P-RAM, suite et fin.

1 Procédure mystère

On définit les deux opérateurs suivants pour un tableau $A = [a_0, a_1, \ldots, a_{n-1}]$ de n entiers :
– PRESCAN(A) renvoie le tableau $[0, a_0, a_0 + a_1, a_0 + a_1 + a_2, \ldots, a_0 + a_1 + \ldots + a_{n-2}]$
– SCAN(A) renvoie le tableau $[a_0, a_0 + a_1, a_0 + a_1 + a_2, \ldots, a_0 + a_1 + \ldots + a_{n-1}]$
Nous avons vu en cours comment réaliser ces deux opérateurs en temps $O(\log n)$ sur une P-RAM EREW.

▷ **Question 1** *Étant donné un tableau de booléens Flags, que fait la procédure suivante ?*

```
SPLIT(A,Flags)
1: Idown ← PRESCAN(not(Flags))
2: Iup ← n - REVERSE(SCAN(REVERSE(Flags)))
3: Pour i = 1 à n en parallèle :
4:   Si Flags(i) Alors
5:     Index[i] ← Iup[i]
6:   Sinon
7:     Index[i] ← Idown[i]
8: Result ← PERMUTE(A,Index)
9: Renvoyer Result
```

Les noms des différentes fonctions sont relativement intuitifs ; en particulier, REVERSE *renverse le tableau, et* PERMUTE(A,*Index*) *réordonne le tableau A selon la permutation Index. L'horrible expression* REVERSE(SCAN(REVERSE(*Flags*))) *effectue simplement un* SCAN *à partir de la fin du tableau Flags, dont les éléments sont considérés comme des entiers.*

Voici un exemple d'utilisation :

A	=	[5	7	3	1	4	2	7	2]
Flags	=	[1	1	1	1	0	0	1	0]
Idown	=	[0	0	0	0	*0*	*1*	2	*2*]
Iup	=	[*3*	*4*	*5*	*6*	7	7	*7*	8]
Index	=	[3	4	5	6	0	1	7	2]
Result	=	[4	2	2	5	7	3	1	7]

Quel est le coût de la fonction SPLIT *?*

On considère la procédure MYSTÈRE suivante :

```
MYSTÈRE(A, Number_Of_Bits)
1: Pour i = 0 à Number_Of_bits − 1 :
2:   bit(i) ← tableau indiquant si le i-ème bit des éléments de A est à 1
3:   A ← SPLIT(A,bit(i))
```

▷ **Question 2** *Faire tourner la procédure sur $A = [5, 7, 3, 1, 4, 2, 7, 2]$ avec Number_Of_Bits $= 3$. Que fait la procédure* MYSTÈRE *?*

▷ **Question 3** *Avec des entrées de taille $O(\log n)$ bits, quelle est la complexité avec n processeurs ? Et avec seulement p processeurs ? Quelles sont les valeurs de p les plus intéressantes ?*

2 Composantes connexes

On souhaite concevoir un algorithme CREW qui permette de calculer les composantes connexes d'un graphe $G = (V, E)$ dont les sommets sont numérotés de 1 à n. Plus précisément, on cherche un algorithme qui renvoie un tableau C de taille n tel que $C(i) = C(j) = k$ si et seulement si i et j sont dans la même composante connexe et k est le plus petit indice des sommets de cette composante.

Définition 1. À toute étape de l'algorithme, on appellera pseudo-sommet étiqueté par i l'ensemble de sommets $j, k, l, \cdots \in V$ tels que $C(j) = C(k) = C(l) = \cdots = i$. On assimilera le pseudo-sommet i étiqueté par i au sommet étiqueté par i.

Un des invariants de l'algorithme est que le plus petit indice des sommets constituant un pseudo-sommet étiqueté par i est i et que les sommets appartenant à un pseudo-sommet sont dans la même composante connexe. Cette assertion est donc vraie si on initialise C par : pour tout $i \in V = [\![1, n]\!]$: $C(i) = i$. Ceci signifie que chaque processeur se considère au départ comme sommet de référence de sa composante connexe. L'objectif de l'algorithme est de modifier ce point de vue égocentrique.

Définition 2. Une arborescence k-cyclique ($k \geqslant 0$) est un graphe orienté faiblement connexe (c'est-à-dire tel que le graphe non orienté sous-jacent est connexe) tel que :
- tout sommet a un degré sortant égal à 1 et
- il existe exactement un circuit de longueur $k + 1$.

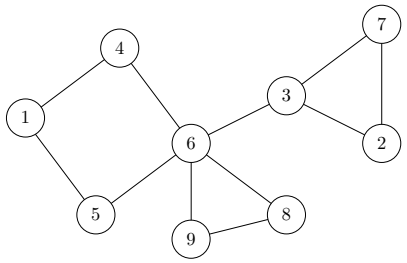
On appelle étoile une arborescence 0-cyclique dans laquelle toutes les arêtes sont incidentes à la racine et l'indice de la racine est le plus petit indice dans l'étoile.

L'invariant précédent est donc que le graphe orienté $(V, \{(i, C(i)) \mid i \in V\})$ est constitué d'étoiles. On peut donc identifier pseudo-sommets et étoiles, le centre de l'étoile étant l'indice du pseudo-sommet. Le calcul des composantes connexes s'effectue en enchaînant plusieurs fois de suite les deux fonctions suivantes :

```
GATHER()
1: Pour tout i ∈ S en parallèle :
2:    T(i) ← min{C(j) | {i,j} ∈ E, C(j) ≠ C(i)}
          {si l'ensemble est vide, on associe C(i)}
3: Pour tout i ∈ S en parallèle :
4:    T(i) ← min{T(j) | C(j) = i, T(j) ≠ i}
          {si l'ensemble est vide, on associe C(i)}

JUMP()
5: Pour tout i ∈ S en parallèle :
6:    B(i) ← T(i)
7: Pour j = 1 à log n :
8:    Pour tout i ∈ S en parallèle :
9:       T(i) ← T(T(i))
10: Pour tout i ∈ S en parallèle :
11:   C(i) ← min{B(T(i)), T(i)}
```

▷ **Question 4** *On considère le graphe suivant.*



Appliquer la fonction GATHER *sur ce graphe, puis la fonction* JUMP*, puis la fonction* GATHER*, et ainsi de suite. Il sera instructif d'observer l'effet des opérations sur les graphes orientés* $(V, \{(i, T(i)) \mid i \in V\})$ *et* $(V, \{(i, C(i)) \mid i \in V\})$.

▷ **Question 5** *Montrez que la fonction* GATHER *peut s'exécuter en temps* $O(\log n)$ *avec* n^2 *processeurs.*

▷ **Question 6** *Montrer qu'après l'application de la fonction* GATHER*, les composantes connexes contenant plusieurs pseudo-sommets induisent des arborescences* 1*-cycliques dans le graphe orienté* $(V, \{(i, T(i)) \mid i \in V\})$*. On notera également que le plus petit pseudo-sommet d'une arborescence* 1*-cyclique appartient au cycle.*

▷ **Question 7** *Montrer que la fonction* JUMP *transforme une arborescence* 1*-cyclique en étoile (ou pseudo-sommet).*

▷ **Question 8** *Montrer qu'après* $\lceil \log n \rceil$ *enchaînements des fonctions* GATHER *et* JUMP*, les composantes connexes du graphe sont représentées par les pseudo-sommets induits par* C*.*

▷ **Question 9** *Quelle est la complexité de l'algorithme ? Combien de processeurs sont utilisés ?*