X Info PC 2006

Question 1

Q1 : la programmation récursive serait évidente!

```
> n_test:=3:
> r_test:=array(1..n_test,[5,2,4]):
  d_test1:=array(1..n_test,[1,1,2]):
  d_test2:=array(1..n_test,[1,2,1]):
> coutDe:=proc(r,d)
     local k,n,pos1,pos2,cout,deplace;
        pos1:=0;pos2:=0;cout:=0;n:=op(2,op(2,eval(r)));
        deplace:=proc(dk) if dk=1 then abs(pos1-r[k]) else
  abs(pos2-r[k]) fi end;
        for k from 1 to n do
          cout:=cout+deplace(d[k]);
         if d[k]=1 then pos1:=r[k] else pos2:=r[k] fi;
                          od;
     cout end:
> coutDe(r_test,d_test1);
> coutDe(r_test,d_test2);
```

Question 2

Soit d de cout minimum. Si elle commence par 1 c'est bon, si d commence par 2, on obtient une suite de même coût en échangeant les rôles des deux têtes (elles partaient toutes deux du même endroit)

Question 3

Il faut, |r| fois choisir "1" ou "2" : $2^{|r|}$, si on a fixé d[1]=1, on en a $2^{|r-1|}$

Question 4

Pour (10,3) : après le premier déplacement les têtes sont en 0 et 10, pour aller en 3, il vaut mieux bouger celle en 0; une séquence optimale est <1,2>

Pour (3,10) : après le premier déplacement les têtes sont en 0 et 3, pour aller en 10, il vaut mieux bouger celle en 3; une séquence optimale est <1,1>

Question 5

Il faut, comparer r[2] à 0 et à r[1] et utiliser la tête la plus proche

```
> coutOpt2=proc(r1,r2) if r2>r1/2 then [1,1] else [1,2] fi end;
```

$$coutOpt2 = (\mathbf{proc}(r1, r2)\ \mathbf{if}\ 1 / 2{*}r1 < r2\ \mathbf{then}\ [1, 1]\ \mathbf{else}\ [1, 2]\ \mathbf{end\ if\ end\ proc})$$

Question 6

L'idée est d'utiliser la tête la plus proche

- Avec (20, 9, 1) cela fait déplacer 1 (qui va en 20) puis 2 qui va en 9 puis 2 qui revient en 1

Le coût total aura été 20 + 9 + 8 = 37

Question 7

On nomme 1 la première tête déplacée. Les stratégies possibles sont (1,1,1), (1,1,2), (1,2,1) et (1,2,2)

```
> n_Q7:=3:r_Q7:=array(1..n_Q7,[20,9,1]):
  d_Q7_1:=array(1..n_Q7,[1,1,1]):d_Q7_2:=array(1..n_Q7,[1,1,2])
  :
  d_Q7_3:=array(1..n_Q7,[1,2,1]):d_Q7_4:=array(1..n_Q7,[1,2,2])
  :
> coutDe(r_Q7,d_Q7_1);
```

39

```
> coutDe(r_Q7,d_Q7_2);
```

32

```
> coutDe(r_Q7,d_Q7_3);
```

48

```
> coutDe(r_Q7,d_Q7_4);
```

37

Question 8

```
> d1:=array(1..3,[1,1,1]):d2:=array(1..3,[1,1,2]):
  d3:=array(1..3,[1,2,1]):d4:=array(1..3,[1,2,2]):
> coutOpt3:=proc(r1,r2,r3) local r,cout_min,seq_min;
          r:= array(1..3,[r1,r2,r3]);
          cout_min:= coutDe(r,d1);seq_min:=d1;
          if coutDe(r,d2)<cout_min then cout_min:=coutDe(r,d2);
  seq_min:=d2 fi;
          if coutDe(r,d3)<cout_min then cout_min:=coutDe(r,d3);
  seq_min:=d3 fi;
          if coutDe(r,d4)<cout_min then cout_min:=coutDe(r,d4);
  seq_min:=d4 fi;
          seq_min end:
> coutOpt3(20,9,1);
```

d2

Question 9

Après avoir satisfait la requète n on aura i= n ou j= n. On se trouvera donc dans la ligne ou colonne k. En inspectant les valeurs figurant dans

cette ligne ou colonne on trouvera la plus petite

Ce n'est qu'après avoir lu les questions qui viennent que j'ai compris que (i,j) ne désigne PAS (i,j) mais (r(i),r(j))

