

ALGO6

④ TD2 ALGO6 - TD2 - Séance 3 : aléatoire et simulation.

Exercice 1.

① nextInt(6) + 1
 ② string pile (double p){
 int a = nextDouble();
 if (a > p){
 return face;
 } else {
 return pile;
 }
 }

③ a = nextInt(6)
 if a
 switch (a) {
 case 0, 2, 4 : return "a";
 case 1, 3 : return "n";
 case 5 : return "s";
 }

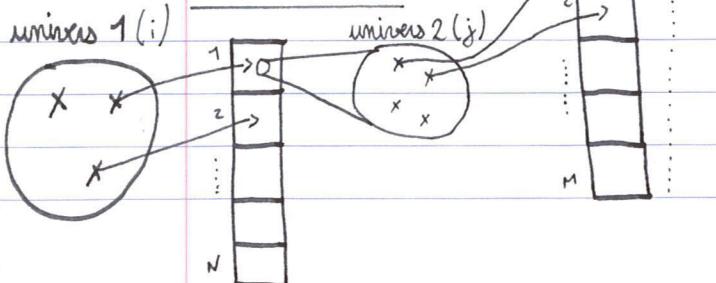
Génération testée

① $P(\text{we, shall}) = 1$
 ② $P(\text{fight, im}) = \cancel{2/7}$ 4/7
 $P(\text{fight, on}) = \cancel{2/7}$ 3/7

$$P(\text{fight, } j) \forall j \notin \{\text{im, on}\} = 0$$

$$\textcircled{3} P(\text{im, } j) \forall j = \longrightarrow P(\text{im, France}) = 1/5$$

$$\textcircled{4} P(i, j) \forall j, \forall i = cf \text{ exo 3-} \quad P(\text{im, the}) = 4/5$$

Exercice 3 -

```

hashmap < string, hashmap < string, int >> Pij = new hashmap;
while (wordi > hasNext) { // wordi not comment.
    string wordj = wordi.next();
    if (Pij.containsKey(wordi)) {
        if (Pij.get(wordi).containsKey(wordj)) {
            int c = Pij.get(wordi).get(wordj) + 1;
            Pij.get(wordi).put(wordj, c);
        } else { Pij.get(wordi).put(wordj, int(1)); }
    } else {
        Pij.put(wordi, new Hashmap());
        Pij.get(wordi).put(wordj, new int(1));
    }
    wordi = wordj;
}

```

Exercise 4-

```

i = 0; gen = new Random();
start = "We";
while i < 100 {
    i = i + 1;
    double Ni = 0
    for (String sw; Pij.get(stw1).get(stw2).keySet().size() == 0; Ni = Ni + Pij.get(stw1).get(sw),
    }
    int m = gen.nextInt(Ni);
    for (String sw, Pij.get(start).keySet()) {
        if (sum < m) {
            sum = sum + Pij.get(stw1).get(sw);
        } else {
            stw2 = sw;
            break;
        }
    }
    stw1 = stw2;
    print(stw, " ")
}

```

ALGO6

① CM Algorithmes randomisés

Idee : Utiliser un générateur aléatoire pour résoudre un problème.

Exemple : coupe minimale en TD

pour i de 1 à m faire

generer une coupe
L actualiser la coupe min

proba d'obtenir une coupe min $\geq \frac{2}{m(m-1)}$

Exemple : Bogo sort

Tant que paquet non-trié

L mélanger uniformément (aléatoirement)

pour $m = 32$

Temps moyen de tri

Tri rapide

TD2 - analyse du pire cas $\mathcal{O}(m^c)$

" " meilleur cas $\Omega(m \log n)$

analyse en moyenne : uniformité sur le choix de l'entrée.

coût moyen $\mathcal{O}(m \log n)$

Idee : faire un mélange préalable du tableau -

L randomisation de l'algorithme .

On peut aussi randomiser durant l'exécution du tri :

choisir un pivot

segmenter $E < E >$

fusion tri(E_1) pivot tri(E_2)

→ uniformément dans E . On s'affranchi de la donnée.

Test de primalité

Exemple : Protocole RSA : basé sur un produit de deux nombres premiers très grands p, q .

$31 \times 13 = 403$: difficile à factoriser en facteurs premiers.

On cherche à obtenir les entiers premiers à partir de p x q.

Crible d'Eratosthène (III avant JC)

2	3	4	5	6	7	8	9	10	11	12	13
14	15	16	17	18	19	20	21	22	23	24	25
26	27	28	29	30	31						

→ on passe sur tout les entiers, ce qui donne une complexité un peu trop grande.

Théorèmes sur les nombres premiers : $\pi(n) = \text{nb de nombres premiers entre } 1 \text{ et } n \leq n$

$$\pi(n) \approx \frac{n}{\log n}, \quad n = 1000 \quad \frac{n}{\log n} \approx 144$$

Idée : générer uniformément un entier entre 2 et n_{MAX}

→ vérifier si n est premier si non, recommencer.

Nombre de tirages à faire en moyenne : $\frac{1}{\log n_{MAX}}$

Petit théorème de Fermat (qui on a jamais vu et qui on découvre) :
si p est premier alors pour tout $a \neq 0 \pmod p$, $a^{p-1} \equiv 1 \pmod p$

Preuve : $a \cdot a^3 \cdot a^4 \cdot a^5 \dots a^{(p-1)} \rightarrow$ des puissances mod p
 $= 2 \times 3 \times \dots \times (p-1) \pmod p$
donc $a^{p-1} (p-1)! \equiv (p-1)! \pmod p$
 $a^{p-1} \equiv 1 \pmod p$

Pseudo-premiers (n)

si $2^{n-1} \equiv 1 \pmod n$ alors

retourner premier : peut-être

sinon

retourner composé : réponse correcte

On test avec des valeurs différentes 2, 3, ...

07/02/2018

ALG-06

① CM Algorithme de miller - rabin -

MR (m, s)

```
pour i = 1 à m faire
    a = random (1 m-1)
    si tremain (a, m) alors
        ↘ retourne "composé"
    sinon
        ↘ retourne premier
```

Tremain (a, m) renvoie 1 si le nb est composé ou premier.

```
m-1 = 2t * u      t ≥ 1
x0 = au              u impair
pour i = 1 à t faire
    xi = xi-12 mod m
    si xi = 1 et xi-1 ≠ 1 alors
        ↘ return true
    si xi ≠ 1 alors
        ↘ return mai
    sinon
        ↘ return fausse
```

Conclusion - En répétant l'algorithme, on peut montrer que la probabilité

$$\text{d'erreur est } \leq \frac{1}{2^s}$$

↳ Le nombre est dit premier qu'il est composé.

①

(TD) TDG - Min-cut randomisé : algorithme de Krager

Exercice 1-

- ④ 11 pour la coupe minimale
- ② a. 6 7 8
- b. 11 8
- c. ~~9 10 11 12 8~~ 12 8

- ③ car on a plus de chance de contacter deux sommets qui ont plusieurs arêtes qui les relient, par conséquent la coupe ~~maximale~~ trouvera généralement peu d'arêtes, la coupe est donc "assez" bonne.
- ④ À faire.

a Exercice 2-

- ① $\frac{m-h}{m}$, h étant le nombre d'arêtes de la coupe et m le nombre d'arêtes.

$$\textcircled{2} \quad \textcircled{a} \quad \textcircled{b} \quad \textcircled{c} \quad c. \sum d(v) \geq h m$$

$$\textcircled{3} \quad \frac{m-h}{m} = 1 - \frac{h}{m}$$

$$2m \geq hm$$

$$m \geq \frac{hm}{2}$$

$$m-1 \rightarrow \text{coupe min. } h.$$

$$\geq 1 - \frac{2}{m-1} \dots$$

TD 4: Min-cut randomisé: algorithme de Krager

Concept : Génération aléatoire, contrôle de la probabilité d'échec

Auteur: Nicolas Gast

Méthode : algorithme glouton, randomisation

L'objectif de cette séance est d'étudier un algorithme randomisé résolvant le problème Coupe-Min (MinCut). Ce problème peut être résolu par un algorithme polynomial déterministe en cherchant un flot maximum (p.e., en utilisant l'algorithme de Ford-Fulkerson). Dans ce TD, nous allons explorer une technique se basant sur un générateur aléatoire, introduite par Karger en 1993. On obtient ainsi un algorithme dit randomisé, qui fournit un bon résultat avec une probabilité que l'on peut estimer.

