Corrigé X - MP/PC - 2006.

Épreuve d'informatique de l'X - 2006 - MP/PC

Disque dur à deux têtes

Question 1 : On rappelle que n est supposé être une constante du programme.

```
1  coutDe := proc(r,d)
2  local i, cout, pos1, pos2;
3  pos1 := 0; pos2 := 0;
4  cout := 0;
5  for i from 1 to n do
6    if d[i] = 1
7    then cout := cout + abs(r[i]-pos1); pos1 := r[i];
8    else cout := cout + abs(r[i]-pos2); pos2 := r[i];
9    fi;
10 od;
11 return(cout);
12 end;
```

Question 2 : On considère l'ensemble des coûts des séquences de déplacements qui satisfont le bloc de requêtes r. C'est un partie non vide $\mathbb{N}$. Elle admet donc un plus petit élément qui correspond au coût optimal. On considère une séquence de déplacements d qui satisfait le bloc de requêtes r et de coût minimal. Si cette séquence commence par 1, on a terminé, sinon on remplace les 1 par des 2 et les 2 par des 1, la nouvelle requête aura le même coût que la précédente et sera donc de coût optimal et commencera par 1.
conclusion : Pour toute requête on peut trouver une séquence de déplacements de coût optimal commençant par un 1.

Question 3 : Pour chaque déplacement, on peut choisir de déplacer la tête 1 ou la tête 2. Donc si n représente le nombre de déplacements à effectuer, on aura 2^n séquences de déplacements possibles. Ce nombre se réduit à 2^{n-1} si on impose de déplacer d'abord la première tête.

Question 4 : Pour trouver la séquence de déplacements de coût minimal, il suffit de calculer les coûts des requêtes $\langle 1,1\rangle$ et $\langle 1,2\rangle$. On obtient le tableau de coût suivant :

	$\langle 10,3\rangle$	$\langle 3,10\rangle$
$\langle 1,1\rangle$	17	10
$\langle 1,2\rangle$	13	13

Conclusion : Pour la requête $\langle 10,3\rangle$, c'est la séquence $\langle 1,2\rangle$ qui est de coût minimal et pour la requête $\langle 3,10\rangle$ c'est la requête $\langle 1,1\rangle$ qui est de coût minimal.

Question 5 :

```
1  coutOpt2 := proc(r1,r2)
2  local cout_1_1, cout_1_2;
3  cout_1_1 := coutDe([r1,r2],[1,1]);
4  cout_1_2 := coutDe([r1,r2],[1,2]);
5  if cout_1_1 <= cout_1_2
6       then return([1,1])
7       else return([1,2])
8  fi;
9  end;
```

Question 6 : La requête $\langle 20,9\rangle$ nous conduit à choisir la séquence $\langle 1,2\rangle$ puis pour la requête $\langle 9,1\rangle$ conduit à la séquence $\langle 1,2\rangle$ ou $\langle 2,1\rangle$. Pour recoller nos deux séquences de déplacements, il faut choisir de déplacer en premier dans la deuxième séquence, la tête qui a été déplacée en dernier dans la première séquence, puisque ces déplacements correspondent à la même requête. On obtient donc au final la séquence $\langle 1,2,1\rangle$.

Question 7 : Toutes les requêtes possibles sont $\langle 1,1,1\rangle$, $\langle 1,1,2\rangle$, $\langle 1,2,1\rangle$ et $\langle 1,2,2\rangle$ qui conduisent aux coûts respectifs : 39, 32, 48 et 37. On voit que la stratégie précédente à conduit à la séquence $\langle 1,2,1\rangle$ qui a un coût supérieur à la séquence $\langle 1,2,2\rangle$.

Question 8 :

```
1  coutOpt3 := proc(r1,r2,r3)
2  local cout, sequence, cout112, cout121, cout122;
3  cout := coutDe([r1,r2,r3],[1,1,1]);
4  sequence := [1,1,1];
5  cout112 := coutDe([r1,r2,r3],[1,1,2]);
6  if cout112 < cout then cout := cout112; sequence := [1,1,2] fi;
7  cout121 := coutDe([r1,r2,r3],[1,2,1]);
8  if cout121 < cout then cout := cout121; sequence := [1,2,1] fi;
9  cout122 := coutDe([r1,r2,r3],[1,2,2]);
10 if cout122 < cout then cout := cout122; sequence := [1,2,2] fi;
11 return(sequence);
12 end;
```

Question 9 : Quand on a le tableau cout_n, il suffit de trouver en n^{e} ligne la valeur minimale : coût optimal $= \min_{0 \leqslant i \leqslant n} \text{cout}_n[i][n]$.

