Disque dur à deux têtes (X MP-PC 2006)

Durée : 2 heures

Ce problème étudie des stratégies de déplacement des têtes d'un disque dur afin de minimiser le temps moyen d'attente entre deux requêtes au disque dur (de lecture ou d'écriture). Dans ce problème, le disque dur est représenté par une demi-droite $[0,+\infty[$ et possède deux têtes de lecture/écriture. Chacune des têtes peut aller indifféremment à n'importe quelle position sur le disque pour y lire ou écrire une donnée. Les deux têtes peuvent être au même endroit ou encore se croiser. On ne s'intéresse qu'aux temps de déplacement des têtes et non aux temps de lecture/écriture. Les deux têtes ont la même vitesse de déplacement. Le temps de déplacement d'une tête est supposé égal à la distance qu'elle parcourt.
Une requête r est un entier positif ou nul représentant l'emplacement du disque auquel l'une des deux têtes doit se rendre. Initialement les deux têtes sont chacune à la position 0.
Le disque dur est muni d'une mémoire (appelée cache) qui permet d'enregistrer n requêtes ($n > 0$) avant de les traiter. À chaque bloc de n requêtes présentes dans le cache, le contrôleur du disque dur doit alors satisfaire ce bloc de requêtes, dans leur ordre d'arrivée, en minimisant le déplacement total des deux têtes. L'ordre importe puisqu'une opération d'écriture peut précéder une autre opération de lecture ou d'écriture. Il faut donc déterminer pour chacune des n requêtes le numéro de la tête à déplacer de manière à minimiser la somme totale des temps de tous les déplacements.

Partie I. Coût d'une séquence de déplacements

Un bloc de n requêtes est représenté par une suite de n entiers positifs ou nuls $[r_0, r_1, \ldots, r_{n-1}]$ rangés dans une liste nommée `R`. Une séquence de déplacements $[d_0, d_1, \ldots, d_{n-1}]$ est une suite de n entiers, exclusivement 0 ou 1, rangés dans une liste nommée `D` (de même taille que `R`) indiquant à l'étape i qui de la première tête ($d_i = 0$) ou de la deuxième tête ($d_i = 1$) doit se déplacer à la position r_i ($1 \leqslant i \leqslant n$).

Le *coût* d'une séquence de déplacements est la somme totale des distances parcourues par chacune des têtes. Ainsi pour le bloc de requêtes $[5, 2, 4]$, le coût de la séquence de déplacements $[0, 0, 1]$ est $5 + 3 + 4 = 12$, alors que le coût de $[0, 1, 0]$ vaut $5 + 2 + 1 = 8$.

Question 1. Proposer une fonction `cout(R, D)` prenant en argument les deux listes `R` et `D` et retournant le coût de la séquence de déplacements `D` pour le bloc de requêtes `R`.

Question 2. Combien de séquences de déplacements peuvent satisfaire un bloc de requêtes `R` ?

Le *coût optimal* d'une suite de requêtes `R` est le plus petit coût des séquences de déplacements satisfaisant le bloc de requêtes `R`.

Question 3. Justifier qu'il existe toujours une séquence de déplacements de coût optimal qui commence par 0, c'est-à-dire commençant par déplacer la première tête.

Partie II. Coût optimal pour deux requêtes

Dans cette partie, le cache est de taille 2 ($n = 2$). Il n'y a donc que deux requêtes r_0 et r_1. **Par convention, la première tête sera toujours celle qui bouge sur la première requête.** Les seules séquences de déplacements possibles seront donc $D = [0, 0]$ et $D = [0, 1]$.

Question 4. Donner une séquence de déplacements de coût minimal pour chacun des deux blocs de requêtes $[10, 3]$ et $[3, 10]$.

Question 5. Expliquer brièvement comment choisir la tête effectuant le second déplacement dans le cas d'un cache ne contenant que deux requêtes.

Question 6. Proposer une fonction `optimal2(R)` prenant en argument un bloc de **deux** requêtes $[r_0, r_1]$ et retournant une liste $[d_0, d_1]$ de coût minimal. Si plusieurs séquences de coût minimal existent, on pourra librement choisir celle retournée.

Partie III. Coût optimal pour trois requêtes

Dans cette partie, le cache est de taille 3 ($n = 3$). Il y a donc trois requêtes r_0, r_1 et r_2. Par convention, la première tête sera toujours celle qui bouge sur la première requête ($d_0 = 0$).

Question 7. En supposant que l'on ait utilisé le critère de la question 5 pour déterminer quelle tête s'est déplacée pour la seconde requête, et que l'on utilise ce même critère pour déterminer celle qui se déplace pour la troisième requête, déterminer la séquence de déplacement d que l'on obtient pour la requête $[20, 9, 1]$, et le coût correspondant.

Question 8. Montrer que cette stratégie n'est pas optimale.

Question 9. Proposer une fonction `optimal3(R)` prenant en argument une liste `R` représentant un bloc de **trois** requêtes $[r_0, r_1, r_2]$ et retournant une liste `D` de longueur trois contenant une séquence de déplacements $[d_0, d_1, d_2]$ de coût minimal. Si plusieurs séquences de coût minimal existent, on pourra librement choisir celle retournée.

