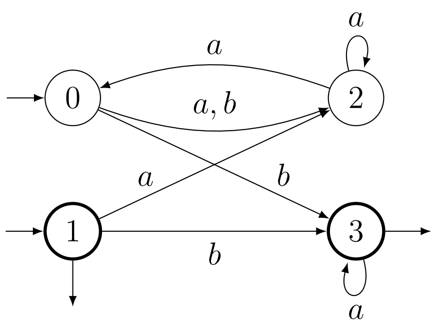
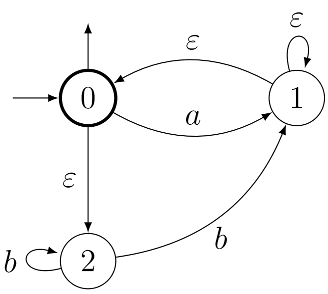
Durée : 4h. Aucun matériel électronique autorisé. Aucun document autorisé. Si une fonction ou un résultat mathématique est introduit dans l'énoncé, vous pouvez l'utiliser dans les questions suivantes, y compris si vous n'avez pas proposé de définition de cette fonction, de démonstration du résultat. Les exercices sont complètement indépendants. Dans toutes les questions de programmation, chaque fonction auxiliaire (comprendre, non spécifiquement demandée dans l'énoncé) doit être accompagnée d'une description rapide de son comportement et du sens attaché à ses arguments.

Exercice 1 : Automates : exercices de base

Q. 1 *En appliquant exactement l'algorithme de déterminisation du cours,* donner un automate déterministe et complet équivalent à l'automate ci-dessous. On attend sur la copie à la fois l'automate résultat et la table qui a permis de le construire. Aucun point ne sera accordé pour un automate déterministe équivalent obtenu par une autre méthode.



Q. 2 Donner l'automate obtenu, *en appliquant exactement l'algorithme de suppression des ε-transitions du cours,* sur l'automate suivant, en traitant les états dans l'ordre $0, 1, 2$. On attend une représentation de l'automate obtenu après le traitement de chaque état.



Q. 3 Sur l'alphabet $\Sigma = \{a, b, c\}$ on considère le langage L suivant.

$$L \overset{\text{déf}}{=} \{w \in \Sigma^\star \mid w \neq \varepsilon \text{ et } \forall i \in [\![1, |w|]\!], w_i = b \Rightarrow (i \neq 1 \text{ et } i \neq |w| \text{ et } w_{i-1} = w_{i+1})\}$$

Montrer que L est reconnaissable en fournissant un automate $\mathcal{A}$, à 4 états, dont L est le langage.

Exercice 2 : Accessibilité dans un graphe non orienté

Soit $G = (S, A)$ un graphe non orienté. On note $n = |S|$ et on suppose que $S = [\![1, n]\!]$. De plus on note $m = |A|$. Pour $(u, v) \in S^2$, on dit que u et v sont *reliés* dans G, ce qu'on note $u \sim v$ s'il existe une chaîne entre u et v dans G. On cherche à calculer la relation $\sim$, c'est-à-dire à déterminer pour chaque paire de sommets s'ils sont reliés par une chaîne du graphe. On cherche à calculer cette relation sous la forme d'une matrice de booléens $(T[u][v])_{(u,v) \in S^2}$ indiquant si $u \sim v$.

Sommets intérieurs Si $(\gamma_k)_{k\in[\![0,l]\!]}$ est une chaîne élémentaire[♣] de G, on note $I(\gamma)$ l'ensemble des sommets intérieurs de γ, *i.e.* les sommets de γ qui n'en sont pas les extrémités.

$$I(\gamma) \stackrel{\text{déf}}{=} \{\gamma_k \mid k \in [\![1, l-1]\!]\}$$

On remarque qu'une chaîne ayant seulement 0 ou 1 arête ne possède aucun sommet intérieur, autrement dit si $l \leqslant 1$, alors $I(\gamma) = \emptyset$ et la réciproque est vraie.

