C36212

3è ANNÉE

INFORMATIQUE II

DUREE : 5 HEURES

Aucun document n'est autorisé

L'usage de toute calculatrice est interdit

Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.

Le sujet comporte 7 pages

Le sujet porte sur la manipulation de formules booléennes. Il comporte deux problèmes indépendants. Les questions peuvent être résolues en admettant les résultats des questions précédentes. L'usage de la calculatrice est interdit.

Quelques rappels sur les formules booléennes

On se donne un ensemble $\mathcal{V} = \{a, b, c, \ldots, a_1, b_1, c_1, \ldots\}$ infini dénombrable de variables propositionnelles.

On construit les formules booléennes à partir des variables de $\mathcal{V}$, des constantes 0 (faux) et 1 (vrai), et des connecteurs usuels, $\neg$ (non), $\vee$ (ou), $\wedge$ (et). Les connecteurs $\vee$ et $\wedge$ peuvent être n-aires. On note $var(F)$ l'ensemble des variables apparaissant dans une formule F.

Une affectation est une fonction de $\mathcal{V}$ dans $\{0,1\}$. On étend la notion d'affectation aux formules en utilisant les tables de vérités. Une affectation σ satisfait la formule F si $\sigma(F) = 1$; on dit que σ est une solution de F. On dit que σ falsifie F si $\sigma(F) = 0$. Une formule F est satisfiable s'il existe une affectation σ telle que $\sigma(F) = 1$. Elle est insatisfiable sinon. Deux formules F et G (ne portant pas forcément sur les mêmes ensembles de variables) sont dites équivalentes pour la satisfiabilité si on a (F satisfiable) $\Leftrightarrow$ (G satisfiable).

Conseils pour la rédaction des algorithmes

Les algorithmes peuvent être donnés dans la notation de votre choix, pourvu qu'elle soit suffisamment précise (vous pourrez vous inspirer de la rédaction des deux algorithmes donnés dans le sujet). Vous pourrez supposer que les types de données utilisés sont munis des primitives dont vous avez besoin sans que vous ayez à les redéfinir : on demande simplement que leurs spécifications soient rappelées clairement, et le cas échéant, de justifier leur complexité.

En particulier, le type de données « liste » sera supposé muni de toutes les primitives nécessaires à la manipulation d'ensembles d'objets (insertion et retrait d'un élément, copie d'un ensemble, union et intersection de deux ensembles...), ce qui ne vous dispense pas d'en donner (et d'en justifier) la complexité.

1 Formules CNF

Un littéral est soit une variable (littéral positif), soit sa négation (littéral négatif). Par souci de simplicité, on note $\neg p$, l'opposé d'un littéral p (sachant que $\neg\neg p$ et p ont les mêmes tables de vérité). Une clause est une disjonction de littéraux aussi manipulée comme l'ensemble de ses littéraux. Une formule sous forme normale conjonctive (CNF) est une conjonction de clauses aussi manipulée comme l'ensemble de ses clauses. On rappelle que toute formule admet une formule CNF équivalente (ayant les mêmes variables et étant satisfaite par les mêmes affectations). On remarque qu'une CNF contenant la clause vide est insatisfiable (la clause vide étant assimilée à la constante 0).

Une formule est dite k-CNF, $k \geq 1$, si c'est une formule CNF dont toutes les clauses comportent au plus k littéraux. Elle est dite exact-k-CNF si toutes ses clauses comportent exactement k littéraux. Finalement, elle est dite k,j-CNF, $k \geq 1, j \geq 0$ (respectivement exact-k,j-CNF) si c'est une formule k-CNF (respectivement exact-k-CNF) et si aucune de ses clauses ne comporte plus de j littéraux positifs. Par convention, on définit la taille $|C|$ d'une clause C comme son nombre de littéraux et la taille $|F|$ d'une formule CNF F comme la somme de la taille de ses clauses.

Soient deux clauses C et D. On dira que la résolvante R de C et D existe, si d'une part il existe un littéral p et des clauses C' et D' tels que $C = \{p\} \cup C'$ et $D = \{\neg p\} \cup D'$, d'autre part aucun autre littéral n'apparait sous des formes opposées dans C' et D'. Si R existe, $R = C' \cup D'$.

Le problème SAT consiste à déterminer si un ensemble de clauses est satisfiable. On rappelle que le problème SAT est NP-complet.

