

Oraux blancs MPI

type CCINP

Florian Bourse

Exercice A.

Dans cet exercice, on s'intéresse à l'utilisation de sémaphores pour gérer la concurrence entre fils d'exécution. On suppose que les sémaphores sont tous initialisés avec la valeur 0. Pour un sémaphore S , on note :

- $P(S)$ l'opération qui décrémente S ou met le fil d'exécution en file d'attente si sa valeur est 0.
- $V(S)$ l'opération qui réveille un fil d'exécution en attente s'il en existe ou incrémente la valeur du sémaphore sinon.

On supposera également pour l'instant que les opérations P et V ne sont exécutés qu'une seule fois par sémaphore.

On considère les pseudo-codes suivants exécutés en parallèle par deux fils d'exécution, après initialisation de deux sémaphores S_1 et S_2 :

```
void A(void *args){  
    printf("A1");  
    V(S_2);  
    P(S_1);  
    printf("A2");  
}
```

```
void B(void *args){  
    printf("B1");  
    V(S_1);  
    P(S_2);  
    printf("B2");  
}
```

1. On souhaite créer un graphe de dépendance entre les instructions des différents fils d'exécutions. On suppose que pour chaque fil d'exécution, les instructions sont effectuées dans l'ordre donc chacune des instructions dépend de la précédente. Que peut-on dire des dépendances entre les $P(S_i)$ et les $V(S_i)$? Tracer le graphe de dépendance pour les 8 instructions des fonctions A et B.
2. Que représente l'ordre d'exécution des instructions pour ce graphe? Donner l'ensemble des chaînes de caractères qui peuvent être affichées par les fonctions A et B.
3. Proposer un algorithme permettant de détecter une situation menant à un interblocage.
4. On s'autorise à présent à utiliser plusieurs fois les opérations P et V sur le même sémaphore. Généraliser la synchronisation imposée par les fonction A et B à un nombre arbitraire de fils d'exécutions A_1, \dots, A_n .

Exercice B. L'exercice suivant est à traiter dans le langage C

Le code fourni peut être compilé avec la commande

```
gcc *.c -o test
```

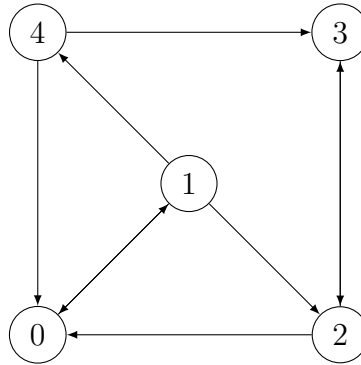
puis exécuté avec la commande

```
./test
```

Dans cet exercice, tous les graphes seront orientés. Les graphes seront représentés par leur matrice d'adjacence : $G = (S, A)$, avec $S = \{s_0, \dots, s_{n-1}\}$ est représenté par la matrice \mathbf{G} telle que

$$G_{i,j} = \begin{cases} 1 & \text{si } (s_i, s_j) \in A \\ 0 & \text{sinon} \end{cases}$$

Un programme en C vous est fourni dans lequel le graphe suivant est représenté par la variable `g_exemple`.



Une fonction `erdos_renyi` vous est également fournie pour générer un graphe aléatoire sur n sommets dans lequel chaque arête existe avec probabilité p .

Dans ce sujet nous nous intéressons à compter ou à détecter les triangles dans un graphe, un triangle étant un cycle de longueur 3.

1. Donner le nombre de triangles contenus dans le graphe donné en exemple.
2. Compléter la fonction `est_triangle` qui permet de vérifier si le triplet (s_0, s_1, s_2) est un triangle du graphe G .
3. Écrire une fonction `int ** init_matrice(int n)` renvoyant une nouvelle matrice de dimensions $n \times n$ ne contenant que des 0.
4. Écrire une fonction `int ** prod(int n, int ** A, int ** B)` calculant le produit \mathbf{AB} de deux matrices de dimensions $n \times n$. On rappelle que si $\mathbf{C} = \mathbf{AB}$, alors on a :

$$C_{i,j} = \sum_{k=0}^{n-1} A_{i,k} B_{k,j}$$

5. Montrer que la puissance k -ième de la matrice \mathbf{G} contient le nombre de chemins de longueurs k , plus précisément, $G_{i,j}^k$ contient le nombre de chemins de longueur exactement k de i à j .
6. En déduire une fonction `int compte_triangles(int n, int ** g)` permettant de compter le nombre de triangles d'un graphe.

7. Afin d'améliorer la complexité, on souhaite utiliser l'algorithme de multiplication de matrices de Strassen. En décomposant les matrices par bloc :

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix}, \quad B = \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix}, \quad C = \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

En posant :

- $M_1 = (A_{1,1} + A_{2,2})(B_{1,1} + B_{2,2})$
- $M_2 = (A_{2,1} + A_{2,2})B_{1,1}$
- $M_3 = A_{1,1}(B_{1,2} - B_{2,2})$
- $M_4 = A_{2,2}(B_{2,1} - B_{1,1})$
- $M_5 = (A_{1,1} + A_{1,2})B_{2,2}$
- $M_6 = (A_{2,1} - A_{1,1})(B_{1,1} + B_{1,2})$
- $M_7 = (A_{1,2} - A_{2,2})(B_{2,1} + B_{2,2})$

On a :

- $C_{1,1} = M_1 + M_4 - M_5 + M_7$
- $C_{1,2} = M_3 + M_5$
- $C_{2,1} = M_2 + M_4$
- $C_{2,2} = M_1 - M_2 + M_3 + M_6$

Implémenter cet algorithme.

8. Déterminer la relation de récurrence vérifiée par la complexité en temps de l'algorithme de Strassen.
9. On s'intéresse maintenant uniquement à la détection de triangle, c'est-à-dire renvoyer V si le graphe contient un triangle et F sinon. On propose l'algorithme probabiliste suivant :
- (a) tirer une arête (u, v) aléatoirement ;
 - (b) tester si v possède un successeur qui a pour successeur u .
- À quelle famille d'algorithmes appartient-il ?
10. Proposer un algorithme permettant de détecter les triangles avec une fiabilité supérieure à 95 pourcents.