DS6-Durée 3h

07 mai

1 Logique (45min - 8pts)

1 Soit la formule propositionnelle A définie comme $((\neg p \vee \neg q \vee r) \wedge (p \Rightarrow q)) \Rightarrow (q \Rightarrow r)$

1. Donner la représentation syntaxique de la formule A sous forme d'arbre.
2. Donner la table de vérité de la formule A.
3. Donner une forme normale conjonctive et une forme normale disjonctive équivalentes à la formule A.
4. Cette formule est-elle une tautologie ? justifier votre réponse.
5. Cette formule est-elle satisfiable ? si oui donner un modèle.
6. Donner le nombre total de modèles de la formule A.

2 On considère des formules propositionnelles construites uniquement à partir des formules atomiques $\bot$ et $\top$ et pour chaque variable propositionnelle i, un opérateur binaire $IF_i(p,q)$ qui se comporte comme la formule p si i est vrai et comme la formule q sinon. On note IP l'ensemble des formules ainsi construites.

Si x, y et z sont des variables propositionnelles, on peut traduire la formule propositionnelle $x \vee (y \wedge \neg z)$ par une formule de IP (notée A dans la suite) : $A = IF_x(\top, IF_y(IF_z(\bot, \top), \bot))$.

Cette formule est vraie exactement lorsque x est vrai ou lorsque y est vrai et z est faux.

Soit v une valuation et p une formule de IP, on définit la fonction $valip(v,p)$ qui représente la valeur de vérité de la formule p pour la valuation v.

Cette définition est faite de manière récursive sur la structure de p :

— $valip(v, \top) = V$

— $valip(v, \bot) = F$

— $valip(v, IF_j(p,q)) = valip(v,p)$ si $v(j) = V$ et $valip(v,q)$ sinon

On vérifie que pour toute valuation v, la valeur de vérité de $x \vee (y \wedge \neg z)$ est bien la même que celle de sa traduction A.

1. Suivant le même principe, donner une formule de IP qui corresponde à la traduction de la formule $y \Rightarrow (x \vee \neg z)$. Justifier que la traduction a la même table de vérité que la formule initiale.
2. Soit p une formule de IP et i une variable propositionnelle, donner une expression plus simple pour $IF_i(p,p)$ qui est logiquement équivalente mais qui ne dépend pas de i.
3. Donner une formule de IP qui soit équivalente à la négation de A.
4. Construire une fonction neg qui étant donnée une formule p de IP calcule une nouvelle formule de IP dont la valeur est la négation de la valeur de p dans toute valuation v.

   On définira la fonction par des équations récursives puis le code en OCAML en utilisant le type suivant :

   ```
   type iformule = Top | Bottom | If of int * iformule*iformule;;
   ```
5. On remarque qu'il est inutile d'avoir deux nœuds IF_i faisant référence à la même variable i emboités dans la même formule. Soit la formule $C = IF_x(IF_x(\top, \bot), IF_y(IF_z(\bot, \top), IF_y(\top, \bot)))$ construire la table de vérite de C et donner une forme simplifiée de C toujours dans IP dans laquelle il n'y a pas deux nœuds IF_i sur la même variable qui sont emboités. C'est-à-dire que pour toute sous-formule $IF_i(p,q)$, la variable i n'apparaît ni dans p, ni dans q.
6. On se donne un ordre strict sur les variables propositionnelles. Dans nos exemples, on prendra $x < y < z$. On s'intéresse maintenant à des formules de IP ordonnées, c'est-à-dire que pour toute sous-formule $IF_i(p,q)$, les variables j qui sont dans p ou dans q sont toutes strictement plus grandes que i.
   (a) Dire si les formules suivantes sont ordonnées ou non :
      i. $IF_x(\top, IF_y(IF_z(\bot, \top), \bot))$
      ii. $IF_x(\top, IF_x(IF_y(\bot, \top), \bot))$
      iii. $IF_x(\top, IF_z(IF_y(\bot, \top), \bot))$

   (b) Soient les formules de IP ordonnées $P = IF_x(IF_z(\bot, \top), \top)$ et $Q = IF_x(IF_y(\top, \bot), IF_z(\top, \bot))$. Donner une formule de IP ordonnée qui soit équivalente à la conjonction de P et de Q.

(c) Construire une fonction `conj` qui étant données deux formules p et q de IP ordonnées, calcule une nouvelle formule de IP ordonnée dont la valeur est la conjonction des valeurs de p et de q dans toute valuation v. On définira la fonction par des équations récursives. On ne demande d'écrire le code mais seulement de définir les relations par induction.

(d) On suppose maintenant que l'on prend une formule de IP ordonnée. Comment peut-on tester que cette formule est insatisfiable ? Dire pourquoi l'hypothèse que la formule est ordonnée a de l'importance.

