Option Informatique en Spé MP et MP*
OD : découpage d'un texte, le corrigé

Question 1 Cette première question était assez difficile. Pour ce genre de tâche, qui finalement relève de l'analyse lexicale, nous pouvons nous inspirer des automates finis : à tout instant, nous nous trouvons dans l'un des deux états suivants : en train de lire un mot ; en train de sauter des espaces consécutifs.

```
let décompose_en_mots s =
 let rec lire_mot accu début fin =
  if début < 0
   then (sub_string s (début + 1) (fin - début))::accu
   else if s.[début] <> ‘ ‘
    then lire_mot accu (début - 1) fin
    else sauter ((sub_string s (début + 1) (fin - début))::accu) début
 and sauter accu p =
  if p < 0 then accu else
   if s.[p] = ‘ ‘
    then sauter accu (p - 1)
    else lire_mot accu p p
 in sauter [] (string_length s - 1);;
```

Question 2 Après décomposition de la ligne en mots, nous extrayons de la liste obtenue le plus long préfixe respectant la condition de l'énoncé. L'extraction se poursuit tant que l'on dispose d'une liste de mots non vide, et dont la tête peut tenir dans la ligne en cours de constitution.

```
let prend_une_ligne n s =
 let rec aux accu dispo = function
  | t::q when string_length t <= dispo ->
    aux (t::accu) (dispo - string_length t - 1) q
  | x -> (rev accu,x)
 in aux [] n s;;
```

Question 3 Tant que la liste de mots n'est pas vide, nous en extrayons une ligne.

```
let découpe_en_lignes n s =
 let rec aux accu = function
  | [] -> rev accu
  | x -> let (y,z) = prend_une_ligne n x in aux (y::accu) z
 in aux [] (décompose_en_mots s);;
```

Question 4 Le texte peut être composé si et seulement s'il ne contient aucun mot de longueur supérieure à n.

Question 5 L'énoncé interdit l'emploi de `do_list`, puisque les espaces doivent *séparer* les mots.

```
let rec imprime_ligne = function
 | [] -> ()
 | [x] -> print_string x; print_newline()
 | t::q -> print_string t; print_string " "; imprime_ligne q;;
```

Question 6 Ici, en revanche, nous pouvons utiliser `do_list`.

```
let imprime_texte n s =
 do_list imprime_ligne (découpe_en_lignes n s);;
```

Question 7 La somme des longueurs des inserts est au moins égale à $(q-1)k$, et elle est strictement inférieure à $(q-1)(k+1)$ de par la définition de k. Par ailleurs, la longueur n de la ligne est la somme des longueurs des mots et des inserts qui la composent. D'où la relation $(q-1)k \leqslant n - v < (q-1)(k+1)$ dont on déduit $k = \left\lfloor \dfrac{n-v}{q-1} \right\rfloor$.

Question 8 Appliquons les formules précédentes pour calculer v, q, k. Décidons de construire une liste formée de p inserts de largeur k, suivis de $q-1-p$ inserts de largeur $k+1$; p est défini par la condition $pk+(q-1-p)(k+1)=n-v$ soit $p=(q-1)(k+1)-n+v$. La fonction `aux` est spécifiée comme suit : `aux p k` construit une liste de p entiers égaux à k. Un `map` adroit transforme une telle liste en liste d'inserts. Nous réglons au niveau de cette fonction le cas d'une ligne réduite à un mot : il est clair que la justification ne peut pas être réalisée. D'ailleurs, les calculs précédents impliqueraient une division par zéro.

```
let répartit_espaces n lm =
 let v = it_list (prefix +) 0 (map string_length lm)
 and q = list_length lm in
 if q = 1 then [] else
 let k = (n-v)/(q-1) in
 let p = (k+1)*(q-1)-n+v in
 let aux nb lg = list_of_vect (make_vect nb lg) in
  map (fun i -> make_string i `*`) ((aux p k) @ (aux (q-1-p) (k+1)));;
```

Question 9 Commençons par fusionner la liste des mots avec la liste des inserts, en notant bien que leurs longueurs ne sont pas égales :

```
let intègre_espaces lm le =
 let rec aux accu = function
  | (m::q,m'::q') -> aux (m'::m::accu) (q,q')
  | ([m],[]) -> rev(m::accu)
  | (_,_) -> failwith "trop ou pas assez d'espaces"
 in aux [] (lm,le);;
```

Il ne reste plus qu'à enchaîner les deux fonctions que nous venons d'écrire :

```
let justifie_ligne n s =
 intègre_espaces s (répartit_espaces n s);;
```

Question 10 Confions à une fonction `imprime_ligne_justifiée` la tâche d'imprimer une ligne justifiée. L'emploi de `do_list` allège notablement les écritures.

```
let rec imprime_ligne_justifiée l =
 do_list print_string l;
 print_newline();;

let imprime_texte_justifié n s =
 do_list imprime_ligne_justifiée
  (map (justifie_ligne n) (découpe_en_lignes n s));;
```

Question 11 Une condition suffisante pour que chaque ligne (sauf peut-être la dernière, voir question suivante) soit justifiée est qu'elle contienne au moins deux mots ; ceci sera certainement réalisé si la somme des longueurs de deux mots consécutifs est toujours strictement inférieure à n.

Question 12 La longueur occupée par les mots de la dernière ligne sera en moyenne $n/2$. Les inserts seront donc exagérés ; la solution la plus simple consiste à appliquer la justfication à toutes les lignes, sauf à la dernière.

► La justification d'un texte est, bien entendu, plus compliquée que ce problème ne le laisse entrevoir. Par exemple, si nous disposons de sept espaces pour quatre inserts, ceux-ci seront égaux à un ou deux espaces : leur variation au sein de la ligne sera désagréable à l'œil. En pratique, nous utiliserons un *quantum* d'insert beaucoup plus petit que l'espace (notion qu'il serait d'ailleurs difficile de définir). Nous devons également faire en sorte que les inserts ne varient pas trop d'une ligne à l'autre (ce qui interdit donc l'emploi d'un algorithme glouton comme celui que l'on a envisagé). Au besoin, nous appliquerons une *césure* si nous ne trouvons pas de point de coupure acceptable entre deux mots : cette césure devra respecter de nombreuses règles, qui d'ailleurs diffèrent d'une langue à l'autre même lorsque celles-ci sont basées sur le même système d'écriture (alphabet romain).

► Vous vous convaincrez de la complexité du problème en lisant un journal : la composition en colonnes étroites rend la justification très difficile. Signalons un dernier point esthétique qui, à ma connaissance, n'est pris en compte par aucun logiciel à l'heure actuelle : ce que les typographes appellent *cheminées*, qui sont des espaces entre mots, qu'un hasard malencontreux a alignés verticalement.

FIN