Introduction en algorithmique

Cours 1 – Algorithmique des tableaux

Catalin Dima

Objectifs du cours

- Manipulation des tableaux et applications : ensembles, relations, représentation graphiques...
- Algoritmes de tri et de recherche.
- Techniques de programmation :
  - Algorithmes gloutons.
  - Algorithmes de type backtracking (avec retour en arrière).
  - Programmation dynamique.
- Introduction à la recursivité.
- Introduction à la complexité.
- Structures de données élémentaires
  - Piles, files et autres listes.
  - Arbres.
- Éléments de complexité des algorithmes.
- Notion de graphe, qqs algorithmes de chemins dans les graphes.

Ressources

- D. Knuth, The art of computer programming.
- Th. Cormen, Ch. Leiserson, R. Rivest, Introduction à l'algorithmique.
- Et d'innombrables autres livres de programmation...
- Notes de cours
  `http ://www.univ-paris12.fr/lacl/dima/enseignement/`.
- Ressources pour la programmation C (voir cours !).
- Pages du manuel `man 3 printf`
- En ligne :
  `http ://www.linux-kheops.com/doc/ansi-c/Introduction_ANSI_C.htm`

Evaluation

- Examen : 67% de la note finale.
  - 3/4 sujets = programmes à écrire !
- Note TP : 33% de la note finale.

Tableaux en C

- Déclarations, forme la plus simple : `type nom[taille_constante] ;`

```
int tableau1[100],tableau2[22];
float tableau3[2];
```

- Déclaration avec initialisation

```
int t[3] = { 1, 2, 3 };      // { 1, 2 } accepte aussi !
float abcd[] = { 0.1, 2.3, 4.5 }
```

- Taille implicite !
- Déclarations de tableaux constants :

```
const long v[3] = { 1, 2, 3 }; // aucune case n'est modifiable !
```

- Impossible de déclarer des tableaux de taille = valeur d'une variable !

```
const a = 3;
int v[a];    // interdit par le compilo !
```

Indices de tableaux

Utilisation d'un élément d'un tableau : `nom[expression_indice]`.

- Premier indice : 0 !
- Dernier indice : taille – 1 !

```
int v[5] = { 1, 2, 3, 4, 5 };
v[0] = 6;       // correct !
v[i] = v[i+1]   // correct, tant que 0 < i < 3 !
v[5] = 5;       // element inexistant !!
```

- Instruction incorrecte – erreur d'exécution ! (non pas de compilation !)
- Indice = expression entière :

```
int i,j,t[20],a;
scanf("%d%d",&i,&j);
t[i-2*j] = 2*a;
```

- Interdit d'affecter un tableau à un autre :

```
int b[5], a[5] = { 1, 3, 5, 7, 9 };
b=a;       // erreur de compilation !!
```

- Alors comment faire ?...

Lecture & affichage

- On applique ce qu'on a appris avec `printf` et `scanf` :

```
int a[4], i;
scanf("%d",&i);  scanf("%d%d",&a[2],&a[i]);
```

- Lecture & affichage de tous les éléments :

```
int a[10], i=0;
while(i<10){
   printf("Introduire a[%d] = ", i);  // a[i] !
        // erroné d'écrire printf("Introduire a[i] = "); !
   scanf("%d",&a[i]);
   i++;
}
       // ecrire maintenant un for
       // qui affichera les éléments du tableau
.....
   printf("Element a[%d] = %d\n", a[i], i);
....
```

- Rappel : `\n` = retour charriot.

Tableaux et fonctions

- Paramètres d'une fonction modifiant un tableau :"le tableau + **la dimension** !
- Exemple : fonction affichage d'un tableau

```
void affiche-tbl(int tab[], int dim){
      // mettre tab[100] ne changerait rien !
int i; //indice
for (i=0; 1<dim; i++)
   printf("a[%d]=%d",i,tab[i]);
}
```

Appel :

```
int vecteur[6];
affiche-tableau(vecteur,6);
```

- Ne jamais oublier de déclarer la dimension comme paramètre
- ... alors comment afficher 100 éléments d'un tableau qui n'en a que 6 ? !

