1.
```
enTetedeSuffixe :=proc(mot,tab,k)
local tMot,tTab,reponse,i;
tMot :=taille(mot);
tTab :=taille(tab);
reponse :=true;
if (k+tMot>tTab+1) then
        reponse :=false
else
        i :=1;
        while (reponse=true) and (i<tMot+1) do
                if (mot[i]<>tab[k+i-1]) then
                        reponse :=false
                fi;
                i :=i+1
        od;
fi;
return reponse;
end;
```

2.
```
rechercherMot :=proc(mot,tab)
local tMot,tTab,reponse,i;
tMot :=taille(mot);
tTab :=taille(tab);
reponse :=false;
if (tMot<tTab+1) then
        i :=1;
        while (i+tMot-1<tTab) and (reponse=false) do
                reponse :=enTetedeSuffixe(mot,tab,i);
                i :=i+1
        od;
fi;
return reponse;
end;
```

3.
```
compterOccurences :=proc(mot,tab)
local iMax,nbreOcc,i;
iMax :=taille(tab)-taille(mot)+1;
nbreOcc :=0;
for i from 1 to iMax do
        if enTetedeSuffixe(mot,tab,i) then
                nbreOcc :=nbreOcc+1
        fi;
od;
return nbreOcc;
end;
```

4.
```
frequenceLettre :=proc(tab)
local frqTab,i,mot;
freqTab :=allouer(26);
for i from 1 to 26 do
        mot :=tab[i..i];
        freqTab[i] :=compterOccurences(mot,tab)
od;
return freqTab;
end;
```

5.
```
afficherFrequenceBigramme :=proc(tab)
local i,mot,tTab;
tTab :=taille(tab);
for i from 1 to tTab-1 do
        mot :=tab[i..i+1];
        afficherMot(tab,i,2);
        print(compterOccurences(mot,tab))
od;
return;
end;
```

6. Solution recursive :

```
comparerSuffixes :=proc(tab,k1,k2)
local tTab,debut1,debut2;
tTab :=taille(tab);
if k2>tTab then
        return k2-k1
elif k2<k1 then
        return comparerSuffixes(tab,k2,k1)
else
        debut1 :=tab[k1];
        debut2 :=tab[k2];
        if debut1<>debut2 then
                return debut1-debut2
        else
                return comparerSuffixes(tab,k1+1,k2+1)
        fi;
fi;
```

Solution non recursive :

```
comparerSuffixes :=proc(tab,k1,k2)
local tTab,ecart,i;
if k2<k1 then
        return comparerSuffixes(tab,k2,k1)
else
        tTab :=taille(tab);
        ecart :=0;
        i :=0;
        while (ecart=0) and (k2+i<tTab+1) do
                ecart :=ecart+tab[k1+i]-tab[k2+i]
                i :=i+1;
        od;
        if (ecart=0) and (k1<k2) then
                ecart :=1
        fi;
fi;
return ecart;
end;
```

7.

```
calculerSuffixes :=proc(tab)
local tTab,tabS,i,j,suff1,suff2;
tTab :=taille(tab);
tabS :=allouer(tTab);
for i from 1 to tTab do
        tabS[i] :=i
od;
for i from tTab downto 2 do
        for j from 1 to i-1 do
                suff1 :=tabS[j];
                suff2 :=tabS[j+1];
                if comparerSuffixes(tab,suff1,suff2)>0 then
                        tabS[j] :=suff2;
                        tabS[j+1] :=suff1
                fi;
        od;
od;
return tabS;
end;
```

8.
   ```
   comparerMotSuffixe :=proc(mot,tab,k)
   local tTab,tMot,ecart,i;
   tTab :=taille(tab);
   tMot :=taille(mot);
   ecart :=0;
   i :=1;
   while (ecart=0) and (k+i-1<tTab+1) and (i<tMot+1) do
       ecart :=ecart+mot[i]-tab[k+i-1]
       i :=i+1;
   od;
   if (ecart=0) and (tMot>tTab-k+1) then
       ecart :=1
   fi;
   return ecart;
   end;
   ```

