Informatique tronc commun
X 2003 : l'enclos du robot

Corrigé

26 novembre 2005

Les questions 1.6 et suivantes reproduisent avec son aimable autorisation le corrigé de Mickaël Péchaud, disponible sur http://www.eleves.ens.fr/home/pechaud/

1 Frontière Sud-Ouest

1.1 Question:

```
> sudouest := proc (x1,y1,x2,y2)
> (x1 <= x2) and (y1 <= y2) ;
> end ;
        sudouest := proc(x1, y1, x2, y2) x1 <= x2 and y1 <= y2 end proc

> sudouest (1,2,3,4) ; ;
                                    true
> nordouest := proc (x1,y1,x2,y2)
> (x1 <= x2) and (y1 >=y2)
> end ;
       nordouest := proc(x1, y1, x2, y2) x1 <= x2 and y2 <= y1 end proc

> sudest := proc (x1,y1,x2,y2)
> (x1 >= x2) and (y1 <= y2)
> end ;
         sudest := proc(x1, y1, x2, y2) x2 <= x1 and y1 <= y2 end proc

> nordest := proc (x1,y1,x2,y2)
> (x1 >= x2) and (y1 >= y2)
> end ;
        nordest := proc(x1, y1, x2, y2) x2 <= x1 and y2 <= y1 end proc
```

1.2 Question:

```
echange := proc (a,b,i,j)
```

```
 local tmp_a,tmp_b ;
 tmp_a := a[i] ; a[i] := a[j] ; a[j] := tmp_a ;
 tmp_b := b[i] ; b[i] := b[j] ; b[j] := tmp_b ;
 end ;

> t := [1,2,3] ;
                                t := [1, 2, 3]

> u := [4, 5, 6] ;
                                u := [4, 5, 6]

> echange ('t','u',1,3) ;
                                      4

> t ;
                                   [3, 2, 1]

> u ;
                                   [6, 5, 4]
```

1.3 Question: Les points P_0 et P_1 définissent la partie sud-ouest de l'enveloppe.

1.4 Question:

```
frontiereSO :=proc(a, b, size)
 local l, bool, i, j, c ;
 global aSO, bSO, nSO ;
 l :=[] ; nSO :=0 ;
 for i from 1 to size do
  bool :=true ;
   for j from 1 to size do
    if sudouest([a[j], b[j]],[a[i], b[i]]) then
       if not(i=j) then bool :=false ; fi ; fi ;
   od ;
  if bool then l :=[op(l), i] ; nSO :=nSO+1 ; fi ;
 od ;
#l contient maintenant la liste
#des points de l'enveloppe SO

c :=1 ;
for i in l do
 echange(a, b, i, c) ;
 c :=c+1 ;
od ;

#Les éléments SO sont maintenant rangés au début.
#Il n'y a plus qu'à les ordonner
for i from 1 to nSO do
 for j from i to nSO do
  if (a[i]>a[j]) then echange(a, b, i, j) fi ;
 od ;
od ;
```

```
    #on recopie maintenant tout ça
    #dans aSO et bSO (ça n'est pas de-
mandé par l'énoncé)
     aSO :=array(1..nSO) ;
     bSO :=array(1..nSO) ;
     for i from 1 to nSO do
      aSO[i] :=a[i] ;
      bSO[i] :=b[i] ;
     od ;
    RETURN() ;
    end :

    #nous allons tester ceci sur un pe-
tit exemple. Pour celà, on
    #va créer une fonction d'affichage à par-
tir de 2 tableaux.

    affiche :=proc(a, b, size)
     local l, i ;
     l :=[] ;
     for i from 1 to size do
      l :=[op(l), [a[i], b[i]]] ;
     od ;
     plot(l, 0..12, 0..12, style=point) ;
    end :

    a :=array([1, 4, 7, 9, 8, 12, 2, 4, 7, 2]) :
    b :=array([6, 4, 6, 10, 2, 4, 7, 8, 3, 10]) :

    affiche(a, b, 10) ;
```

Notez qu'on peut envisager une version plus compacte de cet algorithme, qui toutefois n'utilise pas les questions précédentes :
Considérons le premier point du tableau : il est dans la frontière sud-ouest, excepté s'il y a un point de même abcisse plus bas que lui dans le tableau. On peut donc, quitte à réorganiser nos quelques premiers points (d'abcisse identique), partir d'un point de la frontière Sud-Ouest.
Puis, à chaque point P considéré lors de notre parcours, notons $prec(P)$ le dernier point de la frontière sud-ouest rencontré.