Fonctions qui modifient un tableau

- Maintenant on veut lire un tableau dans une fonction.
- Le tableau et la dimension seront les paramètres.
- Similaire au cas affichage, **à quelques différences près (très importantes !)**

```
void lire-tableau(int tab[], int *dim){
        // int dim serait une grande erreur !
int i; //indice
printf("introduire la dimension");
scanf("%d",dim);
        // eh oui ! c'est un pointeur, et pas un entier !
for (i=0; 1<*dim; i++) // la valeur pointée par dim !
   scanf("%d",tab[i]);
}
```

- Appel :

```
int a[20], n;
lire-tableau(a,&n);
        // et on lit donc la dimension comme dans un scanf !
```

- Pourquoi `int dim` serait une erreur dans la déclaration de `lire-tableau` ?

Algorithmes élémentaires

Somme des éléments d'un tableau d'entiers :

- Comment on la calcule ?
  - Au début : $somme = 0$
  - Jusqu'au 1er élément : $somme = 0 + a[0] = somme\_anterieure + a[0]$.
  - Jusqu'au 2e : $somme = 0 + a[0] + a[1] = somme\_antrieure + a[1]$.
  - ....

```
int somme(int tab[], int dim){
          // la somme renvoyée comme résultat !
  int i, somme;
  i=somme=0;
  while (i<dim){
     somme = somme + a[i++]; // ou somme += a[i++];
  }       // écrire aussi avec un for !
}
```

Algorithmes élémentaires

Max des éléments d'un tableau d'entiers :

- Technique "bête" : on vérifie, pour chaque élément, si c'est le max...
  - Sûrement trop d'allers-retours dans le tableau !
- Algorithme : calculer les max intermédiaires :
  `max{calculé_entre_0_et_i} =`
  `max ( max{calculé_entre_0_et_i-1}, a[i] ) ;`
- `max{calculé_entre_0_et_-1} = le plus grand entier !!`

```
i=0;
max = INT_MAX;
while (i<10){
   max = (max<a[i]) ? max : a[i] ;
   i++;
}
```

- Opération ternaire : `expr_booleenne ? expr1 : expr2`
  - Si `expr_booleenne` vraie, alors retourne `expr1`, sinon, retourne `expr2`.
- Comparer cette solution à l'idée "bête" !
  - Combien d'opérations sur le tableau `{ 7,4,6,9,15,2,10 }` ?

Algorithmes élémentaires

- Chercher si un élément se trouve dans un tableau :

```
int se-trouve(int elem, int tab[], int dim)
  // elem est l'element qu'on cherche dans tab
  // renvoie la POSITION de l'élément dans le tableau
int i=0, pos=-1;
while (i<dim){
   if (tab[i]==elem) {
      pos=i;
      break;  // on l'a trouvé ! il faut sortir de la boucle !
   }
   i++;   // sinon, il faut continuer à chercher !
}
return( ...); // quoi ?
```

- Quelles sont les valeurs possibles de pos ? Que signifie chacune ?
- Comment utiliser cette fonction ? (appel)
- Et si on veut renvoyer seulement 1 si élément trouvé, et 0 si pas trouvé, comment faire ?

```
if (.....) // comment tester si on a bien trouvé l'élément ?
   return (1);
else return (0)
// faut-il changer aussi l'entête de la fonction ?
```

Insertion dans un tableau

- Insérer un élément dans un tableau trié :
  - Trouver l'emplacement pour le nouvel élément.
  - Si déjà présent, terminer.
  - Déplacer les éléments plus grands d'une case vers la droite.
- Trouver l'emplacement de `x` : l'endroit où `a[i] < x <a[i+1]`
  - Propriété à tester pour tous les `i`.
  - Mais on peut faire mieux : tester seulement si `x < a[i]` !
  - Et on s'arrête lorsque `x < a[i]` !
    - Car, comme on a dû avoir `x > a[i-1]` lors de l'itération précédente, on a trouvé la position.
- Ne pas oublier de sortir *avec une information particulière* lorsque `x` se trouve déjà dans le tableau !