Un multigraphie $G = (S, M)$ un graphe dans lequel chaque arête peut apparaître plusieurs fois. Il est composé d'un ensemble de sommets S et d'une liste d'arêtes non orientées M . Une coupe $C \subset M$ est un ensemble d'arêtes de G tel que, si on enlève ces arêtes, le graphe partiel $G'' = (S, M \setminus C)$ a 2 composantes connexes non connectées. Le problème de la coupe minimale est de trouver une coupe ayant un minimum d'arêtes.

On définit la contraction d'un multigraphie $G = (S, M)$ par une arête (x, y) comme étant le multigraphie $G' = (S', M')$ dans lequel :

1. Les sommets x et y sont remplacés par un sommet z ;
2. on supprime toutes les arêtes entre x et y ; et
3. on remplace les arêtes de la forme (x, u) ou (y, u) par (z, u) .

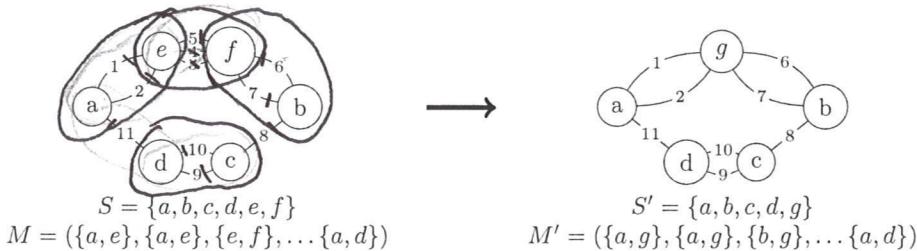


FIGURE 1 – Contraction du multigraphie par l'arête $\{e, f\}$.

Exercice 1: Génération d'une coupe aléatoire

On se donne l'algorithme suivant :

Génère_coupe(G)

Input : Un multigraphie G

Output : G contient 2 sommets, les arêtes entre ces sommets forment une coupe de G

for $i = 1$ à $n - 2$ **do**

Choisir une arête a uniformément dans le multigraphie G ;
 $G \leftarrow$ contraction(G, a) ;

Retourner G ;

Algorithme 1 : Génération d'une coupe aléatoire.

1. Quel est la taille de la coupe min sur le multigraphie de la Figure 1 ?
2. On fait tourner l'algorithme sur le graphe à 6 sommets de la Figure 1. Quel est la taille de la coupe min lorsque les arêtes enlevées successivement sont (on ignore les arêtes déjà enlevées et l'algorithme s'arrête lorsqu'il n'y a plus que deux sommets) :
 - a) [3, 4, 1, 9, 11, 10, 6, 7, 5, 2, 8]
 - b) [10, 3, 1, 7, 4, 9, 11, 6, 8, 5, 2]
 - c) [6, 7, 11, 5, 9, 4, 2, 8, 1, 10, 3]

3. A votre avis, pourquoi la coupe générée est en générale assez bonne ?
4. Comme travail personnel (à la maison mais pas à rendre), montrer que cet algorithme génère bien une coupe.

Exercice 2: Propriétés de la coupe aléatoire

On veut estimer la probabilité que cette algorithme fournit une coupe minimale. Le calcul exact étant difficile, nous allons minorer cette probabilité et montrer qu'elle est suffisante pour avoir un algorithme efficace.

Notations : on note n le nombre de sommets initial de G et m son nombre d'arêtes. on suppose que la taille minimale d'une coupe est k ; on choisit une coupe minimale C_{min} (on pourrait en avoir plusieurs). On note p la probabilité que l'algorithme 1 génère la coupe C_{min} .

1. Quelle est la probabilité de ne pas tomber sur une arête de C_{min} au premier tirage aléatoire ?
2. On cherche à majorer cette probabilité. Pour cela, on veut minorer m . Parmi les inégalités suivantes, dire ce qui est vrai :
 - $m \geq k$ ✓
 - $m \geq m$ ✓
 - $m \geq kn/2$
3. Donner un minorant du nombre d'arêtes du multi-graphe graphe G' après la première contraction. Minorer la probabilité de ne pas tomber sur une des arêtes de C_{min} au second tirage.
4. Procéder par récurrence pour minorer la probabilité de ne pas tomber sur une des arêtes de C_{min} au $i^{ème}$ tirage.
5. Calculer un minorant de la probabilité que l'algorithme 1 génère la coupe minimale C_{min} .

$$m = \text{nb de sommet initial de } G. \quad 11 / \frac{5 \times 12}{2} = \frac{55}{2} : 27,5$$

Exercice 3: Garantir la probabilité d'erreur

Comme on ne génère pas "la bonne coupe" à tous les coups, on répète le tirage et on garde le meilleur résultat. Cela revient à générer un échantillon de taille suffisante T .

Répétition_coupe(G, T)

Input : multigraphe $G = (S, M)$, entier T (Taille de l'échantillon)

Output : Meilleure coupe obtenue

C_{min} ;

for $i = 1$ à T do

└ $C_{courant} \leftarrow$ Génère_Coupe(G) ;
 if $|C_{courant}| < |C_{min}|$ then
 └ $C_{min} \leftarrow C_{courant}$

Retourner C_{min} ;

Algorithme 2 : Répétition de tirages

1. Minorer la probabilité que l'algorithme 2 fournit le bon résultat en fonction de T .
2. Montrer que pour une taille d'échantillon de $T = n^2/2$, la probabilité d'obtenir une coupe minimale est minorée par $1 - 1/e$.
3. On se donne une probabilité α et un graphe de taille n ; donner la taille d'échantillon telle que la probabilité d'avoir généré une coupe minimale soit supérieure à α . Donner la valeur de la taille de l'échantillon pour $\alpha = 0.99$ et $n = 100$.

On rappelle que la fonction \log étant concave, on a $\log(1 + x) \leq x$ pour tout $x \in (0, \infty)$, ce qui implique que pour tout a :

$$\left(1 + \frac{a}{n}\right)^n = e^{n \log(1 + \frac{1}{n})} \leq e^a.$$

De plus, on a

$$\lim_{n \rightarrow \infty} \left(1 + \frac{a}{n}\right)^n = e^a.$$

② CM Tables de hachage (suite)

voir le schéma du cours précédent avec l'univers des clefs et le chainage

voir l'exemple avec les numéros d'étudiants.

Fonctions de hachage:

idée : - uniformité

- mélange : 2 clés "proches" donnent 2 valeurs hachées "lointaines"

=> modèle uniforme, clé = entier (de grande taille)

Méthode : ① $h(x) = x \bmod M$: Pas efficace pour le mélange. Ne prend pas en compte toute l'information dans x .

$$\text{② } h_a(x) = \left\lfloor M \left(\begin{array}{c} \xrightarrow{\text{coefficent reel}} \\ a x \bmod 1 \end{array} \right) \right\rfloor$$

partie décimale de ax .

, partie entière inférieure.

exemple:

$$x : 12537$$

$$a : \sqrt{2}$$

$$ax : 17729, \underbrace{9959}_{ax \bmod 1}$$

$$M : 13$$

$$\lfloor 13 \times (ax \bmod 1) \rfloor = 12$$

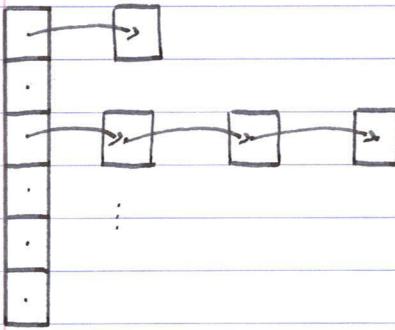
⚠ Difficile de prouver que h a de bonnes propriétés.

=> tous les langages modernes ont leurs propres fonctions de hachage.

Utilisation comme empreinte d'un objet.

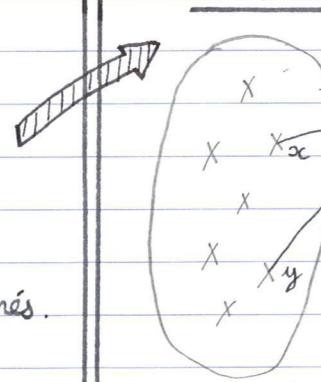
si M est grand, la probabilité de collision est très faible. C'est donc un outil de vérification. c'est utilisé en sécurité.

Structures de données



-> Tableau de listes chaînées.

Tableau de listes implicites:



insertion facile / suppression difficile

Stratégie de collision:

- mette dans la première case suivante disponible.

$$h(x,i) = h(x) + i \bmod M.$$

↳ nb de collision.

