Homework Assignment 2

Matthieu Boyer

21 novembre 2023

Table des matières

1 Exercise 1

1.1 Question 1

For i, j in $|1, n|^2$, we have $AB_{i,j} = A_{i*}B_{*j} = \sum_{k=1}^{n} A_{i,k}B_{k,j}$, thus AB is computed by computed, for all i, j the product $A_{i,k}B_{k,j}$ and thus uses at most $\sum_{k=1}^{n} a_k b_k$ multiplications.

1.2 Question 2

The number of multiplications and additions is two times the number of multiplications. We just need to get a majoration of the number of multiplications. Yet, since $a_k \leq n$ for all k, $\sum_{k=1}^{n} a_k b_k \leq n \sum_{k=1}^{n} b_k = mn$. Then the number of multiplications and additions required is $\mathcal{O}(mn)$.

1.3 Question 3

Multiplying a matrix in $\mathcal{M}_{ap,bp}$ by a matrix $\mathcal{M}_{bp,cp}$ can, by seeing the matrices as block matrices, be seen as multiplying two matrices in $\mathcal{M}_p$ the number of times we need to compute the product of matrix in $\mathcal{M}_{a,b}$ by a matrix in $\mathcal{M}_{b,c}$:

$$\begin{pmatrix} \alpha_{1,1} & \dots & \alpha_{1,bp} \\ \vdots & & \vdots \\ \alpha_{ap,1} & \dots & \alpha_{ap,bp} \end{pmatrix} = \begin{pmatrix} A_{1,1} & \dots & A_{1,b} \\ \vdots & & \vdots \\ A_{a,1} & \dots & A_{a,b} \end{pmatrix} \text{ where } A_{i,j} = \begin{pmatrix} \alpha_{ip+1,jp+1} & \dots & \alpha_{ip+1,j(p+1)} \\ \vdots & & \vdots \\ \alpha_{i(p+1),jp+1} & \dots & \alpha_{i(p+1),j(p+1)} \end{pmatrix} \in \mathcal{M}_p$$

Thus, we can multiply a matrix in $\mathcal{M}_{ap,bp}$ by a matrix $\mathcal{M}_{bp,cp}$ in $M(a,b,c)M(p,p,p)$ multiplications. Thus :

$$M(ap, bp, cp) \leq M(a,b,c)M(p,p,p)$$

1.4 Question 4

- If $0 \leq r \leq \alpha$: $w(1,r,1)$ is the smallest number k such that $M(n,n^r,n) = \mathcal{O}(n^{k+o(1)})$. But, again by seeing A an $n \times n^\alpha$ matrix as a $n \times n^r$ matrix next to a $n, n^\alpha - n^r$ matrix and same for B, we get $M(1,r,1) \leq M(1,\alpha,1)$ and thus $w(1,r,1) \leq w(1,\alpha,1) = 2$.
- If $\alpha \leq r \leq 1$: by seeing a $n \times n^r$ matrix A as a $n^{\frac{1-r}{1-\alpha}} \times n^{\frac{(1-r)\alpha}{1-\alpha}}$ bloc matrix with blocks of size $n^{\frac{r-\alpha}{1-\alpha}} \times n^{\frac{r-\alpha}{1-\alpha}}$ and applying the reasoning from 3. we get that :

$$\begin{aligned} M(n,n^r,n) &= M\left(n^{\frac{1-r}{1-\alpha}} \cdot n^{\frac{r-\alpha}{1-\alpha}}, n^{\frac{(1-r)\alpha}{1-\alpha}} \right. \\ &\qquad \left. \times n^{\frac{r-\alpha}{1-\alpha}}, n^{\frac{1-r}{1-\alpha}} \cdot n^{\frac{r-\alpha}{1-\alpha}}\right) \\ &\leq M\left(n^{\frac{1-r}{1-\alpha}}, n^{\frac{(1-r)\alpha}{1-\alpha}}, n^{\frac{1-r}{1-\alpha}}\right) \\ &\qquad \times M\left(n^{\frac{r-\alpha}{1-\alpha}}, n^{\frac{r-\alpha}{1-\alpha}}, n^{\frac{r-\alpha}{1-\alpha}}\right) \\ &= \mathcal{O}\left(\left(n^{\frac{1-r}{1-\alpha}}\right)^{w(1,\alpha,1)} \left(n^{\frac{r-\alpha}{1-\alpha}}\right)^{\omega}\right) \\ &= \mathcal{O}\left(n^{\frac{2*(1-r)+(r-\alpha)\omega}{1-\alpha}}\right) \end{aligned}$$