- Soit $y_P \geq y_{prec(P)}$. Dans ce cas, P n'est pas sur la frontière sud-ouest : en effet, comme le tableau est parcouru dans l'ordre des abcisses croissantes, $prec(P)$ est au sud-ouest de P.
- Soit $y_P < y_{prec(P)}$ Dans ce cas, P est sur la frontière sud-ouest.

Cet algorithme est la base de la solution en temps linéaire du programme : nous rangeons ici les points en deux groupes (celui de la frontière, et les autres), *en place*, c'est à dire en ne stockant que les données incluses dans le tableau initial. En stockant les points dans deux listes différentes, qui seraient ensuite concaténées en temps linéaire, nous pourrions obtenir un algorithme de classement linéaire.

Preuve

On montre d'abord que s'il existe un point P' au sud-ouest d'un point P tel que $x_P \geq x_{prec(P)}$ alors il existe un point P'' au sud-ouest de P'.
Supposons qu'un point P' distinct de P est au sud-ouest de P. Alors $x_{P'} \leq x_P$, donc P' a été rencontré avant x_P, et $y_{P'} \leq y_P$ donc P' est plus bas que P. En particulier, $y_{P'} < y_{prec(P)}$.
Nous savons donc que $x_{P'} > x_{prec(P)}$, sinon P' serait au sud-ouest de $prec(P)$, ce qui est absurde car $prec(P)$ est sur la frontière sud-ouest.
Or P' n'est pas sur la frontière sud-ouest, puisqu'il a déja été rencontré et qu'il est distinct de $pre(P)$. Donc il existe un point P'' au sud-ouest de P'.
On peut donc montrer, en supposant par l'absurde que P n'est pas sur la frontière sud-ouest (*i.e.* qu'il existe un point qui est à son sud-ouest), qu'il existe une infinité de points entre P et $prec(P)$. Ceci est absurde.

Il ne nous reste plus qu'à appliquer l'algorithme précédent à notre tableau : nous le parcourons dans le sens des abcisses croissantes, et nous conservons en permanence deux variables : `c` valant l'indice du point courant considéré et `p` l'indice du dernier point de la frontière rencontré.

Si le point courant est en-dessus, alors il ne fait pas partie de la frontière sud-ouest. Dans le cas contraire, il est membre de la frontière, et nous le faisons alors percoler juste après le dernier élément de la frontière trouvé, avant d'augmenter l'indice `p`. A la fin du programme, on renvoie le nombre de points dans la frontière `k`.

```
frontiereSO = proc(a, b, n)
local c, p, i, j, k;
    c := 1;
    p := 1;
    k := 1;
    while a[c + 1] = a[1] and b[c + 1] < b[c] do for j from n - c - 1 to
        n - 2 do echange(a, b, n - j - 1, n - j);
        od;
        c := c + 1;
    od;
    for i from c to n do
        if b[c] < b[p] then for j from n - c to n - p + 1 do
                echange(a, b, n - j - 1, n - j)
           od;
          p := i;
          k := (k+1);
        fi;
    od;
end;
```

On en tire aisément une version linéaire de cet algorithme :

```
frontiereSO :=proc(a,b,n)
local ;
  c := 1 ;
  p := 1 ;
  k := 1 ;
  liste_frontiere := [] ;
  liste_autres := [] ;
  while a[c+1] = a[1] and b[c+1] < b[c] do
    liste_autres := [op(liste_autres),a[c]] ;
    c :=c+1 ;
  od ;
  for i from c to n do
    if b[c]<b[p] then
      liste_frontiere :=[op(liste_frontiere),(a[i],b[i])] ;
      p := i ;
      k := k+1 ;
    else
      liste_autres := [op(liste_autres),(a[i],b[i])] ;
    fi ;
  od ;
  for i from 1 to k do
    (abc,ord) := liste_frontiere[i] ;
    a[i] :=abc ;
    b[i] :=ord ;
  od ;
  for i from (k+1) to n do
    (abc,ord) := liste_autres[i-k] ;
    a[i] := abc ;
    b[i] := ord ;
  od ;
  k ;
end ;
```

