Algorithmique III. L2 Informatique I41.

TD 9. Tri par dénombrement, par répartition et tri lexicographique [1]

EXERCICE 1. **(1)** Calculez la dispersion des deux listes

(*a*) $L_1 := [1,1,2,3,1,8,1,2,1,1]$,

(*b*) $L_2 := [3,1,1,2]$.

(2) Quelle est la dispersion minimale et la dispersion maximale d'une liste ? Justifiez.

Solution. **(1)** (*a*) On a $\min L_1 = 1$ et $\max L_1 = 8$, l'étendue de L_1 est donc égale à $8-1+1=8$. Le nombre de valeurs distinctes de L_1 est égal à 4, on en déduit que $\Delta(L_1) = 1 - \frac{4}{8} = \frac{1}{2}$.
(*b*) On a $\min L_2 = 1$ et $\max L_2 = 3$, l'étendue de L_2 est donc égale à $3-1+1=3$. Le nombre de valeurs distinctes de L_1 est égal à 3, on en déduit que $\Delta(L_2) = 1 - \frac{3}{4} = \frac{1}{4}$.
(2) La dispersion minimale est obtenue pour toute liste L constante puisque le nombre de valeurs distinctes est égal à 1 et l'étendue est aussi égale à 1, donc $\Delta(L) = 0$. Pour une liste de longueur n, l'étendue E peut être arbitrairement grande avec 2 valeurs distinctes dans la liste, minimisant ainsi le numérateur de la fraction, autrement dit $\Delta(L) = 1 - \frac{2}{E}$ qui tend vers 1 si $E \to +\infty$. Au sens mathématique, il n'existe donc pas de dispersion *maximale*, puisque la valeur 1 ne peut pas être atteinte, mais 1 est la borne supérieure pour la dispersion.

EXERCICE 2. Démontrez que l'ensemble des entiers relatifs $\mathbb{Z}$ est équipotent à l'ensemble des entiers naturels $\mathbb{N}$ en exhibant une bijection $\varphi : \mathbb{Z} \to \mathbb{N}$. Indication : associez les nombres négatifs aux entiers impairs et les nombres positifs aux entiers pairs.

Solution. On considère l'application $\iota : \mathbb{Z} \to \mathbb{N}$ définie par :

$$\iota(n) = \begin{cases} -(2n+1) & \text{si } n < 0 \\ 2n & \text{si } n \geqslant 0 \end{cases} \tag{1}$$

Montrons que ι est bijective. Soit $(n, n') \in \mathbb{Z} \times \mathbb{Z}$ tel que $\iota(n) = \iota(n')$. Si n et n' sont de même signe, on a soit $-(2n+1) = -(2n'+1)$ et donc $n = n'$ s'ils sont strictement négatifs, soit $2n = 2n'$ et donc $n = n'$ s'ils sont positifs. S'ils sont de signes opposés on a $-2(n+1) = 2n'$ et donc $-2(n-n') = 1$, ce qui est impossible puisque leur différence est un entier relatif et on en conclut que ι est injective. Considérons à présent un entier naturel m. S'il est pair, disons $m = 2k$, alors $k \geqslant 0$ et $\iota(k) = m$, il admet donc un antécédent k. S'il est impair disons $m = 2k+1$ avec $k \geqslant 0$ alors $-(k+1) < 0$ et $\iota(-(k+1)) = m$, il admet là encore un antécédent ce qui prouve que ι est surjective et par conséquent bijective, montrant ainsi que $\mathbb{N} \approx \mathbb{Z}$.

NB. Une telle application permet indirectement d'indexer une liste L avec des index négatifs en composant L avec l'application $\iota : L[\iota(n)]$ où $n \in \mathbb{Z}$.

EXERCICE 3. On rappelle l'algorithme de comparaison lexicographique :

```
    ALGORITHME COMPARAISON-LEXICO(u,v):{-1,0,+1}
    DONNEES
     · u,v: mots sur un alphabet
    VARIABLES
     · i: entier
    DEBUT
     · i ← 1
(a) · TQ ((i <= |u|) ET (i <= |v|) ET (u[i] = v[i])) FAIRE
    ·  · i ← i + 1
    · FTQ
(b) · SI ((i > |u|) ET (i > |v|)) ALORS
    ·  · RENVOYER 0
(c) · SINON SI ((i > |u|) OU ((i <= |v|) ET (u[i] < v[i]))) ALORS
    ·  · RENVOYER -1
(d) · SINON
    ·  · RENVOYER +1
    · FSI
    FIN
```

ALGO. 1. Comparaison pour l'ordre lexicographique.

(1) Démontrez que l'algorithme s'arrête.

(2) Faites la preuve de correction partielle.

1. Version du 28 avril 2023, 11 : 21

Solution. **(1)** La variable i est initialisée à 1 et incrémentée d'une unité à chaque passage dans la boucle, la suite de ses valeurs est strictement croissante et non majorée, elle dépassera donc la longueur du mot u ou celle du mot v ce qui invalidera la condition d'entrée dans la boucle et l'algorithme s'arrête.

(2) Pour montrer que l'algorithme est correct, il faut s'assurer, d'une part que l'on n'accède jamais à une cellule d'un mot inexistante et d'autre part que l'algorithme renvoie toujours la bonne valeur.

Nous faisons l'hypothèse que dans notre pseudo-langage algorithmique, l'évaluation d'une forme disjonctive (resp. conjonctive) se fait de gauche à droite, et s'arrête dès que l'une des clauses est satisfaite (resp. n'est pas satisfaite). Ainsi, la condition `u[i]=v[i]` de la boucle `(a)` n'est évaluée que si $i \leqslant |u|$ et $i \leqslant |v|$ tout comme la condition `u[i]<v[i]` en `(c)`. On n'accède donc jamais à une cellule inexistante.

