Algorithmique IV (UE-41) - TD 2.

TD 2. Machine RAM et langage algorithmique [1]

Pour tous les programmes de la machine RAM qui traitent une liste, une valeur particulière sert de balise de fin de lecture et n'est pas considérée comme une donnée.

EXERCICE 1. Écrivez un programme sur la machine RAM qui lit les données sur la bande d'entrée et les écrit sur la bande de sortie.

Solution. On suppose que la valeur 0 sert de balise de fin de lecture.

```
>00: READ    ; ACC ← ENTREE[I++]
 01: JUMZ 4  ; SI ACC = 0 SAUT EN #04 (FIN)
 02: WRITE   ; SORTIE[j++] ← ACC
 03: JUMP 0  ; SAUT EN #00 (NOUVELLE LECTURE)
>04: STOP    ; ARRET
```

EXERCICE 2. Écrivez un programme sur la machine RAM qui lit les données sur la bande d'entrée et les écrit dans l'ordre inverse en sortie.

Solution. Le caractère séquentiel des entrées/sorties impose de mémoriser toutes les valeurs de la bande d'entrée pour les écrire dans l'ordre inverse sur la bande de sortie. Le registre R_1 sert à l'adressage indirect pour mémoriser ces valeurs, le premier étant le registre R_2. La valeur 0 sert de balise de fin de lecture.

```
 00: LOAD #2  ; ACC ← 1
 01: STORE 1  ; R[1] ← ACC (INIT. INDEX)
>02: READ     ; ACC ← ENTREE[I++]
 03: JUMZ 7   ; SI ACC = 0 FIN LECTURE (SAUT EN #07)
 04: STORE @1 ; R[R[1]] ← ACC (RANGEMENT VALEUR)
 05: INC 1    ; R[1] ← R[1] + 1
 06: JUMP 2   ; SAUT EN #02
>07: DEC 1    ; R[1] ← R[1] - 1
 08: LOAD #1  ; ACC ← 1
 09: SUB 1    ; ACC ← ACC - R[1]
 10: JUMZ 14  ; SI ACC = 0 FIN (SAUT EN #14)
 11: LOAD @1  ; ACC ← R[R[1]] (CHARGEMENT VALEUR)
 12: WRITE    ; SORTIE[j++] ← ACC
 13: JUMP 7   ; SAUT EN #07
>14: STOP     ; ARRET
```

1. Version du 14 mars 2025, 14 : 04

EXERCICE 3. Écrivez un programme sur la machine RAM qui lit les données sur la bande d'entrée et renvoie la plus petite valeur sur la bande de sortie, on suppose que les valeurs lues sont $\geqslant 0$ et que la liste n'est pas vide.

Solution. Une valeur négative sert de balise de fin de lecture. Le minimum est rangé dans le registre R_1 qui est initialisé à la première valeur de la liste. Le registre R_2 permet de mémoriser la valeur lue avant de faire la soustraction avec R_1 dont le résultat permet de savoir s'il faut ou non mettre à jour R_1.

```
 00: READ     ; ACC ← ENTREE[I++]
 01: STORE 1  ; MIN ← ACC (INITIALISATION MIN)
>02: READ     ; ACC ← ENTREE[I++]
 03: JUML 11  ; SI ACC < 0 FIN (SAUT EN #11)
 04: STORE 2  ; R[2] ← ACC
 05: SUB 1    ; ACC ← ACC - MIN
 06: JUML 8   ; SI ACC < 0 NOUVEAU MIN (SAUT EN #08)
 07: JUMP 2   ; SAUT EN #02 (LECTURE SUIVANTE)
>08: LOAD 2   ; ACC ← R[2]
 09: STORE 1  ; MIN ← ACC (MISE A JOUR MIN)
 10: JUMP 2   ; SAUT EN #02 (LECTURE SUIVANTE)
>11: LOAD 1   ; ACC ← MIN
 12: WRITE    ; SORTIE[J++] ← ACC
 13: STOP     ; ARRET
```

EXERCICE 4. Écrivez un programme sur la machine RAM qui lit les données sur la bande d'entrée et renvoie les deux valeurs maximales sur la bande de sortie. On supposera que la bande contient au moins deux valeurs hors balise de fin de lecture.