Chaînes élémentaires dans les sous-graphes. Pour tout $(u, v) \in S^2$, on note $\mathcal{C}_{u,v}$ l'ensemble des chaînes élémentaires de u à v. De plus, pour tout $k \in [\![0, n]\!]$, on définit $\mathcal{C}_{u,v}^{(k)}$ l'ensemble des chaînes élémentaires, de u à v, dont tous les sommets intérieurs sont dans $[\![1, k]\!]$.

$$\mathcal{C}_{u,v}^{(k)} \stackrel{\text{déf}}{=} \{\gamma \in \mathcal{C}_{u,v} \mid I(\gamma) \subseteq [\![1, k]\!]\}.$$

Famille de relations $\sim_k$. On définit finalement, pour $k \in [\![0, n]\!]$, la relation binaire $\sim_k$ sur S comme suit.

$$\forall (u, v) \in S^2, u \sim_k v \stackrel{\text{déf}}{\Leftrightarrow} \mathcal{C}_{u,v}^{(k)} \neq \emptyset$$

Autrement dit, $u \sim_k v$ dès lors qu'il existe une chaîne de u à v n'utilisant, pour sommet intérieurs, que des sommets de $[\![1, k]\!]$.

Q. 1 Pour $k \in [\![1, n]\!]$, la relation $\sim_k$ est-elle une relation d'équivalence ?

Q. 2 Justifier en quoi le calcul des relations $(\sim_k)_{k\in[\![0,n]\!]}$ permet de répondre au problème du calcul de $\sim$.

On cherche à établir des relations de récurrence permettant le calcul des $(\sim_k)_{k\in[\![0,n]\!]}$.

Q. 3 Que vaut $\sim_0$?

Q. 4 Soient $u \in S$, $v \in S$, $k \in [\![1, n]\!]$. Proposer et démontrer une relation entre $u \sim_k v$ et $u \sim_{k-1} k$, $k \sim_{k-1} v$ et $u \sim_{k-1} v$.

Q. 5 Quel problème pose l'implémentation d'une telle relation de récurrence ?

Dans les algorithmes qui suivent, la famille de relations $(\sim_k)_{k\in[\![0,n]\!]}$ sera représentée par un tableau tri-dimensionnel T indicé par $[\![0, n]\!] \times S^2$ contenant des booléens :

$$\forall k \in [\![0, n]\!], \forall (u, v) \in S^2, T[k][u][v] = \mathsf{V} \text{ si et seulement si } u \sim_k v$$

Q. 6 Proposer un algorithme permettant de calculer $\sim$, tout en évitant le problème soulevé en **Q. 5**.

Q. 7 Dans le cas où la complexité spatiale de l'algorithme proposé n'est pas en $\mathcal{O}(n^2)$, proposer une amélioration permettant d'atteindre une telle complexité spatiale.

Q. 8 De quel algorithme au programme de MP2I l'algorithme 6 est-il proche ?

Q. 9 Au moyen d'un parcours de graphe, proposer une solution algorithmique alternative au problème du calcul de $\sim$. Préciser la complexité temporelle de cette solution.

Exercice 3 : Listes gauches [Ccmp 2023]

[♣]. *i.e.* ne passant pas deux fois par un même sommet

1. Préliminaires

Présentation du sujet

L'objectif de cet exercice est de construire des *listes à accès direct* : une liste à accès direct est un type de donnée abstrait qui permet, d'une part, d'empiler et de dépiler efficacement un élément en tête de liste et, d'autre part, d'accéder efficacement au k^e élément de la liste, pour n'importe quel indice k valide.
Dans la première section, nous étudions un système de numération : la représentation binaire gauche des entiers naturels. Dans la deuxième section, nous étudions les arbres binaires parfaits. Dans la troisième section, nous implémentons le type liste à accès direct par la structure de données concrète *liste gauche*, que nous construisons en exploitant les résultats obtenus dans les sections précédentes.
Dans tout l'énoncé, un même identificateur écrit dans deux polices de caractère différentes désignera la même entité, mais du point de vue mathématiques avec la police en italique (par exemple n) et du point de vue informatique avec celle en romain avec espacement fixe (par exemple `n`).

