Révisions

1 Ordonnancement sur 1 processeur...

On veut ordonnancer un ensemble de n tâches indépendantes $T_1, T_2, \ldots, T_n$. La durée de T_i est p_i. On suppose que l'ordonnancement débute au temps $t = 0$, et on note C_i la date de la fin de l'exécution de la tâche T_i. On dispose d'un seul processeur.

▷ **Question 1** *Quel est l'ordre d'exécution optimal si l'objectif est de minimiser la somme des dates de fin* $\sum_{i=1}^{n} C_i$ *?*

▷ **Question 2** *Quel est l'ordre d'exécution optimal si l'objectif est de minimiser la somme pondérée des dates de fin, i.e.* $\sum_{i=1}^{n} w_i.C_i$*, où* w_i *est un poids positif associé à la tâche* T_i *?*

2 Permutation de boucles

On considère un nid de boucle parfait de profondeur n, qui comprend m dépendances uniformes. Soit D la matrice de taille $n \times m$, obtenue à partir du signe des composantes des vecteurs de dépendance. Si d_i est le i-ème vecteur de dépendance, on stocke le signe $+$, 0 ou $-$ de ses composantes dans la i-ème colonne de D. Par exemple si $n = 3$ et $d_1 = (2, -2, 0)^t$, on stockera $(+, -, 0)^t$ dans la première colonne de D.

▷ **Question 3** *Donner la matrice* D_1 *pour le nid suivant :*

```
for i = 1 to n do
  for j = 1 to n do
    for k = 1 to n do
      S1 : a(i+1, j, k) = a(i, j, k) + 2
      S2 : b(i, j, k+1) = b(i, j, k) - 1
      S3 : c(i+1, j+1, k+1) = c(i, j, k) + 1
    endfor
  endfor
endfor
```

▷ **Question 4** *Donner la matrice* D_2 *pour le nid suivant :*

```
for i = 1 to n do
  for j = 1 to n do
    for k = 1 to n do
      S1 : a(i, j, k+1) = a(i, j-1, k) + a(i-1, j, k+2) + 2
    endfor
  endfor
endfor
```

▷ **Question 5** *Donner un nid dont la matrice* D_3 *soit*

$$D_3 = \begin{pmatrix} + & + & + & 0 & 0 & 0 \\ + & 0 & 0 & + & 0 & 0 \\ 0 & + & 0 & 0 & + & 0 \\ 0 & 0 & + & 0 & 0 & + \end{pmatrix}$$

L'algorithme de parallélisation fonctionne ainsi :

- Si une ligne de la matrice D (disons la i-ème) ne contient que des 0, alors on place cette boucle à l'extérieur du nid et on et on l'exécute en parallèle : on permute les boucles, la i-ème boucle devient la première boucle, et on remplace le *for* par un *forpar*.
- Si aucune ligne de la matrice D ne contient que des 0, on sélectionne la ligne qui contient le plus de $+$ et aucun $-$, on place la boucle correspondante à l'extérieur et on l'exécute en séquentiel.

On exécute alors récursivement l'algorithme sur les boucles et les dépendances restantes, i.e. qui n'ont pas été satisfaites par la séquentialisation de la boucle.

Par exemple pour D_1, on choisit de séquentialiser soit la première soit la troisième boucle. Si on choisit la première, pas besoin de permuter, la première et troisième dépendances sont satisfaites. Il reste deux boucles à l'intérieur, avec la matrice réduite $\overline{D}_1 = \begin{pmatrix} 0 \\ + \end{pmatrix}$. On en déduit le code parallèle :

```
forseq i
  forpar j
    forseq k
      ...
```

Bien sûr, on aurait pu choisir d'abord la boucle sur k et obtenir :

```
forseq k
  forpar j
    forseq i
      ...
```

▷ **Question 6** *Faire tourner l'algorithme sur l'exemple avec D_2.*

▷ **Question 7** *Faire tourner l'algorithme sur l'exemple avec D_3.*

▷ **Question 8** *Prouver la correction de l'algorithme. Pour cela, prouver qu'on a le droit de déplacer à l'extérieur et paralléliser une boucle correspondant à une ligne nulle, et préciser quelles sont les dépendances qui sont satisfaites après le choix d'une boucle, son déplacement à l'extérieur et sa séquentialisation.*

▷ **Question 9**

Pensez-vous que cet algorithme conduise à un nombre maximal de boucles parallèles ? (utiliser l'exemple avec D_3)

▷ **Question 10** *Montrer que trouver le nombre maximal de boucles parallèles est NP-complet, par réduction à partir de SET-COVER : étant donnés un ensemble X à n éléments, k sous-ensembles de X et une borne B, peut-on trouver B sous-ensembles parmi les k tels que tout élément de X appartienne à au moins l'un de ces B sous-ensembles ? On pourra supposer que la matrice ne contient aucun $-$.*

3 P-RAM : Rupture de symétrie déterministe

On veut concevoir un algorithme EREW qui permet de sélectionner "beaucoup" d'objets dans une liste chaînée sans jamais choisir deux objets adjacents dans la liste. Chaque objet est associé à un processeur mais le numéro du processeur n'est pas relié à l'ordre des éléments dans la liste.

▷ **Question 11** *Concevoir un algorithme qui permet de sélectionner un nombre optimal d'objets en temps $O(log\ n)$ sur une EREW.*

Dans la suite on va montrer qu'il est possible d'exrtaire "beaucoup" d'éléments en un temps $O(log^* n)$ où

$$log^* n = min\{i | log^i n \leq 1\}$$

Dans l'expression précédente, log^i représente la composition de i fois la fonction log. La fonction log^* a une croissance très lente, puisque $log^*(2^{65536}) = 5$.

