TD – Partie 1. Structures de données

1. Types abstraits de données et structures linéaires

Exercice 1. *Algorithmes sur les tableaux*
Pour chacun des problèmes suivants, donner un algorithme pour le résoudre en utilisant le TAD tableau vu en cours, prouver la correction de l'algorithme et analyser sa complexité.

1. Calculer le deuxième plus grand élément d'un tableau d'entiers de taille ≥ 2. *Pour éviter toute ambiguité sur la définition, on suppose que tous les éléments sont distincts.*
2. Retourner un tableau, c'est-à-dire produire le tableau $[T_{[n-1]}, T_{[n-2]}, \dots, T_{[0]}]$ où $n = \text{TAILLE}(T)$.

Exercice 2. *Algorithmes sur les listes*
Pour chacun des problèmes suivants, donner un algorithme pour le résoudre en utilisant le TAD liste vu en cours, prouver la correction de l'algorithme et analyser sa complexité.

1. Calculer la longueur de la liste.
2. Calculer le maximum d'une liste non vide d'entiers.
3. Retourner une liste, c'est-à-dire produire la liste qui contient les mêmes éléments mais en sens inverse.

Exercice 3. *Pile, file, file à priorité*
Décrire comment réaliser une pile et une file à l'aide d'une file de priorité.

Exercice 4. *File et tableaux circulaires*
On rappelle que dans la réalisation d'une file à l'aide d'un tableau T de taille N, on stocke deux indices (d, f) pour indiquer que les éléments de la file sont dans le sous-tableau $T_{[d,f[}$. Après un certain nombre d'appels à ENFILER et DÉFILER, on peut avoir d et f proches de N : cela signifie que les premières cases du tableau sont inutilisées. Pour éviter ce gâchis, on peut utiliser T de manière *circulaire* : si on atteint $f = N$, on *boucle* en utilisant le début du tableau.

1. On utilise un tableau de taille $N = 4$, et on effectue la suite d'opérations suivantes : ENFILER(F, a), ENFILER(F, b), ENFILER(F, c), DÉFILER(F), ENFILER(F, d), DÉFILER(F), DÉFILER(F), ENFILER(F, e), ENFILER(F, f).

i. Si on utilise le tableau de manière *non-circulaire*, quelle opération renvoie une erreur ?

ii. Décrire l'état du tableau après chaque opération s'il est utilisé de manière circulaire ?

iii. Exprimer, en français, la condition pour qu'il n'y ait pas d'erreur dans les deux cas : tableau circulaire et non-circulaire.

2. Écrire les algorithmes pour les quatre opérations de File, en utilisant un tableau de manière circulaire.

Exercice 5. *File avec deux piles*

On souhaite réaliser le TAD File à l'aide de deux piles. On définit donc $F = (E, S)$ où E est la *pile d'entrée* et S la *pile de sortie*. Quand on enfile un élément dans F, on l'empile sur E. Pour défiler, si S est vide, on transfère tout le contenu de E dans S, puis ensuite on dépile un élément de S.

1. Écrire les opérations NVFILE, ESTVIDE et ENFILER à partir des opérations sur les piles. Quelles sont leurs complexités ?

2. **i.** Écrire l'opération DÉFILER à partir des opérations sur les piles.

ii. Justifier que la file obtenue suit bien la politique FIFO.

iii. Quelle est la complexité dans le *pire des cas* de l'opération DÉFILER ? Et dans le *meilleur des cas* ?

3. On souhaite montrer que les opérations ENFILER et DÉFILER ont une complexité *amortie* $O(1)$, c'est-à-dire que partant d'une file vide, toute suite de N opérations ENFILER/DÉFILER coûte *au total* $O(N)$ (d'où $O(1)$ par opération).

i. Supposons qu'il y a m opérations ENFILER et n opérations DÉFILER ($m + n = N$). Quelle relation doivent satisfaire m et n ?

