X Informatique MP/PC 2006 — Corrigé

Ce corrigé est proposé par Jean-Baptiste Rouquier (ENS Lyon) ; il a été relu par Walter Appel (Professeur en CPGE) et Céline Chevalier (ENS Cachan).

Le thème de ce sujet est la répartition des ordres de lecture et écriture entre deux têtes d'un disque dur, de façon à minimiser la distance totale parcourue, donc la durée de ces opérations. Il s'agit surtout d'un prétexte à un exercice intéressant, car le modèle est assez éloigné de la réalité. On y voit un exemple (avec étapes détaillées) de programmation dynamique, qui est une technique classique de programmation.

Le sujet est bien équilibré entre questions théoriques et questions de programmation. Contrairement au sujet de l'an dernier, celui-ci est bien détaillé. La difficulté est progressive sur l'ensemble du sujet (on fera attention aux indices dans la dernière sous-partie). Le découpage en sous-parties est surtout un regroupement des quelques questions détaillant un même but.

Nous avons rédigé les codes source en Maple, car ce langage est le plus connu en classes préparatoires (par ceux qui ne suivent pas l'option informatique), même s'il est plus destiné au calcul formel qu'à la programmation. D'autres langages sont autorisés pour cette épreuve, dont Caml (Caml Light et OCaml), C, C++, Java et Pascal. Les fonctions écrites n'utilisant qu'une syntaxe simple, le code est compréhensible même si vous ne maîtrisez pas Maple.

Indications

Première partie

1 Définir deux variables `position_tete1` et `position_tete2` pour mémoriser la position des têtes, ainsi que `resultat` pour accumuler la distance parcourue.

2 Considérer une séquence optimale (on montrera qu'il en existe une). Que se passe-t-il si on échange les deux têtes ?

4 Le premier déplacement est fixé ; examiner les deux cas du second.

5 Comparer les coûts des deux possibilités et renvoyer le tableau correspondant.

Seconde partie

6 La règle de décision de la question 5 consiste à satisfaire une requête par la tête la plus proche.

8 Énumérer brutalement toutes les possibilités et renvoyer le tableau correspondant à la bonne.

9 Comme il n'y a pas de contrainte sur la configuration finale, chercher celle qui coûte le moins cher.

10 Remarquer (c'est sous-entendu dans l'énoncé) qu'il est inutile de déplacer une tête après avoir satisfait une requête (on la déplacera s'il le faut lors de la prochaine requête), puis justifier les égalités une à une. Pour la deuxième, quelles étaient les configurations possibles juste avant de satisfaire la k^{e} requête ? Pour la troisième, considérer l'échange des deux têtes.

11 Si l'on fait les mises à jour dans l'ordre de la question 10, on est assuré de ne pas modifier une case avant de la lire.

12 Créer un tableau `cout` puis utiliser les questions 9, 10 et 11. Précisément, initialiser le tableau `cout` selon la question 10, en déduire $\texttt{cout}_n$ grâce à la question 11 et extraire enfin le résultat comme indiqué à la question 9.

13 À l'étape k, `cout`$[i]$ contient le coût d'une séquence optimale finissant dans la configuration (i, k). Adapter alors les égalités de la question 10 puis les fonctions `mettreAJour` et `coutOpt`. L'ordre des mises à jour est différent de celui de la question 11.

Partie I

A. Coût d'une séquence de déplacements

1 On définit deux variables `position_tete1` et `position_tete2`, de type entier, pour mémoriser la position courante des têtes, ainsi que `resultat` qui accumule les distances parcourues. On peut alors lire les requêtes une à une et appliquer la définition du coût. Comme proposé dans l'énoncé, `n` est une variable globale.

```
coutDe := proc (r,d)
local position_tete1, position_tete2, resultat, i;
  # position initiale des têtes :
  position_tete1 := 0;
  position_tete2 := 0;
  resultat := 0;
  for i from 1 to n do
    if d[i] = 1
    then
      resultat := resultat + abs(r[i] - position_tete1);
      position_tete1 := r[i];
    else
      resultat := resultat + abs(r[i] - position_tete2);
      position_tete2 := r[i];
    fi;
  od;
  resultat;
end;
```

2 Justifions tout d'abord l'existence d'une séquence de déplacements optimale. Pour cela, il suffit de constater que le nombre de séquences de déplacements étant fini, l'ensemble des coûts possibles est dès lors également fini : il admet un minimum.

