TP2 – Les tableaux

Projet de programmation M1

7 Octobre 2014

Exercice 1. [Mesures d'hygiène] Le but de cet exercice pédagogique est de vous montrer les bonnes habitudes à prendre quand vous manipulez des tableaux et les erreurs à éviter.

1. Écrivez une fonction `print_array` qui prend en argument un tableau d'entiers et écrit ses éléments séparés par un espace, puis saute une ligne. Pouvez-vous vous contenter simplement d'un seul argument pour cette fonction ?
   Morale : Quand vous écrivez une fonction qui prend en argument un tableau et que vous prévoyez de le parcourir, vous devez aussi prendre sa longueur en argument puisqu'en C, il n'y a aucun moyen de trouver la longueur d'un tableau à partir du tableau seul. C'est ensuite votre responsabilité d'appeler vos fonctions avec la bonne taille de tableau.
2. Écrivez une fonction qui prend en argument un tableau et met la valeur de la première case (*la case 0 !*) à 0. Appelez-la dans `main` sur le tableau `int t[] = {1,2,3}`. Puis affichez `t[0]`. Que remarquez-vous ?
   Morale : Le comportement ici diffère de ce que vous avez observé la semaine dernière quand l'argument était de type int. Nous expliquerons les détails de ce comportement plus tard.
3. Dans le `main`, déclarez un tableau de taille 5 (`int t[5]`), appelez `print_array(t,5)`. Compilez et exécutez deux fois le programme. Que remarquez-vous ?
   Morale : Ne pas initialiser les valeurs dans un tableau peut mener à des comportements bizarres.

Réponse. *On donne toute les fonctions d'un coup, testées dans le* `main` *:*

```c
#include <stdio.h>

void print_array(int t[], int n) {
  int i;
  for (i=0;i<n;i++) {
    printf("%d ", t[i]);
  }
  printf("\n");
}

void change(int t[]) {
  t[0] = 0;
}

int main() {
  int t1[5],t2[]={1,2,3};
  change(t2);
  print_array(t2,3);
  print_array(t1,5);
}
```

Exercice 2. [Le tri sélection] Le tri sélection est un algorithme de tri célèbre que nous nous proposons d'étudier ici. Il fonctionne ainsi :

- Trouver la position i_{max} du maximum du tableau
- Échanger les valeurs de la case i_{max} et de la dernière case du tableau
- Recommencer en ignorant la dernière case du tableau, puis l'avant-dernière etc.

On commence par découper cet algorithme en opérations élémentaires qu'on va implémenter avant.

1. Écrivez une fonction `int indice_max(int t[], int n)` qui étant donné un tableau `t` et un entier `n`, renvoie la position du plus grand élément de `t` parmi les `n` premiers éléments de `t`.

2. Écrivez une fonction `void echange(int t[], int i, int j)` qui étant donné un tableau `t` et deux indices `i`,`j`, échange les valeurs des cases d'incice `i` et `j`. *Testez-la !* (cf Exercice 1, question 2 pour le comportement de ces fonctions).
3. Utilisez les deux fonctions précédentes pour écrire la fonction `void tri_selection(int t[], int n)` qui trie le tableau `t` de taille `n`.

Réponse. *Voici une implémentation complète du tri sélection :*

```c
#include <stdio.h>

void echange(int t[], int i, int j) {
  int tmp = t[i];
  t[i] = t[j];
  t[j] = tmp;
}

int indice_max(int t[], int n) {
  int i, imax=0;
  for(i=0; i<n; i++) {
    if (t[i] > t[imax]) {
      imax = i;
    }
  }
  return imax;
}

void tri_selection(int t[], int n) {
  int j;
  for(j=n;j>0;j--) {
    int imax = indice_max(t, j);
    echange(t,imax,j-1);
  }
}

void main() {
  int t[] = {1,3,2,6,4,0}, i=0;
  tri_selection(t,6);
  for(i=0;i<6;i++) {
    printf("%d ", t[i]);
  }
  printf("\n");
}
```

Exercice 3. [Plus grande somme contiguë] Dans cet exercice, on souhaite implémenter une fonction `int max_sub(int t[], int n)` qui, étant donné un tableau d'entier `t` et sa taille `n`, renvoie la plus grande somme qu'on peut faire en additionnant des éléments consécutifs du tableau. Par exemple pour le tableau `int t[] = {-2,10,-7,10,1,-5}`, la plus grande somme d'éléments consécutifs qu'on peut faire est $10 + -7 + 10 + 1 = 14$.

1. Proposez une première implémentation en calculant directement la valeur maximale pour tout i, j, $0 \leq i \leq j < n$, de la somme `t[i]+t[i+1]+...+t[j]`. Combien d'additions votre algorithme effectue-t-il ?

   Réponse. *En fixant i, il y a $n - i$ sommes à calculer. Quand on a calculé* `t[i]+t[i+1]+...+t[j]`, *on a besoin d'une seule addition supplémentaire pour calculer* `t[i]+t[i+1]+...+t[j+1]`. *Donc pour calculer toutes les sommes qui commencent à la case i, on a besoin de $n - i - 1$ additions. Donc en tout : $\sum_{i=0}^{n-1} n - i - 1 = O(n^2)$.*

2. On va trouver une meilleure solution qui fait un nombre linéaire d'étape en la taille du tableau. Pour cela, on va calculer pour tout $i < n$, une valeur m_i qui contient la plus grande somme d'éléments consécutifs se terminant à la case i.
   (a) Que vaut m_0 ? Exprimez ensuite m_{i+1} en fonction de m_i.
   Réponse. $m_0 =$`t[0]` *puisqu'il y a un seul sous-tableau qui finit en* 0. *Puis* $m_{i+1} = \max(m_i +$ `t[i + 1]`$,$ `t[i + 1]`$)$
   (b) Comment trouver à partir de cela la somme qui nous intéresse ?