Solution. On rangera la valeur maximale (`MMAX` dans le code) et la deuxième valeur maximale (`MAX` dans le code) dans les registres R_1 et R_2 respectivement. Le registre R_3 permettra de sauvegarder la valeur courante pour les calculs.

```
 00: READ     ; INITIALISATION MMAX (R[1]) ET MAX (R[2])
 01: STORE 1  ;   ↓
 02: READ     ;   ↓
 03: STORE 2  ; FIN INITIALISATION
 04: SUB 1    ; CALCUL R[2] - R[1]
 05: JUMG 23  ; SI R[2] > R[1] ECHANGER R[1]<>R[2] (SAUT EN #23)
>06: READ     ; ACC ← ENTREE[i++] (LECTURE x)
 07: JUMZ 30  ; FIN
```

```
 08: STORE 3   ; R[3] ← ACC (SAUVEGARDER x)
 09: SUB 1     ; ACC ← ACC - R[1] (x - MMAX)
 10: JUMG 18   ; SI x > MMAX MISE A JOUR MMAX ET MAX (SAUT EN #18)
 11: LOAD 3    ; RECHARGER x
 12: SUB 2     ; ACC ← ACC - R[2]
 13: JUMG 15   ; SI x > MAX MISE A JOUR MAX (SAUT EN #15)
 14: JUMP 6    ; SAUT EN #06 (NOUVELLE LECTURE)
>15: LOAD 3    ; RECHARGER x
 16: STORE 2   ; MISE A JOUR MAX (R[2])
 17: JUMP 6    ; FIN MISE A JOUR MAX
>18: LOAD 1    ; MISE A JOUR MMAX ET MAX
 19: STORE 2   ;   ↓
 20: LOAD 3    ;   ↓
 21: STORE 1   ;   ↓
 22: JUMP 6    ; FIN MISE A JOUR MMAX ET MAX
>23: LOAD 1    ; ECHANGE VALEURS R[1] <> R[2]
 24: STORE 3   ;   ↓
 25: LOAD 2    ;   ↓
 26: STORE 1   ;   ↓
 27: LOAD 3    ;   ↓
 28: STORE 2   ;   ↓
 29: JUMP 6    ; FIN ECHANGE
>30: LOAD 1    ; AFFICHAGE RESULTATS
 31: WRITE     ;   ↓
 32: LOAD 2    ;   ↓
 33: WRITE     ; FIN AFFICHAGE RESULTATS
 34: STOP      ; ARRET
```

EXERCICE 5. Calculez la complexité dans le meilleur et dans le pire des cas des algorithmes des exercices 1, 2 et 3.

Solution. Dans toute la suite n désigne la taille des données (sans compter la balise de fin de lecture). Pour l'exercice 1, le nombre d'instructions décodées ne dépend que du nombre de valeurs de la liste et pas de la nature de ces valeurs, il n'y a pas de meilleur ou de pire des cas. Les instructions de lecture `#00` et le test `#01` sont exécutée $n+1$ fois, les instructions d'écriture `#02` et le saut `#03` sont exécutées n fois et l'instruction d'arrêt `#04` une seule fois. On a donc

$$T(n) = 4n + 3.$$

Même remarque préliminaire pour l'exercice 2 qui affiche les éléments de la liste dans l'ordre inverse, la complexité ne dépend que de la taille de la liste, pas de ses valeurs. Les instructions `#00`, `#01` (initialisation du compteur) et `#14` (arrêt) ne sont exécutées qu'une seule fois. Les instruction `#02` et `#03` et `#07` à `#10` sont exécutées $n+1$ fois, les instructions `#04` à `#06` et `#11` à `#13` le sont n fois. On a

