, R, L \setminus \{\alpha_i\})$;

(ii) sinon, itérer avec $(O, R, L) \leftarrow (O, R \cup \{\alpha_i\}, L \setminus \{\alpha_i\})$.

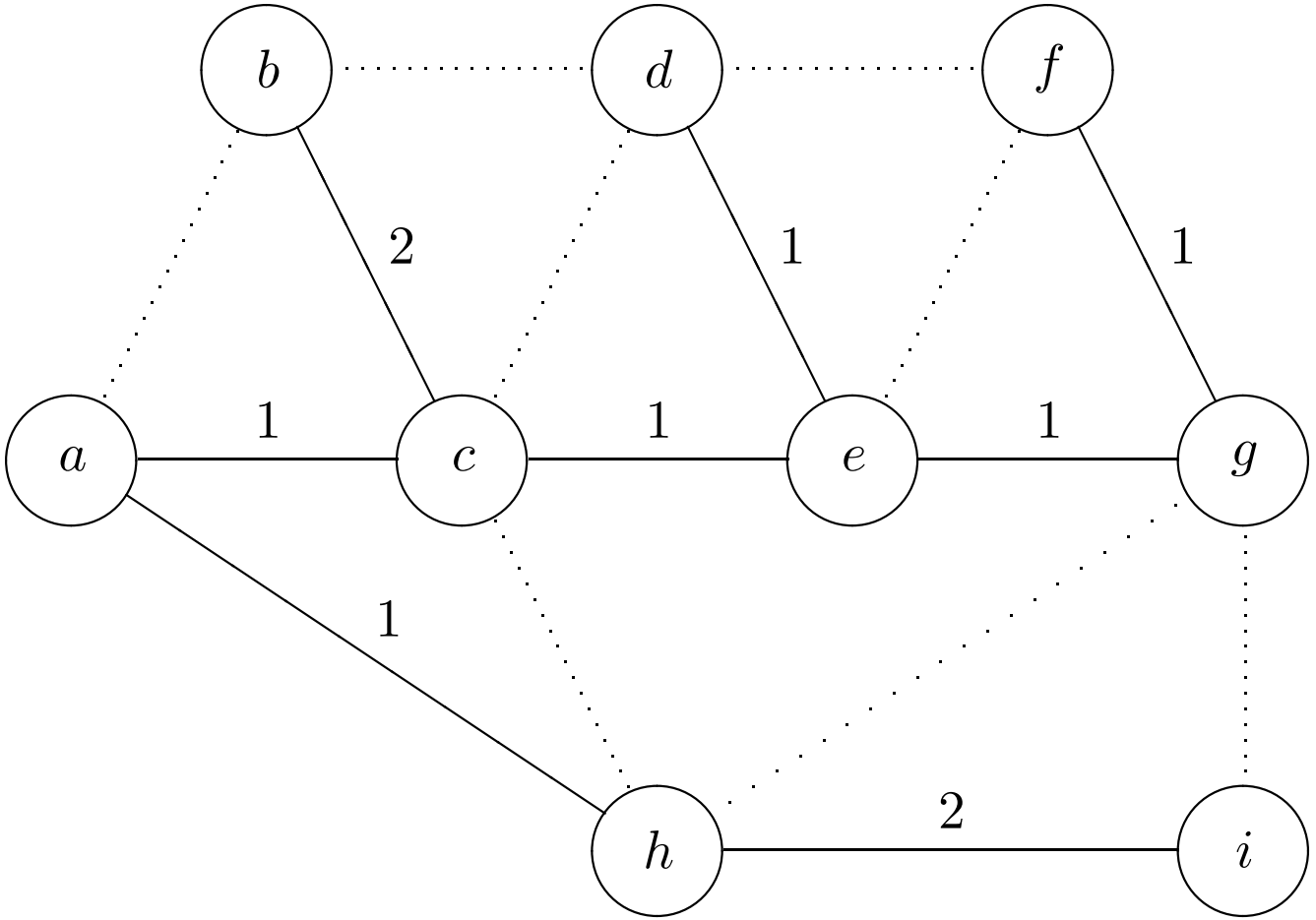


Fig. 3: un arbre obtenu par l'algorithme de Kruskal

Algorithme de Kruskal
une variante

Algorithme :

Initialisation On trie par ordre de coût *décroissant* les arêtes, obtenant une liste $(\alpha_i)_{0 \leqslant i < p}$; on pose $(O, R, L) \leftarrow (\emptyset, \emptyset, A)$.

Boucle principale Pour i de 0 à $p-1$ faire ce qui suit :

(i) si $(S, (O \cup L) \setminus \{\alpha_i\})$ est encore connexe, itérer avec $(O, R, L) \leftarrow (O, R \cup \{\alpha_i\}, L \setminus \{\alpha_i\})$;

(ii) sinon, itérer avec $(O, R, L) \leftarrow (O \cup \{\alpha_i\}, R, L \setminus \{\alpha_i\})$.

Le problème dit *union-find*

On part d'une partition d'un ensemble fini X de cardinal n en n singletons : $X = \{x_0\} \cup \{x_1\} \cup \cdots \cup \{x_{n-1}\}$.
Cette partition va évoluer, de plus en plus grossière.

