On considère un ensemble fini Σ de cardinal s que l'on appelle *alphabet*. Cet alphabet peut être l'alphabet latin habituel à 26 lettres, mais dans certaines questions on considèrera un alphabet à deux lettres seulement.
Pour tout entier naturel non nul p, une fonction de $\{1, 2, \cdots, p\}$ à valeur dans Σ est appelée un *mot* de p lettres (ou de longueur p) sur Σ. En procédant de gauche à droite, on dira que $x(1)$ est la première lettre du mot x, que $x(2)$ est la deuxième lettre et ainsi de suite.
Par convention, il existe un seul mot de longueur 0 appelé *mot vide* et noté ϵ.

1. Soit p un naturel non nul, quel est le nombre de mots de moins de p lettres sur un alphabet de s lettres ?
2. Dans cette question on prend pour Σ l'alphabet latin de 26 lettres. Un mot de p lettres sur cet alphabet est représenté informatiquement par une liste de p caractères.
   Le tableau suivant définit deux telles listes et les désigne par des variables `x` et `y`. Les longueurs sont désignées par `m` et `n`. Dans toute cette question ces variables désigneront toujours ces objets.

mot	longueur	liste
`bonbon`	`m:=6`	`x:=[b,o,n,b,o,n]`
`quelbonbonbon`	`n:=13`	`y:=[q,u,e,l,b,o,n,b,o,n,b,o,n]`

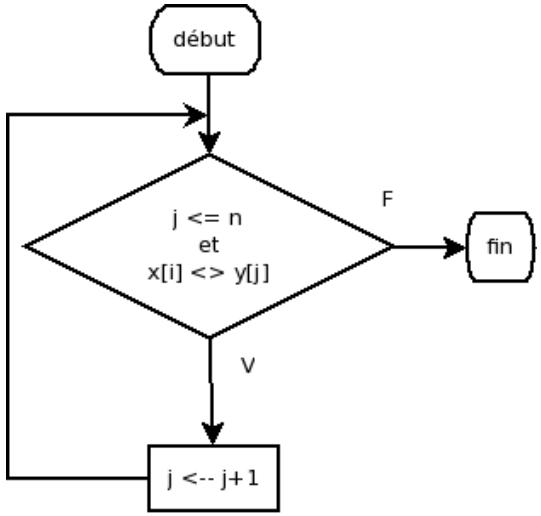

FIG. 1 – Une boucle

a. Dans la figure 1, le symbole `<>` signifie $\neq$. Au moment de l'étape `debut`, les variables `i` et `j` désignent toutes les deux l'entier 1. Que désignent-elles au moment de l'étape `fin` ?

b. Que désignent les variables `i` et `j` au moment de l'étape `fin` du diagramme de la figure 2.

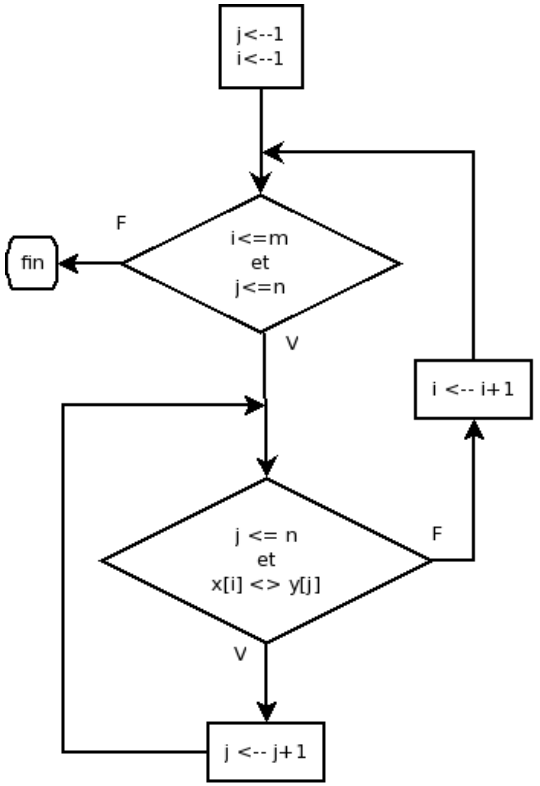

FIG. 2 – Un algorithme

3. Implémenter en syntaxe Maple, l'algorithme présenté dans la figure 2. Il est inutile de définir une procédure.
4. Soit $m \leq n$ deux entiers naturels non nuls. On dira qu'un mot x de m lettres est un *sous-mot* d'un mot y de n lettres si et seulement si il existe une

application φ telle que :

$$\varphi \text{ strictement croissante de } \{1,2,\cdots,m\} \text{ dans } \{1,2,\cdots,n\}$$
$$x = y \circ \varphi$$

(on rappelle que selon la définition du début, un mot est une fonction)

a. Montrer que « bonbon » est un sous-mot de « quelbonbonbon ». On précisera toutes les fonctions φ vérifiant les propriétés caractérisant la définition d'un sous-mot,

b. Définir, en syntaxe Maple, une procédure `est_sous_mot` telle que :
- `x` désignant une liste de `m` caractères associée à un mot x,
- `y` désignant une liste de `n` caractères associée à un mot y,
- l'appel `est_sous_mot(x,m,y,n)` renvoie 1 si x est un sous-mot de y et renvoie 0 sinon.

