Algorithmique et Programmation, IMA 3

Cours 5 Constantes en C - Algos de tri

Laure Gonnord
`http://laure.gonnord.org/pro/teaching/`
`Laure.Gonnord@polytech-lille.fr`

Université Lille 1 - Polytech Lille

Partie 1 d'après A. Miné (ÉNS Ulm)

Définition de constantes symboliques

En C :
`#define CST valeur`

- `CST` : identificateur, par convention en majuscules,
- `valeur` : texte arbitraire,
- doit occuper une ligne complète,
- pas de point-virgule ; final.

Effet : dans la suite du programme, `CST` est remplacé par `valeur`.

Application des constantes symboliques

Application :
écrire du code *paramétrique* là où le C interdit les variables.

```c
#define N 10
int t1[N],t2[N]
void f(){
  int i;
  for (i=0;i<N;i++)
    t1[i] = t2[i] + 1;
}
```

▶ Pour changer la taille de tous les tableaux, il suffit de changer une seule ligne.

Danger des constantes symboliques

Définition de constante symbolique $\neq$ affectation de variable !

- affectation : évaluation,
- constante symbolique : substitution littérale.

$\Rightarrow$ danger de “capture” syntaxique.

Exemple d’erreur :

```
#define N x+y
z = 3*N; /* signifie z = 3*x+y, pas z = 3*(x+y) */
         /* x et y peuvent aussi etre symboliques!
```

Solution :

```
#define N ((x)+(y))     /* plus su^r */
```


Énoncé du problème

But

- On va trier des tableaux d’**entiers** de taille N.
- On dispose du test de comparaison entre entiers

Exemple :

11	-2	1515	42	2048	28	11	-78

devient

-78	-2	11	11	28	42	1515	2048

▶ Let’s go !
▶ Attention transparents sans exemple ni dessin, donc, en faire !

Action auxillaire

On dispose de l’action auxilliaire **permuter** de signature (ou prototype) :
`permuter(T:Tableau[N] d’Entiers,ind1:Entier,ind2:Entier)`
qui permute les valeurs des éléments d’indices `ind1` et `ind2` du tableau T.

Tri Sélection

Principe :

- Je cherche le minimum du tableau et je le permute avec la case d'indice 0.
- Je cherche le minimum du tableau restant (le sous tableau $T[1..N-1]$) et je le permute avec la case indice 1
- Je cherche .....

▶ Algo, Correction, Complexité

Sélection

<u>Action</u> *tri-sélection(T)*
  <u>D/R</u>: T : Tableau[N] d'entiers
  <u>L</u>: ideb,i : Entiers
  <u>L</u>: imin : Entier `{Indice de l'élément minimum courant}`
  <u>Pour</u> *ideb* **<u>de</u>** *0* **<u>à</u>** *N-2* **<u>Faire</u>**
    `{Recherche de l'indice de l'élément minimum}`
    imin := ideb;
    <u>Pour</u> *i* **<u>de</u>** *ideb + 1* **<u>à</u>** *N-1* **<u>Faire</u>**
      <u>Si</u> *T[i]* $<$ *T[imin]* **<u>alors</u>**
        imin := i
      <u>Fsi</u>
    <u>Fpour</u>
    permuter(T,ideb,imin)
  <u>Fpour</u>
<u>FAction</u>

Sélection : analyse

Correction : « L'algorithme tri-selection conserve les éléments du tableau T et les trie dans l'ordre croissant »

Invariant : « à la fin du tour $ideb$, le sous-tableau $T[0..ideb]$ contient les $ideb+1$ plus petits éléments de T, dans l'ordre croissant de leurs valeurs »

Coût (nb comparaisons) : $(N-2)+(N-3)+\dots 1 = O(N^2)$.

Tri Bulle

Variante du tri précédent :

- Je parcours tout le tableau en permutant toute suite d'éléments non ordonnés (donc le maximum est ensuite à sa place)
- Je recommence sur le sous-tableau $[0..N-2]$

▶ Algo, Correction, Complexité

Bulle

Action *TriBulle(T)*
  D: T :tab[N]
  L: i,j,tmp : Entiers
  Pour i **de** 0 **à** $N-2$ **Faire**
    Pour j **de** 0 **à** $N-i-2$ **Faire**
      Si *(t[j + 1] < t[j])* **alors**
        permuter(T,j+1,j)
      Fsi
    Fpour
  Fpour
FAction

Bulle : analyse

Correction : on prouve l'**invariant** suivant : « Après le tour de boucle `i`, sous-tableau `T[N-1-i..N-1]` contient les $i+1$ plus gros éléments du tableau, triés.»

Coût : $O(N^2)$ comparaisons quel que soit le tableau.

Tri Insertion

«Tri des cartes à jouer» :

- Je trie les 2 premières cartes.
- Je regarde la troisième et l'insère à sa bonne place (par décalages vers la droite).
- ...