On change de point de vue : au lieu de se concentrer sur les opérations, on se concentre sur les éléments qui sont dans la file. Soit x un élément qui passe dans F pendant la suite d'opérations.

ii. Combien de fois, au plus, l'opération EMPILER est appelée sur x ?

iii. Si x est extrait de F, combien de fois x a-t-il été renvoyé par un appel à DÉPILER ? Et si x reste dans F à la fin des opérations ?

iv. En déduire qu'une suite de N opérations ENFILER/DÉFILER effectue au plus $3N$ appels à EMPILER/DÉPILER.

2. Arbres binaires et tas

Exercice 6. *Algorithmes sur les arbres binaires*

Pour chacun des problèmes suivants, donner un algorithme pour le résoudre en utilisant le TAD arbre binaire vu en cours, prouver la correction de l'algorithme et analyser sa complexité.

1. Calculer le nombre de feuilles d'un arbre binaire.
2. Calculer le symétrique d'un arbre binaire par rapport à un axe vertical. *Tout enfant gauche devient un enfant droit, et réciproquement.*

Exercice 7. *Parcours d'arbre binaire*

1. Écrire l'algorithme de parcours en profondeur préfixe d'un arbre binaire sous forme itérative. *Utiliser une pile pour stocker les nœuds à visiter.*
2. **i.** Écrire l'algorithme de parcours en largeur d'un arbre binaire. *Utiliser une file pour stocker les nœuds à visiter.*
ii. Justifier que si l'arbre est quasi-complet, l'ordre de traitement des sommets (en commençant à 0) fournit à chaque sommet un numéro n_x tel que $n_{\text{PARENT}(x)} = \lfloor (n_x - 1)/2 \rfloor$.

Exercice 8. *Évaluation d'expressions arithmétiques*

On souhaite évaluer des expressions arithmétiques de la forme $(1 + ((2 + 3) \times (4 \times 5)))$, complètement parenthésées et ne contenant que des opérateurs binaires. On utilise un algorithme très simple dû à E. W. Dijkstra. Premièrement, on suppose (ou vérifie) que l'expression est bien parenthésée. Ensuite, on utilise deux piles : celles des opérandes et celle des opérateurs ($+$, $\times$, …). On ignore les parenthèses ouvrantes. On parcourt l'expression et on applique les règles suivantes :

1. si on lit un entier, on l'empile sur la pile des opérandes ;
2. si on lit un opérateur, on l'empile sur la pile des opérateurs ;
3. si on lit une parenthèse fermante, on dépile un opérateur et deux opérandes ; on effectue le calcul et on empile le résultat sur la pile des opérandes.

1. Appliquer l'algorithme sur l'expression ci-dessus.
2. Écrire l'algorithme. *Utiliser une boucle de la forme « Pour chaque caractère c de l'expression : » et des tests « ESTPARENTHÈSEOUVRANTE(c) », etc.*
3. **i.** Justifier qu'une expression complètement parenthésée décrit de manière unique un arbre binaire.
ii. Dessiner l'arbre correspondant à l'expression ci-dessus. Appliquer les trois parcours en profondeur sur l'arbre obtenu. À quel parcours correspond l'expression parenthésée ?
iii. Interpréter l'algorithme en terme de modification d'arbre binaire. En déduire sa correction.

Exercice 9. *Tas*

1. Soit T un tas à n éléments. Les affirmations suivantes sont-elles correctes ?
   - **i.** $T_{[0]}$ est l'élément maximal.
   - **ii.** $T_{[n-1]}$ est l'élément minimal.
   - **iii.** L'élément minimal est dans une case d'indice $\geq (n-1)/2$.
2. - **i.** Parmi les tableaux suivants, identifier les tas : $[25, 18, 13, 7, 12, 10, 6, 9, 7]$, $[17, 11, 7, 13, 10, 4]$, $[21, 6, 12, 13, 7, 2, 4, 5, 8]$. *Dessiner l'arbre quasi-complet représenté par le tableau, puis vérifier si c'est un tas.*
   - **ii.** Pour ceux qui ne le sont pas, identifier le(s) nœud(s) mal placé(s), et appliquer soit MONTER soit DESCENDRE pour obtenir un tas.
3. Montrer qu'un tableau trié par ordre décroissant est un tas.
4. Monter que les feuilles d'un tas à n éléments sont exactement les nœuds de numéro $\lfloor n/2 \rfloor$ à $n-1$.