Question 1 :

- *En introduisant une nouvelle variable, donner une formule exact-3-SAT équivalente pour la satisfiabilité à la formule* $(p \Leftrightarrow \neg q)$.
- *Soient F et G deux formules CNF et C et D deux clauses telles que $F = \{C, D\} \cup G$ et soit R la résolvante de C et D (on suppose qu'elle existe). Montrer que F est satisfiable si et seulement si $F \cup \{R\}$ est satisfiable.*

Question 2 : *Montrer que tout ensemble de clauses peut être récrit en une formule exact-3-SAT équivalente pour la satisfiabilité.*

Question 3 : *Quelle est la complexité du problème exact-3,2-SAT ?*

Question 4 : *Soient n et k deux entiers strictement positifs, $n > k$, dénombrer le nombre de clauses de taille inférieure ou égale à k que l'on peut construire sur un ensemble de n variables.*

On note $\setminus$ la différence ensembliste. On considère l'algorithme (dit de saturation) d'un ensemble F de clauses suivant :

SATURATION :
Donnée : une formule CNF F
Résultat : une formule CNF G
Variables locales : C_1, C_2, D, R : clauses ; FINI : booléen.
début :
  $G \leftarrow F$
  FINI $\leftarrow$ faux
  tant que non FINI **faire**
    FINI $\leftarrow$ vrai
    pour tous les couples de clauses C_1, C_2 de G **faire**
      si la résolvante R de C_1 et C_2 existe
      et si il n'existe pas de clause D de G telle que $D \subseteq R$
      alors faire
        pour toutes les clauses D telles que $R \subset D$ **faire**
          $G \leftarrow G \setminus \{D\}$
        fait
        $G \leftarrow G \cup \{R\}$
        FINI $\leftarrow$ faux
      fait
    fait
  fait
  retourner G
fin

Remarque : $R \subset D$ si $R \subseteq D$ et $R \neq D$.

Question 5 :
- *Prouver que l'algorithme SATURATION termine pour toute formule CNF F.*
- *Prouver que la formule G produite par l'algorithme SATURATION est équivalente pour la satisfiabilité à la formule F de départ.*

Question 6 : *En supposant que les clauses sont codées par des listes de littéraux et les ensembles de clauses par des listes de clauses, analyser la complexité de la boucle interne de l'algorithme SATURATION (la boucle « pour tous les couples ... »).*

Question 7 : *En s'aidant des réponses aux questions précédentes, analyser la complexité de l'algorithme SATURATION lorsqu'on l'applique sur des formules 2-SAT. Qu'en conclure sur la complexité du problème 2-SAT ?*

On considère l'algorithme (dit de propagation unitaire) dans un ensemble de clauses F suivant :

PROPAGATION-UNITAIRE
Donnée : une formule CNF F
Résultat : une formule CNF G
Variables locales : C, D : clauses ; p, q : littéraux ; L : ensemble de littéraux ; FINI : booléen.
début
  $G \leftarrow F$
  $L \leftarrow \emptyset$
  FINI $\leftarrow$ faux
  tant que non FINI et que G ne contient pas de clause vide **faire**
    FINI $\leftarrow$ vrai
    si G contient une clause $C = \{p\}$ telle que $p \notin L$ **alors faire**
      $L \leftarrow L \cup \{p\}$
      FINI $\leftarrow$ faux
      pour toutes les clauses D de G, $D \neq C$, telles que $p \in D$ **faire**
        $G \leftarrow G \setminus \{D\}$
      fait
      pour toutes les clauses D de G, telles que $\neg p \in D$ **faire**
        $D \leftarrow D \setminus \{\neg p\}$
      fait
    fait
  fait
fin

Question 8 : *Analyser la complexité de l'algorithme PROPAGATION-UNITAIRE.*

On admettra qu'en choisissant les structures de données adéquates, on peut mettre en œuvre un algorithme de propagation unitaire dont la complexité est en $\mathcal{O}(|F|)$.

Question 9 : *Soient F une formule 2-CNF, p un littéral et G la formule obtenue à partir de $F \cup \{p\}$ par propagation unitaire.*

- *Que peut-on dire de la satisfiabilité de F et G dans le cas où G ne contient pas la clause vide (pour cela, on étudiera ce qu'il advient des clauses de la forme $\{q, x\}$ et $\{\neg q, x\}$, où q est un des littéraux « propagés » par l'algorithme) ?*
- *En utilisant le fait que F est satisfiable si et seulement si $F \cup \{p\}$ est satisfiable ou bien si $F \cup \{\neg p\}$ est satisfiable, en déduire un algorithme de complexité linéaire (en temps) pour tester la satisfiabilité des formules 2-SAT.*

Question 10 : *Démontrer que la propagation unitaire est complète sur les k,1-CNF (pour tout $k \geq 1$), c'est à dire qu'une formule k,1-CNF est insatisfiable si et seulement si l'algorithme PROPAGATION-UNITAIRE produit la clause vide.*

2 Systèmes d'équations booléennes

On considère maintenant des systèmes (c'est à dire des ensembles) d'équations booléennes, construits à partir de l'ensemble $\mathcal{V}$ des variables propositionnelles des connecteurs $\neg$, $\vee$ et $\wedge$ et d'un ensemble infini dénombrable de noms de formules $\mathcal{G} = \{G_1, G_2, \ldots\}$ de la façon suivante : l'ensemble des systèmes d'équations booléennes est le plus petit ensemble tel que :

- L'ensemble vide $\emptyset$ est un système d'équations booléennes.
- Si S est un système d'équations booléennes, $f_1, \ldots, f_n$ sont des noms de formules apparaissant dans S ou des variables propositionnelles et si G_i est un nom de formule n'apparaissant pas dans S alors les trois ensembles suivants sont des systèmes d'équations booléennes :
  - $S \cup \{G_i = \neg f_1\}$,
  - $S \cup \{G_i = f_1 \vee \ldots \vee f_n\}$,
  - $S \cup \{G_i = f_1 \wedge \ldots \wedge f_n\}$.