Travail attendu

Pour répondre à une question, il est permis de réutiliser le résultat d'une question antérieure, même sans avoir réussi à établir ce résultat. Des rappels de programmation sont faits en annexe et peuvent être utilisés directement.
Selon les questions, il faudra coder des fonctions à l'aide du langage de programmation C exclusivement, en reprenant le prototype de fonction fourni par le sujet, ou en pseudo-code (i.e. dans une syntaxe souple mais conforme aux possibilités offertes par le langage C). Il est inutile de rappeler que les entêtes `<assert.h>`, `<stdbool.h>`, etc. doivent être inclus.
Quand l'énoncé demande de coder une fonction, sauf indication explicite de l'énoncé, il n'est pas nécessaire de justifier que celle-ci est correcte ou de tester que des préconditions sont satisfaites. On suppose que le type `int` n'est jamais sujet à des débordements.
Le barème tient compte de la clarté des programmes : nous recommandons de choisir des noms de variables intelligibles ou encore de structurer de longs codes par des blocs ou par des fonctions auxiliaires dont on décrit le rôle. Lorsqu'une réponse en pseudo-code est permise, seule la logique de programmation est évaluée, même dans le cas où un code en C a été fourni en guise de réponse.

2. Représentation binaire gauche des entiers naturels

Mise en place

Soient m un entier naturel et N un entier naturel non nul. Il est classique d'appeler *représentation binaire standard de l'entier m sur N chiffres*, ou plus simplement *représentation standard*, toute suite finie $b = (b_n)_{0 \leqslant n < N}$ de longueur N telle que, pour tout indice n compris entre 0 et $N-1$, le chiffre b_n appartient à $\{0, 1\}$ et l'égalité suivante est vérifiée

$$m = \sum_{n=0}^{N-1} b_n 2^n.$$

Définition : Nous appelons *représentation binaire gauche de l'entier m sur N chiffres* toute suite finie $g = (g_n)_{0 \leqslant n < N}$ de longueur N telle que,
(i) pour tout indice n compris entre 0 et $N-1$, le chiffre g_n appartient à $\{0, 1, 2\}$,
(ii) l'égalité suivante est satisfaite

$$m = \sum_{n=0}^{N-1} g_n (2^{n+1} - 1),$$

(iii) le chiffre « 2 » n'apparaît qu'au plus une fois parmi les chiffres $(g_n)_{0 \leqslant n < N}$,

(iv) s'il existe une position p telle que le chiffre g_p est 2, alors, pour tout indice n compris entre 0 et $p - 1$, le chiffre g_n est nul.

De manière plus courte, nous parlons simplement de *représentation gauche*.

La figure 1 ci-dessous donne une représentation standard et une représentation gauche sur quatre chiffres des seize premiers entiers. Conformément à l'usage habituel, nous écrivons toute représentation, qu'elle soit standard ou gauche, sous la forme d'un mot $b_{N-1} \cdots b_0$ ou $g_{N-1} \cdots g_0$ dans lequel les chiffres de poids faibles sont écrits à droite.

Entier	Repr. standard	Repr. gauche
0	0000	0000
1	0001	0001
2	0010	0002
3	0011	0010
4	0100	0011
5	0101	0012
6	0110	0020
7	0111	0100

Entier	Repr. standard	Repr. gauche
8	1000	0101
9	1001	0102
10	1010	0110
11	1011	0111
12	1100	0112
13	1101	0120
14	1110	0200
15	1111	1000

FIGURE 1 – Représentations standard et gauche des seize premiers entiers

Q. 1 Soit c un entier. Donner la représentation standard de l'entier dont une représentation gauche est $10 \cdots 0$ (avec c chiffres nuls) en justifiant sommairement. Faire de même avec l'entier dont une représentation gauche est $20 \cdots 0$ (avec c chiffres nuls).