$$T(n) = 12n + 9.$$

Pour l'exercice 3, l'estimation *exacte* des fonctions de complexité devient fastidieuse (d'où l'introduction des notations asymptotiques). Dans toutes les situations, les instructions `#00`, `#01` (initialisation du minimum), `#11`, `#12` et `#13` (affichage du résultat et arrêt) ne sont exécutées qu'une seule fois. L'instruction de lecture `#02` et le test `#03` sont exécutés n fois. Les instructions `#04` à `#06` sont exécutées $n-1$ fois et les deux instructions de saut de retour `#07` et 10 sont au total appelées $n-1$ fois (la répartition entre les deux dépend de l'instance traitée). En revanche les instructions `#08` et `#09` de mise à jour du minimum dépendent de l'instance, elles ne sont jamais exécutées dans le meilleur des cas et $n-1$ fois dans le pire des cas. On a donc :

$$\check{T}(n) = 6n + 1 \quad \text{et} \quad \hat{T}(n) = 8n - 1.$$

EXERCICE 6. Réécrivez les algorithmes précédents en langage algorithmique et en considérant les données de la bande d'entrée et de la bande de sortie comme des listes.

Solution. Notons pour le premier exercice que l'on pourrait directement renvoyer la liste en entrée, sans faire une "copie", cette version a principalement pour objectif de calquer le fonctionnement de l'algorithme sur la machine RAM.

Les algorithmes auxilaires `InsTete(@L,x)` et `InsQueue(@L,x)` insèrent la valeur x en tête de liste et en queue de liste L respectivement. On supposera toujours que les données sont conformes à ce qui est attendu afin de ne pas compliquer inutilement les algorithmes. Ainsi dans la recherche du minimum, la liste en entrée est supposée contenir au moins un élément et dans la recherche des deux plus grandes valeurs, la liste est supposée contenir au moins deux termes.

```
ALGORITHME Entree-Sortie(L):liste
DONNEES
 · L: liste d'entiers
VARIABLES
 · M: liste d'entiers
 · i: entier
DEBUT
```

```
 · M ← []
 · i ← 1
 · TQ (i <=: L|)
 ·  · InsQueue(M,L[i])
 ·  · i ← i + 1
 · FIN
 · RENVOYER M
FIN

ALGORITHME Inverser(L):liste
DONNEES
 · L: liste d'entiers
VARIABLES
 · M: liste d'entiers
 · i: entier
DEBUT
 · M ← []
 · i ← 1
 · TQ (i <=: L|)
 ·  · InsTete(M,L[i])
 ·  · i ← i + 1
 · FIN
 · RENVOYER M
FIN

ALGORITHME Chercher-Minimum(L):entier
DONNEES
 · L: liste d'entiers
VARIABLES
 · min,i: entiers
DEBUT
 · min ← L[1]
 · i ← 2
 · TQ (i <=: L|)
 ·  · SI (L[i] < min) ALORS
 ·  ·  · min ← L[i]
 ·  · FSI
 ·  · i ← i + 1
 · FIN
 · RENVOYER min
FIN

ALGORITHME Chercher-2Max(L):couple
DONNEES
 · L: liste d'entiers
VARIABLES
 · mmax,max,i: entiers
DEBUT
 · mmax ← L[1]
 · max ← L[2]
 · SI (max > mmax) ALORS
 ·  · ECHANGER(mmax,max)
 · FSI
 · i ← 3
 · TQ (i <=: L|)
 ·  · SI (L[i] > mmax) ALORS
 ·  ·  · max ← mmax
 ·  ·  · mmax ← L[i]
 ·  · SINONSI (L[i] > max)
 ·  ·  · max ← L[i]
 ·  · FSI
 ·  · i ← i + 1
 · FIN
 · RENVOYER (mmax, max)
FIN
```

EXERCICE 7. Soit $b = b_0 b_1 \dots b_{n-1}$ une séquence binaire de n bits. On définit son *poids* $\pi(b)$ comme le nombre de ses bits non-nuls :

$$\pi(b) = \#\{i \in [\![0, n-1]\!] \mid b_i \neq 0\}. \tag{1}$$

(1) Écrivez en pseudo-langage algorithmique le calcul du poids d'une séquence binaire. La liste codant la séquence est indexée à partir de 0.

(2) Faites la preuve de correction de cet algorithme.

