TP 2 : tableaux et polynomes

Programmation en C (LC4)

Semaine du 23 janvier 2006

1 Un algorithme de tri

Exercice 1 Écrire une fonction `int separe(unsigned int i,int j,int t[],int x)`, qui réordonne le tableau `t` entre les indices `i` et `j`, de manière à regrouper en premier tous les éléments dont le `x`ème bit vaut 0, et donc en second tous ceux dont le `x`ème bit vaut 1. La fonction doit renvoyer en résultat l'indice de la frontière entre les deux paquets.

► **Correction**

```
int separe(int i, int j, unsigned int t[], int x) {
  int k=i;
  int y=1<<x;
  for (;i<=j;i++) {
    if ((t[i]&y)==0) {
      unsigned int tmp=t[i];
      t[i]=t[k];
      t[k]=tmp;
      k++;
    }
  }
  return k;
}
```

On suppose que le type `unsigned int` fait 32 bits. (C'est le cas sur les PCs.)

À l'aide de la fonction `separe`, il est possible de trier un tableau :
- on commence par séparer sur le trente deuxième bit
- les éléments du premier paquet sont inférieurs à ceux du second, puisque ceux du premier sont inférieurs à 2^31, tandis que ceux du second sont supérieurs ou égaux à 2^31
- il suffit donc de trier chacun des deux paquets indépendement
- pour ce faire, on procède récursivement à l'intérieur de chaque paquet, en séparant cette fois sur le trente et unième bit
- et ainsi de suite jusqu'au dernier bit

Exercice 2 Écrire une fonction prenant en argument un tableau et le triant par la méthode décrite ci-dessus.

► **Correction**

```
void tri_rec(int i, int j, unsigned int t[],int x) {
  int k;
  if (x<0) return;
  k=separe(i,j,t,x);
  if (k>i+1) tri_rec(i,k-1,t,x-1);
  if (k<j) tri_rec(k,j,t,x-1);
```

```
}

void tri(int n,unsigned int t[]) {
  tri_rec(0,n-1,t,31);
}
```

2 Polynomes

On représente un polynome par un tableau de `double` contenant ses coefficients (par exemple le coefficient de degré i dans la case d'indice i).

La libc fournit plein de fonctions pour traiter les nombres de types `double`. Pour les utiliser, il faut mettre un `#include<math.h>` au début de votre fichier `.c`, et ajouter un `-lm` sur la ligne de commande de compilation (par exemple `gcc -lm truc.c`). En particulier, la fonction `pow` qui calcule une puissance, et la fonction `floor` qui calcule la partie entière.

Exercice 3 Écrire une fonction `double evalue(double P[],int d,double x)` qui évalue le polynome `P` de degrée d sur le nombre `x`.

▶ **Correction**

```
double evalue(double P[], int d, double x) {
  int i;
  double y,z;
  for (i=0,y=1,z=0;i<=d;i++,y*=x) {
    z+=y*P[i];
  }
  return z;
}
```

Exercice 4 Écrire une fonction `double racine(double P[],int d, double a, double b, double precision)` qui calcule par dichotomie une valeur approchée à `precision` près d'une racine du polynome `P` de degré `n` situé entre `a` et `b`. On suppose que `P` n'a pas le même signe en `a` et en `b`.

▶ **Correction**

```
double racine(double P[], int d, double a, double b, double precision) {
  int sgna=evalue(P,d,a)>0;
  while (b-a>precision) {
    double c=(a+b)/2;
    int sgnc=evalue(P,d,c)>0;
    if (sgnc==sgna) a=c; else b=c;
  }
  return a;
}
```

Exercice 5 Écrire une fonction `void decalage(double P[],int d, double x)` qui transforme `P` en `P(X+x)`.

▶ **Correction**

```
double binomial(int n, int p) {
 return fact(n)/fact(p)/fact(n−p);
}
```

```
void decalage(double P[],int d,double x) {
  int i;
  for (i=1;i<=d;i++) {
    int j; double y;
    for (j=i−1,y=x;j>=0;j−−,y∗=x) {
     P[j]+=pow(P[i],i−j)∗binomial(i,j)∗y;
    }
  }
}
```

Exercice 6 Écrire une fonction

> **void** trace(**double** P[],**int** d,**double** precision,**double** x,**double** y,
> **int** largeur, **int** hauteur,**int** image[])
>
> qui dessine dans le tableau `image` le graphe du polynome `P` de degré `n`. Le coin inférieur gauche de l'image est en (x, y), l'image fait `largeur` pixels de large, `hauteur` pixels de haut, et la distance horizontale ou verticale entre deux pixels adjacents est de `precision`. Si vous affichez un point par abcisse, la courbe sera discontinue quand la dérivée est trop grande. Essayez d'y remédier.

Quelques rappels du ternier TP pour la manipulation des images :

On représente une image par un tableau d'entiers, chaque entier représentant la couleur d'un pixel. Ici, une couleur sera juste un entier entre 0 (noir) et 255 (blanc), représentant un niveau de gris. Les pixels sont donnés dans l'ordre suivant :
– le premier pixel est le coin supérieur gauche
– suivent les pixels de sa ligne, de gauche à droite
– suit la deuxième ligne, toujours de gauche à droite
– et ainsi de suite jusqu'en bas
Le pixel de coordonnées (x, y) s'obtiendra donc par `image[x+width*y]`.

Vous pouvez récupérer les fichiers `ppm.c`, `ppm.h` à l'adresse :

`http://www.eleves.ens.fr/~lhabert/LC4/tp2/`

Le fichier `ppm.c` fournit les deux fonctions suivantes :

void charge_image_ppm (**const char**∗ filename, **int** width, **int** height, **int** image[]);
void enregistre_image_ppm (**const char**∗ filename, **int** width, **int** height, **const int** image[])

qui permettent de charger depuis ou enregistrer dans un fichier une image.

Il faudra avoir au préalable compilé `ppm.c` avec la commande `gcc -c ppm.c` (à faire une seule fois), puis compiler votre `.c` (mettons que vous l'avez appelé `image.c`) à vous avec `gcc ppm.o image.c` (en fait, `gcc -lm ppm.o image.c`).

► **Correction**

```
int ordonnee(double y,double y0,double precision) {
  int j=floor((y-y0)/precision);
  return j;
}

void trace(double P[],int d,double precision,double x,double y,
           int largeur, int hauteur,int image[]) {
  int i,j,k;
  j=ordonnee(evalue(P,d,x),y,precision);
  if (j>=0&&j<hauteur) image[j*largeur]=0;
  for (i=1;i<largeur;i++,x+=precision,j=k) {
    int l;
    k=ordonnee(evalue(P,d,x),y,precision);
    if (j<0||j>=hauteur||k<0||k>=hauteur) continue;
    if (j<k) {
      for (l=j;l<(j+k)/2;l++) image[i-1+largeur*l]=0;
      for (;l<=k;l++) image[i+largeur*l]=0;
    } else {
      for (l=k;l<(j+k)/2;l++) image[i+largeur*l]=0;
      for (;l<=j;l++) image[i-1+largeur*l]=0;
    }

  }
}

int main(int argc,char**argv) {
  int image[400*400];
  int i;
  double P[3]={
    2,-2,1
  };
  for (i=0;i<sizeof(image)/sizeof(int);i++) image[i]=255;
  trace(P,2,1./400,1,1,400,400,image);
  enregistre_image_ppm(argv[1],400,400,image);
  exit(0);
}
```