5. Lorsque x et y sont deux mots sur Σ de longueurs respectives m et n, on désigne par $M(x,y)$ l'ensemble des applications φ vérifiant les conditions de la question 4. On note alors $\mu(x,y)$ le nombre d'éléments de $M(x,y)$.
En particulier $M(x,y)$ est non vide et $\mu(x,y) \neq 0$ si et seulement si $m \leq n$ et x est un sous-mot de y.

a. Lorsque x et y sont des mots de longueur supérieure ou égale à 2, on note x_1 et y_1 les premières lettres de x et de y et on définit des mots x' et y' en supprimant la première lettre.
Par exemple

$$\left.\begin{array}{l} x = bonbon \\ y = quelbonbonbon \end{array}\right\} \Rightarrow \left\{\begin{array}{ll} x_1 = b & x' = onbon \\ y_1 = q & y' = uelbonbonbon \end{array}\right.$$

Lorsque $x_1 = y_1$, montrer que $\mu(x,y) = \mu(x',y') + \mu(x,y')$.
Lorsque $x_1 \neq y_1$, montrer que $\mu(x,y) = \mu(x,y')$.

b. Proposer un algorithme récursif qui prend en entrée des listes `x`, `y`, des longueurs `m`, `n` et qui renvoie la valeur de $\mu(x,y)$.

1. Corrigé

1. Il est important de remarquer qu'un mot n'est pas un ensemble de lettres, l'ordre compte. Un mot est représenté par une fonction. Compter les mots de différentes tailles, c'est donc dénombrer des fonctions dont le dommaine de définition varie. On classe ces fonctions suivant le nombre de lettres en partant de 0 avec le seul mot vide. Comme le nombre de fonctions d'un ensemble à k éléments dans un ensemble à s éléments est s^k (voir un cours de dénombrement), le nombre de mots de moins de p lettres est :

$$1 + s + s^2 + \cdots + s^p = \frac{s^{p+1} - 1}{s - 1}$$

2. a. La figure 1 représente une boucle dans laquelle les variables `i` et `j` représentent des compteurs de lettres respectivement dans le premier mot `x` et dans le deuxième mot `y`.
La variable `i` n'est assignée nulle part dans cette boucle, elle désigne donc toujours 1 à la fin.
En revanche, la boucle incrémente `j` jusqu'à trouver une lettre d'indice `j` égale à la première lettre de `x`. C'est à dire la cinquème lettre `b` de `y` dans le cas particulier considéré. À la fin, la variable `j` désigne donc 5.

b. La figure 2 incorpore la boucle précédente dans une autre qui incrémente le compteur `i` de `x` tant que l'extrémité de l'un des mots n'est pas dépassée.
On parcourt donc le premier `bonbon` de `quelbonbonbon` et on finit avec `i` en dehors de `x` et `j` pointant vers le deuxième `n` de `y`.
En conclusion, à la fin, `i` désigne 7 et `j` désigne 10.

3. En syntaxe Maple, l'algorithme de la 2 se code de la manière suivante :

```
 #pour pouvoir tester
x := ['b','o','n','b','o','n'];
y := ['q','u','e','l','b','o','n','b','o','n','b','o','n'];
m := 6;
n := 13;

i :=1; j:=i;
while i<=m and j<=n do
  while i<=m and x[i]<>y[j] do
    j:=j+1;
  od;
```

```
  i:=i+1;
od:

print(i,j);
```

Les quatre premières et la dernière des lignes ne sont pas formellement demandées par l'énoncé, mais elles sont indispensables si on veut exécuter le code pour le tester.

4. a. Il est bien évident que `bonbon` est un sous-mot de `quelbonbonbon`. Pour le montrer, il suffirait de trouver une injection strictement croissante. En fait, on les forme toutes. Il est commode de visualiser le numéro des lettres dans des tableaux

b	o	n	b	o	n
1	2	3	4	5	6

q	u	e	l	b	o	n	b	o	n	b	o	n
1	2	3	4	5	6	7	8	9	10	11	12	13

On peut alors écrire les 7 injections strictement croissantes de `bonbon` dans `quelbonbonbon` en les présentant dans un tableau.