Réponse. *Prenons la plus grande sous-somme S consécutive du tableau. Elle se termine en $i < n$. Donc $S = m_i$. D'où $S = \max_{i<n} m_i$.*

(c) Implémentez cet algorithme, connu sous le nom d'algorithme de Kadane.

Réponse. *Implémentation complète de toutes les fonctions de cet exercice.*

```c
#include <stdio.h>

int max_sub_naif(int t[], int n) {
  int i,j;
  int max = t[0], somme = 0;

  for (i=0;i<n;i++) {
    for(j=i;j<n;j++) {
      somme += t[j];
      if (somme > max) {
        max = somme;
      }
    }
    somme = 0;
  }
  return max;
}

int max_sub(int t[], int n) {
  int max=t[0], i, maxending=t[0];
  for (i=1;i<n;i++) {
    if (maxending+t[i] < t[i])
      maxending = t[i];
    else
      maxending = maxending+t[i];
    if (max < maxending) max = maxending;
  }
  return max;
}

int main() {
  int t[] = {-2,10,-7,10,1,-5};
  printf("%d\n", max_sub(t,5));
  return 0;
}
```

Exercice 4. [Polynôme] On se propose ici d'encoder les polynômes (à coefficients entiers) par des tableaux de la manière suivante : un polynôme $P(X) = \sum_{i=0}^{d} a_i X^i$ de degré d est représenté par un tableau `t` de taille au moins $d+1$ tel que `t[i] =` a_i.

1. Écrivez une fonction `int evalue(int x, int p[], int degre)` qui évalue le polynôme p de degré `degre` sur x. Attention, le degré du polynôme est-il la taille du tableau ? Combien d'additions/-multiplications votre fonction effectue-t-elle en fonction du degré de p ? En remarquant qu'on peut écrire un polynôme ainsi $((a_d X + a_{d-1})X + \ldots + a_1)X + a_0$ (forme de Horner), proposez une implémentation de `evalue` qui effectue seulement d multiplications et d additions.

   Réponse. *Si on évalue naïvement le polynôme, on calcule x^i à chaque boucle, soit en le calculant explicitement à chaque boucle (et en faisant donc $O(d^2)$ multiplications), soit en gardant une variable y qu'on multiplie par x à chaque tour de boucle, voir corrigé (et dans ce cas on fait $2d$ multiplications : une pour le compteur et une avec le coefficient). On peut montrer que la méthode proposée à la fin de la question est optimale !*

2. Écrivez une fonction `void derive(int p[],int degre)` qui dérive le polynôme p.
3. Écrivez une fonction `void somme(int p[], int degrep, int q[], int degreq, int retour[])` qui $p + q$ dans le tableau retour. Quel taille doit avoir ce tableau ? Faites de même pour le produit $p \times q$.

Réponse. *Voici un exemple d'implémentation des fonctions sur les polynômes :*

```c
#include <stdio.h>
```

```
void print_poly(int p[], int degre) {
  int i;
  printf("%d", p[0]);
  for(i=1;i<=degre;i++)
    printf("+%d*x^%d", p[i],i);
  printf("\n");
}

int evalue_naif(int x, int p[], int degre) {
  int xi = 1, somme = 0, i;
  for(i=0;i<=degre;i++) {
    somme = somme + xi*p[i];
    xi = xi*x;
  }
  return somme;
}

int evalue(int x, int p[], int degre) {
  int somme = 0, i;
  for(i=degre;i>=0;i--) {
    somme = somme*x+p[i];
  }
  return somme;
}

void derive(int p[], int degre) {
  int i;
  for(i=0;i<degre;i++) {
    p[i] = p[i+1]*(i+1);
  }
}

int max(int x,int y) {
  if (x < y) return y;
  else return x;
}
int min(int x,int y) {
  if (x < y) return x;
  else return y;
}

// Retourne le coefficient i du poly meme si i > degre
int coeff(int p[], int degre, int i) {
  if (i <= degre) {
    return p[i];
  }
  else {
    return 0;
  }
}

void somme(int p[], int degrep, int q[], int degreq, int retour[]) {
  int degrer = max(degrep, degreq), i;
  for(i=0; i <= degrer; i++) {
    retour[i] = coeff(p, degrep, i)+coeff(q, degreq, i);
  }
}

void produit(int p[], int degrep, int q[], int degreq, int retour[]) {
  int degrepq = degrep + degreq;
  int i,k;
  for (i=0;i<=degrepq;i++) {
    // calcule du coeff de x^i
    retour[i]=0;
    for (k=0;k<=i;k++) {
      retour[i] = retour[i] + coeff(p, degrep, k)*coeff(q, degreq, i-k);
    }
  }
}
```

```
int main() {
  int p[] = {1,2,3}; // p = 1+2x+3x^2
  int q[] = {1,4,5,2}; // q = 1+4x+5x^2+2x^3
  int pplusq[4]; // degre 3 donc 4 coefficient
  int pfoisq[6];

  somme(p,2,q,3,pplusq);
  produit(p,2,q,3,pfoisq);

  print_poly(pplusq,3);
  print_poly(pfoisq,5);

  derive(q,3);
  print_poly(q,2);

  printf("%d\n", evalue(4, p, 2));
}
```