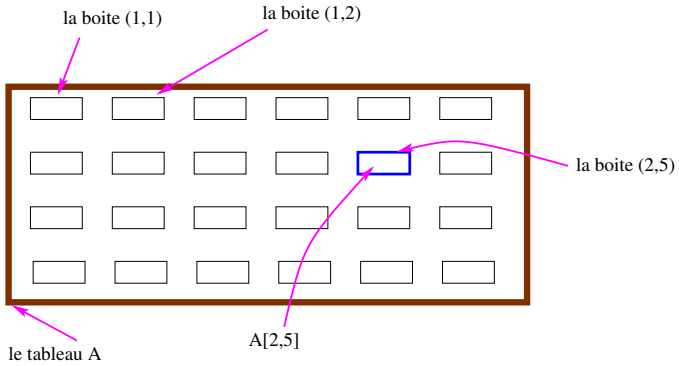

FIG. 1 – tableau à deux dimensions

I. Tableaux et tables

Feuilles de calcul à télécharger Ptab.mw ou Ptab.mws .

1. Principes.

Seul le type de données tableau (`array`) est au programme de l'épreuve de polytechnique.
Il est utile de connaître le type `table` car on génère souvent ce genre de données sans s'en rendre compte. Le type de données `table` est plus général que le type tableau. Dans les deux cas, iI s'agit d'une *famille de boîtes*.
Dans un tableau, les boîtes sont bien rangées en ligne, en rectangle, en rectangles superposés, etc. Les boîtes sont repérées par un nombre (le numéro de la boîte), un couple de nombres, un triplet de nombres et ainsi de suite. On parlera de tableau à une, deux, trois dimensions ou davantage. Un tableau à deux dimensions est analogue à une matrice.

Dans une table, les boîtes ne sont pas particulièrement rangées mais chacune a un nom qui permet d'accéder à son contenu. Un tableau c'est donc une table avec des noms particuliers pour les boîtes.
Dans le vocabulaire informatique, le nom d'une boîte est une *clé*, le contenu d'une boîte est une *valeur*. Dans la syntaxe Maple si `A` est le nom d'une table ou d'un tableau, une clé disons `1/2` doit figurer entre crochets juste à droite du nom de la table.
Une table est proche d'une fonction mathématique dont l'ensemble de définition est fini ; cet ensemble de définition étant l'analogue de l'ensemble des clés de la table. *Attention à l'utilisation de crochets au lieu de parenthèses.*

```
A[1/2];
```

est évalué à la valeur contenue dans la boîte de nom `1/2`.

2. Exemples de commandes pour des tables

```
> restart;
  L:=[a,b,c];
  L[1];L;
  L[2]:=machin;
  L;
  whattype(L);
  L[4]:=truc;
  T:=table();
  T[1]:=machin;T[2]:=truc;
```

Une table est utile pour stocker des données dont le nombre n'est pas connu à l'avance.

```
> whattype(T);
  T;
```

Les tables et les tableaux ont des règles d'évaluation particulières (comme les procédures). Si le nom `T` a été assigné à un objet de ces types, l'évaluation de `T` renvoie seulement le nom `T`. L'appel de `op(T)` renvoie la structure de la table et `op(op(T))` renvoie les composants de la table qui sont la fonction d'indexation (s'il y en a une) et la liste des équations donnant les valeurs que contient la table.

```
> eval(T);op(T);
  dec:=table([0=1,1=2,3=0]);
  dec:=table([(0)=1,(1)=2,(3)=0]);
  dec[1];
  LongOnde:=table([rouge=610,bleu=480,violet=450,vert=520]);
  LongOnde[rouge];
  DS1:=table([Blanchet=20,Gauss=18,Markov=11]);
  DS1[Markov];DS1[titi];DS1;eval(DS1);print(DS1);
```

