>

X Info 2011 PSI

Préliminaire 1 : graphisme

J'ai essayé de trouver dans les outils Maple un "dessin par carreaux" ... je n'ai pas trouvé. Jadis il y avait des RasterPlot, des matrixplot ??

Alors, pour pouvoir tester les fonctions de ce problème, on va le fabriquer

```
> with(plots): with(plottools): interface(rtablesize=21): with(ArrayTools):
      with(ListTools):
> carreau:=proc(x,y,h,c) rectangle([x,y+h],[x+h,y],color=c) end:
> display(carreau(1,2,1,RGB(0,0,0)),carreau(3,2,1,RGB(0.9,0.9,0.9)),carreau(5,2,
      1,RGB(1,1,1))):
> essai := map(k->carreau(k,2,1,RGB(k/15,k/15,k/15)),[$0..15]):
> display(essai,scaling=constrained,axes=none,style=patchnogrid);
```

On fabrique quelques "grilles" standard

```
> grille:=proc(n) [$0..(n-1)] end:
> prefixe:=proc(l,x) map(k->[x,k],l) end:
> prefixe( grille(3),u):
> bigrille:=proc(n) map(k->op(prefixe(grille(n),k)),grille(n)) end:
> bigrille(5):
> bigrille2:=proc(h,l) map(k->op(prefixe(grille(l),k)),grille(h)) end:
> bigrille2(3,4);
```

CoulDec et CoulCroit sont les deux gammes de gris de luminosité Decroissante ou Croissante. Les versions "Num" ne contiennent que la teinte de gris entre 0 et P (selon ce que dit l'énoncé X PSI)

```
> CoulDec:=proc(P,x,y) RGB(1-x/P,1-x/P,1-x/P) end:
> CoulCroit:=proc(P,x,y) RGB(y/P,y/P,y/P) end:
> CoulDecNum:=proc(P,x,y) P-x end: CoulCroitNum:=proc(P,x,y) y end:
```

carF définit les carreaux de la lettre F utilisée comme modèle

```
> carF:=proc(M) local x,y; x:=M[1]; y:=M[2]; if ((x=0) or ((y=6) and (x<=6))
      or (y=10)) then carreau(x,y,1,CoulCroit(10,x,y)) else carreau(x,y,1,CoulDec(10,
      x,y)) fi end:
```

carFnum est la version purement numérique de cette même chose, carPnum : idem avec une frontière Parabolique, carLnum : idem avec la ligne de la page 1

Tout cela est revu après la dernière question pour avoir de "belles images"

```
> carFnum:=proc(M) local x,y; x:=M[1]; y:=M[2]; if ((x=0) or ((y=6) and (x<=6))
      or (y=10)) then CoulCroitNum(10,x,y) else CoulDecNum(10,x,y) fi end:
```

```
> carPnum := proc(M) local x, y; x := M[1]; y := M[2]; if ((x-5)·(x-5) < y)
      then CoulCroitNum(10, x, y) else CoulDecNum(10, x, y) fi end:
> carPnum50 := proc(M) local x, y; x := M[1]; y := M[2]; if (2·(x-25)·(x-25) ≤ 25
      · y) then CoulCroitNum(50, x, y) else CoulDecNum(50, x, y) fi end:
> carLnum := proc(M) local x, y; x := M[1]; y := M[2]; x end:
> carOfnumF := proc(M) carreau(M[1], M[2], 1, RGB(carFnum(M)/10, carFnum(M)/10,
      carFnum(M)/10)) end:
> carOfnumP := proc(M) carreau(M[1], M[2], 1, RGB(carPnum(M)/10, carPnum(M)/10,
      carPnum(M)/10)) end:
> carOfnumP50 := proc(M) carreau(M[1], M[2], 1, RGB(carPnum50(M)/50,
      carPnum50(M)/50, carPnum50(M)/50)) end:
> carOfnumL := proc(M) carreau(M[1], M[2], 1, RGB(carLnum(M)/15, carLnum(M)/15,
      carLnum(M)/15)) end:
> map(carOfnumL, bigrille2(16, 1)):
> display(%, style = patchnogrid, scaling = constrained);
> map(carOfnumF, bigrille(11)):
> display(%, style = patchnogrid, scaling = constrained);
> map(carOfnumP, bigrille(11)):
> display(%, style = patchnogrid, scaling = constrained);
> map(carOfnumP50, bigrille(50)):
> display(%, style = patchnogrid, scaling = constrained);
```

