ISM MP, Informatique
Année 2024/2025

Corrigé du TP Informatique 02

Exercice 1

1. On saisit :

```
def eval(L):
    P=Pile()
    for x in L:
        if type(x)==float or
           type(x)==int:
            P.empiler(x)
        else:
            a=P.depiler()
            b=P.depiler()
            if x=="+":
                P.empiler(a+b)
            if x=="-":
                P.empiler(b-a)
            if x=="*":
                P.empiler(b*a)
            if x=="/":
                P.empiler(b/a)
    return P.depiler()
```

2. On saisit :

```
def inftopost(E):
    res,op=[],Pile()
    k,n=0,len(E)
    chiffre=".0123456789"
    while k<n:
        cur=E[k]
        k+=1
        if cur in chiffre:
            x=cur
            while E[k] in chiffre:
                x+=E[k]
                k+=1
            if "." in x:
                x=float(x)
            else:
                x=int(x)
            res.append(x)
        elif cur in ["+","-","*","/"]:
            op.empiler(cur)
        elif cur==")":
            res.append(op.depiler())
    return res
```

Exercice 2

1. On saisit :

```
    def produit(M1,M2):
        """produit récursif de deux matrices carrées de taille 2**N"""
        if len(M1)==1:
            return M1*M2
        else:
            A1,B1,C1,D1=scinde(M1);A2,B2,C2,D2=scinde(M2)
            P1=produit(A1,A2);P2=produit(B1,C2)
            P3=produit(A1,B2);P4=produit(B1,D2)
            P5=produit(C1,A2);P6=produit(D1,C2)
            P7=produit(C1,B2);P8=produit(D1,D2)
            return fusion(P1+P2,P3+P4,P5+P6,P7+P8)
```

2. On teste :

```
>>> M1=rd.random((4,4));M2=rd.random((4,4))
>>> np.dot(M1,M2)
array([[ 2.13111079,  1.58658899,  1.37798278,  ...
>>> produit(M1,M2)
array([[ 2.13111079,  1.58658899,  1.37798278,  ...
```

3. On saisit :

```
def strassen(M1,M2):
    """produit récursif par l'algorithme de Strassen"""
    if len(M1)==1:
        return M1*M2
    else:
        A1,B1,C1,D1=scinde(M1)
        A2,B2,C2,D2=scinde(M2)
        X1=strassen(A1+D1,A2+D2);X2=strassen(C1+D1,A2)
        X3=strassen(A1,B2-D2); X4=strassen(D1,C2-A2)
        X5=strassen(A1+B1,D2)
        X6=strassen(C1-A1,A2+B2);X7=strassen(B1-D1,C2+D2)
        return fusion(X1+X4-X5+X7,X3+X5,X2+X4,X1-X2+X3+X6)
```

4. On teste :

```
>>> strassen(M1,M2)
array([[ 2.13111079,  1.58658899,  1.37798278,  ...
```

5. On obtient :

```
Comparaison produit récursif naïf VS Strassen
produit récursif naïf =  54.4931640625
Strassen =  33.05960488319397
```

On constate que l'algorithme de Strassen est beaucoup plus efficace que l'algorithme récursif naïf.

6. La performance de l'algorithme de Strassen tient au fait qu'on effectue seulement 7 multiplications matricielles contre 8 avec l'algorithme naïf. Sur des matrices de grande taille, cet avantage est décisif pour la complexité temporelle, ce qu'on constate lors de l'expérimentation. Les opérations de fusion et scission sur des matrices de $\mathscr{M}_{2^{n-1}}(\mathbb{R})$ et $\mathscr{M}_{2^n}(\mathbb{R})$ sont en $\mathrm{O}(4^n)$, proportionnelles au nombre de coefficients de ces matrices. L'addition de deux matrices de $\mathscr{M}_{2^{n-1}}(\mathbb{R})$ est en $\mathrm{O}(4^n)$ puisqu'il faut réaliser $(2^{n-1})^2$ additions. Notons $\mathrm{P}(2^n)$ le nombre d'opérations faites par `produit` pour des matrices de $\mathscr{M}_{2^n}(\mathbb{R})$. On effectue 4 additions sur des matrices de taille 2^{n-1} plus des opérations de fusion et scission soit un coût en $\mathrm{O}(n^4)$ et 8 appels récursifs sur des matrices de $\mathscr{M}_{2^{n-1}}(\mathbb{R})$. Ainsi

$$\mathrm{P}(2^n) = 8\mathrm{P}(2^{n-1}) + \mathrm{O}(4^n)$$

On pose $$\forall n \in \mathbb{N} \qquad u_n = \frac{\mathrm{P}(2^n)}{8^n}$$

Il vient $$\forall n \in \mathbb{N}^* \qquad u_n - u_{n-1} = \mathrm{O}\left(\frac{1}{2^n}\right)$$