2 Algorithmes gloutons (30min - 8 pts)

2.1 Reconnaissance des situations du cours

3 Un cinéma possède s salles de cinéma. Chaque semaine, le cinéma propose une liste de films à voir. Chaque film correspond à un nombre fini de séances. Chaque séance se déroule selon un horaire (intervalle de temps) précis. Les films n'ont pas tous la même durée. On précise qu'il est impossible de changer les horaires des séances. Un étudiant qui a une journée de libre veut voir le maximum de films pendant cette journée (il peut voir un même film plusieurs fois). Ce problème peut être résolu par un algorithme glouton.

Exposer un algorithme glouton qui trouve une solution optimale à ce problème en identifiant une situation vue en classe. On ne demande pas de prouver que l'algorithme est optimal mais simplement de précisémment établir le lien avec un résultat du cours.

4 On considère une rue sur laquelle se trouve n maisons, tel que la maison i se trouve au kilomètre $k(i)$. On veut construire une ligne de bus sur cette rue, et placer les arrêts de bus de manière à ce que pour chaque maison il y ait un arrêt à moins de d km. On cherche à minimiser le nombre d'arrêts.

Proposer un algorithme **glouton** résolvant le problème.

Comme dans la question précédente, en mentionnant un résultat du cours, justifier que votre algorithme est effectivement optimal.

2.2 Une démonstration

1. Vous souhaitez vous rendre de Liège à Brest en scooter.
2. Votre réservoir vous permet de rouler R km.
3. Vous connaissez la liste des pompes à essence disponibles sur la route, donnée sous la forme d'une liste $S = [d_1, d_2, ..., d_k]$ où chaque d_i donne la distance qui le sépare de son précédent : la première station, notée s_1 est à d_1 kilomètres du départ, la deuxième, notée s_2 à d_2 kilomètres de s_0 etc.
4. On suppose $d_i \leq R$ pour chaque $i \in \{1, \ldots, k\}$.
5. On souhaite faire le moins d'arrêts possibles.

Nous allons considérer l'algorithme glouton suivant :
On s'arrête à la station s_i telle que i est le plus grand indice tel que $\sum_{j=1}^{i} d_j \leq R$.
Ensuite, si on s'est arrêté à l'étape précédente dans une station i_0 alors on s'arrête à l'étape suivante dans la station i telle que i est le plus grand indice tel que $\sum_{j=i_0+1}^{i} d_j \leq R$.

5 Pour un réservoir de $250km$, appliquer l'algorithme avec la liste $[120, 142, 90, 70, 130, 150, 84, 25, 110]$.

6 Montrer qu'il existe une liste d'arrêts de taille minimale telle que le premier arrêt ait lieu dans la station i la plus éloignée possible du départ parmi celles qui se trouvent à moins de R kilomètres du départ.

7 Montrer que si on considère une liste d'arrêts de taille minimale telle que le premier arrêt ait lieu dans la station i la plus éloignée possible du départ pami celles qui se trouvent à moins de R kilomètres du départ, alors la liste privée de ce premier arrêt est une solution optimale au problème dont l'entrée serait $[d_{i+1}, \ldots, d_k]$.

8 Montrer par récurrence sur le nombre de stations sur le chemin que l'algorithme proposé trouve bien une solution optimale.

3 Algorithmique du texte (30min - 8pts)

9 Ecrire en C une fonction naïve `int search (char* m, char* s, int n, int p)` qui étant donné deux chaînes de caractères s et m de longueurs respectives n et p avec $p \leq n$ renvoie le plus petit indice d'une occurence de m dans s si elle existe et -1 sinon.

10 Compresser puis décompresser le message $AAAABBBAABBABB$ avec la méthode de Liv Zempel Welch. Expliquer les étapes.

On rappelle que le code ASCII de la lettre A vaut 65.

11 Nous présentons ici une méthode de compression qui donne, comme pour Huffman, un code de longueur variable pour chaque lettre de l'alphabet.

Voici le principe :

1. On ordonne les caractères suivant l'ordre croissant de leur fréquence.
2. On cherche ensuite à partitionner le tableau de caractères en deux sous-tableaux de manière à ce que les sommes des fréquences des caractères de chacun de ces sous-tableaux soient égales ou "le plus proches possible". Tous les codes des caractères du premier sous-tableau commencent par un 0 et ceux du deuxième sous-tableau par un 1.
3. Puis on continue à partitionner chacun des deux sous-tableaux en deux, en rajoutant un 0 aux codes des caractères des premiers sous-tableaux et en rajoutant un 1 aux codes des caractères des deuxièmes sous-tableaux. On continue ainsi les partitions jusqu'à ce qu'on aboutisse à un sous-tableau réduit à un seul caractère.