Nous allons maintenant préciser le sens de "beaucoup" à partir de la notion d'ensemble indépendant maximal.

Définition. Un ensemble de sommets V' d'un graphe $G = (V, E)$ est indépendant ssi

$$\forall (a, b) \in E, \text{ au plus un élément de } \{a, b\} \text{ est dans } V'.$$

Un ensemble de sommets V' d'un graphe $G = (V, E)$ est indépendant et maximal ssi l'ajout de tout sommet à V' en fait un ensemble non indépendant.

Dans ce qui suit, notre objectif est d'arriver à extraire un ensemble indépendant maximal de la liste en temps $O(log^* n)$. Pour cela, on va commencer par colorier la liste (avec 6 couleurs). On va construire un algorithme qui part d'une $n-$coloration (où la couleur de chaque objet est déterminée par la couleur du processeur qui lui est associé) et qui diminue à chaque étape le nombre de couleurs utilisées.

▷ **Question 12** *Donner un algorithme astucieux pour diminuer le nombre de couleurs utilisées tout en gardant une coloration (on pourra raisonner sur le codage binaire des couleurs).*

▷ **Question 13** *Pourquoi obtient-on 6 couleurs ? Montrer qu'on obtient 6 couleurs après $O(log^* n)$ étapes.*

4 Réponses aux exercices

▷ Question 12, page 2

Pour obtenir une coloration en un temps $O(log^*n)$, il faut que le nombre de couleurs nécessaire r_{k+1} soit de l'ordre de $log(r_k)$. Considérons deux nœuds successifs a et b à l'étape k dont les couleurs sont respectivement codées par

$$C^k(a) =< a_{r_k-1}, a_{r_k-2}, \ldots, a_0 > \text{ et } C^k(b) =< b_{r_k-1}, b_{r_k-2}, \ldots, b_0 >.$$

À l'étape $k+1$, on définit le codage de a par :

$$C^{k+1}(a) =<< i >, a_i >$$

où i est le plus petit indice tel que $a_i \neq b_i$ et $< i >$ le codage binaire de i. Comme $i \in [0, r_k - 1]$, on vérifie que la longueur du codage binaire de i est $\leq \lceil log\ r_k \rceil$. Ainsi si on note

$$C^{k+1}(a) =< a_{r_{k+1}-1}, a_{r_{k+1}-2}, \ldots, a_0 >$$

le codage de a après la $k+1^e$ étape, on a

$$r_{k+1} = \lceil logr_k \rceil + 1$$

Comme la couleur d'un nœud est définie à partir de la couleur de son successeur, il est nécessaire de définir la couleur du dernier nœud d de façon particulière, et on pose

$$C^{k+1}(d) =< 0^{r_{k+1}}, d_0 > .$$

Pour vérifier la correction de l'algorithme, il suffit de vérifier qu'on obtient bien un coloriage à chaque étape, c'est-à-dire que si deux nœuds successifs a et b ont une couleur différente à l'étape k, ils ont bien une couleur différente à l'étape $k+1$.

Si $C^k(a)$ et $C^k(b)$ sont différents, le plus petit indice i où les deux codages diffèrent est bien défini. Soit $C^{k+1}(a) =<< i >, a_i > et C^{k+1}(b) =<< j >, b_j >$ les codages des couleurs de a et b après l'étape $k+1$. Si $i \neq j$, alors les codages sont différents et si $i = j$ alors $a_i \neq b_i$ par construction.

On a donc construit un algorithme qui permet à chaque étape d'obtenir un nouveau coloriage des nœuds de la liste.

▷ Question 13, page 3

Si r_k vaut 3, on vérifie que r_{k+1} vaut également 3, et le nombre de couleurs utilisées stagne. Plus précisément, comme la position à laquelle les codages des couleurs de deux nœuds consécutifs peut être 0, 1 ou 2, les seuls codages qu'on peut obtenir sont $< 000 >$, $< 001 >$, $< 010 >$, $< 011 >$, $< 100 >$, $< 101 >$, ce qui conduit bien à 6 couleurs différentes. Pour obtenir le 6 coloriage promis, on va utiliser l'algorithme suivant :

– **Tant que** $r \geq 5$, Effectuer une étape de l'algorithme de coloriage.

– Effectuer 3 étapes de l'algorithme de coloriage

Les 3 étapes finales nous assurent qu'on obtient bien un $6-$coloriage. Pour étudier la complexité de l'algorithme, il est nécessaire de montrer le lemme suivant :

Dans la boucle **Tant que** on a $r_k \geq 5$ et $r_k \leq \lceil logn \rceil + 2$. Cette propriété est vérifiée de façon immédiate pour $k = 1$, puisque $r_1 \leq \lceil logn \rceil$. Supposons $r_k \geq 5$ et $r_k \leq \lceil log^k n \rceil + 2$. On vérifie que $r_{k+1} =$

$$\lceil log\, r_k \rceil + 1 \leq \lceil log(\lceil log^k\, n \rceil + 2) \rceil + 1 \leq \lceil log(log^k\, n + 3) \rceil + 1 \leq \lceil log(2log^k\, n) \rceil + 1 \leq \lceil log(log^k\, n) + 1 \rceil + 1 \leq \lceil log^{k+1}\, n \rceil + 2$$

Soit $m = log^*n$. On vérifie que $log^{m-1}n + 2 \leq 4$ et donc le nombre d'itérations de la boucle **Tant que** est inférieure à $m - 2$. Le nombre d'étapes total de recoloriage de l'algorithme est $m + 1$.