  The first big O equality comes from a substitution in $M(n,n^\alpha,n) = \mathcal{O}(n^{w(1,\alpha,1)})$ of n by $n^{\frac{1-r}{1-\alpha}}$. We obtain :

$$\begin{aligned} w(1,r,1) &\leq \frac{2 \times (1-r) + (r-\alpha)\,\omega}{1-\alpha} \\ &= \frac{2 \times (1-\alpha) + 2 \times (\alpha - r) + (r-\alpha)\,\omega}{1-\alpha} \\ &= 2 + \frac{\omega \times (\alpha - r) - 2 \times (\alpha - r)}{1-\alpha} \\ &= 2 + \beta(r-\alpha) \end{aligned}$$

1.5 Question 5

Let $1 \leq l \leq n$, a_k, b_k such that $a_k b_k$ is decreasing. If $l = 1$, the result is trivial.
Consider for $i \in [\![1, l-1]\!]$ the quantity $a_i b_j + a_j b_i$, we get :

- If $a_i > a_j$: $a_i b_j + a_j b_i > a_j b_j$
- Else : $a_i b_j + a_j b_i > a_i b_i > a_j b_j$ by hypothesis.

Then we get :

$$\sum_{j=i+1}^{n} a_i b_j + a_j b_i > \sum_{k=l}^{n} a_k b_k$$

since the a_k and b_k are positive. By summing :

$$\sum_{i=1}^{l-1} \sum_{j=i+1}^{n} a_i b_j + a_j b_i > l \sum_{k=l}^{n} a_k b_k$$

But :

$$m_1 m_2 = \sum_{i=1}^{n} a_i \sum_{j=1}^{n} b_j = \sum_{i=1}^{n} \sum_{j=i+1}^{n} a_i b_l > \sum_{i=1}^{l-1} \sum_{j=i+1}^{n} a_i b_j$$

Thus :

$$\sum_{k=l}^{n} a_k b_k < \frac{m_1 m_2}{l}$$

1.6 Question 6

We will first prove correctness by induction :
Initialization : If $m^2 \leq n^2$, it is clear.
Heredity : Since π is bijective, $I \sqcup J = [\![1, n]\!]$, we get that no two columns in A_{*I} and A_{*J} have an element in the same row. Then, if $i, j \in [\![1, n]\!]$, we get :

$$[AB]_{i,j} = \sum_{k \in [\![1,n]\!]} a_{i,k} b_{k,j} = \sum_{k \in I} a_{i,k} b_{k,j} + \sum_{k \in J} a_{i,k} b_{k,j} = [A_{*I} B_{I*}]_{i,j} + [A_{*J} B_{J*}]_{i,j}$$

Hence, the algorithm is correct.

Then, for the complexity we get that :

- If $m \leq n$, the algorithm runs in $\mathcal{O}(mn)$.
- Else, its complexity is the sum of three steps :
  - Calculating $C_1 = A_{*I} B_{I*}$ takes $M(n, l, n) = \mathcal{O}(n^{w(1,r,1)+o(1)})$ multplications for $r = \frac{\ln l}{\ln n} < 1$.
  - From question 5), calculating $C_2 = A_{*J} B_{J*}$ takes at most $\frac{m^2}{l}$ multiplications.
  - Summing the results takes $\mathcal{O}(n^2)$ operations.

We obtain a time complexity in the worst case in $\mathcal{O}(n^{w(1,r,1)+o(1)} + \frac{m^2}{l} + n^2)$, assuming that $m^2 > n^2$. In that case, we let $l = m^{\frac{2}{\beta+1}} n^{\frac{\alpha\beta-2}{\beta+1}}$. Then, the computation of C_1 takes $\mathcal{O}(n^{w(1,r,1)+o(1)}) = \mathcal{O}(n^{2+\beta\left(\frac{\ln l}{\ln n} - \alpha\right)})$ from question 4. Then, we get :

$$\begin{aligned}
n^{2+\beta\left(\frac{\ln l}{\ln n} - \alpha\right)} &= n^{2-\alpha\beta} l^{\beta} \\
&= n^{2-\alpha\beta} m^{\frac{2\beta}{\beta+1}} n^{\frac{\alpha\beta^2 - 2\beta}{\beta+1}} \\
&= m^{\frac{2\beta}{\beta+1}} n^{2-\alpha\beta+\frac{\alpha\beta^2}{\beta+1} - \frac{2\beta}{\beta+1}} \\
&= m^{\frac{2\beta}{\beta+1}} n^{\frac{2}{\beta+1} + \frac{\alpha\beta^2 - \alpha\beta(\beta+1)}{\beta+1}} \\
&= m^{\frac{2\beta}{\beta+1}} n^{\frac{2-\alpha\beta}{\beta+1}}
\end{aligned}$$

Moreover, since the computation of C_2 is done in $\mathcal{O}(\frac{m^2}{l})$, replacing l by its value :

$$\frac{m^2}{l} = m^{2-\frac{2}{\beta+1}} n^{\frac{2}{\beta+1} - \frac{\alpha\beta}{\beta+1}} = m^{\frac{2\beta}{\beta+1}} n^{\frac{2-\alpha\beta}{\beta+1}}$$