Exercice 10. *Créer un tas*

1. Écrire un algorithme TASSER qui prend en entrée un tableau T et réordonne ses éléments pour que T représente un tas. *S'inspirer de* TRITAS.
2. Exécuter l'algorithme TASSER sur le tableau $[1, 2, 3, 4, 5, 6, 7]$.
3. Montrer que la complexité de TASSER est $O(n \log n)$.
4. *(bonus)* On veut montrer que TASSER a une complexité $O(n)$.
   - **i.** Pourquoi cela ne contredit-il pas la question 3 ?
   - **ii.** Montrer que pour $h < \log n$, TASSER appelle DESCENDRE $\lceil n/2^{h+1} \rceil$ fois sur des tas de hauteur h.
   - **iii.** En déduire que la complexité de TASSER est $O\left(\sum_{h=0}^{\lfloor \log n \rfloor} nh/2^h\right)$.
   - **iv.** En déduire le résultat. *Indication. Montrer que* $\sum_{h=0}^{+\infty} h/2^h = 2$.

Exercice 11. *Files de priorité et tri*

1. Supposons qu'on dispose d'une réalisation du TAD file de priorité.
   - **i.** Écrire un algorithme de tri basé sur la file de priorité.
   - **ii.** Analyser la complexité de l'algorithme en fonction des complexités des opérations de la file de priorité.
2. On s'intéresse aux algorithmes obtenus en fonction de la réalisation utilisée pour le TAD file de priorités. Pour chacune des réalisations ci-dessous, donner la complexité obtenue pour l'algorithme et identifier à quel algorithme classique correspond cette méthode.
   - **i.** Tableau non trié.
   - **ii.** Tableau trié.
   - **iii.** Tas.

3. Tableaux dynamiques et ABR

Exercice 12. *Analyse amortie*

1. On considère une pile, avec une opération supplémentaire : MULTIDÉPILER(P, k) dépile les k éléments en haut de la pile et renvoie le $k^{\text{ème}}$. S'il y a moins de k éléments dans la pile, ils sont tous dépilés et le dernier est renvoyé.
   i. Écrire l'algorithme MULTIDÉPILER et analyser sa complexité en nombre d'appels aux opérations de base de la pile.
   ii. On effectue une suite de n opérations EMPILER, DÉPILER et MULTIDÉPILER. Justifier que la complexité amortie par opération est constante.
2. On considère un compteur binaire, qu'on voit comme un tableau de n bits $C_{[0]}, \ldots, C_{[n-1]}$, qui représente l'entier $c = \sum_{i=0}^{n-1} C_{[i]} 2^i$. Le compteur possède une unique instruction qui incrémente la valeur de c.
   i. Combien de bits doivent être modifiés, dans le pire cas, lors d'un incrément ? *Donner un exemple qui atteint le pire cas.*

   On incrémente le compteur de $c = 0$ à $c = N$, et on note $\ell = 1 + \lfloor \log N \rfloor$ le nombre de bits de N.
   ii. Montrer que pour $k \leq \ell$, le $k^{\text{ème}}$ bit est modifié $\lfloor N/2^k \rfloor$ fois au cours des N incréments.
   iii. En déduire que le coût amorti par incrément est $O(1)$.

Exercice 13. *Tableaux dynamiques modifiés*

On rappelle qu'un tableau dynamique $\mathcal{T}$ est représenté par un tableau T de taille N et un nombre d'éléments n qui vérifie à tout instant $n \leq N \leq 4n$. On souhaite utiliser moins de place superflue, et avoir à tout instant $N \leq 2n$.