→ création de grappes (séquences sans trous)

• Augmenter la distance en fonction des collisions.

$$\rightarrow h(x, i) = h_0(x) + i^2 \bmod m.$$

• hachage double.

$$\rightarrow h(x, i) = h_0(x) + i \cdot h_1(x) \bmod m$$

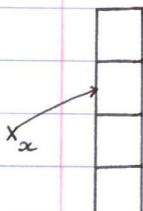
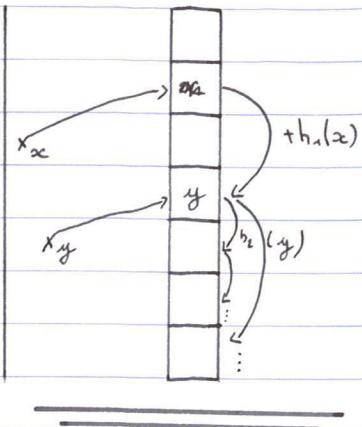
↳ plutôt efficace.

Cout d'une recherche -

postulat

- { table de taille M
- n éléments dans la table
- $n \leq M$

exemple :



La probabilité que x soit haché dans la case h est $1/M$.

La probabilité de collision est de m/M .

Le taux de remplissage de la table est de $(m/M = \alpha) < 1$

Le nombre de tentative pour trouver une case vide suit une loi géométrique de paramètre $(1 - (m/M))$

C'est à dire : C coût de recherche d'une case vide pour x .

$$P(C = h) = ((m/M) \times (m/M) \times (m/M) \dots (m/M))^{(1 - (m/M))}$$

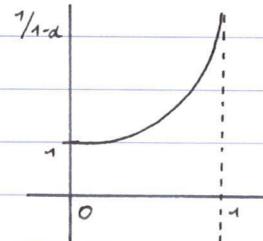
$\xrightarrow{\text{case occupée}} = ((m/M)^{h-1} (1 - (m/M)))$

↓
la borne
 $\alpha = 80\%$

$$= \alpha^{h-1} (1 - \alpha)$$

En moyenne : $E C = \frac{1}{1 - \alpha}$

espérance



$\alpha = 80\%$
 $E C = 5$

Exercice - Trouver l'intersection de deux ensembles.

ensemble 1 : liste de n éléments

ensemble 2 : liste de m éléments

éléments communs aux deux listes.

méthode 1 : Parcourir l'ensemble 1, et pour chaque élément d'un ensemble, on le compare à tous les éléments de l'autre ensemble. Complexité en $O(n \times m)$

méthode 2 : Trier l'ensemble 1, trier l'ensemble 2, puis parcourir simultanément des ensembles triés. Complexité : $n \log n + m \log m + n + m$

④

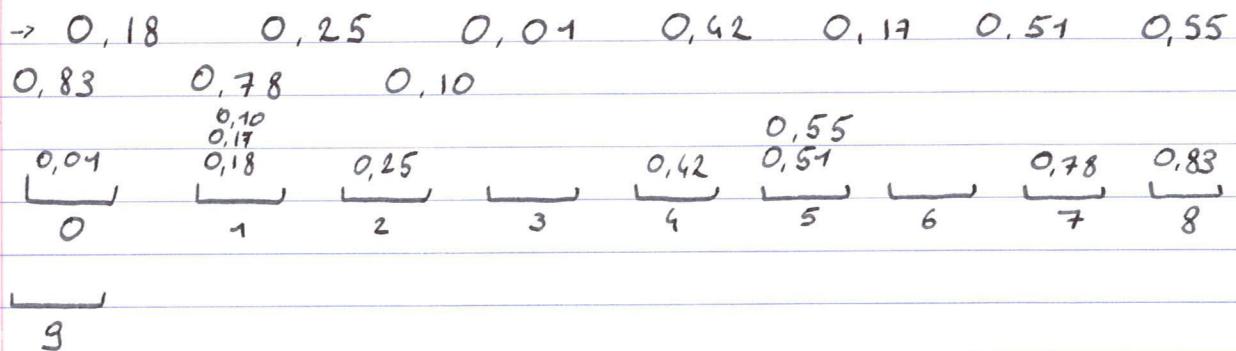
(CM) -- Suite de l'exercice d'intersection d'ensemble --

méthode 3: On utilise une table de hachage, insérer les éléments de E_1 dans la table de hachage. Puis ensuite, on parcours E_2 et on vérifie l'appartenance.

exemple d'un tri en $O(m)$:

postuler $\begin{cases} m \text{ éléments} \\ \text{membres réels entre } 0 \text{ et } 1 \\ x_1, \dots, x_m \end{cases}$

pour i allant de 1 à m faire
mettre x_i dans la case $[x_i; x_m]$



-> Combien d'éléments sont dans la case i ?

N_i suit une loi Binomiale $B(m, 1/m)$

$$\begin{aligned} \mathbb{E}N_i &= 1 \quad \mathbb{V}N_i = m \cdot (1/m) \cdot (1 - (1/m)) = 1 - \frac{1}{m} \\ &= \mathbb{E}(N_i - \mathbb{E}N_i)^2 \\ &= \mathbb{E}N_i^2 - (\mathbb{E}N_i)^2 \end{aligned}$$

$$\begin{aligned} \mathbb{E}N_i^2 &= \mathbb{V}N_i + (\mathbb{E}N_i)^2 \\ &= 1 - (1/m) + 1 \\ &= 2 - (1/m) \leq 2 \end{aligned}$$

Bucket-sort. m en moyenne.

Tables de hachage

I -) Introduction

problème : recherche d'un élément dans un ensemble.
 clé \rightarrow objet

Structure de données associée à l'ensemble.

Ensemble de n objets.

- Tableau : parcours de tableau : $O(n)$
- Liste : parcours de liste : $O(n)$
- Prétraitement : tri des objets / clé
 - recherche dichotomique $O(\log n)$
 - par interpolation $O(\log \log n)$
 dans un tableau.
- ABR : recherche récursive dans l'arbre : $O(\log n)$

II - Principe

Ensemble dynamique :

- insertions
- suppressions
- Opérations globales (recherche ...)
- parcours
- calculs de moyenne
médiane
- ...

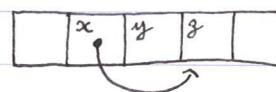
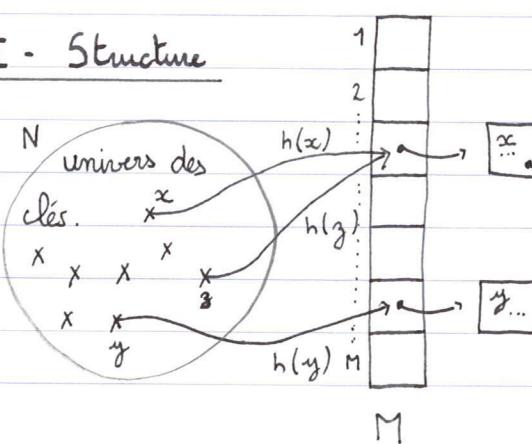
La clé de l'objet donne une information (algorithme) pour retrouver l'objet.

Exemple :

$m = 42$ étudiants	taille 10^8	$O(1)$
id : 8 chiffres	idée 2 : tableau avec le numéro d'étudiant	
		\rightarrow calculer la somme des chiffres du m° d'étudiant modulo 20.

Taux de remplissage de la table : $\frac{N}{M}$ $M = 20$: taille de la table.

III - Structure



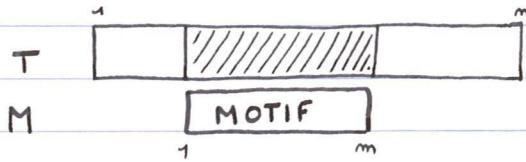
Problèmes : collisions

Tableau de taille $M = 1000$, la proba que x et y soit en collision : $1/1000$.
 collision : haché sur la même case

Nombre moyen de collision dans la table :

• Algorithme de recherche de motif.

Automate des préfixes : Algo de Boyer - moore , Knuth - morris - pratt



$$h_T = h(M)$$

pour $i = 1 \text{ à } m - m$

si $T[i, i+m-1] = M$ alors bravo

ALGO de Karp - robins :

$$h_M = h(M)$$

Pour $i = 1 \text{ à } m - m$

si $h(T[i, i+m-1]) = h_M$ alors

si $T[i, i+m-1] = M$ alors bravo.

Idee : calcul de $h(T[i+1, i+m])$ à partir de $h(T[i, i+m-1])$

Exemple : texte est une suite de chiffre.