(3) Écrivez le même algorithme sur la machine RAM. Conventions : chaque chiffre binaire est rangé dans une cellule de la bande d'entrée et la valeur -1 sert de balise de fin de lecture.

(4) Calculez le nombre minimum, maximum et moyen d'instructions décodées par cette machine RAM. Indication : partitionnez l'ensemble $\Omega := \{0,1\}^n$ selon le poids de ses éléments.

(5) Reprenez le calcul du nombre moyen d'instructions en sommant les poids $\pi(b)$ des 2^n séquences binaires b.

Indications : faites une table des 8 séquences binaires possibles en les énumérant dans l'ordre de l'écriture binaire des entiers de 0 à $2^3 - 1$, par exemple. Observez la répartition des bits dans chacune des 3 colonnes. Que constatez-vous ? Généralisez aux séquences de longueur n.

Solution. **(1)** Voici un algorithme possible :

```
ALGORITHME Poids(B):entier
DONNEES
 · B: liste d'entiers
VARIABLES
 · i,P: entiers
DEBUT
 · P ← 0
 · i ← 0
 · TQ (i < #B) FAIRE
 ·  · SI (B[i] > 0) ALORS
 ·  ·  · P ← P + 1
 ·  · FSI
 ·  · i ← i + 1
 · FTQ
 · RENVOYER P
FIN
```

(2) La variable i est initialisée à la valeur 0 puis incrémentée à chaque passage dans la boucle, ses valeurs successives constituent une suite strictement croissante et non majorée car $(\mathbb{N}, \leqslant)$ est archimédien, assurant à terme la condition d'arrêt `(i >= #B)`.

On note n la taille de la liste B. Considérons le prédicat $P(i)$ suivant : "*P est le poids binaire de la sous-séquence $B[0, i-1]$*", avec pour convention que le poids de la séquence vide est nul. Initialisation : la proposition $P(0)$ est vraie avant l'entrée dans la boucle puisque la sous-séquence $B[0:-1]$ est vide. Hérédité : soit $i \in [\![0, n-1]\!]$ et supposons que la proposition $P(i)$ soit vraie à l'entrée de la boucle. Dans la structure conditionnelle *si*, le bit d'indice i est testé et la variable P est modifiée en conséquence, ainsi juste après ce test, c'est la proposition $P(i+1)$ qui est vraie et après l'incrémentation de la variable i, dernière instruction de la boucle, c'est à nouveau la proposition $P(i)$ qui est vraie. Nous venons de montrer que

$$\forall i \in [\![0, n-1]\!] \quad P(i).$$

En sortant de la boucle, la preuve d'arrêt nous montre que $i = n$ et la proposition $P(n)$ est vraie ce qui prouve la validité de l'algorithme.

(3) Dans le programme ci-dessous, le registre R_1 est initialisé à 0 et contient le poids de la séquence lue.

```
00: LOAD #0   ; ACC ← 0
01: STORE 1   ; R[1] ← ACC
02: READ      ; ACC ← ENTREE[I++]
03: JUML 7    ; SI ACC < 0 SAUT EN #07
04: JUMZ 2    ; SI ACC = 0 SAUT EN #02
05: INC 1     ; R[1] ← R[1] + 1
06: JUMP 2    ; SAUT EN #02
07: LOAD 1    ; ACC ← R[1]
08: WRITE     ; SORTIE[J++] ← ACC
09: STOP      ; ARRET
```

(4) On rappelle que n désigne le nombre de chiffres binaires de la séquence en entrée b. Les 5 instructions `#00`, `#01`, `#07`, `#08` et `#09` ne sont exécutées qu'une seule fois. Les 2 instructions `#02` et `#03` sont exécutées $n+1$ fois, 1 fois pour chacun des n chiffres et une dernière fois pour la balise -1. L'instruction `#04` est exécutée n fois, les instructions `#05` et `#06` sont exécutées $\pi(b)$ fois. Le nombre d'instructions $\eta(b)$ décodées pour traiter l'entrée b est alors :

$$\eta(b) = 3n + 2\pi(b) + 7 \tag{2}$$

Le nombre minimum d'instructions décodées est atteint pour la séquence de n bits tous nuls, i.e. $\pi(b) = 0$, et dans ce cas $\eta(b) = 3n + 7$. Le nombre maximum d'instructions décodées est atteint pour la séquence de n bits tous non-nuls, i.e. $\pi(b) = n$, et dans ce cas $\eta(b) = 5n + 7$.