b	o	n	b	o	n
1	2	3	4	5	6
5	6	7	8	9	10
5	6	7	8	9	13
5	6	7	8	12	13
5	6	7	11	12	13
5	6	10	11	12	13
5	9	10	11	12	13
8	9	10	11	12	13

b. Ce que désigne la variable `j` à la fin de l'algorithme de la 2 permet de déterminer si `x` est un sous-mot de `y`. C'est le cas si et seulement si `j` désigne un nombre inférieur ou égal à n. (ou si `i` désigne un nombre strictement plus grand que m). On peut former la procédure `est_sous_mot` en reprenant le code de la question 4.b.

```
est_sous_mot := proc(x,m,y,n)
 local i,j;
 i :=1; j:=i;
 while i<=m and j<=n do
   while i<=m and x[i]<>y[j] do
     j:=j+1;
   od;
   i:=i+1;
 od:
 if j<= n then
   return(1)
 else
   return(0)
 fi;
end proc:

#test d'appel
x := ['b','o','n','b','o','n'];
y := ['q','u','e','l','b','o','n','b','o','n','b','o','n'];
m := 6;
n := 13;
est_sous_mot(x,m,y,n);
```

5. a. On considère deux mots x et y de longueur respectives m et n.
On suppose d'abord que les deux premières lettres sont égales : $x_1 = y_1$.
Définissons deux parties M_0 et M_1 de $M(x,y)$.

$$\forall \varphi \in M(x,y), \begin{cases} \varphi \in M_0 \Leftrightarrow \varphi(1) = 1 \\ \varphi \in M_1 \Leftrightarrow \varphi(1) \neq 1 \end{cases}$$

Il est clair que M_0 et M_1 partitionnent $M(x,y)$ donc

$$\mu(x,y) = \sharp M_0 + \sharp M_1$$

L'ensemble M_0 est en bijection avec $M(x',y')$ car chaque φ de M_0 est caractérisée par sa restriction à $\{2,\cdots,m\}$ qui réalise une injection de x' dans y'. Donc $\sharp M_0 = \mu(x',y')$.
L'ensemble M_1 est en bijection avec $M(x',y)$ donc $\sharp M_1 = \mu(x',y)$. En effet, pour $\varphi \in M_1$, $\varphi(1) \neq 1$ car $x_1 \neq y_1$ donc $\varphi(i) \neq 1$ pour tous les i entre 1 et m car φ est strictement croissante. Chaque φ de M_1 est caractérisée par sa corestriction à $\{2,\cdots,m\}$ qui réalise une injection de x dans y'.

b. Dans un processus récursif, il faut s'assurer que l'on se ramène à des situations dans lesquelles le processus peut répondre directement sans

s'appeler lui même.

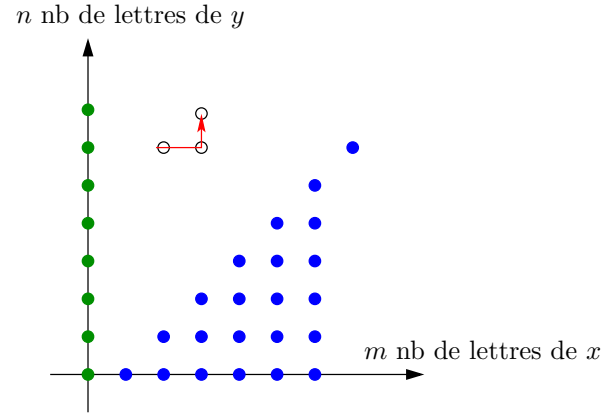

FIG. 3 – Processus récursif

On sait par exemple qu'il n'y a pas d'injection lorsque $m > n$.
La situation est moins claire pour le cas $m = 0$. Il faut chercher à définir arbitrairement $\mu(\epsilon, y)$ pour que les formules récursives soient valables lorsque x contient une seule lettre. On s'aperçoit alors qu'il faut poser

$$\mu(\epsilon, y) = 1 \text{ pour tous les mots } y$$

pour que la formule soit valable avec un mot x de longueur 1. En effet lorsque $x[1] = y[1]$ il y a toujours une injection $1 \to 1$ qui convient et il peut y avoir d'autres injections avec les autres lettres de y. On voit bien sur la figure 3 que si la procédure sait traiter directement les cas $m = 0$ et $n < m$, alors tous les cas seront atteints récursivement. On forme alors le code suivant.

```
 #supprime la première lettre d'une liste
supp := proc(l)
  local i;
  return([seq(l[i],i=2..nops(l))]);
end proc:
#maple accepte de considérer une liste vide

nb_inj := proc(x,m,y,n)
  local xx,yy;
  if m=0 then return(1) fi;
  if n<m then return(0) fi;
  xx:= supp(x);
  yy:= supp(y);
  if x[1]=y[1] then
    return(nb_inj(xx,m-1,yy,n-1)+nb_inj(x,m,yy,n-1));
  else
    return(nb_inj(x,m,yy,n-1));
  fi;
end proc:
```

On a formé une procédure auxiliaire qui supprime le premier mot d'une liste. Le détail de cette procédure n'était pas imposé par l'énoncé.