

Partie I : Tri dénombrement

- 1°) • Construire un tableau C de longueur $\text{MAX_VAL}+1$, ne contenant que des 0.
• Parcourir dans l'ordre croissant le tableau T à trier en incrémentant la case p de C chaque fois qu'on rencontre l'entier p dans T .
• Modifier le tableau C , en ajoutant au contenu de chaque case celui de la case précédente, en commençant par la première case. On obtient ainsi dans la case i de C le nombre d'éléments $\leq i$ contenus dans T .
• Mettre les éléments à leur place.
- Il y a trois parcours de C en plus de sa création, avec trois opérations élémentaires au plus à chaque case (lecture, addition, écriture), deux parcours de T avec une opération élémentaire à chaque case. Donc coût n .

2°)

```
let tri_simple tableau =
  let k = MAX_VAL + 1 in
  let n = t.(0) and c = make_vect k 0 in
  for i = 1 to n do let p=tableau.(i) in
    c.(p) <- c.(p) + 1 done;
  for p = 1 to k-1 do
    c.(p) <- c.(p) + c.(p-1) done;
  for j = 1 to c.(0) do tableau.(j) <- 0 done;
  for p = 1 to k-1 do
    for j = c.(p-1)+1 to c.(p) do
      tableau.(j) <- p done done;;
```

Partie II : Gestion de tables

1°)

```
let vider T = T.(0) <- 0;;
```

2°)

```
let ajouter T p = T.(0) <- T.(0) + 1; T.(T.(0)) <- p;;
```

3°)

```
let concatener T1 T2 =
  for i = 1 to T2.(0) do ajouter T1 T2.(i) done;;
```

- 4°) La complexité de concaténer est de l'ordre de n^2 , longueur de $T2$. En effet ajouter comporte 5 opérations élémentaires, et on l'applique n^2 fois. Les coûts marginaux étant bornés.

5°)

```
let max_valeurs T =
  let n = T.(0) in
  let m = ref (-1) in
  for i=1 to n do
    m:= max T.(i) !m done;
  !m;;
```

6°)

```
let rec nombre_chiffres = fun
  p when p<10 -> 1
  |p -> 1 + nombre_chiffres (p/10);;
```

7°)

```
let max_chiffres T = match T.(0) with
  0 -> 0
  |_ -> nombre_chiffres (max_valeurs T);;
```

- 8°) La complexité est de l'ordre de grandeur de $\text{maxc} + n$. En effet `nombre_chiffres` comporte $\text{maxc} - 1$ divisions et `max_valeurs` $n - 1$ comparaisons et n accès en mémoire.

Partie III : Tri par baquets

11^o) A l'étape suivante, on a `baquet[0] : [|603 ; 7|]` , `baquet[2] : [|20 ; 423 ; 27|]` , `baquet[5] : [|50 ; 453 ; 57|]` et `T : [|603 ; 7 ; 20 ; 423 ; 27 ; 50 ; 453 ; 57|]`. A la dernière étape, on a `baquet[0] : [|20 ; 27 ; 50 ; 57|]` , `baquet[4] : [|423 ; 453 ; 57|]` , `baquet[6] : [|603|]` , et `T` sera trié.

12^o) Invariant de boucle : à l'issue de la boucle d'indice r , les éléments sont classés dans l'ordre lexicographique croissant de leurs r derniers chiffres. La boucle suivante reprend ces éléments dans le tableau dans l'ordre croissant du $(r + 1)$ -ième chiffre.

13^o)

```
let distribuer T r baquets =
  let rieme_chiffre a r =
    let b = puiss10 (r-1) in (a/b) mod 10 in
  for i = 1 to T.(0) do
    let p = T.(i) in
    let k = rieme_chiffre p in
    ajouter baquets.(k) p done ; ;
```

14^o)

```
let tri_baquets T =
  let maxc = max_chiffres T in
  for r = 1 to maxc do
    for i = 0 to 9 do
      vider baquets.(i) done ;
    distribuer T r baquets ;
    vider T ;
    for i = 0 to 9 do
      concatener T baquets.(i) done done ; ;
```

15^o) `max_chiffres` est de coût $maxc + n$, `vider` et `rieme_chiffre` sont de coût constant, `distribuer` est donc de coût n . La concaténation de tous les `baquets.(i)` est de coût n puisque la somme des longueurs des `baquets.(i)` est n . Chacune des boucles r est donc de coût n , d'où le résultat.

16^o) $n \times \ln n$: $maxc$ est au moins de l'ordre de $\ln n$ si tous les nombres sont distincts. Ce coût est atteint s'il s'agit de tous les entiers de 1 à n .

17^o) On traite séparément les nombres en fonction de leur nombre de chiffres. On concatènera les résultats : un nombre de p chiffres est plus petit qu'un nombre de $p + 1$ chiffres. On ne s'occupe donc plus des 0 en tête des nombres.

```
let couper r U V W =
  vider V ; vider W
  for i = 1 to U.(0) do
    let k = U.(i) in
    if k < puiss10 r then ajouter V k
    else ajouter W k ; ;
```

Cette fonction coupe le vecteur U en deux : dans V on place tous les entiers ayant au plus r chiffres, et dans W les autres.

```
let tri_baquets_bis T =
  let r = ref 0 in
  let U = copy_vect T in
  vider T ;
  while U.(0) <> 0 do
    couper !r U V W ;
    tri_baquets V ;
    concatene T V ;
    let U = copy_vect V ;
    incr r done ; ;
```

Je ne suis pas vraiment les indications de l'énoncé, et j'utilise trois tableaux auxiliaires...