Il reste à montrer que l'algorithme renvoie la valeur 0 si et seulement si $u = v$, la valeur -1 si et seulement si u est un préfixe de v ou que u précède v, et la valeur 1 si et seulement si v est un préfixe de u ou que v précède u. En séparant $(u_i \neq v_i)$ en $(u_i < v_i) \vee (u_i > v_i)$, la proposition suivante est vraie à la sortie de la boucle `TQ` en `(a)` :

$$P : \quad (i > |u|) \vee (i > |u|) \vee (u_i < v_i) \vee (u_i > v_i).$$

		$i > \|u\|$	$i > \|v\|$	$u_i < v_i$	$u_i > v_i$	P
1	`(a)`	F	F	F	F	F
2	`(d)`	F	F	F	V	V
3	`(c)`	F	F	V	F	V
4	`(d)`	F	V			V
5	`(c)`	V	F			V
6	`(b)`	V	V			V

Table 1. Table de vérité de la proposition P.

Pour simplifier l'analyse, dressons la table de vérité de cette disjonction en fonction des 4 conditions qui la forment (cf. table 1). Les $2^4 = 16$ interprétations de cette table sont réduites à 6, d'une part parce que les différentes comparaisons entre u_i et v_i ne sont évaluées que si $\neg(i > |u|) \wedge \neg(i > |v|)$ (dans le cas contraire, les cases sont grisées) et d'autre part parce que $(u_i < v_i)$ et $(u_i > v_i)$ ne peuvent être simultanément vraies, éliminant ainsi cette interprétation.

- `(a)` Quand on entre dans la boucle `TQ`, seule l'interprétation 1 est possible.
- `(b)` Dans le `SI`, seule l'interprétation 6 est possible, ce qui entraîne bien l'égalité des deux mots u et v et on renvoie bien 0 si et seulement si $u = v$.
- `(c)` Dans le `SINON SI`, on d'une part $i \leqslant |u|$ ou $i \leqslant |v|$ et deux interprétations possibles. L'interprétation 5 correspond au cas où $i > |u|$ puisque nécessairement $i \leqslant |v|$ et le mot u est donc un préfixe de v. L'interprétation 3 correspond au cas où $i \leqslant |u|$ et $i \leqslant |v|$ et $u_i < v_i$, le mot u précède donc v dans l'ordre lexicographique. On renvoie donc la valeur -1 si et seulement si $u < v$.
- `(d)` Dans le `SINON`, les deux interprétations 4 et 2 restantes correspondent respectivement au cas où v est un préfixe de u et au cas où v précède x dans l'ordre lexicographique. On renvoie donc la valeur 1 si et seulement si $v < u$.

EXERCICE 4. **(1)** Modifiez l'algorithme TriRépartition pour écrire un algorithme `Partition(@L)` qui partitionne une liste L de mots en listes L_k des mots de longueur k et renvoie la table P de ces listes. à l'issue du partitionnement, la liste L est vide. On utilisera librement l'algorithme `SupTete(@L)` qui renvoie la valeur à la tête de la liste L et supprime cette cellule, et l'algorithme `InsQueue(@L,x)` qui insère une liste atomique d'une seule cellule contenant x à la queue de la liste L.

(2) Calculez la complexité de l'algorithme qui réalise cette partition. On supposera que l'algorithme `InsQueue(@L,x)` est en $\Theta(1)$.

Solution. **(1)** La première étape de l'algorithme consiste à calculer la longueur du mot le plus long pour l'allocation des "casiers" de répartition P. Notons que l'algorithme élimine un à un les mots dans la liste d'appel L au fur et à mesure de leur insertion dans les casiers de P. À la sortie de l'algorithme, la liste L est donc vide.

Notons $n := |L|$ la taille de la liste L. La recherche de la longueur du mot le plus long est en $\Theta(n)$ si l'on admet que le calcul de la longueur d'un

```
ALGORITHME Partition(@L):liste
DONNEES
 · L: liste de mots
VARIABLES
 · P: tableau de listes
 · mot: mot
DEBUT
 · lmax ← LongueurMax(L)
 · Allouer(P, lmax, [])
 · TQ (L != []) FAIRE
 ·  · mot ← SupTete(L)
 ·  · InsQueue(P[|mot|],mot)
 · FTQ
 · RENVOYER P
FIN
```

Algo. 2. Partition par les longueurs de mots.

mot est en $\Theta(1)$, ce qui n'est pas toujours pertinent selon le langage utilisé (cf. la fonction `strlen()` du *C* par exemple). L'allocation des casiers de répartition a un coût de $\Theta(l_{\max})$.

(2) Si l'on utilise une structure *ad hoc*, l'insertion en bout de liste peut se faire en $\Theta(1)$ même avec une liste simplement chaînée, il suffit que la structure intègre un pointeur vers le dernier élément de la liste et soit mis à jour pour les différents opérateurs de liste. L'extraction/répartition des n éléments de la liste L se fait donc en $\Theta(n)$. La complexité globale est donc en $\Theta(n) + \Theta(l_{\max}) = \Theta(n)$ si la longueur maximale des mots est faible devant le nombre n de mots de la liste L.

EXERCICE 5. Dans le tri lexicographique, pourquoi avant le tri par répartition sur la k-ème lettre, faut-il concaténer la liste L_k des mots de longueur k *avant* la liste L des autres mots ?

Solution. Si un mot $u = u_1u_2 \dots u_k$ de L_k est préfixe d'un mot $v = u_1u_2 \dots u_k \dots$ de L, alors il sera rajouté à la fin de la liste associée au symbole u_k *avant* le mot v. La concaténation des listes dans l'ordre alphabétique assure que u précèdera v et pas le contraire conformément à l'ordre lexicographique.