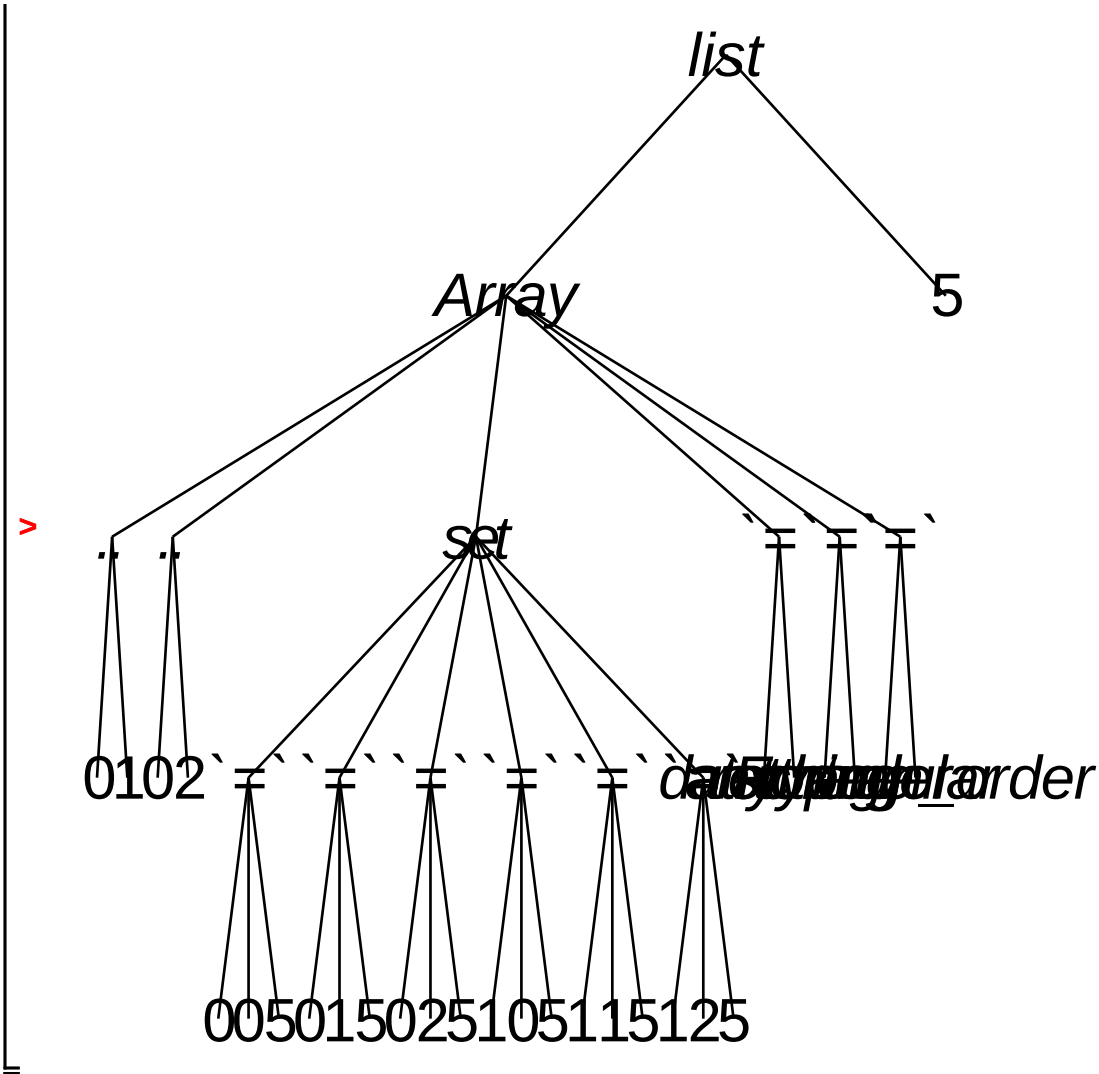
Préliminaire 2 : "allouer"