2 5 3 8 1 9 9 1 0 1 9
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
0 1 2 3 4 4 0 2 0
↳ collision ↳ collision ↳ bravo ↳ collision

$$M = 910$$

$h(x)$: somme des chiffres modulo 5

$$h_M : 0$$

UE ALGO6 — TD2 — Séance 3 : aléatoire et simulation

`java.util.Random` implémente les trois méthodes suivantes :

- `boolean nextBoolean()` //retourne une valeur booléenne aléatoire
- `double nextDouble()` //retourne un double entre 0.0 et 1.0
- `int nextInt(n)` //retourne un int entre 0 inclus et n exclu

Exercice 1. Comment simuler :

- un lancé de dé à 6 faces non pipé;
- un tirage à pile ou face pipé (i.e. pile avec la probabilité p);
- autour d'une roue est écrit «ananas» (Chacune des lettres occupe $\frac{1}{6}$ de la roue). La roue tourne et s'arrête sur une lettre. Simuler le tirage aléatoire des lettres.

Génération de textes

Une manière simple de générer un texte aléatoirement est de considérer que le n^{e} mot du texte ne dépend que du $(n - 1)^{\text{e}}$ mot. Cette manière de faire est utilisée par des comptes twitter : «*Repassant par les rues effrontément en haillons, il composait une étrange musique amoureuse*» est un exemple de *tweet* publié par *le singe* (<http://twitter.com/singe9>).

Pour cela, il faut d'abord fixer les probabilités $P(m, m')$ qu'un mot m soit suivi du mot m' . Ceci se fait à l'aide d'un texte «d'apprentissage».

Exercice 2. Soit le texte d'apprentissage suivant :

We shall fight in France, we shall fight on the seas and oceans, we shall fight in the air. We shall fight on the beaches, we shall fight on the landing grounds, we shall fight in the fields and in the streets, we shall fight in the hills.

Sachant que $P(i, j)$ se calcule à l'aide du nombre d'occurrence du mot i (n_i) et du nombre d'occurrence de la séquence ij (n_{ij}) Quelle est la valeur que ce texte permet de donner aux probabilités suivantes :

- $P(\text{we}, \text{shall})$
- $P(\text{fight}, \text{in}), P(\text{fight}, \text{on})$ et $P(\text{fight}, j) \forall j \notin \{\text{in}, \text{on}\}$
- $P(\text{in}, j) \forall j$
- $P(i, j) \forall j, i$

Exercice 3. Proposer un algorithme qui calcule les $P(i, j)$ en une passe sur le texte.

Exercice 4. Proposer un algorithme qui génère un texte conforme aux $P(i, j)$.

Exercice 5. Pour aller plus loin il faut introduire de la *nouveauté*, on pourra utiliser le lissage de *Good-Turing*¹.

1. Dixit Wikipedia : Good-Turing frequency estimation was developed by Alan Turing and his assistant I. J. Good as part of their efforts at Bletchley Park to crack German ciphers for the Enigma machine during World War II.

TD science 3 - Hachage et complexité moyenne

Exercice 1: Hachage, borne et performance

valeurs possibles de X_i :

0 1 2 3 4 m

$$\left(\frac{m-1}{m}\right)^2$$

$$\frac{1}{m} \left(\frac{m-1}{m}\right)^2$$

$$m \cdot \frac{1}{m} \left(\frac{m-1}{m}\right)^{m-1} \quad \binom{m}{3} \left(\frac{1}{m}\right)^3 \left(\frac{m-1}{m}\right)^{m-3}$$

$$\left. \begin{array}{l} 1 \text{ si la valeur} \\ \text{de } X \\ \text{est } a \\ 0 \text{ sinon} \end{array} \right\} = \mathbb{1}(X=a)$$

\rightarrow booleen

Loi Binomiale $B(1, \frac{1}{m})$

$$P(X_1 = h) = \binom{m}{h} \left(\frac{1}{m}\right)^h \left(1 - \frac{1}{m}\right)^{m-h}$$

$$\begin{aligned} m=100, \quad P(X_1 = 0) &= \left(\frac{99}{100}\right)^{100} = 0,90 \\ m=1000, \quad (X=1) &= 0,09 \end{aligned}$$

Pour une clé qui n'a pas été dans la table

\rightarrow elle est hachée vers une case i

Dans cette case X_i éléments ont déjà été hachés.

Donc on va faire X_i comparaisons.

Le nombre moyen de comparaison est $\sum h_i \times P(X_i = h_i) = m \cdot \frac{1}{m}$

$$E_m = E(X_1 + \dots + X_m)$$

$$m = EX_1 + \dots + EX_m = mEX_1$$

si la clé est la $j+1$ ème insérée: $\frac{j}{m}$

il y a une probabilité $1/m$ de choisir la clé $j+1$.

$$\sum_{j=0}^{m-1} \frac{j}{m} \times \frac{1}{m} = \frac{1}{m \times m} \sum_{j=0}^{m-1} j = \frac{1}{m \times m} \times \frac{j(m-1)}{2} = \frac{\frac{m-1}{2}}{m^2}$$

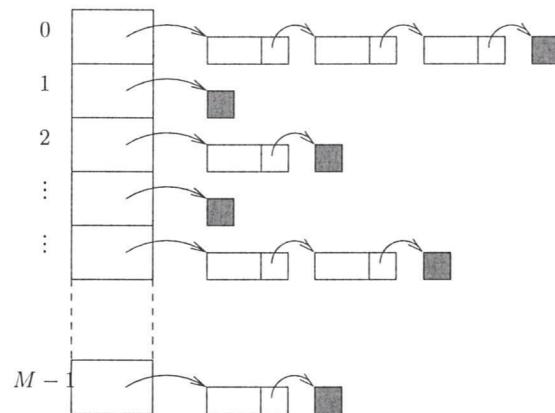
TD2 – Hachage et complexité en moyenne

Concept : Recherche d'élément, analyse en moyenne
 Méthode : Fonction de hachage, probabilité

Auteur: Nicolas Gast

Les tables de hachage sont un exemple classique d'utilisation de fonctions pseudo-aléatoires. Dans ce TP, on s'intéresse à l'analyse en moyenne de ces tables. Selon Knuth, les tables de hachage ont été inventées par H. P. Luhn en 1953, en utilisant la méthode de résolution de conflits à base de par chaînage (comme dans ce TD).

On considère le problème de hachage dans une table de taille m contenant n clés. On considère un hachage fermé, c'est à dire que la résolution de collision se fait par chaînage externe à la table :



On suppose que ces variables aléatoires sont indépendantes entre elles. L'état de remplissage de la table est modélisé par un vecteur de dimension m :

$$X = [X_0, \dots, X_{m-1}], \quad (1)$$

où X_i représente le nombre de clés hachées sur la case i .

On notera $\alpha = n/m$ le taux de remplissage de la table.

Exercice 1: Hachage : borne et performance

On suppose que la fonction de hachage est uniforme. Ceci se modélise en associant à chaque clé k de l'espace des clés une variable aléatoire U_k de loi uniforme à valeurs dans $\{0, \dots, m-1\}$. Ainsi, la variable X_i s'exprime :

$$X_i = \sum_{k=1}^n \mathbf{1}_{\{U_k=i\}}$$

1. Donner la loi de X_i , pour une case i donnée (autrement dit, calculer la probabilité $\mathbb{P}(X_i = \ell)$ pour $\ell = 0, 1, \dots$).
2. Calculer le coût moyen de la recherche d'une clé qui n'est pas dans la table.
3. On veut calculer le coût moyen de la recherche d'une clé que l'on sait dans la table.
 - a) On suppose que j clés ont été insérées dans la table. On insère la clé $j+1$ dans la table (en l'insérant à la fin d'une liste en cas de collision). Quel est le coût de la recherche de cette clé ?
 - b) On choisit une clé au hasard parmi les n clés de la table. À l'aide de la question précédente, calculer le coût de recherche de cette clé (en moyenne).
4. Calculer le coût moyen de l'insertion d'une nouvelle clé dans la table. Calculer le coût moyen de la suppression d'une clé de la table.

5. Calculer la probabilité qu'une case soit vide, donner un équivalent en fonction de α pour n et m grand. En déduire le nombre moyen de cases vides dans la table.
6. On peut montrer que, pour n et m grands, $P(X_i \geq k) \leq (e\alpha/k)^k$
 - a) Application numérique : on suppose que $\alpha = 1/e \approx 1/3$. Exprimer en fonction de k la probabilité qu'une case ait plus de k éléments ? Que vaut cette probabilité pour $k = 2, k = 3$ ou $k = 5$?
 - b) Recommencer la question avec $\alpha = e^{-1}/4 \approx 1/10$.
 - c) Pourquoi peut-on parler de bon comportement de la recherche dans une table de hachage ?
7. Montrer que pour n et m grands, $P(\max_{1 \leq i \leq m} X_i \geq k) \leq m(e\alpha/k)^k$.
 - a) En déduire que, quelque soit la valeur de α , $P(\max_{1 \leq i \leq m} X_i \geq \log m)$ tend vers 0 quand m tend vers l'infini.
 - b) Quelles conclusions en tirer sur la complexité du hachage ?