Q. 2 Déterminer, en justifiant, le plus grand entier naturel M_N qui admet une représentation gauche sur N chiffres. Préciser la représentation gauche de cet entier.

Définition : Soit n_0 un indice compris entre 0 et $N - 1$. On dit que l'indice n_0 est la *position du chiffre de plus fort poids* d'une représentation $g_{N-1} \cdots g_0$ si l'indice n_0 est le plus petit entier tel que, pour tout indice $n > n_0$, le chiffre g_n est nul. On appelle le chiffre g_{n_0} le *chiffre de plus fort poids*.

Q. 3 Soient N un entier naturel non nul, $g = g_{N-1} \cdots g_0$ et $h = h_{N-1} \cdots h_0$ deux représentations gauches d'un même entier m. Démontrer que les chiffres de plus fort poids de g et de h sont de même valeur et à la même position.

Q. 4 Démontrer que tout entier appartenant à l'intervalle $[\![0, M_N]\!]$, où l'entier M_N a été introduit à la **Q. 2**, ne possède au plus qu'une seule représentation gauche sur N chiffres.

Indication C : L'entier N est déclaré comme constante globale. Nous utilisons la structure C déclarée comme suit pour écrire la représentation gauche sur N chiffres d'un entier $g_{N-1} \cdots g_0$.

```c
const int N = 8;

struct RepGauche {
  int position;
  bool chiffres[N];
};
typedef struct RepGauche rg;
```

Le champ `position` repère la position éventuelle du chiffre 2 ; il vaut -1 au cas où le chiffre 2 n'apparaît pas. Pour tout indice n compris entre 0 et $N-1$, la n^e case du champ `chiffres` vaut `true` si le chiffre g_n est non-nul et vaut `false` sinon.
Par exemple, les entiers 15 et 21 ont pour représentation gauche les variables `entier_15` et `entier_21` suivantes.

```
rg entier_15 = { .position = -1,
                 .chiffres = {0, 0, 0, 1, 0, 0, 0, 0}};
rg entier_21 = { .position = 1,
                 .chiffres = {0, 1, 0, 1, 0, 0, 0, 0}};
```

Q. 5 Formaliser sous la forme d'un ou de plusieurs invariants le fait qu'une valeur C de type `rg` est la représentation gauche d'un entier.

Q. 6 Écrire une fonction C `int rg_to_int(rg g)`, qui renvoie l'entier dont g est la représentation gauche. On supposera que l'invariant de la **Q. 5** est satisfait.

Incrémentation et décrémentation

Nous proposons l'algorithme suivant :

ALGORITHME MYSTÈRE :

Entrée : Représentation gauche $g = g_{N-1} \cdots g_0$ d'un certain entier m.

Effet :

- Si aucun des chiffres $(g_n)_{0 \leqslant n < N}$ ne vaut 2, changer le chiffre g_0 en $g_0 + 1$.
- Sinon, en notant p la position du chiffre 2, changer le chiffre g_p en 0 et le chiffre g_{p+1} en $g_{p+1} + 1$.

Nous notons m' l'entier dont la représentation gauche est g après exécution de l'algorithme.

FIGURE 2 – Un algorithme

Q. 7 Vérifier que l'invariant de la **Q. 5** n'est pas rompu par l'algorithme mystère (*cf. figure 2*). Avec les notations m et m' de la figure 2, caractériser, en fonction de l'entier m, la valeur de l'entier m'.

Q. 8 Écrire une fonction C `bool rg_incr(rg *s)` dont la spécification suit :

Précondition : La variable s est un pointeur vers la représentation gauche d'un entier m.

Effet : La valeur pointée par s est modifiée afin de représenter l'entier $m + 1$, si possible.