On fait ci-dessous ce que dit la page 1 de l'énoncé : on va définir une fonction allouer telle que l'on puisse définir une image i par la commande i := allouer(H,L,P) et telle que l'on dispose ensuite de fonctions H, L, P et M telles que H(i), L(i), P(i) et M(i) soient la Hauteur, Largeur, Profondeur et Matrice de i, l'accès au terme de ligne l et de colonne c de M se faisant par M(i)[l,c]

i va être du type liste : [matrice, profondeur], la matrice étant un Array et la profondeur un entier, la hauteur et la largeur sont "cachées" dans la matrice

```
> allouer := proc(H, L, P) [Array(0..H-1, 0..L-1, P), P] end:
> test := allouer(2, 3, 5);
```

L'image qui suit a été obtenue avec la fonction arbre_12 du TD2 appliquée à 'test' (défini ci-dessus)




```
> H := proc(A) op(2, [op(2, A[1])][1]) + 1 end;
> H(test);
> L := proc(A) op(2, [op(2, A[1])][2]) + 1 end;
> L(test);
> P := proc(A) A[2] end;
> P(test);
> M := proc(A) A[1] end;
> M(test);
> M(test)[1, 2];
```

remplisA prend une image déjà déclarée et remplit sa matrice par usage de la fonction de remplissage f, on n'obtient pas un objet graphique, tout est encore numérique

DeNumAGraph prend une image i et fabrique l'objet du graphisme Maple prêt à être affiché

AfficheI combine ce qui précède et l'affichage "à carreaux"

AfficheISP combine ce qui précède et l'affichage "sans carreaux" : Style= Patchnogrid

```
> remplisA := proc(A, f) local l, c;
      for l from 0 to H(A) - 1 do
      for c from 0 to L(A) - 1 do M(A)[l, c] := f(l, c)
   od od; A;end:
> testF := allouer(11, 11, 10) :
> testF := remplisA(testF, (i, j) → carFnum([i, j])) :
> H(testF);
> testP := allouer(11, 11, 10) : testP := remplisA(testP, (i, j) → carPnum([i, j])) :
> testP50 := allouer(50, 50, 50) : testP := remplisA(testP50, (i, j) → carPnum50([i, j])) :
> testL := allouer(1, 16, 16) : testL := remplisA(testL, (i, j) → carPnum([i, j])) :
> carFnum([2, 5]);
> bigrille2(10, 10) :
> DeNumAGraph := proc(i)
          local h, l, p, m, recarreau;
   h := H(i); l := L(i); p := P(i); m := M(i); recarreau := proc(z) local x, y; x := z[1]; y
      := z[2]; carreau(x, y, 1, RGB(m[x, y]/p, m[x, y]/p, m[x, y]/p)) end;

          map(z→recarreau(z), bigrille2(h, l))
   end:
> display(DeNumAGraph(testF), scaling = constrained);
> AfficheI := proc(i) display(DeNumAGraph(i), scaling = constrained) end;
> AfficheI(testP);
> AfficheI(testP50);
> AfficheISP := proc(i) display(DeNumAGraph(i), scaling = constrained, style = patchnogrid)
      end:
> AfficheISP(testP50);
```

I Opérations élémentaires

Question 1

```
> inverser := proc(i) local Max; Max := P(i) - 1; [map(v→Max - v, M(i)), P(i)] end:
> AfficheI(inverser(testF));
```

Question 2

```
> flipH := proc(i)
          local h, l, p, m, N, x, y;
   h := H(i); l := L(i); p := P(i); m := M(i);
   N := Array(0..h - 1, 0..l - 1, 0);
   for x from 0 to l - 1 do
      for y from 0 to h - 1 do
      N[x, y] := m[h - 1 - x, y]
      od
   od;
   [N, p]
   end:
```

```
> AfficheI( flipH(testF) );
```