   Plus précisemment, si un sous-tableau noté $G_{d\to f}$ est l'ensemble des composantes du tableau dont les indices sont compris entre les deux nombres entiers d et f, si $S_{d\to f}$, désigne la somme des fréquences affectées à ces composantes, on cherche la partition $G_{d\to k}, G_{k+1\to,d}$ du sous-tableau $G_{d\to f}$, telle que $|S_{d\to k} - S_{k+1\to f}|$, soit minimum.

Ecrire le code des six premières lettres de l'alphabet dans le cas où le texte à compresser est `ABABABABCDBEBFGEEG` obtenu avec la méthode décrite ci-dessus.

Donner un code obtenu par l'algorithme de Huffman pour le même texte. Expliquer clairement les étapes vous permettant de l'obtenir.

4 Répétitions : à traiter en OCAML (1h15 - 8pts)

Cette partie traite de l'identification de sous-mots répétés dans un mot formé sur un alphabet A donné. Ce problème est particulièrement prégnant en génétique où il s'agit de repérer des répétitions dans des brins d'ADN, ce qui permet de comprendre sa structure. Dans ce cas, on prend simplement $A = \{g, t, a, c\}$.

Soit A un alphabet et soit $m = m_0 m_1 ... m_{n-1}$ un mot de longueur n formé sur A ($m_i \in A$ pour tout i).

Un mot sera représenté par une chaîne de caractères. On extraira d'un mot m le caractère de rang i à l'aide d'une fonction `caractère`(m, i), supposée fournie. Le rang du premier caractère est 0.

4.1 Une famille de relations d'équivalence

Soit $k \in \{1, 2, \cdots, n\}$ et soient $i, j \in \{0, 1, \cdots, n-k\}$. Les rangs i et j sont dits k-équivalents si

$$x_i x_{i+1} \cdots x_{i+k-1} = x_j x_{j+1} \cdots x_{j+k-1}$$

c'est-à-dire si les sous-mots de longueur k commençant aux indices i et j sont identiques.

Ce fait est noté $\langle i, E_k, j\rangle$.

12 Montrer que $\langle i, E_{a+b}, j\rangle$ peut se déduire de deux expressions simples construites sur E_a et E_b.

13 Montrer que $\langle i, E_{a+b}, j\rangle$ peut se déduire de deux expressions simples construites sur E_a, lorsque $b \leqslant a$. Ce résultat sera utilisé dans la suite.

4.2 Manipulation de piles

En Caml, on définit le type `pile` (d'entiers) par :

```
type pile = {
  mutable elements: int list;
};;
```

Le type `pile` possède trois opérations :

— `empile`, qui ajoute un élément au sommet de la pile ;
— `depile`, qui rend l'élément situé au sommet de la pile et le retire de la pile ;
— `vide`, qui rend vrai si et seulement si la pile est vide.

14 Écrire les fonctions `empile`, `depile` et `vide`.

4.3 Calcul des classes d'équivalence de E_1

On considère à partir de maintenant que A est composé des 26 lettres de l'alphabet. On suppose que l'on dispose d'une fonction `numero_lettre` qui à chaque lettre (caractère) associe son rang, compris entre 0 et 25.

On rappelle que si $i \in \{0, 1, \cdots, n-1\}$, la classe d'équivalence de i pour E_1 est le sous-ensemble X_i des entiers $j \in \{0, 1, \cdots, n-1\}$ tels que $\langle i, E_1, j\rangle$. Une classe d'équivalence de E_1 est une partie X de $\{0, 1, \cdots, n-1\}$ pour laquelle il existe $i \in \{0, 1, \cdots, n-1\}$ tel que $X = X_i$.

15 Pour $i, j \in \{0, 1, \cdots, n-1\}$, que signifie $\langle i, E_1, j\rangle$?

16 On note p le nombre de classes d'équivalence de E_1 ; les classes d'équivalence de E_1 seront numérotées par des entiers entre 0 et $p-1$. Elles seront représentées par un tableau (`array`) en OCaml de longueur n (indices numérotés de 0 à $n-1$), dans lequel l'élément de rang i sera le numéro de la classe d'équivalence de i. *Dans la suite, on utilisera le terme vecteur pour désigner un tableau de OCaml.*

Exemple : pour le mot « `abracadabra` », le vecteur $[0, 1, 2, 0, 3, 0, 4, 0, 1, 2, 0]$ donne les classes d'équivalence de E_1. Écrire une fonction `construit_E1` qui pour un mot m donné en argument, construit le vecteur représentant les classes d'équivalence de E_1 ; cette fonction utilisera un seul vecteur de longueur n et aucune autre structure annexe de type liste ou pile.