Soit d une séquence de déplacements optimale pour un bloc de requêtes donné. Si d commence par 1, il n'y a rien à faire.

Dans le cas contraire, d commence par 2. Si l'on échange alors les deux têtes (c'est-à-dire si l'on remplace tous les 1 par des 2 et réciproquement dans la séquence de déplacements), la distance totale est inchangée. On obtient ainsi une nouvelle séquence de déplacements, qui commence par 1.

3 Pour chaque requête, on a le choix entre les deux têtes pour la satisfaire (les deux têtes peuvent se croiser et être au même endroit). Il y a donc 2^n possibilités.

B. Coût optimal pour deux requêtes

4 Le premier déplacement est fixé (c'est 1) : il ne reste qu'à choisir le second. Pour le bloc $\langle 10, 3\rangle$, choisir 1 donne un coût supplémentaire de $|3 - 10| = 7$ tandis que choisir 2 n'ajoute que $|3 - 0| = 3$. Une séquence de déplacements optimale est alors

$$\boxed{\langle 1, 2\rangle}$$

De la même façon, pour le bloc $\langle 3, 10\rangle$, la séquence $\langle 1, 1\rangle$ coûte 10 tandis que la séquence $\langle 1, 2\rangle$ coûte 13. Une séquence de déplacements optimale est donc

$$\boxed{\langle 1, 1\rangle}$$

5 On compare simplement les deux possibilités. La première est $\langle 1,1\rangle$ et coûte $r_1 + |r_2 - r_1|$. La seconde est $\langle 1,2\rangle$ et coûte $r_1 + r_2$. On renvoie donc la première si $|r_2 - r_1|$ est plus petit que r_2, la seconde sinon.

> On choisit arbitrairement la seconde dans le cas $|r_2 - r_1| = r_2$.

```
coutOpt2 := proc (r1, r2)
  if abs (r2 - r1) < r2
  then array([1,1]);
  else array([1,2]);
  fi;
end;
```

Partie II

A. Coût optimal pour trois requêtes

6 La règle de décision de la question 5 consiste à satisfaire chaque requête par la tête la plus proche.

> Une telle optimisation locale sans tenir compte de la suite est appelée algorithme *glouton*. Une théorie permet de caractériser les problèmes où un tel algorithme donne la solution optimale.

Par convention, la tête 1 satisfait la requête 20, la tête la plus proche de la requête $r_2 = 9$ est alors la 2, qui satisfera aussi la requête $r_3 = 1$.

La séquence cherchée est donc $\langle 1,2,2\rangle$ et son coût est $20+9+8=37$.

7 Il y a deux possibilités pour chacune des requêtes r_2 et r_3. Calculons le coût des quatre possibilités.

séquence	$\langle 1,1,1\rangle$	$\langle 1,1,2\rangle$	$\langle 1,2,1\rangle$	$\langle 1,2,2\rangle$
calcul du coût	$20+11+8$	$20+11+1$	$20+9+19$	$20+9+8$
coût	39	32	48	37

L'approche de la question 6 donne la séquence $\langle 1,2,2\rangle$ alors que la séquence $\langle 1,1,2\rangle$ a un coût strictement inférieur. Elle n'est donc pas optimale.

8 Comme suggéré par la question 7, on énumère toutes les possibilités sans chercher plus de finesse.

> Cette stratégie marche à tous les coups mais est souvent d'une lenteur catastrophique. Elle est donc à réserver aux cas où la taille de l'entrée est petite. Pour le cas général, il faudra trouver mieux, mais l'énoncé détaille ici les étapes.