Question 10

- La première condition est une "évidente initialisation" (on part de (0,0))
- La quatrième dit que pour satisfaire la k-ième il faut i ou j égal à k
- La troisième exprime l'évidente symétrie
- La seconde semble ne déplacer que la seconde tête ... mais comme les matrices coût sont symétriques, c'est bon

Question 11

```
> n:=5;
  req:=array(0..n,[0,3,7,1,2,6]):
  cout:=array(0..n,0..n):
  for l from 0 to n do for c from 0 to n do cout[l,c]:= -1 od
  od;cout[0,0]:=0:
```

L'array ne va pas s'afficher "matriciellement" parce que les indices partent de 0

```
> print(cout);
> distance:=proc(x,y) if x=-1 or y=-1 then -1 else abs(x-y) fi
  end;
> mininfini:=proc(x,y) if x=-1 then y else if y=-1 then x else
  min(x,y) fi fi end;
> mettreAJour:=proc(cout,r,k)
      local l,c;
      cout[0,k]:=cout[0,k-1]+distance(r[k],r[k-1]);
          # si on vient de la colonne voisine :
      for l  from 1 to (k-2)
          do cout[l,k]:=cout[l,k-1]+distance(r[k],r[k-1]) od;
          # si on vient de la ligne (k-1) :
      cout[k-1,k]:=cout[k-1,0]+distance(r[k],r[0]); # <-
  initialisation
      for c from 1 to (k-2) do

  cout[k-1,k]:=mininfini(cout[k-1,c]+distance(r[k],r[c]),cout[k
  -1,k]);
          #  print(c," ",cout[k-1,k])<- reste de débogage

                              od;
      for c from 0 to (k-1) do cout[k,c]:=cout[c,k] od; # <-
  symétrisation
      cout end;
> mettreAJour(cout,req,5);
```

```
> print(cout);
```

Question 12

Je suppose que n a été donné ailleurs avant et que la matrice cout a été déclarée elle aussi avant :

```
> n:=5;
  req:=array(0..n,[0,3,7,1,2,6]):
  cout:=array(0..n,0..n):
  for l from 0 to n do for c from 0 to n do cout[l,c]:= -1 od
  od;cout[0,0]:=0:
> coutOpt:=proc(r)
     local k, mini;
     for k from 1 to n do mettreAJour(cout,r,k) od;
     mini:=cout[n,0]; for k from 1 to (n-1) do
  mini:=min(mini,cout[n,k]) od;
  mini end;
> coutOpt(req);
```

metAJour fait (k-1) calculs en remontant la colonne (un par ligne) puis (k-1) calculs pour traiter la ligne précédente :

en fait l'énoncé "fait semblant" de demander le traitement de tout le carré l,c entre 0 et (k-1) ce qui ferait $(k-1)^2$ ... on peut même interpréter l'énoncé comme demandant n^2 calculs à chaque fois

La complexité de cout est la somme des metAJour de 1 à n : selon l'interprétation : $n^2/2$ ou n^3/3 ou n^3 + la recherche finale du minimum qui ne coûte que n

Question 13

C'est justement ce que je viens de faire : en fait, comme je n'y comprenais rien, j'ai dessiné les matrices "cout" et j'y ai vu que :
1) les indices > k n'interviennent jamais (tout est Infini)
2) pour l'indice k : on ne peut venir que de la colonne voisine dans les lignes 1.. k-1, ou bien de la ligne (k-1)