Remarques :
8. ** (Question optionnelle)
 - a) Montrer que pour $k \leq n$: $P(X_i = k) \leq \alpha^k/k!$
 - b) En déduire que $P(X_i \geq k) \leq e^\alpha \alpha^k/(k)!$.
 - c) En déduire que, asymptotiquement, pour b grand : $P(X_i \geq b\alpha) \leq (e/b)^{b\alpha} e^\alpha$.

Remarque : on peut améliorer ce résultat en $P(X_i \geq b\alpha) \leq (e/b)^{b\alpha}$ en se servant du fait que lorsque n et m sont grands, la loi de remplissage est proche d'une loi de Poisson : $P(X_i = k) \approx \alpha^k/k! e^{-\alpha}$.

Exercice 2: Remplissage complet

On chercher à calculer le nombre moyen d'insertions qu'il faut pour avoir au moins un élément dans chacune des m cases de la table. On note T_n ce nombre. Lorsque la table a déjà j cases non vides, on note Y_j le nombre d'insertions à réaliser avant qu'une nouvelle case soit occupée.

1. Que vaut Y_0 ?
2. Donner, en la justifiant, la loi de Y_j . Les variables Y_i sont-elles indépendantes ?
3. Exprimer T_n en fonction des Y_j . En déduire l'espérance de T_n .
4. Commenter votre résultat.
5. * Calculer la variance de T_n . En déduire que T_n est proche de son espérance.

Rappels utiles

$$\sum_{i=1}^n \frac{1}{i} \approx \log n \quad \sum_{i=0}^n \frac{1}{i^2} \approx \pi^2/6$$

Si A et B sont deux variables aléatoires :

- $\mathbb{E}(A + B) = \mathbb{E}(A) + \mathbb{E}(B)$
- Si en plus elles sont indépendantes : $\text{var}[A + B] = \text{var}[A] + \text{var}[B]$.

① TD ALGO6 - Séance 2 : Tables de hachage TDE

1- Chainage

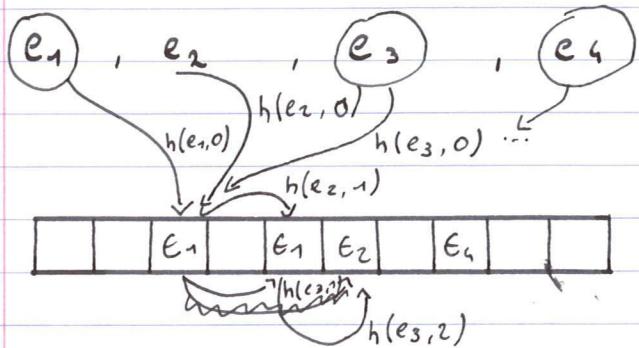
Exercice 1 -

```
Public boolean recherche (objet o) {
    Element e = this.table [o.hashCode() % this.taille];
    while ((e != null) && (!e.value.equals(o))) {
        e = e.next;
    }
    return (e != null);
}
```

```
Public void insertion (objet o) {
    int index = o.hashCode() % this.taille;
    if (!this.recherche ()) {
        Element nouveau = new Element ();
        nouveau.value = o;
        nouveau.next = this.table [index];
        this.table [index] = nouveau;
    }
}
```

```
Public void supprimer (objet o) {
    int index = o.hashCode() % this.taille;
    Element e = table [index];
    Element prec = null;
    while ((e != null) && (!e.value.equals(o))) {
        prec = e;
        e = e.next;
    }
    if (prec == null) {
        table [index] = e.next;
    } else {
        prec.next = e.next;
    }
}
*: if (!this.recherche (o)) { return; }
```

$h : \text{Objet} \times \mathbb{N} \rightarrow \mathbb{N}$



→ Different hash table.

```
bool recherche (objet e) {
    int i, j;
    j = 0; i = h(e, j);
    while ((j < taille) && table[i].present && (!table[i].supprime)) {
        if (!table[i].value.equals(e))
            j = j + 1;
        i = h(e, j);
    }
    return (j < taille) && table[i].present && (!table[i].supprime));
}
```

```
class Element {
    objet value;
    boolean present, supprime;
    Element {
        present = false;
        supprime = false;
    }
}
```

UE ALGO6 — TD2 — Séance 2 : Tables de hachage

On s'intéresse à l'implémentation en Java d'une table de hachage représentant un ensemble d'instances de la classe `Object`.

Dans la librairie standard Java, la classe `Object` fournit les deux méthodes suivantes :

- la méthode `boolean equals(Object o)` permet de comparer l'instance courante avec un autre instance ;
- la méthode `int hashCode()` permet d'obtenir le hachage d'une instance.

1 Chaînage

On considère que les éléments en collision sont placés dans une liste chaînée.

On propose le début d'implémentation suivant :

```
public class Hashtable {  
  
    class Element {  
        Object value;  
        Element next;  
    }  
  
    int taille = 11;  
  
    Element[] table = new Element[taille];  
}
```

Exercice 1. Écrire les méthodes permettant d'insérer, rechercher et supprimer un élément dans la table.

2 Adressage ouvert

L'adressage ouvert consiste à placer les éléments en collision non pas dans une liste chaînée mais dans des cases alternatives du tableau. On définit donc une fonction $h(x, i)$ permettant de trouver une case libre, en essayant successivement $h(x, 0), h(x, 1), \dots$

Exercice 2. En supposant h donnée, implémenter les primitives précédentes avec le principe d'adressage ouvert.

Exercice 3. Donner la définition de h dans les cas suivants :

- les cases alternatives sont les cases adjacentes à `x.hashCode()` ;
- on dispose maintenant non plus d'une mais de deux fonctions de hachages différentes h_1 et h_2 .

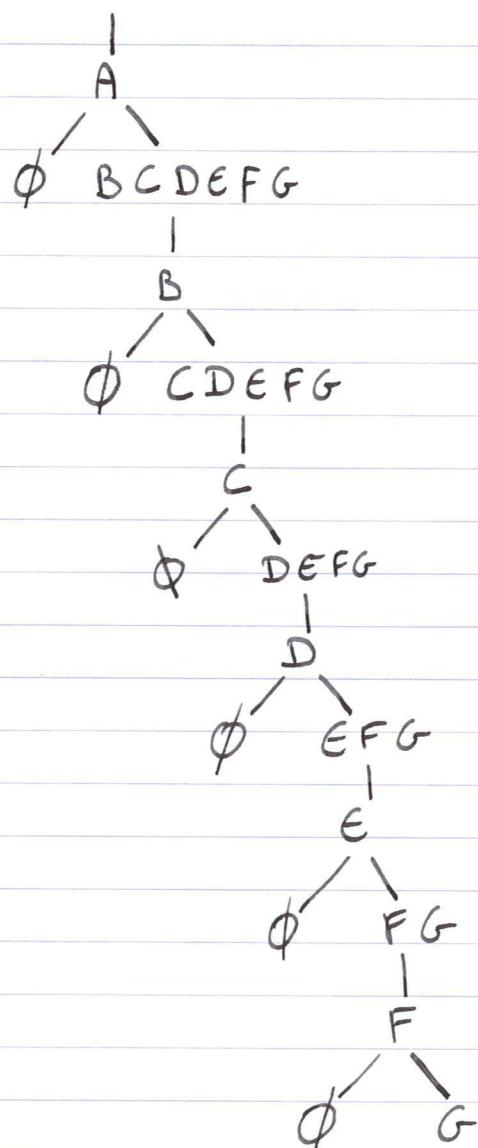
NB : dans les deux cas, il faudrait redéfinir les primitives insérer, supprimer et rechercher pour éviter les appels successifs à `hashCode()`, h_1 et h_2 .



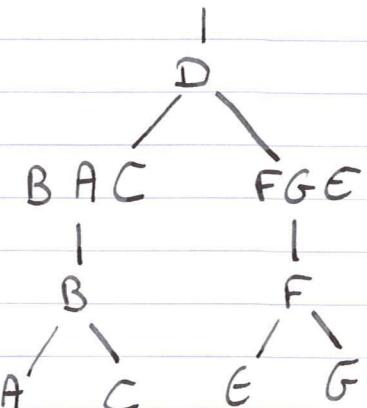
① TD séance 2 -

1. a.)