Partie IV. Coût optimal pour n requêtes

On souhaite à présent déterminer le coût optimal pour n requêtes (sans se préoccuper de la détermination de la séquence de déplacement permettant d'obtenir un tel coût). Par commodité, chacune des deux têtes peut effectuer indifféremment le premier déplacement.

Pour simplifier l'écriture de l'algorithme, on représentera les requêtes comme une liste de taille $n + 1$ où $r_0 = 0$ et $r_1, r_2, \ldots, r_n$ représentent les requêtes à effectuer. Tout se passe comme si on ajoutait une pseudo-requête supplémentaire pour la position zéro en début de liste, requête qui ne contribuera pas au coût puisque les têtes de lecture sont déjà initialement sur cette position.

À un instant donné, la configuration des têtes du disque dur est représentée par un couple (i, j) codant le numéro des deux dernières requêtes respectivement satisfaites par chacune des deux têtes : la première tête a satisfait en dernier la requête r_i et la deuxième tête la requête r_j. Par convention, on suppose que les deux têtes ont satisfait la pseudo-requête $r_0 = 0$. la configuration initiale est donc $(0, 0)$.

À chaque requête r_k (où $0 \leqslant k \leqslant n$), on associe une liste coût_k de taille n tel que $\text{coût}_k[i]$ (avec $i < n$) est égal au coût optimal pour atteindre la configuration (i, k) juste après la requête r_k. On posera $\text{coût}_k[i] = -1$ si la configuration (i, k) ne peut être atteinte. Remarquons que, pour des raisons de symétrie entre les deux têtes, il s'agit également du coût optimal pour atteindre la configuration (k, i).
Prenons par exemple le bloc de requêtes `R = [0, 7, 3, 1, 6, 2, 5]`. On a initialement

coût_0 :

0	-1	-1	-1	-1	-1

puis après la requête r_1

coût_1 :

7	-1	-1	-1	-1	-1

après la requête r_2

coût_2 :

11	10	-1	-1	-1	-1

et après la requête r_3

coût_3 :

13	12	12	-1	-1	-1

Cette dernière liste se lit par exemple de la façon suivante : après la requête $r_3 = 1$,

- si la tête qui n'a pas satisfait r_3 n'a jamais bougé (configuration (0,3)), le coût minimal possible aboutissant dans cette configuration est 13 ;
- si la dernière requête satisfaite par la tête qui n'a pas satisfait r_3 est r_1 (configuration (1,3)), le coût minimal possible est 12 ;
- si la dernière requête satisfaite par la tête qui n'a pas satisfait r_3 est r_2 (configuration (2,3)), le coût minimal possible est 12.

Les -1 restant indiquent simplement des situations impossibles : la configuration $(3, 3)$ n'est pas permise car une seule tête a satisfait la requête r_3, et les configuration $(k, 3)$ avec $k > 3$ ne sont pas permises car la requête r_k n'a pas encore pu être satisfaite.

Question 10. On se place dans la situation où c'est la tête de lecture qui a réalisé la requête r_{k-1} qui réalise la requête r_k. Expliquer, pour $k > 1$, comment calculer $\text{coût}_k[i]$ pour $0 \leqslant i < k-1$ à partir de la liste coût_{k-1} et des r_k.

Question 11. On se place dans la situation où c'est la tête de lecture qui n'a pas réalisé la requête r_{k-1} qui réalise la requête r_k. Expliquer comment déterminer, pour $k \geqslant 1$, $\text{coût}_k[k-1]$ à partir de la liste coût_{k-1} et des r_k.

Question 12. Déterminer les listes coût_4, coût_5 et coût_6.

Question 13. En s'appuyant sur les questions 10 et 11, proposer une fonction `suivant(C, R, k)` prenant en argument la liste `C` représentant coût_{k-1}, la liste `R` contenant le bloc de requêtes $[r_0, r_1, \ldots, r_n]$ et l'entier $k \geqslant 1$ et retournant la liste représentant coût_k.

Question 14. En déduire une fonction `coutsOpt(R)` construisant et retournant une liste `couts` contenant les $n+1$ listes (de taille n) coût_k. Ainsi, `couts[k][i]` devra correspondre à $\text{coût}_k[i]$ et l'appel `coutsOpt([0, 7, 3, 1, 6, 2, 5])` devra retourner :

```
[[0,  -1, -1, -1, -1, -1],
 [7,  -1, -1, -1, -1, -1],
 [11, 10, -1, -1, -1, -1],
 [13, 12, 12, -1, -1, -1],
 [...                   ],
 [...                   ],
 [...                   ]]
```

où les listes incomplètes sont celles déterminées à la question 12.

Question 15. Quelle est la complexité temporelle de cet algorithme ?

Question 16. Comment déduire du résultat retourné par la fonction `coutsOpt(R)` le coût optimal permettant de satisfaire toutes les requêtes ? Proposer une fonction `coutOptimal(R)` retournant le coût optimal associé au bloc de requêtes représenté par la liste `R`.