

Les 26 Septembre, 2 et 3 octobre 2001

Algorithmes de MIN-MAX

Laure Danthony

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

1 Maximum

- **Fonction maxi**

```

function maxi(t:table):integer;
var i,tmp : integer;
begin
  tmp := t[1];
  for i:=2 to long_tab do
    if t[i]>tmp then tmp:=t[i];
  maxi:= tmp
end;
```

- **Complexité** : en notant n la taille du tableau, cet algorithme effectue *toujours* $n - 1$ comparaisons. Au pire, il réalise n affectations, au mieux il n'en réalise qu'une.
- (FACULTATIF) **Pour le calcul des affectations en moyenne**, notons S_n l'ensemble des permutations de $\llbracket 1, n \rrbracket$. Il s'agit alors de calculer :

$$C(n) = \frac{\sum_{d \in S_n} c(d)}{|S_n|},$$

où $c(d)$ désigne le nombre d'affectations de l'algorithme sur le tableau réalisant la permutation d .

Si on note $P_{n,k}$ le nombre de permutations de S_n telles que sur la donnée $[t[1]; t[2]; \dots; t[n]]$ l'algorithme effectue k affectations. On obtient alors :

$$C(n) = \frac{\sum_{k=1}^n k \cdot P_{n,k}}{n!},$$

et il s'agit ensuite de calculer les $P_{n,k}$.

On obtient facilement la relation de récurrence :

$$P_{n+1,k+1} = P_{n,k} + n \cdot P_{n,k+1},$$

en distinguant le cas où le dernier élément du tableau de longueur $n + 1$ est le max des n autres cas.

Si on pose $G_n(z) = \sum P_{n,k} z^k$ ("série génératrice"), on obtient à l'aide de la relation précédente $G_n(z) = (n + z - 1)G_{n-1}(z)$ soit $G_n(z) =$

$z.(z + 1) \dots (z + n - 1).$

Or ce que l'on veut c'est $\frac{\sum_{k=1}^n k.P_{n,k}}{n!} = \frac{G'_n(1)}{n!} = \frac{\frac{n!}{1} + \frac{n!}{2} + \dots + \frac{n!}{n}}{n!}$, d'où finalement :

$$C(n) = \sum_{i=1}^n \frac{1}{i}.$$

- **FACULTATIF Preuve de l'optimalité de l'algo de MAX**

Soit UN algorithme qui résout le problème du MAX et on suppose qu'on l'applique à $[t[1]; t[2]; \dots; t[n]]$, où $t[1]$ est le max. Cet algo renvoie donc $t[1]$.

On montre par l'absurde que tout élément qui n'est pas le max est comparé au moins une fois à un élément plus grand que lui : supposons que $t[2]$ ne soit pas comparé à plus grand que lui. Alors en jouant l'algo considéré sur l'entrée $[t[1]; t[1] + 1; t[3]; \dots; t[n]]$, l'algo se trompe, ce qui contredit l'hypothèse de départ, cqfd.

Or il y a $n - 1$ éléments qui ne sont pas le max. D'où au moins $n - 1$ comparaisons.

2 Maximum et Minimum

- **Procédure minmax naïve :**

```

procedure minmax_naif(t:table);
var min_tmp, max_tmp, i : integer;
begin
  min_tmp:=t[1];
  max_tmp:=t[1];
  for i:=2 to long_tab do
  begin
    if t[i]>max_tmp then max_tmp:=t[i];
    if t[i]<min_tmp then min_tmp:=t[i];
  end;
  writeln('mini : ', min_tmp);
  writeln('maxi : ', max_tmp)
end;
```

- **Algorithme de MIN-MAX :**

```

- pour n impair :
  DEBUT
  M ← t[1], m ← t[1];
  pour i = 2 à n - 1 par pas de 2 faire :
    si t[i] > t[i + 1] alors
      si m > t[i + 1] alors m ← t[i + 1]
      si M < t[i] alors M ← t[i]
    sinon
      si m > t[i] alors m ← t[i]
      si M < t[i + 1] alors M ← t[i + 1]
  imprimer("mini : ", m);
  imprimer("maxi : ", M);
```

FIN

– pour n pair :

algo similaire sauf que l'on commence par comparer $t[1]$ et $t[2]$, la plus grande valeur est affectée à M , la plus petite à m . La boucle va de 3 à $n - 1$.

• **Simulation du "pas de 2" :**

Pour simuler "pour i de 1 à 15 par pas de 2 faire $t[i] \leftarrow toto$ ", on peut (par exemple) prendre une variable auxiliaire k qui servira de variable de boucle et qui est initialement affectée à 0. Comme i doit être incrémenté de 2 à chaque tour, on doit obtenir une relation $i := 2k + c$, avec c à déterminer. On prend donc $c = 1$ pour obtenir $i = 1$ lorsque $k = 0$ (au début de la boucle).

Il reste à déterminer jusqu'où va la variable k . On a la relation $i \leq 15$ soit encore $2k + 1 \leq 15$ et enfin $k \leq 7$. On code donc finalement en PASCAL :

```
for k:=0 to 7 do
  begin
    i:=2*k+1;
    t[i]:=toto
  end;
```

• **Procédure minmax sioux :**

```
procedure minmax_sioux(L:table);
var min_tmp, max_tmp, i,k : integer;
begin
  if long_tab mod 2 = 0 then
  begin
    if L[1]<L[2] then
    begin
      min_tmp := L[1];
      max_tmp := L[2];
    end
    else
    begin
      min_tmp := L[2];
      max_tmp := L[1]
    end;
  end;
  for k:=0 to ((long_tab DIV 2) - 2) do
  begin
    i:=2*k+3;
    if L[i]>L[i+1] then
    begin
      if min_tmp>L[i+1] then min_tmp := L[i+1];
      if max_tmp<L[i] then max_tmp := L[i];
    end
    else
    begin
      if min_tmp>L[i] then min_tmp := L[i];
      if max_tmp<L[i+1] then max_tmp := L[i+1];
    end
  end
end
```

```

end
else {long_tab est impair}
begin
  min_tmp := L[1] ;
  max_tmp := L[1] ;
  for k:=0 to ((long_tab - 1) DIV 2) - 1 do
  begin
    i:=2*k+2;
    if L[i]>L[i+1] then
    begin
      if min_tmp>L[i+1] then min_tmp := L[i+1];
      if max_tmp<L[i] then max_tmp := L[i];
    end
    else
    begin
      if min_tmp>L[i] then min_tmp := L[i];
      if max_tmp<L[i+1] then max_tmp := L[i+1];
    end
  end
end ;
writeln('mini2 : ',min_tmp);
writeln('maxi2 : ',max_tmp)
end;

```

• **Complexité :**

- n impair : 3 comparaisons par boucle sur $\frac{n-1}{2}$ boucles soit $\frac{3n}{2} - \frac{3}{2}$ comparaisons.
- n pair : 1 comparaison initiale et 3 comparaisons par boucle sur $\frac{n-2}{2}$ boucles soit $\frac{3n}{2} - 2$ comparaisons.

Dans les deux cas, on obtient bien $\lceil \frac{n}{2} \rceil + n - 2$ comparaisons