Question 3

Ci-dessous j'aurais dû écrire une ligne pour vérifier l1=l2, être sûr de ne pas avoir confondu haut et bas, etc (zut)

```
> poserV :=proc(i1, i2)    local h1, l1, p1, m1, h2, l2, m2, p2, N, x, y;
    h1 := H(i1); l1 := L(i1); p1 := P(i1); m1 := M(i1);
    h2 := H(i2); l2 := L(i2); p2 := P(i2); m2 := M(i2);
   N := Array(0..(l1 − 1), 0..(h1 + h2) − 1, 0);
    for x from 0 to l1 − 1 do
       for y from 0 to h1 − 1 do
        N[x, y] := m2[x, y]
        od
    od;
    for x from 0 to l1 − 1 do
       for y from h1 to (h1 + h2) - 1 do
        N[x, y] := m1[x, y − h1]
        od
    od;
    [N, p1]
    end:
> AfficheI( poserV( inverser(testF), flipH(testF) ) );
```

Question 4

... mêmes remarques ...

```
> poserH :=proc(i1, i2)    local h1, l1, p1, m1, h2, l2, m2, p2, N, x, y;
    h1 := H(i1); l1 := L(i1); p1 := P(i1); m1 := M(i1);
    h2 := H(i2); l2 := L(i2); p2 := P(i2); m2 := M(i2);
   N := Array(0..(l1 + l2) - 1, 0..h1 − 1, 0);
    for x from 0 to l1 − 1 do
       for y from 0 to h1 − 1 do
        N[x, y] := m1[x, y]
        od
    od;
    for x from l1 to (l1 + l2) - 1 do
       for y from 0 to  h1 − 1 do
        N[x, y] := m2[x − l1, y]
        od
    od;
    [N, p1]
    end:
> AfficheI( poserH( inverser(testF), flipH(testF) ) );
```

II Transferts

Question 5

Pp est à lire "Pprime", etc

```
> Pp := 15 : t := Array(0..10, [1, 0, 8, 3, 4, 6, 7, 9, 11, 13, 15]) :
```

```
> transferer := proc(i, Pp, t)
        local h, l, p, m, N, x, y;
        h := H(i); l := L(i); p := P(i); m := M(i);
        N := Array(0..h - 1, 0..l - 1, 0);
        for x from 0 to h - 1 do
            for y from 0 to l - 1 do
            N[x, y] := t[m[x, y]]
            od
        od;
    [N, Pp]
    end:
> AfficheI(transferer(testF, Pp, t));
```

Question 6

```
> Past := Array(0..10, [10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
> reInverser := proc(i) transferer(i, P(i), Past) end;
```

Ce qui suit ne devrait pas marcher ...j'y utilise reInverser avec 3 variables ! Cela vient d'une mise au point précédente de ce type + de Maple qui prend (par défaut) le premier terme d'une séquence sans vérifier le typage. Rien de tel pour provoquer des amplifications d'erreurs

```
> AfficheI(reInverser(testF, 10, Past));
> AfficheI(reInverser(testF));
```

Question 7

L'exigence de devoir utiliser histo comme variable globale ne gêne pas pour définir la procédure histogramme, mais dans les tests de la question 8 ça provoque des erreurs de partout (quand on omet de ré-initialer les divers histo globaux). Alors j'ai localisé histo et c'est la valeur renvoyée par la fontion histogramme

Si quelqu'un sait dans quel langage c'est mieux en global, merci de me le dire. On économise la création d'un Array de dimension 1, par rapport aux calculs effectués ici c'est insignifiant

```
> histogramme := proc(i)
#       global histo;
        local histo, h, l, p, m, N, x, y, k;
        h := H(i); l := L(i); p := P(i); m := M(i); histo := Array(0..p, 0);
        for k from 0 to p do histo[k] := 0 od;
        for x from 0 to h - 1 do
            for y from 0 to l - 1 do
             histo[m[x, y]] := histo[m[x, y]] + 1
         od od:
         eval(histo)
    end:
> histo := histogramme(testF);
> convert(histo, list);
> convert(%, `+`);
```

```
> convert(histogramme(testP50), list);
```

