Soient p et n deux entiers naturels non nuls avec $p \leq n$.
On veut former un algorithme permettant de tirer aléatoirement et successivement p nombres distincts au hasard entre 1 et n et les placer dans un tableau. Cela revient à tirer sans remise p boules dans une urne contenant n boules numérotées ou encore à former aléatoirement une application injective de $\{1, \cdots, p\}$ dans $\{1, \cdots, n\}$.
Les variables `p`, `n` (constantes) désignent les nombres p et n.
Lorsque `k` désigne un entier k entre 1 et p, l'appel de la procédure

```
random(1..k)()
```

renvoie un nombre entier aléatoire entre 1 et k.
L'algorithme qui vous est demandé doit vérifier certaines contraintes.

- Vous ne devez appeler la procédure `random` que p fois.
- Vous devez utiliser un tableau auxiliaire `B` indexé de 1 à n. Le début de ce tableau contenant les nombres qui n'ont pas encore été tirés.
- Vous devez renvoyer un tableau indexé de 1 à p.

1. Présenter l'algorithme dans un schéma.
2. Implémenter l'algorithme dans une procédure `tirage` de paramètres `p`, `n`.

Corrigé

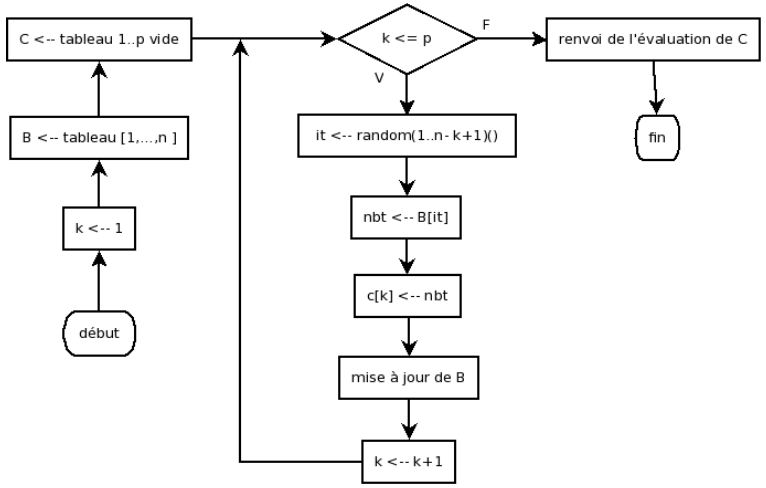

Fig. 1 – Tirage

1. On introduit une variable `k` que l'on va incrémenter de 1 à p. À chaque étape, on tient à jour un tableau `B` indexé de 1 à `n-p+1` et contenant les nombres entre 1 et n qui n'ont pas encore été tirés selon le schéma de la figure 1.
   On peut détailler la mise à jour du tableau auxiliaire.
   Une variable `i` est incrémentée à partir de 1 jusqu'à ce que `B[i]=nbt`, on décale ensuite les valeurs vers la gauche.

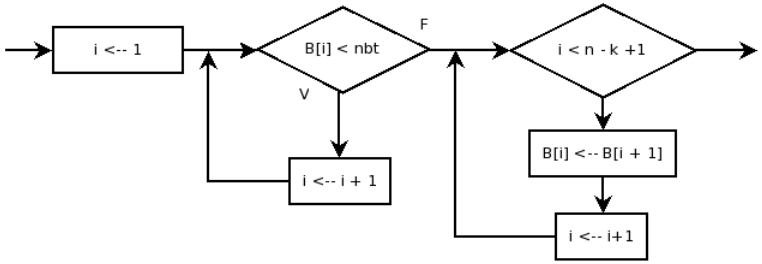

Fig. 2 – Détail de la mise à jour du tableau auxiliaire

Cette création est mise à disposition selon le Contrat
Paternité-Pas d'utilisations commerciale-Partage des Conditions Initiales à l'Identique 2.0 France
disponible en ligne http://creativecommons.org/licenses/by-nc-sa/2.0/fr/

2. L'implémentation en une procédure maple est donnée par le code suivant :

```
tirage := proc(p,n)
 local B, C, k, it, nbt, i  ;
 C:= array(1..p);

 # initialisation de B
 B:= array(1..n);
 for k from 1 to n do
   B[k] := k;
 od;

 # boucle principale
 for k from 1 to p do
   it := rand(1..n-k+1)();
   nbt := B[it];
   C[k] := nbt;

   # mise à jour du B
   i := 1;
   while B[i] < nbt do
     i := i+1;
   od;
   while i < n-k+1 do
     B[i] := B[i+1];
     i := i+1;
   od;
 od;
 return(eval(C));
end proc:
```

Cette création est mise à disposition selon le Contrat
Paternité-Pas d'utilisations commerciale-Partage des Conditions Initiales à l'Identique 2.0 France
disponible en ligne http://creativecommons.org/licenses/by-nc-sa/2.0/fr/