In the end, we get a total complexity in :

$$\mathcal{O}\left(m^{\frac{2\beta}{\beta+1}} n^{\frac{2-\alpha\beta}{\beta+1}} + o(1)\right) + \mathcal{O}\left(m^{\frac{2\beta}{\beta+1}} n^{\frac{2-\alpha\beta}{\beta+1}}\right) + \mathcal{O}(n^2) = \mathcal{O}\left(m^{\frac{2\beta}{\beta+1}} n^{\frac{2-\alpha\beta}{\beta+1} + o(1)}\right)$$

Using the given values for α and β, this time complexity is in

$$\mathcal{O}(m^{0.7} n^{1.21} + n^{2+o(1)})$$

2 Exercise 2

2.1 Question 1

Since concatenation is associative, we can look at the concatenation of a list of length n as the concatenation of the numbers resulting of the concatenation of any partition $A_1, \ldots, A_d$ of the list such that if $i < j$, $a_j \in A_k \Rightarrow a_i \in A_l$ where $l \leq k$. Thus, if the list is sorted, after concatenation of a part of its elements (provided they are neighbours), the obtained list remains sorted.

Since for one and two elements, it is clear that results hold, we only need to cover the case of an insertion. Indeed, if sorting by insertion gives a correct result, then any sorting algorithm gives a correct result. Suppose that a list with n elements $a_0, \ldots, a_{n-1}$ is sorted using the order defined. Let a_n be another number, and suppose it is inserted between a_i and a_{i+1} by insertion sort. Then, suppose that the result we obtain is not the right one. We return to the case of three elements, since the final concatenation can be seen as the one of the results of the concatenation of $(a_0, \dots, a_j)$, (a_n) and $(a_{j+1}, \dots, a_{n-1})$. Indeed, the values are still sorted using our order. The result obtained by the case of three elements would not be right, and thus the list would not be sorted.

In the case of three elements, $a_0 \preceq a_1 \preceq a_2$. By checking all 6 permutations of the elements, we obtain the wanted result.

2.2 Question 2

Consider two integers X and Y. Using their generalised suffix tree (that we suppose to have been preprocessed), we can find the Lowest Common Ancestor of the numbers as a whole in constant time. We then can compare XY and YX in constant time, by looking at the next digit in both numbers from the LCA. If the LCA is X, then, either Y is longer than X and then we return Y or $X = Y$. Else, we return the number with the greatest next digit.

2.3 Question 3

There are $10^{\log_{10}(n)/4} = \sqrt[4]{n}$ numbers that, in base 10, are written using at most $\frac{\log_{10} n}{4}$ digits. Then, we can sort all the integers using counting sort : we create an array A of size $\sqrt[4]{n}$ with only zeros. Then, if we see i in the list of numbers with at most $\frac{\log_{10} n}{4}$ digits, we increase $A[i]$ by 1. This is done in $\mathcal{O}(n)$ since we see each digit at most once. Then, again, by iterating through $i \in [\![0, \sqrt[4]{n} - 1]\!]$ we can add i to a list $A[i]$ times. This is again done in $\mathcal{O}(n)$.

2.4 Question 4

There are at most $\frac{n}{\frac{\log_{10} n}{4}}$ numbers with at least $\frac{\log_{10} n}{4}$ digits. Then using an optimal sorting algorithm, since the comparisons are done in $\mathcal{O}(1)$ by question 2), we get a complexity in :

$$\begin{aligned}
\mathcal{O}\left(\frac{4n}{\log_{10} n} \log_{10}\left(\frac{4n}{\log_{10} n}\right)\right) &= \mathcal{O}\left(4n \times \frac{\log_{10}(4) + \log_{10}(n) - \log_{10}(\log_{10}(n))}{\log_{10}(n)}\right) \\
&= \mathcal{O}\left(n \times \left(1 + \frac{\log_{10}(4)}{\log_{10}(n)} - \frac{\log_{10}\left(\log_{10}(n)\right)}{\log_{10}(n)}\right)\right) \\
&= \mathcal{O}(n)
\end{aligned}$$

2.5 Question 5

We will denote by $|X|$ the number of digits of X and by A, k the input array and its length. Based on the previous questions, we propose the following algorithm :

1. We compute the generalised suffix tree of the numbers. This is done in
$$\mathcal{O}\left(\sum_{i \in [\![0, k-1]\!]} |A[i]|\right) = \mathcal{O}(n)$$
2. We first sort the digits with at most $\frac{\log_{10} n}{4}$ digits. From this, by question 3), we can sort all the numbers is $\mathcal{O}(n)$ time.
3. Then, by question 4), we can sort the rest of the numbers in $\mathcal{O}(n)$ time.
4. We then merge the two sorted arrays in linear time in the sum of the lengths, thus in $\mathcal{O}(n)$.
5. Finally, we can compute the concatenation in $\mathcal{O}(n)$.

This algorithm is correct by question 1), and has time complexity $\mathcal{O}(n)$.