▶ Algo, Correction, Complexité

Insertion

Action *tri-insertion(T)*
  D/R: T : Tableau[N] d'entiers
  L: i : Entier
  L: élt : Entier `{Valeur à insérer}`
  L: ins : Entier `{Indice d'insertion de élt}`
  Pour *i* **de** *1* **à** *N-1* **Faire**
    élt := T[i] `{Initialisation}`
    ins := i
    Tq *(ins ⩾ 1* **et** *T[ins-1] > élt)* **faire**
      T[ins] := T[ins-1];
      ins := ins - 1
    Ftq
    T[ins] := élt `{Insertion de T[i]}`
  Fpour
FAction

Insertion : analyse

Correction : on prouve l'**invariant** suivant : « Au tour i, la boucle insère l'élément $T[i]$ dans le sous-tableau $T[0..i-1]$ déjà trié »

Coût : $O(N)$ au mieux, $O(N^2)$ au pire et en moyenne. Améliorable en $N \ln_2(N)$.

Tri Fusion

Principe «diviser pour régner» :

- Un tableau de taille 1 est trié !
- Je découpe en deux le tableau
- Je trie chacun des sous-tableaux
- Je fusionne

▶ Algo, Correction, Complexité

Tri Fusion - 1

```
Action tri-fusion-bis(T,premier,dernier)
  | D: premier,dernier : Entiers
  | D: T : Tableau[N] d'entiers
  | L: milieu : entier
  | Si premier<dernier alors
  |   | milieu ← (premier+dernier)/2
  |   | tri-fusion-bis(T,premier,milieu)          {Appel récursif 1}
  |   | tri-fusion-bis(T,milieu+1,dernier)        {Appel réc. 2}
  |   | fusion(T,premier,milieu,dernier)
  | Fsi
FAction
```


Tri Fusion - 2

```
Action tri-fusion(T)
  | D: T : Tableau[N] d'entiers
  | Si N>1 alors
  |   | tri-fusion-bis(T,0,N-1)
  | Fsi
FAction
```

▶ Il reste à écrire **fusion**.

Tri Fusion - 3

Dessin ! On va garder en mémoire deux curseurs, `cpt1` et `cpt2`, sur chacun des bouts de tableaux à fusionner.

Action *fusion(T,premier1,dernier1,dernier2)*
- **D**: premier1,dernier1,dernier2 : Entiers
- **D**: T : Tableau[N] d'entiers
- **L**: Ttemp : Tableau[dernier2-premier1+1] d'entiers
- **L**: cpt1,cpt2,premier2 : Entiers
- premier2 := dernier1+1
- cpt2 := premier2
- cpt1 := premier1
- ... suite page suivante ...

FAction

Tri Fusion - 4

Parcours de fusion :

Pour *i* **de** *0* **à** *dernier2-premier1* **Faire**
- **Si** *(c1⩽dernier1* **et** *(t[c1]<t[c2]* **ou** *c2> dernier2))* **alors**
  - fus[i]← t[c1]
  - c1++
- **Sinon**
  - fus[i]← t[c2]
  - c2++
- **Fsi**

Fpour

Pour *i* **de** *0* **à** *dernier2-premier1* **Faire**
- t[premier1+i] ← fus[i]

Fpour

Fusion : analyse

On suppose que fusionne fait bien son travail
Correction : « l'appel à tri-fusion sur un tableau de taille i trie le tableau »

Coût : $O(N \ln_2 N)$ tout le temps. preuve au tableau

Tri Rapide

Principe du pivot
- Je partitionne le tableau en fonction d'un **pivot** (premier élement du tableau).
- Je trie récursivement sur chacun des tableaux à sa gauche et à sa droite.

▶ Algo, Correction, Complexité, en TD !

Tri de tableaux

Théorème

Un tri de tableaux d'entiers par comparaisons ne peut être réalisé en $o(n \ln_2 n)$ comparaisons en moyenne et dans le pire des cas.

▶ Un tri est alors **optimal** si il a une complexité de $\Omega(n \ln_2 n)$ en moyenne et dans le pire des cas.

Tri de tableaux - récapitulatif

On évalue le nombre de comparaisons

Algorithme	Meilleur des cas	En moyenne	Pire des cas
Tri par sélection	$O(N^2)$	$O(N^2)$	$O(N^2)$
Tri à bulle	$O(N^2)$	$O(N^2)$	$O(N^2)$
Tri par insertion	$O(N)$	$O(N^2)$	$O(N^2)$
Tri fusion	$O(N \times \ln_2(N))$	$O(N \times \ln_2(N))$	$O(N \times \ln_2(N))$
Tri rapide	$O(N \times \ln_2(N))$	$O(N \times \ln_2(N))$	$O(N^2)$

Un tri linéaire !

Et si on connaît à l'avance les valeurs des éléments ?
Exemple : tri comptage (ou tri par casiers) :

- On crée autant de casiers que de valeurs possibles
- On compte les occurrences de ces valeurs
- On utilise pour trier

Exemple :

1	10	3	1	3

Tableau d'occurences :

2	0	2	0	3	0	0	0	0	1

et finalement :

1	1	3	3	10

Caractères stable et en place

Stable

Un algo de tri est **stable** si deux valeurs identiques restent dans le même "ordre" à la fin de l'algo.

En place

Un algo de tri est en **place** si il trie sans création d'un tableau auxiliaire.

- Les algorithmes bulle, insertion, sélection sont en place