1. Modifier les algorithmes AJOUTER et SUPPRIMER afin de garantir $N \leq 2n$. *On pourra par exemple faire en sorte qu'après redimensionnement,* $n \simeq \frac{2}{3}N$.
2. Adapter l'analyse pour obtenir le coût amorti par opération. *On souhaite une valeur précise en nombre d'affectations pour comparer à la solution initiale.*
3. *(bonus)* On généralise : on souhaite garantir un remplissage minimal αN, et on vise un remplissage βN après redimensionnement. Exprimer la complexité amortie en fonction de α et β, et trouver la valeur optimale de β en fonction de α.

Exercice 14. *Tableau avec recherche efficace*

On souhaite une structure de données pour stocker un ensemble E de n entiers (distincts), avec deux opérations : INSÉRER(E, x) insère x dans E (s'il n'y était pas déjà) et RECHERCHER(E, x) renvoie VRAI si $x \in E$, FAUX sinon.

1. On utilise un tableau dynamique trié.

i. Quel est le coût de RECHERCHER ? *Nommer l'algorithme.*
ii. Quel est le coût d'INSÉRER ? *Expliciter l'algorithme.*

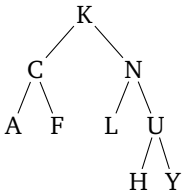
2. On veut améliorer la solution précédente. Pour stocker n éléments, on écrit $n = \sum_{i=0}^{k-1} n_i 2^i$ avec $n_i \in \{0, 1\}$) et on on utilise k tableaux $T^0, \ldots, T^{k-1}$, où T^i est de taille 2^i. Chaque tableau est soit plein (si $n_i = 1$) soit vide (si $n_i = 0$). De plus, chaque tableau plein est trié par ordre croissant (mais aucun ordre n'est imposé entre les éléments de tableaux différents).

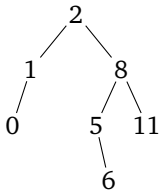
i. Montrer que le nombre total d'éléments contenu dans les tableaux est bien n et donner une borne sur la place totale utilisée, en fonction de n.
ii. Décrire un algorithme pour RECHERCHER, et analyser sa complexité.
iii. Décrire un algorithme pour INSÉRER, et analyser sa complexité.
iv. Calculer le coût amorti par insertion lorsqu'on part d'un ensemble vide et qu'on insère n éléments. *On peut réutiliser certains calculs effectués pour le compteur binaire.*

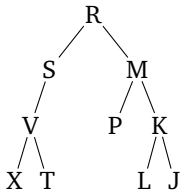
Exercice 15. *Manipuler des ABR*

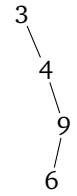
1. Dessiner des ABR de hauteur 6, 3 et 2 dont les nœuds ont le même ensemble de clés : $\{2, 4, 6, 8, 9, 11, 13\}$. Dans chaque cas, donner l'ordre d'insertion qui produit l'arbre dessiné.

2. Parmi les arbres suivants, déterminer ceux qui sont des ABR. *Seules les clés sont représentées.*



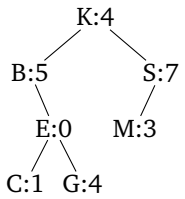






3. Dessiner l'état de l'ABR ci-contre après chacune des opérations suivantes (appliquées dans l'ordre) :

1 Insérer $(Q, 8)$
2 Supprimer B
3 Insérer $(A, 5)$
4 Supprimer K



4. **i.** Appliquer le parcours infixe sur l'ABR de la question précédente. Que remarque-t-on ?
ii. Énoncer et démontrer le résultat observé.