Il s'ensuit que la série téléscopique $\sum [u_{n+1} - u_n]$ converge d'où la convergence de $(u_n)_n$ ce qui prouve que

$$\mathrm{P}(2^n) \underset{n\to+\infty}{\sim} 8^n = (2^n)^3$$

Notons $\mathrm{S}(n)$ le nombre d'opérations faites par `strassen` pour des matrices de $\mathscr{M}_{2^n}(\mathbb{R})$. On effectue 18 additions sur des matrices de tailles 2^{n-1} plus des opérations de fusion et scission soit un coût en $\mathrm{O}(4^n)$ et 7 appels récursifs sur des matrices de $\mathscr{M}_{2^{n-1}}(\mathbb{R})$. Ainsi

$$\mathrm{S}(2^n) = 7\mathrm{S}(2^{n-1}) + \mathrm{O}(4^n)$$

On procède exactement comme avec $\mathrm{P}(2^n)$ et on obtient

$$\mathrm{S}(2^n) \underset{n\to+\infty}{\sim} 7^n = (2^n)^{\log_2(7)} \quad \text{avec} \quad \log_2(7) \simeq 2.8$$

L'algorithme de Strassen[1] a a marqué un tournant dans l'histoire de l'algorithmique puisqu'il constitue le premier algorithme de calcul de produit matriciel qui ne soit pas en $\mathrm{O}(\mathrm{N}^3)$ où N désigne la taille des matrices à multiplier ($\mathrm{N} = 2^n$ dans notre étude précédente). La quête d'un algorithme de produit en $\mathrm{O}(\mathrm{N}^2)$ est un sujet de recherche toujours d'actualité (dernier en date : 2020, algorithme de Josh Alman et Virginia Vassilevska en $\mathrm{O}(n^{2,3728596})$).

Exercice 3

1. On saisit :

```
def percole(milieu,deb):
    """Détermine s'il y a percolation dans milieu depuis deb"""
    chemin=Pile()
    if etat(milieu,deb)==0:
        chemin.empiler(deb)
    pos=deb
    ligne_max=len(milieu)-1
    while pos[0]<ligne_max and not chemin.vide():
        visite(milieu,pos)
        if voisinage(milieu,pos):
            chemin.empiler(pos)
            pos=voisin(milieu,pos)
        else:
            pos=chemin.depiler()
    if pos[0]==ligne_max:
        visite(milieu,pos)
    return not chemin.vide()
```

1. Volker Strassen, né en 1936, mathématicien allemand - Algorithme de Strassen en 1969

2. On saisit :

```
def percolation(milieu):
    """Détermine s'il y a percolation dans milieu depuis première ligne"""
    deb=[0,-1]
    colonne_max=len(milieu[0])-1
    flag_percole=False
    while not flag_percole and deb[1]<colonne_max:
        deb[1]+=1
        flag_percole=percole(milieu,deb)
    return flag_percole
```

3. On saisit :

```
N=100;tab_n=[10,20,30,40];tab_p=np.linspace(0,1,50)

for n in tab_n:
    tab_pc=[]
    for p in tab_p:
        nb_percolation=0
        for k in range(N):
            milieu=[[int(rd.rand()>p)*2 for j in range(n)] for i in range(n)]
            nb_percolation+=percolation(milieu)
        tab_pc.append(nb_percolation/N)
    plt.plot(tab_p,tab_pc)
plt.show()
```

On obtient :

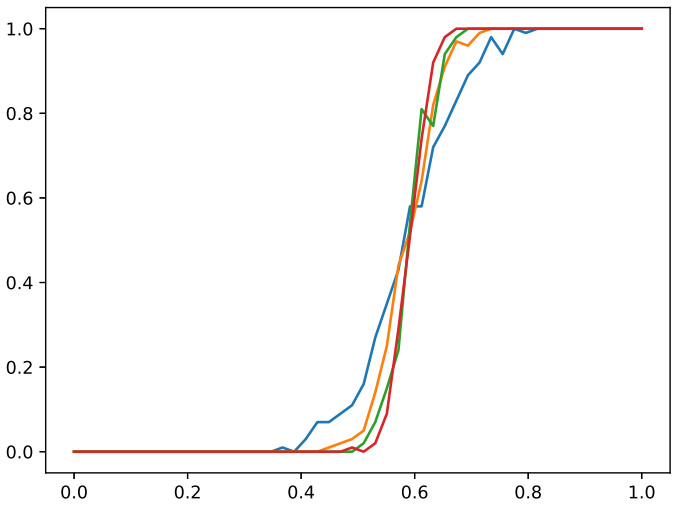

FIGURE 1 – Probabilité critique

On observe un phénomène de *probabilité critique* indépendemment de la taille du milieu $n \times n$. Pour une valeur $p_c \simeq 0.6$, on constate que si $p < p_c$, il n'y a pas percolation et si $p > p_c$, alors il y a percolation. Et cette valeur de probabilité critique p_c ne semble pas dépendre du choix de n.