Trouver la position pour le nouvel élément

Propriété à l'entrée de la boucle, `x>a[i]` !

```
i=present=0;
while (i<taille){
   if (x==a[i]){
       present=1; break;
   }
   if (x<a[i]) break;
   i++;
}
```

- Deux possibilités à la fin de cette boucle :
  - `present==1` : `x` se trouve déjà dans le tableau à l'indice `i`.
  - `present==0&&i==taille` : l'endroit où il faut placer `x` est `taille` !
    Car cela veut dire que `x>a[i]` pour tous les `i` !
  - `present==0&&i<taille` : l'endroit où il faut placer `x` est "entre" `i-1` et `i`.
    Car si `i<taille`, c'est qu'on est sorti par le 2e break, donc `a[i-1]<x<a[i]` !
- Euh, mais ça n'existe pas "entre" deux indices !...
  - Donc il faudra déplacer tout entre `i` et `taille-1` pour faire de la place à `x` !

Déplacer les éléments d'un tableau

- Si on commence par la fin :

```
j=taille;
while (j>i){
   a[j]=a[j-1];
   j--;
}
```

- Donc la nouvelle taille sera `taille+1` !
- On peut aussi le faire en ordre ascendent
  - Mais attention à ne pas perdre le contenu de chaque case !

```
j=i;
while (j<taille){
   a[j+1]=a[j++];    // grave erreur !
}                    // problème ! on perd le contenu de a[j+1] !
```

  - Avec cet algo erroné, on recopie `a[i]` dans toutes les cases jusqu'à `a[taille]` !
- Solution correcte : rajouter une variable "tampon" supplémentaire
  - Pour garder la valeur de `a[j+1]` jusqu'à l'itération suivante.
- Rassembler le tout dans une fonction !

Représentation tabulaire des ensembles

- Un tableau peut représenter un ensemble.
  - Mais bien-sûr, on veut pas de doublons dans la représentation !
  - Donc la lecture d'un ensemble doit aussi vérifier qu'un nouvel élément saisi ne se trouve pas déjà dans le tableau !
  - Appel de la fonction de test d'appartenance dans la fonction de lecture !
- Exemple de lecture d'un ensemble d'entiers :

```
void lit-ens-ent(int tab[], int *dim){
  int i, val;
          // valeur saisie, à vérifier avant de l'insérer
  printf("introduire 0 pour terminer la saisie ");
  dim=0;  // au départ l'ensemble est vide, non ?
  scanf("%d",val);
  while((val!=0) && (se-trouve(tab,dim) != ....) ){
     tab[i++] = val;
     scanf("%d",tab[i]);
  }
}
```

- On peut aussi insérer les éléments en ordre *croissant* !
  - Cela ferait la recherche plus *rapide*, non ?
  - ... mais devrait-on aussi modifier la fonction de recherche ?...

Invariants

- Dans chaque boucle, on calcule une nouvelle version du tableau.
- ... ou une nouvelle valeur d'une variable.
- Cette nouvelle version se calcule de la même manière pour toutes les itérations.
- On parle d'invariant.
- Exemple :

```
i=0;
max = INT_MAX;
while (i<10){
   max = (max>a[i]) ? max : a[i] ;
   i++;
}
```

- Invariant :
  - Au début de la boucle, `max` garde le max du sous-tableau `a[1..i-1]`.

Invariant

```
int i=0, pos=-1;
while (i<dim){
   if (tab[i]==elem) {
      pos=i;
      break;  // on l'a trouvé ! il faut sortir de la boucle !
   }
   i++;   // sinon, il faut continuer à chercher !
}
```

- Invariant au début de la boucle ?
- Essayez aussi avec les autres algorithmes vus en cours !
- Et essayez toujours de construire un algorithme en pensant à ce qui doit être valable au début de chaque boucle.

Opérations sur les ensembles

- Union :
  - On prend tous les éléments du deuxième ensemble et on les met dans le premier.
  - Bien-sûr, en évitant d'insérer des doublons !
  - Mais celà, on sait comment faire, non ?
- Appartenence : ..... ?
  - Revient à faire une recherche dans un tableau !
- Intersection :
  - Copier chaque élément du premier ensemble (tableau) dans le dernier.
  - Puis prendre, un a un, les éléments du deuxième ensemble et vérifier son appartenence dans le dernier.
  - Si résultat de l'appartenence négatif, le rajouter dans le dernier tableau.
- Différence : ..... ?

