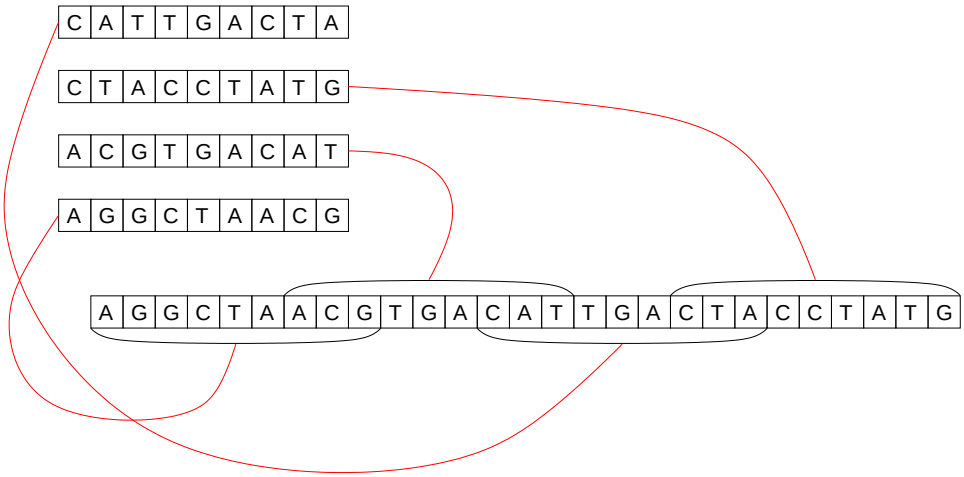
Algorithmique (LI 0436)
Devoir sur table n° 2 & Examen final
Durée : 2 heures
Aucun document autorisé
Questions : (1 et (2 ou 3) et 4)

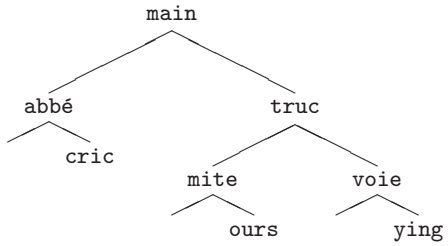
1. **Recherche de facteurs** On suppose qu'on dispose de n chaînes de caractères, toutes de la même longueur l, et qui résultent d'un "découpage" (avec chevauchement) d'une chaîne initiale, comme illustré dans la figure suivante (où $l = 9$ et $n = 4$). On suppose que le chevauchement est de longueur fixe, k (ici $k = 3$). Le problème général qu'on se pose est celui de la reconstitution de la chaîne initiale à partir des n chaînes[4].



   (a) Proposer une formulation générale du problème (en langue naturelle) en termes de préfixe et de suffixe.
   (b) On fait l'hypothèse (très simplificatrice) qu'il y a exactement une solution au problème (il n'y a qu'un moyen d'agencer les n sous-chaînes avec un recouvrement de longueur k). En supposant de plus qu'on dispose de fonctions qui permettent de manipuler les préfixes et les suffixes d'une chaîne donnée, proposer un algorithme qui, étant données n chaînes de longueur l, et étant donné le recouvrement k, donne l'ordre dans lequel il faut placer les n sous-chaînes pour reconstituer la chaîne initiale.
   (c) Quelle est la complexité ($O()$) de cet algorithme ? On ne demande pas de démonstration.
   (d) [Bonus] Quelle est la forme générale de l'algorithme si l'on supprime certaines des simplifications précédentes : il peut ne pas y avoir de solution, ou plusieurs solutions, la longueur du recouvrement peut varier (on maintient l'hypothèse que les chaînes font toutes la même longueur).

[4] Il s'agit d'une version très simplifée d'un des problèmes de séquencement de l'ADN : des enzymes découpent un grand nombre de brins identiques en morceaux, et il s'agit de reconsituer le brin à partir des morceaux. En vrai, ni l, ni k, ni n ne sont fixes.

2. **Arbre binaire** Considérons l'*arbre binaire de recherche* suivant (on suppose qu'on utilise l'ordre lexicographique habituel).



   (a) Rappeler le principe d'organisation des arbres binaires de recherche
   (b) Dessiner l'arbre qui résulterait de l'insertion de l'élément `acre`
   (c) Dessiner l'arbre qui résulterait de la suppression de l'élément `truc`

3. **Piles** Considérons l'algorithme suivant[5]

```
boolean fonction_inconnue (pile p, int x)
{
  if (p.pile_vide())
     return false ;
  if (p.depiler() == x)
     return true ;
  return fonction_inconnue(p, x) ;
}
```

   (a) Détailler le déroulement du programme (« faire tourner ») après l'appel `fonction_inconnue(p, x)` avec `x='f'` et `p` =

a
g
a
f
o
i
u

   (b) Que fait cet algorithme ?
   (c) Proposer une version itérative de l'algorithme.

4. **Parcours d'arbre** On suppose qu'on manipule un arbre "généalogique", qui comprend pour un individu donné, cet individu, ses parents, grands-parents et arrières-grand-parents. Chaque nœud est un enregistrement comportant le nom, le prénom, et l'année de naissance de l'individu. On suppose que l'arbre est complet.

   (a) Combien de nœuds cet arbre contient-il ? Même questions si l'on va jusqu'à la i^{e} génération (parents= 1^{re} gén., grand-parents= 2^{e}, etc.)
   (b) Ecrire un algorithme qui affiche la liste des ancêtres nés avant une date donnée, génération par génération (en commençant par les plus jeunes). Quel type de parcours d'arbre vous paraît le plus adapté ?
   (c) Ecrire un algorithme qui, étant donné un individu (nom, prénom, date), répond vrai ou faux selon que l'individu figure dans l'arbre.
   (d) Ecrire un algorithme qui, pour un arbre généalogique limité à la i^{e} génération, fait la liste des individus qui apparaissent plusieurs fois dans l'arbre. (On suppose résolue la question précédente.)

[5] On suppose que la primitive `depiler()` (1) renvoie la valeur du sommet de pile, et (2) supprime cette valeur de la pile.