Valeur de retour : Booléen `true` si l'incrémentation de m peut avoir lieu et `false` si un débordement se produit car $m + 1$ n'est pas représentable sur le même nombre de chiffres.

Q. 9 Calculer la complexité en temps dans le pire des cas de la fonction `rg_incr` en fonction de N. Comparer avec la complexité de la même opération sur la représentation standard.

Q. 10 Écrire une fonction C `bool rg_decr(rg *s)` dont la spécification suit :

Précondition : Le pointeur s désigne la représentation gauche d'un entier m.

Effet : La valeur pointée par s est modifiée afin de représenter l'entier $m - 1$.

Valeur de retour : Booléen `true` si la décrémentation de m peut avoir lieu et `false` si un débordement se produit car m est nul.

Il est recommandé d'expliquer son intention avant de donner son code.

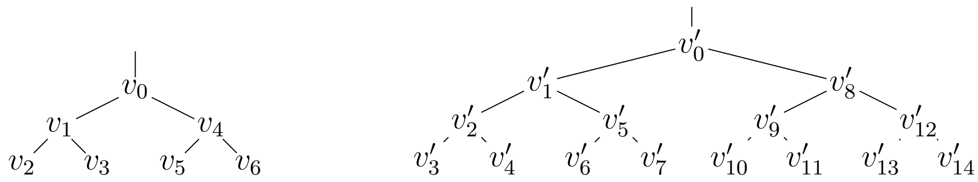


FIGURE 3 – Arbres binaires parfaits de hauteur 2 et de hauteur 3.

3. Arbres binaires parfaits

Opérations sur les arbres binaires

Indication C : Nous représentons les arbres binaires à valeurs entières au moyen du type C `arb` suivant, qui est un pointeur vers une structure contenant la valeur entière dans le champ `valeur` et les deux fils dans les champs `fils_g` et `fils_d`. L'arbre vide se représente par le pointeur `NULL`.

```
typedef struct Noeud *arb;
struct Noeud {
  int valeur;
  arb fils_g;
  arb fils_d;
};
```

L'arbre vide est, par convention, de hauteur -1.

Q. 11 Écrire une fonction C `int hauteur(arb a)` qui calcule la hauteur de l'arbre a.

Q. 12 Écrire une fonction C `arb noeud(int v, arb ag, arg ad)` qui construit un arbre dont la racine a pour étiquette v, dont le fils gauche est a_g et le fils droit est a_d. Dans cette question, il est demandé de se défendre, par le truchement d'une assertion, de toute erreur liée à un échec d'allocation dynamique de mémoire.

Définition : Un *arbre binaire parfait*, ou simplement *arbre parfait*, est un arbre binaire dont tous les nœuds internes ont exactement deux fils, dont toutes les feuilles sont à la même profondeur et dont les nœuds sont étiquetés par des valeurs entières.
Des exemples d'arbres binaires parfaits sont donnés figure 3.

Q. 13 Démontrer que tout arbre binaire parfait de hauteur n possède $2^{n+1} - 1$ nœuds.

Q. 14 Écrire une fonction C `bool est_parfait(arb a, int n)` dont la spécification suit :

Précondition : Le pointeur a désigne la racine d'un arbre binaire (dont les nœuds sont bien tous distincts).

Valeur de retour : Booléen `true` si l'arbre pointé est parfait de hauteur n et `false` sinon.

Q. 15 Calculer la complexité en temps dans le pire des cas de l'exécution de `est_parfait(a, n)` en fonction de l'entier n.

Opérations sur les arbres parfaits

Définition : Soient S une structure de données qui permet de stocker une collection d'entiers et t le cardinal de S. Nous disons que la structure de données S est à *accès direct* s'il existe une manière systématique de numéroter chaque élément de S entre 0 et $t-1$ et s'il existe deux primitives, `acces` et `modif`, qui permettent respectivement de consulter et de modifier n'importe quel élément de S en temps logarithmique en t à partir seulement de son numéro.