Exercice 16. *Algorithmes sur les ABR*

1. **i.** Écrire un algorithme qui renvoie la valeur associée à la clé maximale dans un ABR. Quelle est sa complexité ?
   ii. Que doit-on changer pour obtenir la valeur associée à la clé minimale ?
2. **i.** Écrire un algorithme SUIVANT qui, étant donné un ABR $\mathcal{A}$ et une clé k, renvoie la valeur associée à la plus petite clé de $\mathcal{A}$ qui est supérieure à k. *Si aucune clé n'est supérieure à k, on renvoie une erreur.*
   ii. Que doit-on changer pour obtenir PRÉCÉDENT ?
3. Écrire un algorithme qui prend en entrée un ABR $\mathcal{A}$ et un entier k et affiche les valeurs de nœuds de $\mathcal{A}$ dont les clés sont supérieures à k. *Minimiser le nombre de tests effectués.*

Exercice 17. *File de priorité*

Expliquer comment réaliser une file de priorité avec un ABR, et indiquer la complexité des opérations.

Exercice 18. *Arbre radix*

Un arbre radix, ou trie, est une autre réalisation possible du TAD dictionnaire dans le cas où les clefs sont des mots binaires. C'est un arbre binaire, dont chaque nœud possède une clé de la manière suivante : la racine possède la clé vide (mot de longueur 0) ; si un nœud a la clé c, son enfant gauche (s'il existe) a la clé $c0$ et son enfant droit (s'il existe) la clé $c1$. Chaque nœud est soit vide, soit contient une valeur. S'il est vide, la clé de ce nœud n'est pas dans le dictionnaire. Sinon, la valeur associée à sa clé est la valeur du nœud.

1. **i.** Dessiner l'arbre radix pour le dictionnaire suivant : $\{0 : x, 10 : r, 011 : b, 100 : h, 1011 : p\}$.
   ii. Exprimer la hauteur de l'arbre en fonction des clés contenues dans le dictionnaire.
   iii. Justifier que les feuilles vides sont inutiles.
2. On réalise le TAD dictionnaire à l'aide d'un arbre radix. L'opération NVDICTIONNAIRE est simplement NVARBRE des arbres binaires. On suppose qu'il existe une valeur spéciale $\emptyset$ pour indiquer qu'un nœud est vide.
   i. Proposer un algorithme pour RECHERCHER(D, c) et analyser sa complexité.
   ii. Proposer un algorithme pour INSÉRER(D, c, v) et analyser sa complexité.
   iii. Proposer un algorithme pour SUPPRIMER(c) et analyser sa complexité. *Attention, l'arbre radix ne doit pas contenir de nœud inutile après suppression.*
3. Montrer comment utiliser l'arbre radix pour trier un ensemble de mots binaires par ordre lexicographique, en temps $O(N)$ où N est la somme des longueurs des mots de l'ensemble.

4. Tables de hachage

Rappels

Espérance de X :	$\mathbb{E}[X] = \sum_{v \in V} v \times \Pr[X = v]$
Linéarité de l'espérance :	$\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$
Inégalité de Markov :	$\Pr[X \geq \lambda \mathbb{E}[X]] \leq \frac{1}{\lambda}$
Borne de l'union :	$\Pr[E \vee F] \leq \Pr[E] + \Pr[F]$
Probabilité conditionnelle :	$\Pr[E \mid F] = \Pr[E \wedge F] / \Pr[F]$

Exercice 19. *Tables sans collision*

Une fonction de hachage $h : \{0, \ldots, N-1\} \to \{0, \ldots, m-1\}$ est *sans collision* pour un ensemble $K \subset U$ si pour tout $x, y \in K$, $h(x) \neq h(y)$.

1. Donner une condition nécessaire et suffisante sur K pour qu'il existe une fonction de hachage sans collision pour K.
2. On s'intéresse à l'espérance du nombre de collisions, en fonction de m et $n = |K|$.
   i. Calculer l'espérance dans le modèle idéalisé où h est uniforme parmi les fonctions de $\{0, \ldots, N-1\}$ dans $\{0, \ldots, m-1\}$.
   ii. Calculer l'espérance dans le modèle universel où h est uniforme dans un ensemble universel.
3. On suppose maintenant que $m = n^2$. Montrer, dans les deux modèles idéalisé et universel, qu'avec probabilité $\geq \frac{1}{2}$, h est sans collision pour K.
4. On veut stocker n couples (clé,valeur) dans une table de hachage, sans avoir à gérer les collisions (pas de chaînage ni d'adressage ouvert). Pour cela, on utilise une table de taille $m = n^2$. On tire une première fonction de hachage et on insère tous les éléments dans la table. S'il y a une collision, on recommence avec une nouvelle fonction de hachage, etc.
   i. Quelle est la probabilité que la première fonction de hachage soit sans collision ?
   ii. Quelle est l'espérance du nombre de fonctions de hachage à tester pour en trouver une sans collision ?