3. Exemples de commande pour des tableaux

La commande `V := array(1..10)` crée un tableau à une dimension (un 10-uplet) vide. La commande `A := array(1..m,1..n)` crée un tableau à deux dimensions (une matrice) avec m rangs et n colonnes. Les tableaux ont des règles particulières d'évaluation. Lorsque la nom `A` a été assigné à un tableau, l'évaluation de `A` renvoie le nom et `eval(A)` renvoie la structure du tableau.
On peut utiliser la fonction `op` pour détailler la structure. `op(1,eval(A))` renvoie la fonction d'index s'il y en a une sinon rien.
`op(2,eval(A))` renvoie les bornes.
`op(3,eval(A))` renvoie les entrées du tableau sous forme d'équations clé = valeur.

```
> v := array(1..4):
  for i to 3 do v[i] := i^2 od:
  print(v);
  v[2];
  u:=array(1..n);
  #l'accès à un indice qui n'est pas dans la liste
  # renvoie une erreur:
  v[0];
  A := array(1..2,1..2):
  A[1,2] := x:
  A[1,1];
  A[1,2];
  print(A);
  A := array( symmetric, 1..2,1..2, [ [1,x], [x,x^2] ] ):
  op(1,eval(A));
  op(2,eval(A));
  op(3,eval(A));
```

4. Recherches linéaires

Le problème consistant à trouver la plus grande valeur dans un tableau comporte de nombreuses variantes qui se retrouvent très souvent. Il faut bien maîtriser le schéma conventionnel (figure 2) et l'implémentation Maple.

```
> n:=5
  L:=array(1..n,[1,2,5,2,3]);
  max:=L[1];
  for i from 2 to n do
    if L[i]> max then
```

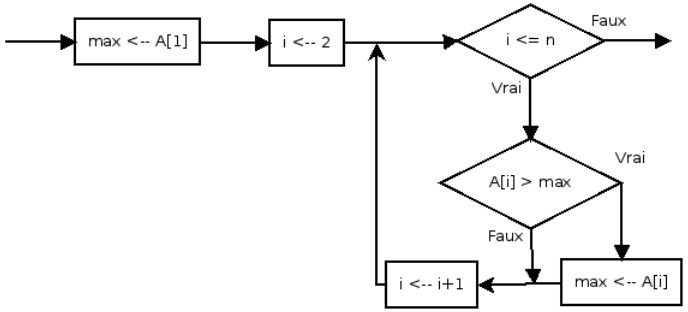

FIG. 2 – recherche de la plus grande valeur

```
      max:=L[i]
    fi;
  od;
  max;
```

Comment doit modifier le schéma conventionnel pour renvoyer la clé de plus grande valeur ?

```
> L:=array([1,2,5,2,3]);
  max:=L[1]; Nmax:=1;
  for i from 2 to n do
    if L[i]> max then
      max:=L[i]; Nmax:=i
    fi;
  od;
  Nmax;
```

Un autre problème fréquent est de trouver la clé pour un objet qui peut être une valeur d'un tableau sans qu'on en soit certain à priori.
On suppose ici que les valeurs d'un tableau `A` (de dimension deux) sont deux à deux distinctes. On se donne un `x` assigné à quelque chose qui est peut être une valeur de `A` . On veut calculer un `cle` tel que

- `cle` soit évalué à 0 si `x` n'est pas une valeur du tableau.
- `A[cle]` soit `x` si c'est une valeur du tableau.

Dans le premier problème, il faut forcément parcourir tout le tableau car on ne peut savoir à l'avance où se situe la plus grande valeur. En revanche dans ce deuxième problème, on peut arrêter le parcours dès que la valeur est trouvée. Il serait plus

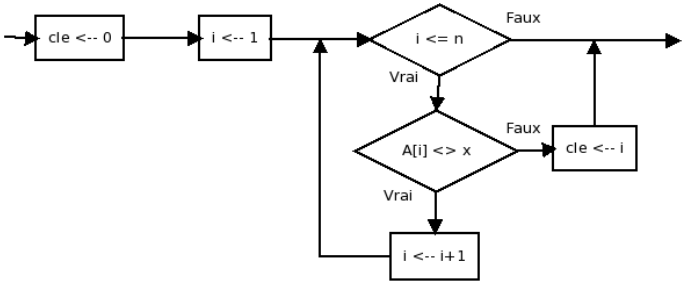

FIG. 3 – recherche d'une valeur

élégant d'implémenter la solution avec une boucle `while` mais il est beaucoup plus facile d'utiliser une *rupture* dans une boucle `for`.