Nous souhaitons vérifier qu'un arbre parfait est une structure de données à accès direct. Nous numérotons les éléments d'un arbre parfait en utilisant l'ordre préfixe de gauche à droite de l'arbre (comme illustré dans la figure 3).

Q. 16 Écrire une fonction C `arb arb_trouve(arb a, int n, int k)` ainsi spécifiée :

Précondition : Le pointeur a désigne la racine d'un arbre binaire parfait de hauteur n. L'entier k satisfait les inégalités $0 \leqslant k < 2^{n+1} - 1$.

Valeur de retour : Pointeur vers le k^{e} nœud de l'arbre dans l'ordre préfixe.

Il est rappelé que la racine d'un arbre est à profondeur 0. Nous donnons la formule de sommation, valable pour tout réel $a \neq 1$ et tout entier t,

$$\sum_{k=0}^{t} k \cdot a^k = \frac{((a-1)t-1)a^{t+1}+a}{(a-1)^2}$$

Q. 17 Nous appelons P la profondeur d'un nœud choisi aléatoirement et uniformément parmi les $2^{n+1} - 1$ nœuds d'un arbre binaire parfait de hauteur n. Calculer l'espérance $\mathbb{E}(P)$ de la variable aléatoire P.

Q. 18 Déterminer la complexité moyenne en temps de l'instruction `arb_trouve(a, n, k)` lorsque la hauteur n et l'arbre a sont fixés et le numéro k est choisi aléatoirement et uniformément dans l'intervalle $[\![0, 2^{n+1} - 2]\!]$.

Q. 19 Écrire une fonction C `int arb_acces(arb a, int n, int k)` qui renvoie la k^{e} valeur de l'arbre a de hauteur n ainsi qu'une fonction C `void arb_modif(arb a, int n, int k, int v)` qui remplace la k^{e} valeur de l'arbre a de hauteur n par la valeur v. Dire finalement si la structure de données arbre binaire parfait est à accès direct.

4. Listes gauches

Les arbres binaires parfaits seuls ne permettent de représenter que des collections d'entiers dont le cardinal est de la forme $2^{n+1} - 1$ où n est entier naturel. Afin de représenter des collections de cardinal quelconque, nous introduisons des suites d'arbres parfaits, aux hauteurs strictement croissantes et savamment choisies.

Définition : Nous appelons *liste binaire gauche de cardinal* m *sur* N *arbres* la structure de données constituée de $N + 1$ arbres binaires parfaits $((a_n)_{0 \leqslant n < N}, e)$ comme suit. Nous décomposons l'entier m selon sa représentation binaire gauche sur N chiffres : $g_{N-1} \cdots g_0$. Pour tout indice n compris entre 0 et $N - 1$, si le chiffre g_n n'est pas nul, l'arbre binaire parfait a_n est un arbre de hauteur n, sinon il s'agit de l'arbre vide. Si le chiffre 2 apparaît parmi les chiffres de la représentation gauche de m et si l'indice p est sa position, alors l'arbre e est un arbre binaire parfait de hauteur p, sinon l'arbre e est l'arbre vide. De manière plus courte, nous parlons simplement de *liste gauche*.
Une liste binaire gauche de cardinal m sur N arbres permet de stocker une collection de m éléments entiers. Il apparaît qu'avec N arbres, une liste binaire gauche peut stocker jusqu'à M_N entiers (où l'entier M_N a été introduit à la **Q. 2**) et que m est inférieur à M_N.

Indication C : Nous adoptons le type C suivant.

```c
struct ListeGauche {
  int hauteur_e;
  arb extra;
  int nb_arbres;
```

```c
  arb *arbres;
};
typedef struct ListeGauche lg;
```

Le champ `hauteur_e` contient la hauteur p de l'arbre exceptionnel e. Le champ `extra` désigne la racine de l'arbre exceptionnel e. Le champ `nb_arbres` contient l'entier N. Enfin, le champ `arbres` désigne un tableau de N pointeurs vers les arbres $(a_n)_{0\leqslant n<N}$ qui a été alloué dynamiquement.