1.5 Question: Ce sont les points P_{11}, P_{15}, $P18$, $P19$

1.6 Question:
En adoptant l'algorithme suggéré par l'énoncé, il suffit simplement de changer les appels à `sudouest`. Dans le cas du second programme proposé, on pourrait envisager de renverser les sens de parcours du tableau, en travaillant sur les indices de boucle.

```
frontiereNO :=proc(a, b, size)
 local l, bool, i, j, c;
 global aNO, bNO, nNO;
 l :=[]; nNO :=0;
 for i from 1 to size do
  bool :=true;
   for j from 1 to size do
    if nordouest([a[j], b[j]],[a[i], b[i]]) then
       if not(i=j) then bool :=false; fi; fi;
   od;
  if bool then l :=[op(l), i]; nNO :=nNO+1; fi;
 od;
c :=1;
for i in l do
 echange(a, b, i, c);
 c :=c+1;
od;
for i from 1 to nNO do
 for j from i to nNO do
  if (a[i]>a[j]) then echange(a, b, i, j) fi;
 od;
od;
 aNO :=array(1..nNO);
 bNO :=array(1..nNO);
 for i from 1 to nNO do
  aNO[i] :=a[i];
  bNO[i] :=b[i];
 od;
RETURN();
end :
```

```
frontiereNE :=proc(a, b, size)
 local l, bool, i, j, c;
 global aNE, bNE, nNE;
 l :=[]; nNE :=0;
 for i from 1 to size do
  bool :=true;
   for j from 1 to size do
     if nordest([a[j], b[j]],[a[i], b[i]]) then
      if not(i=j) then bool :=false; fi;
fi;
   od;
  if bool then l :=[op(l), i]; nNE :=nNE+1; fi;
 od;
c :=1;
```

```
frontiereSE :=proc(a, b, size)
 local l, bool, i, j, c;
 global aSE, bSE, nSE;
 l :=[]; nSE :=0;
 for i from 1 to size do
  bool :=true;
   for j from 1 to size do
    if sudest([a[j], b[j]],[a[i], b[i]]) then
       if not(i=j) then bool :=false; fi; fi;
   od;
  if bool then l :=[op(l), i]; nSE :=nSE+1; fi;
 od;
c :=1;
for i in l do
 echange(a, b, i, c);
 c :=c+1;
od;
for i from 1 to nSE do
 for j from i to nSE do
  if (a[i]>a[j]) then echange(a, b, i, j) fi;
 od;
od;
 aSE :=array(1..nSE);
 bSE :=array(1..nSE);
 for i from 1 to nSE do
  aSE[i] :=a[i];
  bSE[i] :=b[i];
 od;
RETURN();
end :
```

```
for i in l do
 echange(a, b, i, c);
 c :=c+1;
od;
for i from 1 to nNE do
 for j from i to nNE do
  if (a[i]>a[j]) then echange(a, b, i, j) fi;
 od;
od;
 aNE :=array(1..nNE);
 bNE :=array(1..nNE);
 for i from 1 to nNE do
  aNE[i] :=a[i];
  bNE[i] :=b[i];
 od;
RETURN();
end :
```

1.7 Question:

Il suffit maintenant de relier tous ces points de façon appropriée.

Par exemple, pour la partie SO, on parcourt les ponts dans le sens des abscisses croissantes, et on relie deux points consécutifs p et q par une "marche d'escalier", dans le sens droite puis bas.

Cette marche est à l'intérieur de l'enveloppe, car c'est le chemin suivi pour aller de p à q.

Elle est bien une frontière de l'enveloppe, car tout point situé dans le quart de plan inférieur-gauche qu'elle définit est un point SO pour l'ensemble des points union lui-meme, donc il ne sera jamais atteint pour un trajet entre 2 autres points.

Pour joindre SO et SE, on peut démontrer de meme qu'il suffit de joindre le dernier point de SO et le premier de SE.

```
EVLP :=[] :
moveTo :=proc(x, y) global EVLP : EVLP :=[[x, y]] ; end :
lineTo :=proc(x, y) global EVLP : EVLP :=[op(EVLP), [x, y]] ; end :

trace_enveloppe :=proc()
 local i :
 global EVLP :

 #tracé SO
 moveTo(aSO[1], bSO[1]) ;
 for i from 2 to nSO do
  lineTo(aSO[i], bSO[i-1]) ;
  lineTo(aSO[i], bSO[i]) ;
 od ;

 #tracé SE
 lineTo(aSE[1], bSE[1]) ;
 for i from 2 to nSE do
  lineTo(aSE[i-1], bSE[i]) ;
  lineTo(aSE[i], bSE[i]) ;
 od ;

 #tracé NE
 lineTo(aNE[nNE], bNE[nNE]) ;
 for i from 1 to nNE-1 do
  lineTo(aNE[nNE-i], bNE[nNE-i+1]) ;
  lineTo(aNE[nNE-i], bNE[nNE-i]) ;
 od ;

 #tracé NO
 lineTo(aNO[nNO], bNO[nNO]) ;
 for i from 1 to nNO-1 do
  lineTo(aNO[nNO-i+1], bNO[nNO-i]) ;
  lineTo(aNO[nNO-i], bNO[nNO-i]) ;
 od ;

RETURN() ;
end :
```