A B C D E F G



b.) D B F G A E C



10 comparaisons.

$$\frac{m(m-1)}{2} \text{ comparaisons}$$

→ reste les merges à faire.

2.) il existe un exemple (tableau trié) de cout $\frac{m(m-1)}{2}$

≤ Pn

HR

$P_h \leq h^2$ pour tout $h \leq m-1$

① cette hypothèse est vérifiée pour $m=1$

② supposons que la propriété soit vraie à $m-1$.

Soit un pire cas de taille m

$$P_m = (m-1) + C(h) + C(m-1-h)$$

\hookrightarrow plus petit que le pivot L plus grand que pivot

Or $C(h) \leq P_h$

$$C(m-1-h) \leq P_{m-1-h}$$

$$\text{donc } P_m \leq (m-1) + P_h + P_{m-1-h}$$

d'après l'HR $P_h \leq h^2$ et $P_{m-1-h} \leq (m-1-h)^2$

$$\text{donc } P_m \leq (m-1) + \underbrace{h^2 + (m-1-h)^2}_{f(h)} \text{ avec } 0 \leq h \leq m-1$$

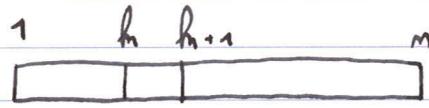
$$f(h) \leq (m-1)^2 \text{ sur } [0; m-1]$$

$$\text{donc } P_m \leq (m-1) + (m-1)^2 = m(m-1) \leq m^2$$

3) Pour une donnée de taille m . T_m

$$C(T_m) = (m-1) + C(T'_h) + C(T'_{m-1-h})$$

\rightarrow Le pivot choisi est l'élément du rang $h+1$. $0 \leq h \leq m-1$



\rightarrow si le pivot est le $h+1^{\text{e}}$ élément

$$\mathbb{E} c(T_m) = (m-1) + M_h + M_{m-1-h}$$

"

M_m || pivot est en $h+1$

$$M_m = \left[\frac{((m-1) + M_h + M_{m-1-h})}{m} \right] \xrightarrow{P(h+1 \text{ soit pivot}) = 1/m}$$

TD3 – Quicksort, analyse en moyenne, ABR



Concept : Analyse en moyenne, arbre de recherche
Méthode : Probabilité, récurrence

Auteur: Nicolas Gast

Le tri rapide (ou quicksort) est un algorithme de tri inventé par Hoare en 1961. Il utilise la méthode de diviser pour régner. Bien que sa complexité dans le pire des cas soit de $O(n^2)$, il a une complexité moyenne de $O(n \log n)$. C'est probablement l'algorithme de tri le plus utilisé. Dans ce TP, nous allons étudier sa complexité en temps et en espace, en faisant le lien avec les arbres binaires de recherche.

Les arbres binaires de recherche ont été introduits par plusieurs personnes de façon indépendante à la fin des années 50. Cette structure de donnée dispose des opérateurs de recherche, du minimum, du maximum, d'insertion et de suppression en un temps linéaire en la hauteur. En pratique, on utilise souvent des variations de ces arbres qui ont une garantie de performance, comme les arbres AVL (introduits par Adelson-Velski and Landis en 1962) ou les arbres 2-3 (introduits par Hopcroft en 1970).

L'algorithme du tri rapide fonctionne de la façon suivante :

- On choisit un indice du tableau k , le pivot, $T[k]$ est la valeur du pivot
- On partitionne les éléments en deux sous-tableaux ;
 - On place tous les éléments plus petits que la valeur du pivot en début de tableau ;
 - On place tous les éléments plus grands que la valeur du pivot en fin de tableau ;
 - On le insère le pivot entre les deux ;
- On trie récursivement les deux sous-tableaux.

Exercice 1: Complexité en temps de Quicksort : pire cas et moyenne

On s'intéresse dans cet exercice au nombre de comparaisons qu'il faut pour effectuer le tri d'un tableau T de n éléments distincts. On note P_n le nombre de comparaisons nécessaires pour trier un tableau de taille n dans le pire des cas et M_n le nombre moyen de comparaisons. On admettra que l'opération de partitionnement coûte $n - 1$ comparaisons.

1. On choisit pour pivot le premier élément d'un tableau (l'élément 1).
 - a) Dérouler l'algorithme sur le tableau $T_1 = [A, B, C, D, E, F, G]$. Combien de comparaisons effectue cet algorithme pour trier ce tableau ?
 - b) Dérouler l'algorithme sur le tableau $T_2 = [D, B, F, G, A, E, C]$. Combien de comparaisons effectue cet algorithme pour trier ce tableau ?
2. (*Complexité en pire cas*) Montrer que $P_n \leq n^2$.
3. (*Complexité en moyenne*) On considère maintenant que le pivot est choisi aléatoirement, de manière uniforme, à chaque étape.
 - a) Quelle est la probabilité que ce pivot correspond à l'élément de rang i dans le tableau ?
 - b) En déduire que

$$M_n = (n - 1) + \frac{1}{n} \sum_{i=0}^{n-1} (M_i + M_{n-i-1}) = (n - 1) + \frac{2}{n} \sum_{i=0}^{n-1} M_i.$$

Que vous rappelle cette formule ? Quelles sont les valeurs initiales de cette suite récurrente ?

- c) Montrer que $M_n = 2n \log(n) + O(1)$ et conclure sur le coût moyen de l'algorithme lorsque le choix du pivot est aléatoire et uniforme.

Indications : (1) calculer $nM_n - (n-1)M_{n-1}$. (2) En déduire que $M_n/(n+1) - M_{n-1}/n \leq 2/n$.
(3) Utiliser le fait que $1 + 1/2 + 1/3 + \dots + 1/n = \log n + O(1)$

Exercice 2: Mélanger pour mieux trier ?

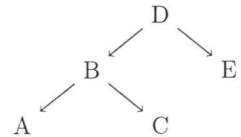
1. Pourquoi si avant de commencer l'algorithme les éléments sont dans un ordre aléatoire (*i.e.*, correspondant à une permutation aléatoire), alors l'algorithme qui choisit pour pivot l'indice 1 a la même complexité moyenne que celui qui choisit un pivot aléatoirement ?

2. Écrire une procédure qui tire une permutation aléatoire d'un tableau de taille n en temps $O(n)$.
3. Pourquoi certaines implémentations de quicksort mélangeant le tableau avant de le trier ?

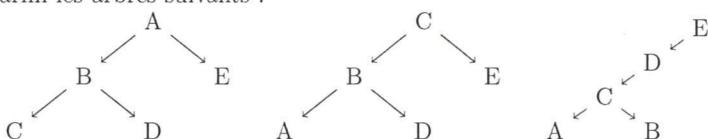
Exercice 3: Complexité en espace et hauteur d'un arbre

La complexité en espace d'un algorithme est la taille de la mémoire qu'il faudrait réservé pour pouvoir exécuter notre algorithme. Dans le cas de Quicksort, cette quantité est liée à la hauteur de l'arbre des appels récursifs.

On peut représenter les appels récursifs à la fonction Quicksort par un arbre d'appel, dans lequel l'étiquette d'un noeud est le contenu du pivot. Par exemple, si l'on trie le tableau $\{D, B, C, A, E\}$ en choisissant pour pivot le premier élément du tableau, on obtient l'arbre d'appel ci-contre.



1. On suppose que la hauteur de l'appel récursif est h . Justifier pourquoi la complexité en espace de Quicksort est $O(h)$.
2. On suppose que le pivot est le premier élément du tableau. Dessiner les arbres d'appels correspondants au tri des tableaux de l'exercice précédents :
 - a) $T_1 = [A, B, C, D, E, F, G]$
 - b) $T_2 = [D, B, F, G, A, E, C]$.
 Quelle est la hauteur de ces arbres ?
3. Justifier pourquoi l'arbre d'appel est un arbre binaire de recherche, c'est à dire un arbre binaire tel que si un noeud a une clé k , alors :
 - les clés des noeuds du sous-arbre gauche sont inférieures ou égales à k .
 - les clés des noeuds du sous-arbre droit sont supérieures ou égales à k .
4. Parmi les arbres suivants :



Lesquels sont des arbres binaires de recherche (*i.e.*, peuvent correspondre à un arbre d'appel de la fonction Quicksort) ?