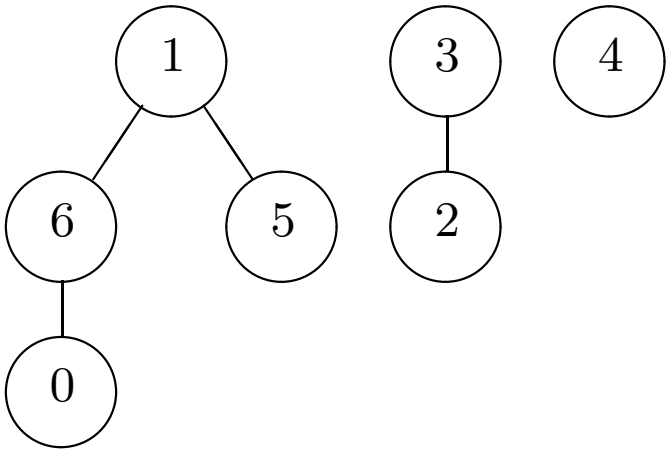
Opération *find* déterminer la classe d'un élément x, et donc savoir dire si deux éléments x et y sont dans la même classe ;

Opération *union* remplacer les classes de deux éléments x et y par la classe réunion

Représentation d'une partition

On utilise une forêt d'arbres : un arbre pour chaque classe, la racine de l'arbre est le représentant privilégié de la classe.

Par exemple la partition $X = \{0, 1, 5, 6\} \cup \{2, 3\} \cup \{4\}$ pourra être représentée par la forêt suivante :



find

Il s'agit de monter au (grand-)père.
Il suffit pour cela de représenter la forêt d'arbres par un tableau donnant pour chaque élément la valeur de son père (ou d'ailleurs d'un quelconque de ses ancêtres...)

union

On raccrochera l'arbre de poids le plus faible à la racine de l'arbre de poids le plus fort : on tiendra donc à jour un tableau conservant pour chaque élément la taille courante du sous-arbre dont il est racine.

Compression des chemins *(path compression)*

À chaque opération *find* réalisée, on associe chaque élément visité directement à la racine de l'arbre. Par exemple, `find 20` produit ceci :

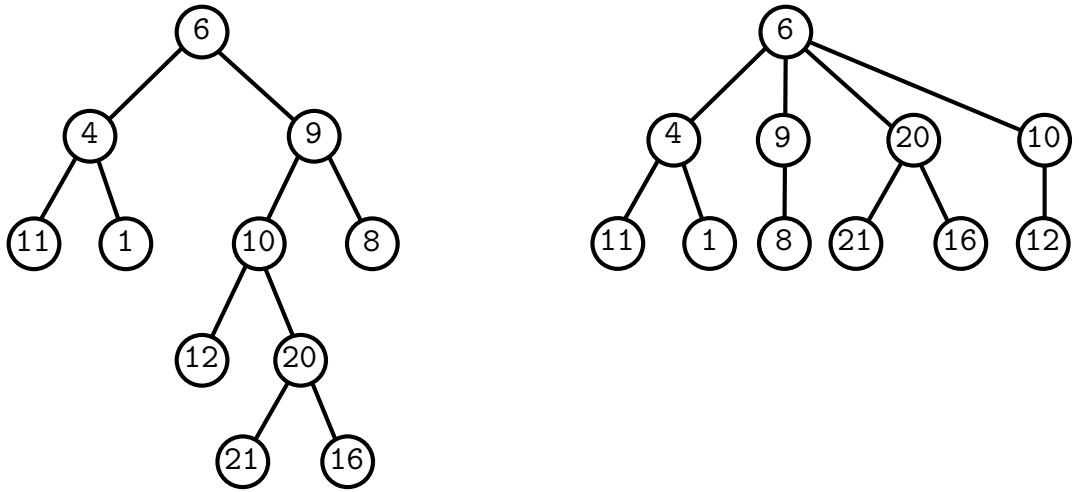


Fig. 4: compression des chemins (avant et après `find 20`)

Évaluation

On peut montrer, mais c'est *vraiment très* difficile, qu'une suite de $n-1$ opérations *union* et de p opérations *find* (avec $p \geqslant n$) se réalise en $O(p\alpha(p,n))$, où la fonction α est en pratique toujours un (petit) $O(1)$ (qui ne dépasse pas 2 en pratique...)

On en déduit que l'algorithme de Kruskal, grâce à cette représentation des partitions, peut tourner en $O(p \lg n)$.

La fonction α

On définit une espèce de *fonction d'Ackermann* en posant $A(0,0) = 0$, $A(p,0) = 1$ si $p \leqslant 1$, $A(0,n) = 2n$ si $n \geqslant 0$, et plus généralement $A(p+1,n+1) = A(p, A(p+1,n))$.

On définit $\alpha(p,n) = \min\{k \geqslant 1, A(k, 4\lceil p/n \rceil > \lg n\}$.

On vérifie que $\alpha(p,n) \leqslant \alpha(n,n) \leqslant 2$ pour $p \geqslant n$ et $n < 2^{65536}$.

$A(p,n)$	0	1	2	3	4	5
$p=0$	0	2	4	6	8	10
$p=1$	1	2	4	8	16	32
$p=2$	1	2	4	16	65536	2^{65536}
$p=3$	1	2	4	65536	$2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}$ (65536 exposants)	$\ldots$