Mais le coût est le bon : couper est de coût $U.(0)$ et la somme de tous les $U.(0)$ qui interviennent est exactement le nombre total de chiffres. L'ensemble des `copy_vect` est de même coût, celui des `tri_baquets` également, celui des `concatene` est de coût n .

Partie I : Approche naïve

- 18^o) Si $L(\mathcal{A})$ n'est pas vide, soit m un mot de longueur minimale. m est nécessairement de longueur au moins n , il passe par $n+1$ états au moins. Il y a donc au moins un état par lequel il passe deux fois. On obtient un mot strictement plus court en supprimant toute la partie située entre ces deux passages. C'est impossible.
- 19^o) On remplace l'ensemble des états finaux par son complémentaire. Un mot de $L(\mathcal{A})$ s'achève à un état final, qui ne l'est plus, il n'est donc pas reconnu. Inversement un mot qui n'est pas dans $L(\mathcal{A})$ s'achève sur un état non final, qui l'est devenu.
- 20^o) On fait le produit cartésien des deux automates : $\langle \Sigma, Q \times Q', T'', I \times I', F \times F' \rangle$ où $((p, p'), a, (q, q')) \in T''$ si et seulement si $(p, a, q) \in T$ et $(p', a, q') \in T'$. Pour aboutir à un état final, il faut et il suffit que le mot soit reconnu par les deux automates.
- 21^o) Sinon $L(\mathcal{A}) \cap \overline{L(\mathcal{A}')} \neq \emptyset$ ou $\overline{L(\mathcal{A})} \cap L(\mathcal{A}') \neq \emptyset$. Dans le premier cas par exemple, d'après 18. , 19. et 20. , il existe un mot m dans $L(\mathcal{A}) \cap \overline{L(\mathcal{A}')}$ de longueur au plus $n \times n' - 1$.
- 22^o) On détermine, et on obtient le majorant $2^{n+n'} - 1$.

Partie II : Approche plus fine

- 23^o) Deux automates à deux états, l'un ne reconnaissant que le mot a , et l'autre tous les mots de longueur impaire.
- 24^o) Il existe un calcul d'origine i , d'extrémité j et d'étiquette $m = a_1 \dots a_k$ si et seulement si on a $(i_p, a_p, i_{p+1}) \in T$ pour tout p avec $i_1 = i$ et $i_{k+1} = j$. On applique la définition de φ_m .
- 25^o) $\varphi_m(e_1)$ vaut e_p où p est l'extrémité du calcul d'étiquette m et d'origine 1. Donc $\varphi_m(e_1).z$ vaut 1 ou 0 selon que le calcul est réussi ou non, c'est à dire selon que le mot est reconnu ou non.
- 26^o) $\Phi_m(E)$ vaut $(e_p; -e'_q)$, où p est l'extrémité du calcul de $\mathcal{A}$ d'étiquette m et d'origine 1, et q l'extrémité du calcul de $\mathcal{A}'$ d'étiquette m et d'origine 1. Donc $\Phi_m(E).Z = \varphi_m(e_1).z - \varphi'_m(e'_1).z'$ vaut 0 si $\varphi_m(e_1).z$ et $\varphi'_m(e'_1).z'$ prennent la même valeur, c'est à dire m est à la fois dans $L(\mathcal{A})$ et $L(\mathcal{A}')$ ou ni dans l'un ni dans l'autre. L'expression vaut 1 si m est dans $L(\mathcal{A})$ et pas dans $L(\mathcal{A}')$ et -1 si c'est le contraire .
- 27^o) Le système générateur de V_k est inclus dans celui de V_{k+1} .
- 28^o) C'est vrai pour φ_m , donc séparément pour les deux parties.
- 29^o) On le démontre pour un générateur de V_k : soit $w = \Phi_m(E)$ avec $|m| \leq k$, et on conclut par linéarité. En effet $\Phi_a(w) = \Phi_{ma}(E)$, et $|ma| \leq k+1$.
- 30^o) L'une des inclusions a déjà été faite.
Soit w un générateur de V_{k+2} . $w = \Phi_m(E)$ avec $|m| \leq k+2$. Si $|w| \leq k+1$, w est dans V_{k+1} , sinon $m = \mu a$, $w' = \Phi_\mu(E) \in V_{k+1}$, et on applique la question 29. à w' , donc ici aussi $w \in V_{k+1}$.
- 31^o) D'après les questions 27. et 30., la suite des sous-espaces V_k de $\mathbb{R}^{n+n'}$ est strictement croissante puis éventuellement constante. La suite des dimensions est une suite d'entiers, $V_0 = \text{Vect}(E)$ est de dimension 1, donc la dimension de V_k est au moins $k+1$, tant que la suite n'est pas constante. Donc la suite est constante au moins à partir de $k = n + n' - 1$.
- 32^o) $\mathcal{A}$ et $\mathcal{A}'$ sont équivalents si et seulement si $\Phi_m(E).Z$ est nul pour tous les $m \in \Sigma^*$. Cela revient à dire que les V_k sont dans l'hyperplan orthogonal de Z . Or $V_k \subseteq V_{n+n'-1}$ pour tout k . Tout revient à savoir si les $\Phi_m(E)$ sont dans l'hyperplan pour tout m vérifiant $|m| \leq n + n' - 1$.
La question 23. montre qu'on ne peut pas obtenir un meilleur majorant général dans le cas d'automates déterministes.
- 33^o) Pour deux automates déterministes le majorant trouvé est $n + n' - 1$. Par déterminisation on obtient le majorant $2^n + 2^{n'} - 1$ pour deux automates quelconques.