Exercice 20. *Hachage parfait*

Le hachage parfait résout le problème des collisions en mettant, dans chaque case non vide de la table, une autre table de hachage. La structure est alors constituée d'une *table principale* $\mathcal{T} = (T, h)$, dont la case $T_{[i]}$ est soit vide, soit contient une *table secondaire* $\mathcal{T}^i = (T^i, h^i)$. Dans les tables secondaires, on n'accepte aucune collision (ni chaînage ni adressage ouvert) : pour cela, on utilise une table de taille $m_i = n_i^2$ pour stocker n_i éléments.

On se place dans un cadre *statique* en deux phases : 1. on initialise la table en insérant n couples (clé, valeur) ; 2. on effectue des appels à RECHERCHER uniquement. On utilise le modèle universel des fonctions de hachage.

1. Pendant l'initialisation, on crée chaque table secondaire $\mathcal{T}^i$ en tirant des fonctions de hachage h^i jusqu'à en avoir une sans collision pour les éléments de cette table. Justifier que l'espérance du nombre total de fonction de hachage à tirer pendant la phase d'initialisation est $O(n)$.
2. Justifier que lors de la phase de requêtes, chaque appel à RECHERCHER coûte $O(1)$ en pire cas.
3. On veut montrer que l'espérance de la taille *totale* de la table est $O(n)$. On compte le nombre total de cases : les cases de T et les cases de chaque T^i. On note K l'ensemble des clés insérées dans $\mathcal{T}$, et on définit les variables aléatoires $X_{k,i}$ pour $k \in K$ et $i \in \{0,\dots,n-1\}$ par $X_{k,i} = 1$ si $h(k) = i$ et 0 sinon.
   - **i.** Pour tout i, exprimer n_i en fonction des $X_{k,i}$, et exprimer la taille totale en fonction de n et des n_i.
   - **ii.** Montrer que $n_i^2 = \sum_{k\in K} X_{k,i} + \sum_{k_1\neq k_2} X_{k_1,i}X_{k_2,i}$.
   - **iii.** Montrer que $\sum_{i=0}^{n-1}\sum_k X_{k,i} = n$.
   - **iv.** Montrer que pour $k_1 \neq k_2$, $\mathbb{E}\left[\sum_{i=0}^{n-1} X_{k_1,i}X_{k_2,i}\right] \leq \frac{1}{n}$.
   - **v.** En déduire que $\mathbb{E}\left[\sum_{i=0}^{n}\sum_{k_1\neq k_2} X_{k_1,i}X_{k_2,i}\right] \leq n-1$.
   - **vi.** En déduire que l'espérance de la taille totale est $\leq 3n-1$.

Exercice 21. *Adressage ouvert*

Soit $\mathcal{T}$ une table de hachage T de taille m contenant n éléments dans laquelle les conflits sont résolus par *adressage ouvert*. On se place dans le cadre idéalisé suivant : on dispose de m fonctions de hachage h_0, …, h_{m-1} telles que pour tout k, $(h_0(k), h_1(k), \dots, h_{m-1}(k))$ est une permutation aléatoire de $\{0,\dots,m-1\}$, et pour $k_1 \neq k_2$ et $0 \leq i, j < m$, $h_i(k_1)$ est indépendant de $h_j(k_2)$.
On effectue une recherche *infructueuse* : on cherche une clé k dans la table qui n'y est pas. Soit X la variable aléatoire qui compte le nombre de cases visitées lors de cette recherche. On a démontré en cours que $\mathbb{E}[X] \leq m/(m-n)$.