5. Quelle est la hauteur maximale d'un arbre de taille n ? Quelle est sa hauteur minimale?
6. On effectue le tri d'un tableau de taille n en choisissant le pivot aléatoirement. On note X_n la hauteur de l'arbre d'appel et on appelle $Y_n = 2^{X_n}$ la hauteur exponentielle de cet arbre. Soit Y_i^g (respectivement Y_i^d) la hauteur exponentielle du sous-arbre gauche (respectivement droit) de l'arbre lorsque ce sous-arbre est de taille i . On note Z_i une variable qui vaut 1 lorsque le sous-arbre gauche a i noeuds et 0 sinon.

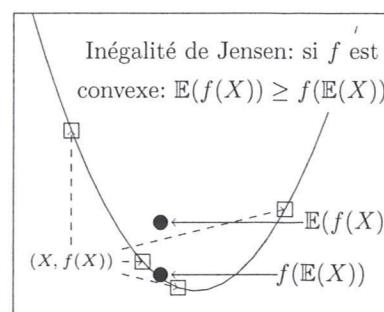
- a) Montrer que $Y_n = 2 \sum_{i=0}^{n-1} Z_i \max(Y_i, Y_{n-i-1})$.
- b) En déduire que

$$\mathbb{E}(Y_n) \leq 2 \sum_{i=0}^{n-1} \frac{1}{n} (\mathbb{E}(Y_i) + \mathbb{E}(Y_{n-i-1})) = \frac{4}{n} \sum_{i=0}^{n-1} \mathbb{E}(Y_i)$$

- c) Calculer $n\mathbb{E}(Y_n) - n\mathbb{E}(Y_{n-1})$ et en déduire que

$$\mathbb{E}(Y_n) \leq \frac{n+3}{n} \mathbb{E}(Y_{n-1}) \leq \frac{(n+3)(n+2)(n+1)}{6} \mathbb{E}(Y_0) = O(n^3).$$

- d) En déduire que $\mathbb{E}(T_n) \leq 3 \log_2 n + O(1)$.
7. Conclure sur la complexité en mémoire moyenne et en pire cas et comparer avec la complexité en temps.

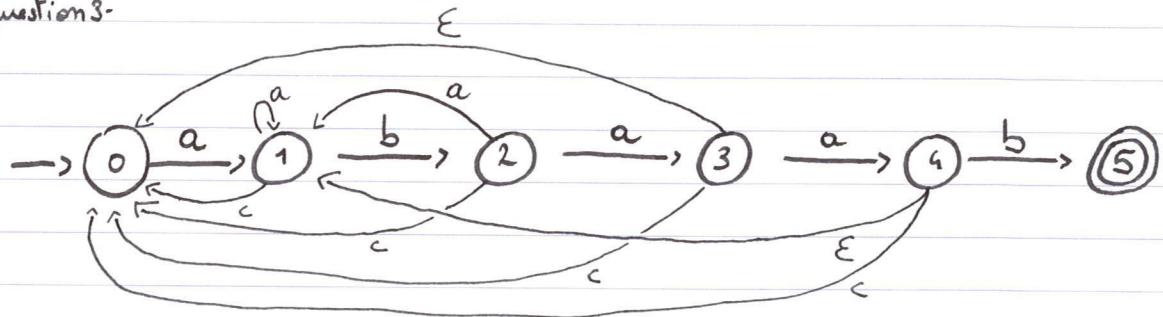


① TDE TDE - séance 1-

question 1: $O(m \cdot m + 1) \times m$

question 2: il est bien. potentiel problème d'indice résolu glorieusement par le .length qui marche tout seul.

question 3-



Connexion :

q	a	b	c
0	1	0	0
1	1	2	0
2	3	0	0
3	4	2	0
4	1	5	0
5	3	0	0

question 4:

motif de longueur k

entrée mbMotif (texte, tab){

mbacc = 0;

état = 0;

lettre = lettre suivante (texte);

for (lettre in T){

 état = tab[état, lettre];

 si état == M.length + 1 alors

 mbacc ++;

return mbacc;

Créer - Table - Transition (Σ, M) {

retourne TT [] []

Σ Alphabet

M motif de taille m

pour q=0 dans m faire

 pour a $\in \Sigma$ faire

 h = q+1

 répéter

 h = h - 1

 tant que M[1... h] = M[1... q].a

 fin pour

 fin pour

 concaténé

UE ALGO6 — TD2 — Séance 1 : Recherche de motif

L'objectif est de rechercher toutes les occurrences d'un motif, un mot ou un groupe de mots, dans un texte. On suppose que le texte est réalisé par un tableau $T[1..n]$, le motif par un tableau $M[1..m]$, formés de caractères pris dans un alphabet fini, par ex. $\{a, b, c, d, \dots, z\}$. Par exemple, le motif abaab est présent dans le texte ababcabbaabaabbac à la position 10.

Question 1. Soit l'algorithme de recherche suivant :

```
Recherche_Motif(T,M)
  { n est la longueur de T }
  { m est la longueur de M }
  pour i dans [0..n-m] faire
    si M[1..m]=T[i+1..i+m] alors afficher ("le motif apparaît à la position " s+1)
```

Si on mesure le coût en nombre de comparaisons entre caractères, quel est le coût de cet algorithme, en fonction de n et m ?

Question 2. Que pensez-vous de l'implémentation suivante, en Java, de cet algorithme? cb; 1

```
void rechercheMotif(String texte, String motif) {
    for (int i=0; i ≤ texte.length()-motif.length(); i++) {
        if (texte.substring(i,i+motif.length()).equals(motif)) {
            System.out.println("le motif apparaît à la position " + i);
        }
    }
}
```

Question 3. Essayant de diminuer le nombre d'opérations, on s'aperçoit qu'il suffirait pour chaque caractère de T lu de connaître le plus long suffixe lu qui est un préfixe de M (au lieu de recommencer à chaque fois toutes les comparaisons). Dans ce but, on construit un automate d'états finis, destiné à reconnaître les préfixes de M . On rappelle qu'un automate est défini par :

- un ensemble fini d'états Q
- un état initial q_0 élément de Q
- un sous-ensemble A de Q formé des *états terminaux*
- un alphabet fini S
- une fonction $d : Q \times S \rightarrow Q$ dite *fonction de transition*

Un automate qui reconnaît un motif M de longueur m possède $m + 1$ états, numérotés de 0 à m , 0 désignant l'état initial, m l'état terminal ; cet automate démarre à l'état $q_0 = 0$, et lit un par un les caractères du texte d'entrée ; lorsque l'automate est dans un état q , et reçoit en entrée un caractère x , il passe dans l'état $d(q, x)$; le motif est reconnu lorsque l'automate arrive à l'état terminal m .

La fonction de transition vérifie la propriété : Si $d(q, x) = k$ alors $M[1..k]$ est suffixe de $M[1..q].x$.

Dessiner un automate qui reconnaît l'exemple précédent, avec l'alphabet $\{a, b, c\}$; donner la table de la fonction de transition. Pour l'état $q = 3$, préciser pour chacun des caractères possibles x le plus grand k tel que $M[1..k]$ soit suffixe de $M[1..q].x$.

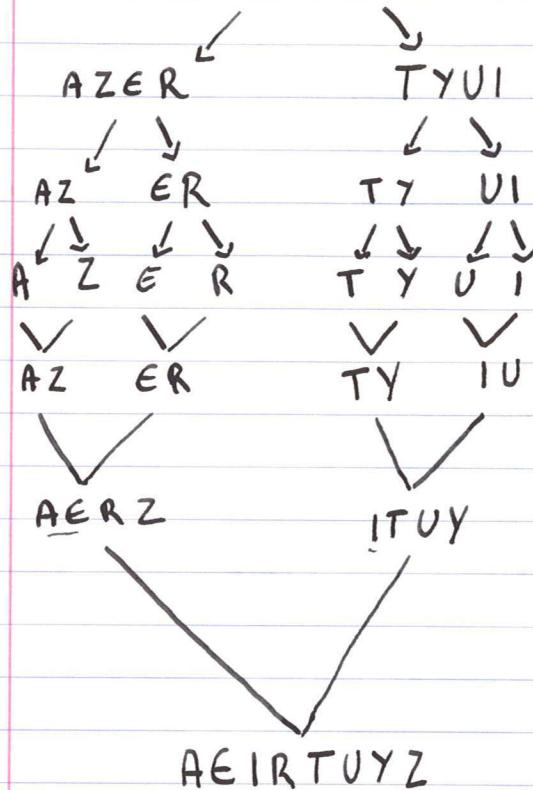
Question 4. Écrire une fonction qui calcule la table de la fonction de transition d'un motif $M[1..m]$. Écrire un algorithme qui recherche toutes les occurrences de ce motif dans un texte $T[1..n]$. Quel est le coût de ce nouvel algorithme ?