Recherche dans tableau *trié*

- On peut chercher d'un bout à l'autre l'élément désiré.
- Mais cela prendra du temps proportionnel à la taille du tableau.
- Dans un dictionnaire on ne cherche pas, page par page, à partir de la première !

Recherche dans tableau *trié*

- On peut chercher d'un bout à l'autre l'élément désiré.
- Mais cela prendra du temps proportionnel à la taille du tableau.
- Dans un dictionnaire on ne cherche pas, page par page, à partir de la première !
- Idée de recherche : aller directement au milieu du tableau `t`.
- ... et comparer le nombre donné `k` avec l'élément du milieu.
- Si pas de chance, deux situations possibles :
  - `t[milieu] < k` : on doit chercher alors dans le sous-tableau `t[0..milieu]` !
  - `t[milieu] > k` : que faire ?
- Comment peut-on itérer cette idée ?
  - On imagine que le tableau commence à `inf` et se termine à `sup` !
  - Le milieu est donc : !

Recherche dans tableau *trié*

- On peut chercher d'un bout à l'autre l'élément désiré.
- Mais cela prendra du temps proportionnel à la taille du tableau.
- Dans un dictionnaire on ne cherche pas, page par page, à partir de la première !
- Idée de recherche : aller directement au milieu du tableau `t`.
- ... et comparer le nombre donné `k` avec l'élément du milieu.
- Si pas de chance, deux situations possibles :
  - `t[milieu] < k` : on doit chercher alors dans le sous-tableau `t[0..milieu]` !
  - `t[milieu] > k` : que faire ?
- Comment peut-on itérer cette idée ?
  - On imagine que le tableau commence à `inf` et se termine à `sup` !
  - Le milieu est donc : `milieu = (inf+sup)/2` !
- Et comment fait-on pour réduire notre recherche dans le sous-tableau `t[0..milieu]` ?

Algorithme de recherche *binaire*

```
inf=0;
sup=n-1;
trouve=0;
while((inf<=sup)&&(!trouve)){
   i=(inf+sup)/2;
   if(el==tableau[i]) trouve=1;
       else if(el<tableau[i]) sup=i-1;
            else inf=i+1;
}
// si trouve==1, alors i est l'indice de l'element !
// sinon il ne se trouve pas dans notre tableau.
```

- Comparer le nombre d'opérations de la recherche binaire avec le nombre d'opérations pour la recherche séquentielle.

Trier des tableaux

- Tri par sélection :
  - Trouver le max du tableau et le mettre dans un 2e tableau.
  - Trouver le 2e max du tableau et le mettre lui aussi dans le 2e.
  - ... et la suite ?...
- Il nous faut un invariant !
  - Après chaque itération, on doit se trouver dans une situation dans laquelle on peut appliquer la même séquence d'opérations !
  - Après chaque itération, il faut enlever le max du tableau !

```
m=0;
for(i=0;i<n;i++){
    j=trouve_max(t,n); // renvoie l'indice où se trouve max
    resultat[m] = t[j];
    m++;
    supprime(j,t,n);   // on sait l'écrire, non ?
}
```

Tri par insertion

- Supposons qu'à chaque itération, on a, dans le tableau `res`, une partie de notre tableau initial qui est déjà triée.
- Et que donc dans `t` il y a le reste du tableau initial (mois ce qui se trouve déjà dans `res`).
- Invariant : comment supprimer encore un élément de `t`, en l'insérant dans `res` au bon endroit ?
- On utilise une modification de la recherche binaire, car `res` est déjà trié !

```
m=0;
for (i=0;i<n;i++){
   j=recherche_binaire(t[i],res,m);
         // t[i] devrait être mis à cet endroit !
   insertion(t[i],j,res,m);
   m++;   // ça suffit pas de le faire dans insertion, pourquoi ?
}
```