Question 8

Pour tester la Q8 on va ajouter 10 partout à testF (on peut nommer cela un palichonnage)

```
> testFplus10 := allouer(11, 11, 20) :
> testFplus10 := remplisA(testFplus10, (i,j) →10 + carFnum([i,j])) :
> M(testFplus10) :
> display(DeNumAGraph(testFplus10), scaling=constrained);
> testPplus10 := allouer(11, 11, 20) :
> testPplus10 := remplisA(testPplus10, (i,j) →10 + carPnum([i,j])) :
> display(DeNumAGraph(testPplus10), scaling=constrained);
```

La fonction cumul prend un Array et cumule ses valeurs. Je n'ai pas compris pourquoi Size(A) est du genre [1 11] au lieu de 11 .... en mettant [2] on récupère le 11 voulu

Quand on a ainsi récupéré 11 il ne faut pas mettre 11 mais 10 car les indices partent de 0 ... voilà un bon paquet d'erreurs coûteuses en temps !

On découvre aussi que lors de l'affichage "texte" d'un Array, Maple "s'économise" du travail en n'écrivant pas les valeurs 0, comme c'est souvent cela que l'on regarde ici, ça trouble

```
> cumul :=proc(A) local S, taille, k;
      taille := Size(A)[2]; #      print(taille);
  S := Array(0 .. taille, A[0]); # print(S);
      for k from 1 to (taille − 1) do S[k] := S[k − 1] + A[k] od; eval(S) end:
> histogramme(testF);
> cumul(histogramme(testF)); cumul(histogramme(testP));
> Size(histogramme(testFplus10))[2];
> histogramme(testFplus10); histogramme(testPplus10);
> cumul(histogramme(testFplus10))[10];
> P(testFplus10); P(testPplus10);
> map(z→histogramme(testFplus10)[z], [$0 ..20]); map(z
      →histogramme(testPplus10)[z], [$0 ..20]);
> PlusGris :=proc(A) local k, histoA;
            histoA := histogramme(A); k := 0;
            while (histoA[k]=0) do k := k + 1 od; k end;
> PlusGris(testFplus10);
> histogramme(testFplus10);
> vprime :=proc(i, v)
    local h, l, p, m, vmin, vp, k, hisA, chisA, coeff;
        h := H(i); l := L(i); p := P(i); m := M(i);
        vmin := PlusGris(i); hisA := histogramme(i); chisA := cumul(hisA);
  print(vmin); print(hisA); print(chisA); print("h", h);
      vp := Array(0 ..p, 0); coeff := p/(h·l − hisA[vmin]);
  print(coeff);
```

```
            for k from vmin  to p do vp[k] := round(coeff· (chisA[k] − hisA[vmin])) od;
            print(vp[vmin], vp[vmin + 1], vp[p]);
            vp
    end:
> M(testFplus10);
> Mprime := map(vprime(testFplus10, M(testFplus10)), M(testFplus10));
> egtestFplus10 := remplisA(testFplus10, (i, j) →Mprime[i, j]) :
> display(DeNumAGraph(egtestFplus10), scaling = constrained, style = patchnogrid);
> display(DeNumAGraph(testF), scaling = constrained, style = patchnogrid);
```

J'attribue le blanc sale qui remplace le blanc propre à la fonction round ... (?) Il faudrait ré-essayer avec floor et ceiling

Question 9

L'image est blanche : la fonction M est constante égale à P. l'histogramme donne donc 0,0,0....H*L,

vmin ... est mal défini, cela peut être P ou bien ne pas être défini

Si vmin=P, le dénominateur de v' est 0, la fonction egaliser renverra une erreur

Si vmin n'est pas défini v' est "encore moins" défini

L'idée de poser une question concernant une phrase de l'énoncé qui est justement peu claire ... semble peu claire, peut être fallait il ici faire un discours sur les procédures qui renvoient des exceptions (?). Aucun étudiant de PSI n'a entendu parler de cela!