(5) Le nombre d'instructions ne dépend que de la longueur et du poids de la séquence b, on partitionne alors l'ensemble $\{0,1\}^n$ de cardinal 2^n en $n+1$ classes

$$\Omega_p := \{b \in \{0,1\}^n \mid \pi(b) = p\} \qquad p \in [\![0, n]\!]. \tag{3}$$

Pour construire une séquence de poids p de longueur n, il faut choisir p bits égaux à 1 parmi ces n bits, on vérifie donc aisément que $\#\Omega_p = \binom{n}{p}$. On en déduit que le nombre moyen d'instructions $\overline{T}(n)$ est donné par

$$\begin{aligned}\overline{T}(n) &= \frac{1}{2^n}\sum_{p=0}^{n}(3n+2p+7)\binom{n}{p} \\ &= \frac{1}{2^n}\Big[(3n+7)\underbrace{\sum_{p=0}^{n}\binom{n}{p}}_{2^n} + 2\underbrace{\sum_{p=1}^{n}p\binom{n}{p}}_{n2^{n-1}}\Big].\end{aligned}$$

Les classes Ω_p formant une partition de Ω, la formule de sommation permet d'affirmer que la somme de leurs cardinaux est égale au cardinal de la réunion Ω, i.e. 2^n. Il reste donc à évaluer la seconde somme. On peut

utiliser la formule du pion ou encore considérer le polynôme $P(X) := (1+X)^n$ et son développement à l'aide de la formule du binôme de Newton :

$$P(X) = \sum_{p=0}^{n} \binom{n}{p} X^p. \tag{4}$$

On dérive les deux expressions de $P(X)$, à savoir $(1+X)^n$ et (4) :

$$\begin{aligned} P'(X) &= n(1+X)^{n-1} \\ &= \sum_{p=1}^{n} p \binom{n}{p} X^{p-1}. \end{aligned}$$

On en déduit que

$$\sum_{p=1}^{n} p\binom{n}{p} = P'(1) = n2^{n-1}, \tag{5}$$

ce qui nous permet de conclure que

$$\overline{T} = 4n + 7. \tag{6}$$

(5) En repartant de l'équation (2), on obtient aisément

$$\begin{aligned} \overline{T}(n) &= \frac{1}{2^n} \sum_{b=0}^{2^n-1} (3n + 2\pi(b) + 7) \\ &= (3n+7) + \frac{1}{2^{n-1}} \sum_{b=0}^{2^n-1} \pi(b). \end{aligned}$$

Pour illustrer le raisonnement, observons la table des $2^3 = 8$ séquences binaires de longueur 3 (cf. table 1). On constate que chaque colonne contient autant de 0 que de 1.

b_0	b_1	b_2	$\pi(b)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	2
1	0	0	1
1	0	1	2
1	1	0	2
1	1	1	3
4	4	4	12

Table 1. Poids des séquences binaires de longueur 3.

Démontrons le à l'aide d'une récurrence forte. Montrons que chacun des n chiffres d'une séquence binaire de longueur n prend autant de fois la valeur 0 que la valeur 1 quand on décrit les 2^n séquences possibles, autrement dit 2^{n-1} fois. C'est vrai pour $n = 1$ puisqu'il n'y a que deux séquences de longueur 1, les valeurs 0 et 1 y apparaissent chacune $2^0 = 1$ fois. Par récurrence, on construit toutes les séquences de longueur $n+1$ en préfixant toutes les séquences de longueur n d'une part par 0 et d'autre part par 1, la propriété reste donc vraie au rang $n+1$ si elle est vraie pour tous les rangs inférieurs ou égaux à n.

Pour calculer la somme des poids des séquences binaires de longueurs n, il suffit donc d'additionner les poids en colonnes plutôt qu'en ligne, soit $n\,2^{n-1}$ pour retrouver (6).