Un système d'équations booléennes peut être représenté par un graphe orienté acyclique (DAG) dont les nœuds sont étiquetés par des variables, des noms de formules ou des connecteurs, chaque variable ou nom de formule n'étiquetant qu'un seul nœud. Par exemple, le système d'équations booléennes suivant S_1 :

$$\begin{aligned} G1 &= G2 \vee G3 \\ G2 &= e1 \wedge G4 \\ G3 &= G4 \wedge e3 \\ G4 &= \neg\, e2 \end{aligned}$$

est représenté par le DAG D_1 (Figure 1).

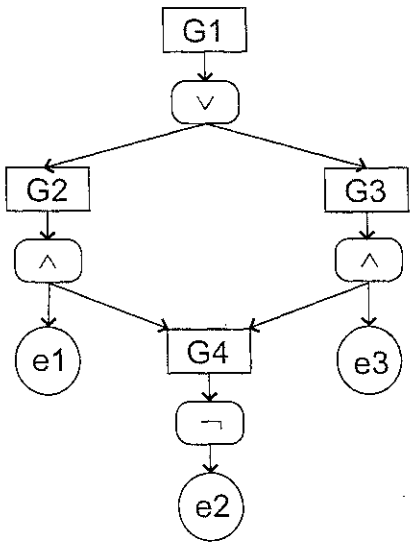

FIG. 1 – Le DAG D_1 représentant S_1.

On associe de façon naturelle une formule booléenne à chaque nœud du DAG étiqueté par un nom de formule. Par exemple, dans le DAG D_1, on associe :

- $\neg e_2$ au nœud étiqueté par G_4,
- $(e_1 \wedge \neg e_2)$, au nœud étiqueté par G_2,
- $(\neg e_2 \wedge e_3)$ au nœud étiqueté par G_3,
- et $(e_1 \wedge \neg e_2) \vee (\neg e_2 \wedge e_3)$ au nœud étiqueté par G_1.

On suppose que l'on dispose du type de données « DAG ». Ce type de données permet en particulier d'accéder à la liste des nœuds fils et à la liste des nœuds pères d'un nœud, ainsi que de marquer les nœuds lors d'un parcours.

Question 11 : *Écrire un algorithme polynomial en temps prenant en entrées un DAG D et un nœud n de D et produisant en sortie la liste des variables étiquetant les nœuds descendants de n (c'est à dire les nœuds du sous-DAG dont n est la racine). Donner la complexité de cet algorithme.*

On définit le poids d'un nœud de la façon suivante :
- 1 si c'est une racine (i.e. s'il n'a pas de nœud père),
- la somme des poids de ses nœuds pères sinon.

Question 12 : *Écrire un algorithme polynomial en temps prenant en entrée un DAG D et calculant le poids de chaque nœud de D. Donner sa complexité.*

On suppose à partir de maintenant et dans toute la suite du sujet que :
- Les connecteurs $\vee$ et $\wedge$ sont binaires.
- Tous les DAG considérés n'ont qu'une seule racine.

La polarité $pol(v)$ d'une variable v dans un DAG avec une seule racine est définie de la façon suivante :
- $pol(v)$ vaut 1 si tous les chemins de la racine au nœud étiqueté par v contiennent un nombre pair (incluant 0) de nœuds étiquetés par des négations.
- $pol(v)$ vaut -1 si tous les chemins de la racine au nœud étiqueté par v contiennent un nombre impair de nœuds étiquetés par des négations.
- $pol(v)$ vaut 0 dans les autres cas.

Question 13 : *Écrire un algorithme polynomial en temps prenant en entrée un DAG D et calculant la polarité de chaque nœud de D étiqueté par une variable. Donner la complexité de cet algorithme.*

Un système d'équations booléennes est sous forme normale négative (NNF) si les négations n'apparaissent que sur les nœuds pères des nœuds étiquetés par des variables. On rappelle les lois de de Morgan :

$$\begin{aligned} \neg(F \vee G) &\equiv \neg F \wedge \neg G \\ \neg(F \wedge G) &\equiv \neg F \vee \neg G \end{aligned}$$

Question 14 : *En utilisant les lois de de Morgan, écrire un algorithme polynomial en temps, dont on précisera la complexité, prenant un DAG D en entrée et produisant un DAG D' en sortie tel que :*

- *D' n'a qu'une seule racine,*
- *D' est sous forme normale négative,*
- *la formule associée à la racine de D' est équivalente (au sens habituel des tables de vérité) à la formule associée à la racine de D.*

Idée : On pourra introduire de nouveaux noms de formules.

Question 15 : *Écrire un algorithme polynomial en temps prenant en entrée un DAG D et produisant en sortie une formule CNF F telle que F est équivalente pour la satisfiabilité à la formule associée à la racine de D. Donner la complexité de cet algorithme.*

FIN DE L'ÉPREUVE