Question 10

Dans la fonction qui suit les niveaux de gris iront de 0 à P'-1 : avec P'=2 on aura ce dont parlait l'énoncé avec 1

```
> reduire :=proc(i, Pp)
            local h, l, p, m, N, x, y;
            h := H(i); l := L(i); p := P(i); m := M(i);
            N := Array(0 ..h − 1, 0 ..l − 1, 0);
            for x from 0 to h − 1 do
                for y from 0 to l − 1 do
                N[x, y] := floor( m[x, y]·Pp / p )
                od
            od;
    [N, Pp]
    end:
> AfficheI(reduire(testF, 8));
> AfficheI(reduire(testF, 4));
> AfficheI(reduire(testF, 2));
```

III Tramage

Question 11

Les noms M et m étant utilisés depuis le début, je re-nomme T la trame-exemple que l'énoncé nomme M, elle servira à fabriquer l'image K (comme dans l'énoncé)

```
> T := Array(0..3, [3, 2, 1, 0]) :
> faisK := proc(trame, h, l)
      [Array(0..h - 1, 0..l - 1, (i, j) → T[irem(j, 4)]), 3] end;
> faisK(T, 13, 12);
> AfficheI(faisK(T, 23, 40));
```

J'avais dans un premier temps comparé les mi et mk ... il faut les normaliser par la division par pi et pk (et c'est plus rapide en multipliant l'autre)

```
> tramer := proc(i, k)    local hi, li, pi, mi, hk, lk, pk, mk;
      hi := H(i); li := L(i); pi := P(i); mi := M(i);   hk := H(k); lk := L(k); pk := P(k);
      mk := M(k);
    # il faudrait des tests d'égalité des h,l ..
      [Array(0..hi - 1, 0..li - 1, (i, j) → if pk·mi[i, j] > pi·mk[i, j] then 1 else 0 fi), 1]
  end:
> tramer(testF, faisK(T, 13, 12));
> AfficheI(tramer(testF, faisK(T, 13, 12)));
> AfficheI(testP50);
> AfficheI(tramer(testP50, faisK(T, 51, 51)))
```

Question 12

```
> echelleV := proc(P) Array(0..P - 1, map(k → P - 1 - k, [$0..P - 1])) end;
> faisTrameK := proc(trame, h, l, P)
      [Array(0..h - 1, 0..l - 1, (i, j) → echelleV(P)[irem(j, P)]), P - 1] end;
> AfficheI(faisTrameK(echelleV(7), 13, 15, 7));
> AfficheI(tramer(testF, faisTrameK(echelleV(17), 13, 15, 7)));
```

Question 13

```
> trame16 := [[1, 5, 10, 14], [3, 7, 8, 12], [13, 9, 6, 2], [15, 11, 4, 0]];
> emart16 := map(Reverse, trame16);
> J16 := Array(0..3, 0..3, trame16); pasJ16 := Array(0..3, 0..3, emart16);
> faisTJ := proc(h, l)
      [Array(0..h - 1, 0..l - 1, (i, j) → if is(iquo(i, 4) + iquo(j, 4), even)
    then J16[irem(j, 4), irem(i, 4)] else pasJ16[irem(j, 4), irem(i, 4)] fi), 15] end;
> faisTJ(8, 8);
> AfficheI(faisTJ(8, 8));
> AfficheI(tramer(testF, faisTJ(11, 11)));
> AfficheI(tramer(testP50, faisTJ(51, 51)))
```

Question 14

Ce que veut l'énoncé n'est pas bien clair !

```
> tramerReduire := proc(i, k) reduire(tramer(i, k), 2) end;
> AfficheI(tramerReduire(testP50, faisTJ(51, 51)))
```

Comme l'image était déjà de P égal à 2, lui appliquer la fonction réduire n 'a rein fait ... L'énoncé dit ensuite de le coder pour une image de profondeur arbitraire ... c'est déjà ce qui avait été demandé et fait

```
> coulPfinal := proc(n, M) local x, y; x := M[1]; y := M[2]; if ((2·x − n)·(2·x − n) ≤ n
      ·y) then CoulCroitNum(n, x, y) else CoulDecNum(n, x, y) fi end:
> n := 200 : Pfinal := allouer(n, n, n); Pfinal := remplisA(Pfinal, (i, j) → coulPfinal(n,
      [i, j])) :
> AfficheISP(Pfinal);
> AfficheISP(tramerReduire(Pfinal, faisTJ(n, n)))
```