15/01/2018

Algo 6

① Exercice 1 - Tri fusion "classique"

1-a) AZERTYUI



→ on prend et insere le minimum des tableaux à chaque fois.

b) $\approx \log m$

2-a) fusion (T_1, T_2)

```
entiers a, b, c = 1;  
if (T1[a] < T2[b])  
    if (T1[a] < T2[b])  
        T[c] = T1[a];  
        c++;  
        a++;  
    else  
        T[c] = T2[b];  
        c++;  
        b++;  
return T;
```

algorithme en $O(m)$. $m_1 + m_2$
b) oui.

3.) ~~tri fusion (T₁, T₂)~~

~~tri fusion base (T[0..m/2], T[m/2..m])~~

~~tri fusion rec (T₁, T₂, l..r)~~

~~rec fusion (T₁, T₂)~~

~~T₁ = tri fusion rec (T[0..m/2], T[m/2..m])~~

~~T₂ = tri fusion rec (T[0..m/2], T[m/2..m])~~

Connexion
fusion:

i = 1

j = 1

pour h = 1 à m₁ + m₂ faire

si i = m₁ alors $\begin{cases} T[h] = T_2[j], \\ j++ \end{cases}$

sinon si j = m₂ + 1 alors $\begin{cases} T[h] = T_1[i], \\ i++ \end{cases}$

sinon si T₁[i] < T₂[j] alors $\begin{cases} T[h] = T_1[i], \\ i++ \end{cases}$

sinon $\begin{cases} T[h] = T_2[j], \\ j++ \end{cases}$

fusion:

Algorithme de tri partition fusion

Données: un tableau de m = 2^h éléments comparable

Résultat: le même tableau trié.

cas de base $\begin{cases} \text{si } T \text{ est réduit à 1 élément} \\ \text{l'ne rien faire} \end{cases}$

décomposition $\begin{cases} \text{sinon} \\ \quad \text{tri PF} (T_1, 1..m/2) \\ \quad \text{tri PF} (T_2, m/2 + 1..m) \\ \quad \text{fusion} (T_1, T_2) \end{cases}$

photo +

TD1 – Tri partition/fusion (merge sort)

Concept : Diviser pour régner; Analyse de coût

Méthode : Décomposition récursive

Auteur: Nicolas Gast

Le tri fusion est un algorithme de tri utilisant le principe de “diviser pour régner”, inventé par John von Neumann en 1945. Il effectue au plus $O(n \log n)$ opérations, qui est la complexité en temps dans le pire cas optimale parmi les algorithmes de tri à comparaison. Le tri fusion (ou ses variantes, comme Timsort) est utilisé par des bibliothèques standards de plusieurs langages, comme Python, Java ou Perl.

Dans ce TD, on propose d'étudier la complexité du tri fusion, et d'une variante moins connue : le tri fusion en place.

Exercice 1: Tri fusion “classique”

1TOY

L'algorithme de tri fusion est un algorithme récursif qui consiste à trier les deux moitiés du tableau séparément puis à fusionner les deux sous-tableaux triés ainsi obtenus. Pour trier un tableau de taille n , il s'opère récursivement de la façon suivante :

- On trie le sous tableau constitué des $\lceil n/2 \rceil$ premiers éléments du tableau.
 - On trie le sous tableau constitué des $\lfloor n/2 \rfloor$ derniers éléments du tableau.
 - On fusionne les deux tableaux triés en un seul tableau trié.
1. On veut trier le tableau de caractères “AZERTYUI”.
 - a) Écrire les valeurs des paramètres et les valeurs de retour de tous les appels récursifs à la fonction `tri_fusion`.
 - b) Compter le nombre de comparaisons qu'effectue cet algorithme sur cet exemple.
 2. Écrire (en pseudo-code) une fonction `fusion(T1, T2)` qui prend en entrée deux tableaux triés par ordre croissant (de taille n_1 et n_2) et rend un nouveau tableau de taille $n_1 + n_2$ contenant les éléments de $T1$ et $T2$ triés par ordre croissant.
 - a) Quel est la complexité de votre algorithme ?
 - b) Si ce n'est pas le cas : améliorer votre algorithme pour qu'il effectue au plus $n_1 + n_2$ opérations.
 3. Écrire (en pseudo-code), une fonction récursive `tri_fusion(T)` qui effectue le tri de T par ordre croissant.
 4. Soit $C(n)$ le temps que met le tri fusion pour trier un tableau de taille n . Donner une formule de récurrence pour $C(n)$ et la résoudre.

Exercice 2: Tri fusion en place

Au prix d'une complexité plus grande, on peut effectuer le tri fusion sans utiliser de tableau extérieur. Pour cela, on modifie l'opération de fusion. L'idée est la suivante : pour fusionner deux sous tableaux V et W , on découpe chaque sous-tableau en deux parties de même taille (pour simplifier, on suppose pour l'instant que $n = 2^k$). Ces parties sont notées V_1, V_2 et W_1, W_2 . On effectue alors les opérations de fusion suivantes (voir Figure 1) :

- `fusion(V1, W1)`
- `fusion(V2, W2)`
- `fusion(V2, W1)`

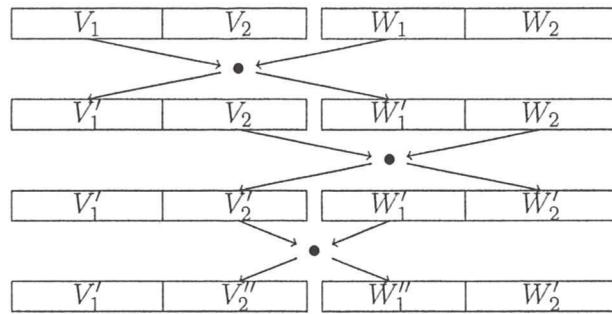


FIGURE 1 – Les opérations de fusion du tri de fusion en place

1. Soit $D(n)$ le nombre de comparaisons qu'effectue l'algorithme de fusion en place pour fusionner deux tableaux de taille $n/2 = 2^{k-1}$ et $n/2 = 2^{k-1}$.
 - a) Écrire une formule de récurrence pour $D(n)$.
 - b) En déduire la complexité de la fusion en place lorsque $n = 2^k$.
 - c) En déduire la complexité du tri en place lorsque $n = 2^k$.
 2. Montrer que l'opération de fusion est correcte.
 3. Écrire la fonction `fusion_en_place` en pseudo-code.
 4. Écrire la fonction `tri_en_place` en pseudo-code.
 5. Généraliser l'algorithme et l'étude de la complexité à des tableaux de taille quelconque (différent de 2^k).

Exercice 3: Exercice à rendre (extrait du deuxième Quick 2014-2015)

Pour un problème donné, un algorithme naïf a une complexité en $O(n^3)$. On a aussi trouvé deux solutions de type diviser pour régner :

Div1 Découper le problème de taille n en deux sous-problèmes de taille $n/2$ et les recombiner en temps $O(n^2)$.

Div2 Découper le problème en quatre sous-problèmes de taille $n/3$ et les recombiner en temps $O(n)$.

1. Quelle est la complexité de Div1 ?
 2. Quelle est la complexité de Div2 ?
 3. Quelle solution choisir : naïf, div1 ou div2 ?

Rappel : L'étude de la complexité d'algorithme de type *diviser pour régner* implique souvent des formules de récurrences reliant $C(n)$ à $C(n/b)$. Le résultat suivant est alors souvent utile :

Théorème 1 (Master theorem, Theorem 4.1 du livre de Cormen et al.). Soit $a \geq 1$ et $b > 1$ deux constantes. Soit $f(n)$ une fonction et $C(n)$ définie par la récurrence :

$$C(n) = aC\left(\frac{n}{h}\right) + f(n).$$

$C(n)$ a les bornes asymptotiques suivantes :

$$\begin{aligned} C(n) &= \Theta(n^{\log_b a}) && \text{si } f(n) = O(n^{\log_b a - \varepsilon}) \text{ pour un } \varepsilon > 0 ; \\ C(n) &= \Theta(n^{\log_b a} \log n) && \text{si } f(n) = \Theta(n^{\log_b a}) \\ C(n) &= \Theta(f(n)) && \text{si } f(n) = \Omega(n^{\log_b a + \varepsilon}) \text{ pour un } \varepsilon > 0 \text{ et si } f(n) \geq acf(n/b) \text{ pour } c \geq 1. \end{aligned}$$