9.
   ```
   rechercherMot2 :=proc(mot,tab,tabS)
   local tTabS,compMot,presence,compMot2,milieu,tabSdebut,tabSfin;
   tTabS :=taille(tabS);
   if tTabS=1 then
       compMot :=comparerMotSuffixe(mot,tab,tabS[1]);
       if compMot=0 then
           presence :=true
       fi;
   elif tTabS=2 then
       compMot :=comparerMotSuffixe(mot,tab,tabS[1]);
       compMot2 :=comparerMotSuffixe(mot,tab,tabS[2]);
       if compMot*compMot2=0 then
           presence :=true
       fi;
   else
       milieu :=floor((1+tTabS)/2);
       compMot :=comparerMotSuffixe(mot,tab,tabS[milieu]);
       if compMot=0 then
           presence :=true
       elif compMot<0 then
           tabSdebut :=tabS[1..milieu-1];
           presence :=rechercherMot2(mot,tab,tabSdebut)
       else
           tabSfin :=tabS[milieu+1..tTabS];
           presence :=rechercherMot2(mot,tab,tabSfin)
       fi;
   fi;
   return presence;
   end;
   ```
10. <u>rechercheMot :</u>
    $N(tTab) = O(tTab)$.
    <u>rechercheMot2 :</u>
    $N(tTab) = O(\log_2 tTab)$.
    $tTab$ est très grand. La méthode en $\log_2 tTab$ est une bonne méthode.

11.
    ```
    rechercherPremierSuffixe :=proc(mot,tab,tabS)
    local tTabS,compMot,premier,compMot2,milieu,tabSdebut,tabSfin;
    tTabS :=taille(tabS);
    premier :=0;
    if tTabS=1 then
        compMot :=comparerMotSuffixe(mot,tab,tabS[1]);
        if compMot=0 then
            premier :=tabS[1]
        fi;
    elif tTabS=2 then
    ```

```
          compMot :=comparerMotSuffixe(mot,tab,tabS[1]) ;
          compMot2 :=comparerMotSuffixe(mot,tab,tabS[2]) ;
          if compMot=0 then
                    premier :=tabS[1]
          elif compMot2=0 then
                    premier :=tabS[2]
          fi ;
else
          milieu :=floor((1+tTabS)/2) ;
          compMot :=comparerMotSuffixe(mot,tab,tabS[milieu]) ;
          if compMot>0 then
                    tabSfin :=tabS[milieu+1..tTabS] ;
                    premier :=rechercherPremierSuffixe(mot,tab,tabSfin)
          else
                    if compMot=0 then
                              premier :=tabS[milieu]
                    fi ;
                    tabSdebut :=tabS[1..milieu-1] ;
                    premier :=rechercherPremierSuffixe(mot,tab,tabSdebut)
          fi ;
fi ;
return premier ;
end ;
```

12.
```
compterOccurences2 :=proc(mot,tab,tabS)
if rechercherPremierSuffixe(mot,tab,tabS)=0 then
          return 0
else
          return rechercherDernierSuffixe(mot,tab,tabS)-rechercherPremierSuffixe(mot,tab,tabS)+1
fi ;
end ;
```

13.
```
afficheFrequenceKgramme :=proc(tab,tabS,k)
local tTab,i,j,mot,occMot ;
tTab :=taille(tab) ;
i :=1 ;
while i<tTab+1 do
          j :=tabS[i] ;
          if j+k<tTab+1 then
                    mot :=tab[j..j+k-1] ;
                    afficherMot(tab,j,k)
                    occMot :=compterOccurences2(mot,tab,tabS) ;
                    print(occMot) ;
                    i :=i+occMot
          else i :=i+1
          fi ;
od ;
return ;
end ;
```