1. On souhaite borner $\Pr[X \geq t]$ pour un t fixé. Pour cela, on définit pour tout j l'évènement E_j : « les j premières cases visitées sont occupées ».
   - **i.** Exprimer l'événement « $X \geq t$ » en fonction d'un E_i.
   - **ii.** Montrer que pour $j \geq 0$, $\Pr[E_{j+1}|E_j] = (n-j)/(m-j)$.
   - **iii.** En déduire que pour $1 \leq t \leq m$, $\Pr[X \geq t] \leq (n/m)^{t-1}$.

On étudie maintenant le scénario suivant : on part de la table vide (de taille m) et on insère successivement n clés, avec $n \leq m/2$. On rappelle qu'une insertion doit trouver une première case vide : c'est l'équivalent d'une recherche infructueuse.

2. On note X_i le nombre de cases visitées lors de la $i^{\text{ème}}$ insertion, et $X = \max_{1\le i\le n} X_i$.

i. Montrer que pour tout i, $\Pr[X_i > \lfloor 2\log n\rfloor] \le 1/n^2$.

ii. Montrer que $\Pr[X > \lfloor 2\log n\rfloor] < 1/n$.

iii. En déduire que l'espérance de X est $O(\log n)$. *Couper* $\mathbb{E}[X] = \sum_{t=0}^{n} t\Pr[X=t]$ *en deux sommes* ($t \le \lfloor 2\log n\rfloor$ *et* $t > \lfloor 2\log n\rfloor$) *et borner indépendamment chacune des deux.*

Exercice 22. *Une famille quasi-universelle*

Pour $w > 0$, soit I_w l'ensemble des entiers impairs entre 1 et $2^w - 1$. Pour $\ell < w$ et $a \in I_w$, on définit $h_a(x) = (ax \bmod 2^w) \operatorname{quo} 2^{w-\ell}$ où quo et mod désignent le quotient et le reste dans la division euclidienne. On s'intéresse à la famille $\mathcal{H}_{w,\ell} = \{h_a : a \in I_w\}$.

1. **i.** On écrit ax en binaire, avec les bits $b_0, b_1, \ldots$ Exprimer $h_a(x)$ en fonction des b_i. *Faire un dessin !*

ii. Montrer que $h_a(x) \in \{0, \ldots, 2^\ell - 1\}$ pour tout $x \ge 0$.

2. **i.** Montrer que pour tout x, $y \in I_w$, il existe un unique $a \in I_w$ tel que $ax \bmod 2^w = y$.

ii. En déduire que pour tout x, $y \in I_w$, $\Pr_a[ax \bmod 2^w = y] = 1/2^{w-1}$.

3. Soit $x < y$. Montrer que $h_a(x) = h_a(y)$ si et seulement si $h_a(y-x) = 0$ ou $h_a(y-x) = 2^\ell - 1$.

4. On va montrer que la famille $\mathcal{H}_{w,\ell}$ est quasi-universelle. On fixe pour cela deux entiers positifs $x < y < 2^w$, on pose $z = y - x$ et on écrit $z \bmod 2^w = q2^r$ où $q \in I_{w-r}$ et $r < w$.

i. Montrer que pour tout $v \in I_{w-r}$, $\Pr_a[az \bmod 2^w = v2^r] = 1/2^{w-r+1}$.

ii. En déduire que

$$\begin{cases} \Pr[h_a(z)=0] = \Pr[h_a(z) = 2^\ell - 1] = 0 & \text{si } r > w-\ell, \\ \Pr[h_a(z)=0] = 0 \text{ et } \Pr[h_a(z) = 2^\ell - 1] = 1/2^{\ell-1} & \text{si } r = w-\ell, \text{ et} \\ \Pr[h_a(z)=0] = \Pr[h_a(z) = 2^\ell - 1] = 1/2^\ell & \text{si } r < w-\ell. \end{cases}$$

iii. En déduire que pour tout $x \ne y$, $\Pr[h_a(x) = h_a(y)] \le 1/2^{\ell-1}$.