Opérations simples sur les listes gauches

Q. 20 Écrire une fonction C `lg lg_init(int N)` qui renvoie une liste gauche de cardinal nul sur N arbres.

Q. 21 Écrire une fonction C `int lg_card(lg l)` qui calcule le cardinal m de la liste gauche ℓ.

Définition : Afin de numéroter l'ensemble des éléments d'une liste gauche $\ell = ((a_n)_{0\leqslant n<N}, e)$, nous parcourons d'abord les éléments de l'arbre exceptionnel e dans l'ordre préfixe, puis nous parcourons les éléments des arbres $a_0, a_1, \ldots, a_{N-1}$ dans l'ordre préfixe. Les numéros sont attribués aux valeurs rencontrées par ordre de première rencontre.

Q. 22 Écrire une fonction C `arb lg_trouve(lg l, int k)` qui renvoie le k^{e} nœud de la liste gauche ℓ. On supposera que k est un indice valide $(0 \leqslant k < m)$.

Q. 23 Calculer la complexité en temps dans le pire des cas de `lg_trouve(l, k)` en fonction de la capacité maximale M_N de la liste gauche ℓ.

Q. 24 Écrire une fonction C `int lg_acces(lg l, int k)` qui renvoie la k^{e} valeur de la liste gauche ℓ ainsi qu'une fonction C `void lg_modif(lg l, int k, int v)` qui remplace la k^{e} valeur de la liste gauche ℓ par la valeur v.

Ajout et suppression en tête de liste gauche

Q. 25 Soient v une valeur entière et $\ell = ((a_n)_{0\leqslant n<N}, e)$ une liste gauche de cardinal m, avec $m < M_N$, qui contient les éléments $v_1, \ldots, v_m$ dans cet ordre. Décrire, en fonction de la liste gauche ℓ, la liste gauche $\ell' = ((a'_n)_{0\leqslant n<N}, e')$ de cardinal $m+1$ dont les éléments sont $v, v1, \ldots, v_m$ dans cet ordre. En déduire le principe d'une fonction C `bool lg_empile(int v, lg l)` réalisant l'insertion de la valeur v en tête de la liste gauche ℓ où le booléen résultat vaut `true` si l'ajout a eu lieu et vaut `false` si un débordement de capacité de la liste se produit. On ne demande pas le code complet de cette fonction.

Q. 26 Déterminer la complexité en temps dans le pire des cas de la fonction `lg_empile`.

Q. 27 Donner le principe d'une fonction C `bool lg_depile(int *w, lg l)` réalisant le retrait de l'élément de tête de la liste gauche ℓ et son affectation à l'adresse w. Le booléen résultat vaut `true` si le retrait a eu lieu et vaut `false` si l'opération a échoué en raison d'une liste vide. On ne demande pas le code complet.

Q. 28 Donner la complexité en temps dans le pire des cas de la fonction `lg_depile`.

Q. 29 Discuter la possibilité d'obtenir une complexité plus faible à la **Q. 28**, quitte à modifier légèrement la définition du type `lg`.

Q. 30 Soit N un entier et $\ell = ((a_n)_{0\leqslant n<N}, e)$ une liste gauche. Discuter la possibilité de modifier en place et avec une faible complexité en temps la liste gauche ℓ de sorte que le nombre d'arbres N devienne $N+1$ et que les mêmes éléments deumeurent dans la liste.

Les listes binaires gauches, ou *skew binary lists*, ont été inventées par Eugene Myers en 1983.

Rappels de programmation en C

L'expression `1 << n` représente le décalage de la valeur 1 sur n bits : elle a pour valeur l'entier 2^n. Cette opération a un coût algorithmique constant.

Le type `pthread_t` désigne des fils d'exécution.

$\therefore$ Fin du sujet $\therefore$