Corrigé contrôle d'informatique.

Épreuve d'informatique de l'X - 2004 - PSI

Question 1 : Pour $s = 0$, on a $E = \langle 0, 1, 2, 3, ..., (n-1)\rangle$ et pour $s = 1$ avec $c_i \sim c_j$ on obtient deux solutions qui sont $E = \langle 0, 1, \ldots, i, \ldots, i, \ldots, (n-1)\rangle$ et $E = \langle 0, 1, \ldots, j, \ldots, j, \ldots, (n-1)\rangle$ où les i et les j sont en position i et j dans les tableaux précédents.

Question 2 :

```
representant := proc(E :: array, i :: nonnegint)
local ami;
 ami := E[i];
 while (E[ami] <> ami) do ami := E[ami] od;
 return(ami);
end;
```

Question 3 :

```
sontAmis := proc(E::array, i::nonnegint, j::nonnegint)
   return(evalb(representant(E, i) = representant(E, j)));
end proc;
```

Question 4 : On choisit comme représentant de la fusion des cercles d'amis de i et de j, le représentant du cercle d'amis de i. On remarque que si i et j appartiennent au même cercle d'amis, il n'y a pas de modification du tableau E. La fonction «`représentant`» s'exécutant en temps linéaire par rapport à n (au plus un parcours dans le tableau E), il en est de même pour la fonction «`ajout`» (on a au plus deux parcours du tableau E).

En conclusion, on a un temps d'exécution linéaire.

```
ajout := proc(E::array, i::nonnegint, j::nonnegint)
   E[representant(E, j)] := representant(E, i);
   return() ;
end proc;
```

Question 5 : Si à l'arrivée le tableau E vérifie $E[i] = representant(E, i)$ pour tout i de $[\![0, n-1]\!]$, la fonction « `représentant`» s'exécutera en temps constant. Ce sera le cas si on ajoute les $c_i \sim c_{i+1}$ par ordre croissant pour i variant de 0 à $n-2$ pour lequel on obtient alors $E = \langle 0, 0, 0, ..., 0\rangle$. Par contre l'ajout des $c_i \sim c_{i+1}$ par ordre décroissant, pour i variant de $n-2$ à 0, nous donne $E = \langle 0, 0, 1, 2, \ldots, n-2\rangle$ qui entraîne le fait que la fonction «`representant`» s'exécute en temps linéaire.
Une solution pour garantir un meilleur temps d'exécution est de prendre en compte les relations $c_i \sim c_j$ à condition de les avoir rangé par ordre «croissant» avant. Dans la relation $c_i \sim c_j$, on impose $i < j$ et on dira que $c_{i_k} \sim c_{j_k}$ est inférieur à $c_{i_k} \sim c_{j_k}$ si et seulement si $(i_k = i_{k+1}$ et $j_k <= j_{k+1})$ ou $i_k < i_{k+1}$.

Question 6 :

```
couleurNO := proc(T :: array, i :: nonnegint, j :: nonnegint)
if i=0 or j=0 then return(0)
              else return(T[i-1,j-1])
fi;
end;

couleurN := proc(T :: array, i :: nonnegint, j :: nonnegint)
if i=0 then return(0)
       else return(T[i-1,j])
fi;
end;

couleurO := proc(T :: array, i :: nonnegint, j :: nonnegint)
if j=0 then return(0)
       else return(T[i,j-1])
fi;
end;
```

Question 7 :

```
nc := 0; C := 20;
E := array(0 .. C-1);
for i from 0 to C-1 do E[i] := i od: # on initialise E

marquage := proc(T::array)
local i, j, K, n, o, no;
global nc, E, C;
   for i from 0 to R - 1 do
    for j from 0 to R - 1 do
       K := T[i,j];
       if K <> 0 then
           no := couleurNO(T, i, j);
           if no <> 0 then
             T[i,j] := no
           else
             o := couleurO(T, i, j);
             n := couleurN(T, i, j);
             if n = 0 and o <> 0 then
                T[i,j] := o
             elif n <> 0 and (o = 0 or o = n) then
                 T[i,j] := n
             elif n <> 0 and o <> 0 then
                 ajout(E, n, o); T[i,j] := n;
             else
                 nc := nc + 1; T[i,j] := nc;
             end if;
           end if;
        end if;
    end do;
  end do;
  return() ;
end proc;
```

En considérant que le nombre de couleurs utilisées est du même ordre de grandeur que le nombre d'objets qui a priori est faible devant le nombre de pixels, les fonctions «`couleurO`», «`couleurN`», «`couleurNO`», la fonction `marquage` va s'exécuter en temps quadratique.

Question 8 :

```
diffusion := proc(T::array)
local i, j, R;
global E;
   R := longueur(T);
   for j from 0 to R - 1 do
     for i from 0 to R - 1 do
      if T[i, j] <> 0
         then T[i, j] := E[representant(E, T[i, j])]
      end if;
     end do;
  end do;
  return() ;
end proc;
```

La fonction «`representant`» s'exécutant, dans le pire des cas en temps linéaire par rapport à C, la double boucle de la fonction «`diffusion`» va s'exécuter en $O(C.R^2)$

```
colorier := proc(T::array)
  marquage(T);
  diffusion(T);
end;
```