Ci-dessous coutL désigne "coutLigne" un array de dimension 1 qui va tout stocker, les fonctions "distance" ou ou "miniinfini" ne servent plus à rien maintenant que l'on a compris, mais j e les ai laissées

```
> n:=5:req:=array(0..n,[0,3,7,1,2,6]):
  coutL:=array(0..n):for c from 0 to n do coutL[c]:= -1
  od:coutL[0]:=0:
> mettreAJourL:=proc(cout,r,k)
      local l,c;
      if k=1
      then cout[0]:=distance(r[0],r[1])
      elif k=2
```

```
            then cout[1]:=cout[0]+distance(r[2],r[0]);
            cout[0]:=cout[0]+distance(r[1],r[2])
            else                # si on vient "de la ligne (k-1)"
    :
                cout[k-1]:=cout[0]+distance(r[k],r[0]); # <-
    initialisation
                for c from 1 to (k-2) do
    cout[k-1]:=mininfini(cout[c]+distance(r[k],r[c]),cout[k-1]);
                  #  print(c," ",cout[k-1,k])    #<- reste de
    débogage
                              od;
           # si on vient de la colonne voisine :
           for l  from 0 to (k-2) do
    cout[l]:=cout[l]+distance(r[k],r[k-1]) od;
        fi;    cout end;
> mettreAJourL(coutL,req,5);
> print(coutL);
> coutOptL:=proc(r)
     local k, mini;
     for k from 1 to n do mettreAJourL(coutL,r,k) od;
     mini:=coutL[0]; for k from 1 to (n-1) do
    mini:=min(mini,coutL[k]) od;
    mini end;
> coutOptL(req);
```

Le temps d'éxécution est $n^2/2$

Remarques personelles

Après coup, la présentation avec les n+1 matrices creuses semble ridicule

C'est en fait avec ce genre de maladresses que l'on (du moins moi) met au point les idées, et au lieu de "faire le malin" en montrant l'objet fini sans les intermédiaires, l'auteur de ce problème a laissé voir sa première idée!

Bravo et merci

A satisfaire : 0 3 7 1 2 6

6	∞	∞	∞	∞	∞	∞	∞
5	∞	∞	∞	∞	∞	∞	∞
4	∞	∞	∞	∞	∞	∞	∞
3	∞	∞	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	∞	∞	∞
1	∞	∞	∞	∞	∞	∞	∞
0	0	∞	∞	∞	∞	∞	∞
Cout_0	0	1	2	3	4	5	6

6	∞	∞	∞	∞	∞	∞	∞
5	∞	∞	∞	∞	∞	∞	∞
4	∞	∞	∞	∞	∞	∞	∞
3	∞	∞	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	∞	∞	∞
1	3	∞	∞	∞	∞	∞	∞
0	∞	3	∞	∞	∞	∞	∞
Cout_1	0	1	2	3	4	5	6

6	∞	∞	∞	∞	∞	∞	∞
5	∞	∞	∞	∞	∞	∞	∞
4	∞	∞	∞	∞	∞	∞	∞
3	∞	∞	∞	∞	∞	∞	∞
2	7	10	∞	∞	∞	∞	∞
1	∞	∞	10	∞	∞	∞	∞
0	∞	∞	7	∞	∞	∞	∞
Cout_2	0	1	2	3	4	5	6

6	∞	∞	∞	∞	∞	∞	∞
5	∞	∞	∞	∞	∞	∞	∞
4	∞	∞	∞	∞	∞	∞	∞
3	∞	∞	∞	∞	∞	∞	∞
2	7	10		8	∞	∞	∞
1			10	16	∞	∞	∞
0			7	13	∞	∞	∞
Cout_3	0	1	2	3	4	5	6

6	∞	∞	∞	∞	∞	∞	∞
5	∞	∞	∞	∞	∞	∞	∞
4	∞	∞	∞	∞	∞	∞	∞
3	13	16	8		13	∞	∞
2				8	9	∞	∞
1				16	17	∞	∞
0				13	14	∞	∞
Cout_4	0	1	2	3	4	5	6

6	∞	∞	∞	∞	∞	∞	∞
5	∞	∞	∞	∞	∞	∞	∞
4	14	17	9	13		10	∞
3					13	14	∞
2					9	13	∞
1					17	21	∞
0					14	18	∞
Cout_5	0	1	2	3	4	5	6