Question 10 : La ligne «$\texttt{cout}_0[0][0] = 0$ `et` $\texttt{cout}_0[i][j] = \infty$ `pour tout` $i \neq 0$ `ou` $j \neq 0$» exprime le fait qu'à l'instant initial les deux têtes sont en position 0 pour un coût nul et toutes les autres configurations sont inaccessibles;

La ligne « $\texttt{cout}_k[i][k]$ `est le minimum de` $|r_k - r_j| + \texttt{cout}_{k-1}[i][j]$ `pour` $0 \leqslant j \leqslant n$;» exprime le fait que le coût optimal pour atteindre la configuration (i, k) il faut passer par un configuration (i, j) en $(k-1)$ coups ($cout_{k-1}[i][j]$ étant infini si ($i \neq k-1$ et $j \neq k-1$).

$$cout_{k-1}[i][j] = \min_{1 \leqslant j \leqslant n} \left[\underbrace{|r_k - r_j| + cout_{k-1}[i][j]}_{(*)} \right]$$

$(*)$ représente le coût minimal pour atteindre la configuration (i, k) en k requêtes en passant par la configuration (i, j) atteinte en $k-1$ requêtes.

la ligne «$\texttt{cout}_k[k][j] = \texttt{cout}_k[j][k]$» exprime le fait que la coût optimal pour notre suite de requêtes ne dépend pas du choix de la première tête à déplacer.

Question 11 : On commence par écrire les fonctions calculant la somme et le minimum de deux entiers de $\mathbb{N} \cup \{\infty\}$ quand ∞ est représenté par l'entier $Infini = -1$.

```
1  Infini := -1;
2
3  somme := proc(a,b)
4  if a = Infini or b = Infini
5          then return(Infini)
6          else return (a+b);
7  fi;
8  end;
9
10 mini := proc(a :: integer, b :: integer)
11    if a = Infini then return(b)
12  elif b = Infini then return(a)
13  elif a < b then return(a) else return(b);
14 fi;
15 end;
```

On écrit maintenant la fonction de mise à jour en suivant les formules données à la question 10 :

```
1  mettreAJour := proc(cout, r, k)
2  local temp, i, j, m, rk;
3  global n;
4  temp := array(0..n,0..n);
5  # initialisation du tableau temp
6  for i from 0 to n do
7    for j from 0 to n do
8      temp[i,j] := Infini;
9    od;
10 od;
11 rk := r[k];
12 for i from 0 to n do
13       m := Infini;
14       for j from 0 to n do
15          m := mini(somme(abs(r[k]-r[j]),cout[i,j]),m)
16       od;
17       temp[i,k] := m;
18       temp[k,i] := m;
19 od;
20
21 for i from 0 to n do
22   for j from 0 to n do
23     cout[i,j] := temp[i,j];
24   od;
25 od;
26 end;
```

Question 12:

```
1  coutOpt := proc(r :: array)
2  local cout, i, j, k, rbis, res;
3  global n;
4   # On recrée une liste de requêtes avec un 0 au début
5  rbis := array(0..n);
6  rbis[0] := 0;
7  for i from 1 to n do
8      rbis[i] := r[i]
9  od;
10  # On crée le tableau $cout_0$.
11 cout := array(0..n,0..n);
12 for i from 0 to n do
13   for j from 0 to n do
14       cout[i,j] := Infini;
15   od;
16 od;
17 cout[0,0] := 0;
18  # On fait les mises à jour nécessaires
19 for k from 1 to n do
20    mettreAJour(cout,rbis,k);
21 od;
22 res := cout[0,n];
23  # On cherche le coût optimal
24 for i from 1 to n-1 do
25    res := mini(cout[i,n],res);
26 od;
27 return(res);
28 end;
```

Pour évaluer le temps d'exécution, on étudie les différentes boucles. La première boucle (lignes 7-9) et la dernière (lignes 24-26) sont en $O(n)$. La deuxième boucle (lignes 12-16) est en $O(n^2)$. La troisième boucle exécute n fois la fonction `mettreAJour` qui a une complexité quadratique ($O(n^2)$), le temps de cette troisième boucle sera en $O(n^3)$.

conclusion : `couOpt` s'exécute de $O(n^3)$.

Question 13:

```
1  coutOpt := proc(r)
2  local i, j, cout, k, res, rbis;
3  global n, Infini;
4  cout := array(0..n);
5  for i from 0 to n do
6     cout[i] := Infini;
7  od;
8  cout[0] := 0;
9   # On recrée une liste de requêtes avec un 0 au début
10 rbis := array(0..n); rbis[0] := 0;
11 for i from 1 to n do
12     rbis[i] := r[i]
13 od;
14  # On calcule l'équivalent des cout_k.
15 for k from 1 to n do
16    # calcul de la configuration (k-1,k)
17    cout[k-1] := cout[0] + abs(rbis[k]-rbis[0]);
18    for j from 1 to k-2 do
19       cout[k-1] := mini(cout[j]+abs(rbis[k]-rbis[j]),cout[k-1]);
20    od;
21    # calcul des configurations (i,k) pour i < k-1
22    for i from 0 to k-2 do
23       cout[i] := cout[i]+abs(rbis[k] - rbis[k-1]);
24    od;
25 od;
26  # On cherche le coût optimal
27 res := Infini;
28 for i from 0 to n do
29    res := mini(cout[i],res);
30 od;
31 return(res);
32 end;
```

Pour évaluer le temps d'exécution, on étudie les différentes boucles. Les deux premières (lignes 5-7 et lignes 11-13) et la dernière (lignes 28-30) ont un coût linéaire en n (soit en $O(n)$).

La double boucle (lignes 15-25) à un coût proportionnel à $\sum_{k=0}^{n-1} k = \frac{(n-1)n}{2} = O(n^2)$. Au final, on peut dire que le temps d'exécution est en $O(n^2)$ (temps quadratique en n).

conclusion : ici `couOpt` s'exécute de $O(n^2)$.