Exercice 23. *Case la plus remplie*

Soit $h : U \to \{0, \ldots, n-1\}$ une fonction de hachage aléatoire. On insère n clefs dans une table $\mathcal{T}$ de taille n à l'aide de h, en utilisant une résolution par chaînage. On souhaite connaître l'espérance de la case de $\mathcal{T}$ la plus remplie.

1. **i.** Soit j un indice entre 0 et $n-1$. Quelle est l'espérance du nombre d'élément en case j ?

ii. Pourquoi on ne peut pas conclure directement ?

2. Soit X_j la variable aléatoire qui compte le nombre d'éléments en case $T_{[j]}$.
   i. Montrer que $\Pr[X_j \geq k] \leq \binom{n}{k}\frac{1}{n^k} \leq 1/k!$.
   ii. En déduire que $\Pr[X_j \geq k] < \frac{1}{(k/2)^{k/2}}$.
3. On pose $k = \frac{4c\log n}{\log\log n}$, pour une certaine constante c.
   i. Justifier que $\frac{2c\log n}{\log\log n} \geq \sqrt{\log n}$ pour n suffisamment grand.
   ii. En déduire que pour n suffisamment grand, $\Pr[X_j \geq k] \leq \frac{1}{n^c}$.
   iii. En déduire que la probabilité que la case la plus remplie possède plus de $\frac{4c\log n}{\log\log n}$ éléments est $\leq 1/n^d$ pour une constante d à déterminer.
4. On note M le nombre d'éléments dans la case la plus remplie, et on veut borner $\mathbb{E}[M]$.
   i. Montrer que pour tout k, $\mathbb{E}[M] \leq k\Pr[M \leq k] + n\Pr[M > k]$.
   ii. En déduire que $\mathbb{E}[M] = O(\log n/\log\log n)$.

Exercice 24. *Filtres de Bloom*

Les filtres de Bloom permettent de stocker de manière très compressée un ensemble (statique, c'est-à-dire duquel on ne supprime jamais d'élément). La contrepartie est la présence de faux-positifs : le filtre répond parfois que x appartient à l'ensemble alors que ça n'est pas le cas.

Un filtre de Bloom pour un ensemble de taille n est donné par un entier m (la taille de la représentation) et k fonctions de hachage $h_1, \ldots, h_k$ indépendantes. Un ensemble X est représenté par un mot booléen w de taille m. L'ensemble vide est représenté par le mot $0\cdots0$. Pour insérer un nouvel élément x, on passe à 1 les k bits de w d'indices $h_1(x), \ldots, h_k(x)$. Un bit peut être mis plusieurs fois à 1. Pour tester si un élément y appartient à x, on vérifie si $w_{h_j(y)}$ vaut 1 pour $1 \leq j \leq k$: si c'est le cas, on répond « oui » et sinon on répond « non ».

Dans la suite, on suppose qu'on a construit la représentation w d'un ensemble X de taille n. On se place dans le modèle aléatoire pour les fonctions de hachage.

1. Laquelle des deux réponses de l'algorithme de recherche est toujours exacte ?
2. Montrer que le i-ème bit w_i de w vaut 1 si et seulement s'il existe $x \in X$ et j tels que $h_j(x) = i$.
3. Quelle est la probabilité p que le i-ème bit de w soit égal à 0 ? *On rappelle qu'on se place dans le modèle aléatoire, et que la probabilité dépend du choix des fonctions de hachage.*

On fait maintenant l'hypothèse qu'une fraction p des bits de w sont à 0.

4. Pourquoi cette hypothèse ne découle pas de la question précédente ?
5. Soit $y \notin X$. Quelle est la probabilité d'obtenir un faux-positif, c'est-à-dire que l'algorithme de recherche réponde « oui » sur l'entrée y ?
6. Montrer qu'en prenant $k = m\ln 2/n$, cette probabilité est exponentiellement petite. *On pourra utiliser, entre autres, que* $1 - x \geq e^{-2x}$ *pour* $x \leq 1/2$.