

Les 26 Septembre, 2 et 3 octobre 2001

## Algorithmes de MIN-MAX

*Laure Danthony*<http://www.ens-lyon.fr/~ldanthon/>

### But du TP, consignes

Ce TP présente des algorithmes de calcul du maximum, puis du maximum et du minimum (simultanément) d'un tableau d'entiers.

Pour la manipulation des entrées-sorties de tableaux, on fournit cette fois dans un fichier `minmax.pas` les procédures `saisie` et `visu` qui permettent respectivement la saisie au clavier et la visualisation à l'écran de tableaux.

Les tableaux utilisés dans ce TP auront une taille bornée, et sont déclarés de la manière suivante :

```
const long_tab = 25;
type table= array[1..long_tab] of integer;
```

Les algorithmes considérés ici effectuent des *affectations* et leur seul critère de décision est la *comparaison* de deux éléments ( $<$  ou  $=$ ). En aucun cas ils ne peuvent effectuer des opérations arithmétiques, comme l'addition et la multiplication.

Comme d'habitude, on rappelle :

1. Utilisez avec exagération le papier et le crayon (**DESSINS**) ...
2. Les tests sont **INDISPENSABLES** : les utiliser sans modération
3. N'hésitez pas à embêter la khôlleuse, elle est là pour ça ...

*Et maintenant, au travail!*

## 1 Maximum

On s'intéresse tout d'abord au calcul du maximum d'un tableau.

► Exo :

1. Ecrire une **fonction** `maxi(t:table):integer` qui retourne le maximum du tableau  $t$ .
2. Evaluer le nombre de comparaisons et le nombre d'affectations (au pire, au mieux) de cette fonction.
3. Pour le calcul des affectations en moyenne (de l'algorithme naturel, qui est certainement celui que vous avez codé), on pourra chez soi lire la correction de ce TP. On notera qu'il est équivalent à  $\ln(n)$  comparaisons.

On a le résultat suivant :

Cet algorithme de MAXIMUM est optimal pour la complexité en comparaisons dans le pire des cas.

Les redoublants essaieront (très rapidement) de prouver ce résultat en montrant tout d'abord que tout élément qui n'est pas le MAX est comparé au moins une fois à un élément plus grand que lui. Les autres se reporteront à la feuille de correction.

A partir de maintenant, on ne s'intéresse plus qu'à la complexité **dans le pire des cas et en nombre de comparaisons** des algorithmes considérés.

## 2 Maximum et Minimum

On cherche ici à calculer simultanément le maximum et le minimum d'un tableau  $t$ .

► Exo : Ecrire une procédure "naïve" `minmax_naif(t:table)` qui imprime à l'écran le minimum et le maximum du tableau  $t$ . La tester et donner sa complexité.

L'idée pour améliorer l'algorithme est de regrouper les éléments à comparer *par paires*, i.e. de comparer à chaque tour de boucle  $t(2k)$  et  $t(2k+1)$ , et ensuite comparer *un seul des deux* à `min_tmp` (lequel?), et l'autre à `max_tmp`.

Dans la suite, on distinguera bien les cas où la longueur du tableau est paire des cas où elle est impaire.

► Exo :

1. EN S'APPUYANT SUR DE NOMBREUX EXEMPLES, écrire sur une feuille l'algorithme décrit plus haut, en pseudo-code. Vous pouvez dans un premier temps utiliser des instructions du type *pour  $i$  de 1 à 15 par pas de 2*.
2. Réfléchir à la façon de simuler l'instruction précédente qui n'existe pas en PASCAL.
3. Coder enfin l'algorithme en PASCAL.
4. Le tester sur des exemples judicieux (longueur de tableau paire et impaire, le minimum sur les bords, ...).
5. Montrer que la complexité de cet algorithme sur un tableau de longueur  $n$  peut s'écrire  $\lceil \frac{n}{2} \rceil + n - 2$ .

Vos collègues taupins ayant suivi(subi?) l'option info pourront avec un peu de travail montrer :

Cet algorithme de MIN-MAX est optimal pour la complexité en comparaisons dans le pire des cas.