17 Donner la complexité de cette fonction en termes du nombre de lectures et d'écritures dans un vecteur.

18 On autorise maintenant la fonction `construit_E1` à utiliser un vecteur supplémentaire. Réécrire `construit_E1` de sorte qu'elle ait une complexité en $O(n)$.

4.4 Classes d'équivalence des E_{a+b}

On cherche dans cette section à construire les classes d'équivalence de E_{a+b}, connaissant les classes d'équivalence de E_a et sachant que $b \leqslant a$.

Comme précédemment, les p classes d'équivalence de E_a sont numérotées de 0 à $p-1$.

Soit v le vecteur donnant les numéros des classes d'équivalence de E_a ; on rappelle que v est un vecteur à n composantes. Le but de cette section est de construire w le vecteur à n composantes donnant les numéros des classes d'équivalence de E_{a+b}.

Pour construire les classes d'équivalence de E_{a+b}, on commence par regrouper les indices des lettres de m en sous-ensembles, de façon que deux éléments d, e quelconques d'un tel sous-ensemble vérifient $\langle d+b, E_a, e+b\rangle$. On remarque que seuls sont concernés les indices d vérifiant $d + b \leqslant n - a$.

19 Combien y aura-t-il de sous-ensembles au maximum ?

20 On souhaite construire ces différents sous-ensembles. On utilisera pour cela un vecteur de piles, chaque pile correspondant à un sous-ensemble. Expliquer comment on peut simplement construire ce vecteur de piles. *On ne demande pas d'écrire de programme à ce stade.* Certaines piles peuvent être vides.

21 Une pile donnée contient donc des indices tels que deux valeurs successives d et e vérifient $\langle d+b, E_a, e+b\rangle$. Si de plus ces deux valeurs successives vérifient $\langle d, E_a, e\rangle$, que peut-on en déduire d'utile pour construire les classes de E_{a+b} ?

22 Cependant, a priori la méthode de construction des piles de la question **20** ne garantit pas que tous les éléments d'une pile qui appartiennent à une même classe d'équivalence de E_a seront placés successivement. On cherche donc à modifier la solution de la question **20** pour ajouter cette propriété.

Nous allons commencer par créer un premier vecteur de piles (intermédiaire), contenant tous les éléments de v, et tel que si deux éléments d, e appartiennent à la même pile, alors $\langle d, E_a, e\rangle$. Écrire une fonction `empileClassesEa` qui fournit ce résultat, en fonction du vecteur v et du nombre p de classes d'équivalence de E_a.

23 À partir de ce vecteur de piles intermédiaire, modifier la méthode proposée à la question **20** de façon à obtenir un vecteur de piles final dans lequel les éléments d'une pile qui appartiennent à une même classe d'équivalence de E_a sont placés successivement. La solution sera réalisée sous forme d'une fonction `sousEnsembles` prenant en paramètre le vecteur v, vecteur de piles intermédiaire, et la valeur b.

24 En utilisant la propriété remarquée dans la question **21**, écrire une fonction `classesAplusB` qui calcule les classes d'équivalence de E_{a+b}. Cette fonction rendra le vecteur w et prendra en paramètre le vecteur v, le vecteur (final) de piles issu de la fonction de la question précédente et la valeur b.

25 Écrire finalement la fonction `classesAversAplusB` qui calcule le vecteur w à partir du vecteur v.

4.5 Construction des classes d'équivalence des E_k, $k > 1$

26 Soit q un entier naturel tel que 2^q soit inférieur ou égal à n. Montrer que l'on peut construire les classes d'équivalence de E_{2^q} en appliquant q fois `classesAversAplusB`.

27 Écrire une fonction `classesEq` qui, à un mot m de longueur n et à un entier quelconque $k \leqslant n$, associe les classes d'équivalence de E_k.

4.6 Sous-mots répétés

28 Expliquer comment `classesEq` permet d'obtenir la liste des sous-mots répétés d'une certaine longueur ℓ.

On cherche maintenant à trouver l'entier ℓ *maximum* pour lequel il y a des sous-mots répétés.

29 Quel test simple sur le résultat de `classesEq` permet de savoir si un mot m contient des sous-mots répétés de longueur ℓ ?

30 Nous avons vu en **26** que E_{2^q} peut être construit avec q appels à `classesAversAplusB`. En vertu de ce résultat et de la question précédente, on doit donc pouvoir déterminer le plus petit entier q tel qu'il n'existe pas de sous-mots répétés de longueur 2^q dans m.

Comment alors déterminer en un minimum d'opérations le plus grand entier ℓ ($\ell < 2^q$) tel que m contienne des sous-mots répétés de longueur ℓ ?

On ne demande pas de programmer cette opération.