Algorithme polynomial pour le problème 2-SAT

Contents

I) Préliminaires

Ceci est un exercice (très) difficile. Faites des dessins de graphes pour bien comprendre les constructions et algorithmes définis, et étudiez des exemples avant de vous lancer dans le cas général !

I.1) Formules en 2-CNF

Définition 1 [Formule sous forme 2-CNF]

> Une formule logique est dites en 2-CNF si elle est de la forme :
>
> $\varphi = \bigwedge_i p_i \vee q_i$
>
> où p_i et q_i sont des littéraux, c'est-à-dire soit des variables x, soit des négations de variables $\neg x$.
>
> Les $p_i \vee q_i$ sont appelés les clauses.

Exemple 2

La formule $\varphi = (a \vee b) \wedge (\neg a \vee \neg b) \wedge (c \vee \neg c)$ est en 2-CNF.

Les formules en 2-CNF seront représentées en OCaml par le type suivant :

```
type litteral = Var of int | Not of int;;
type formule_2cnf = (litteral * litteral) list;;
```

On va s'intéresser à la résolution de formules en 2-CNF, c'est à dire de trouver des valeurs des variables telles que la formule soit vrai, si c'est possible.

On dit qu'une formule est *satisfiable* si une telle solution existe.

I.2) Graphe d'implication

Pour cela, on va utiliser une construction appelée *graphe d'implication.*

Définition 3 [Graphe d'implication]

> Pour une formule φ en 2-CNF, on appelle **graphe d'implication** le graphe orienté dont :
> - les sommet S sont les littéraux de *phi* (c'est à dire, toutes les variables de φ et leurs négations)
> - pour chaque clause $p \vee q$, on place les arrêtes $\neg p \to q$ et $\neg q \to p$
>
> *nb : en fait, on peut voir la clause $p \vee q$ comme l'implication "non p implique q", et donc les arrêtes de graphe sont bien les implications.*

Les graphes en OCaml seront représentés par liste d'adjacence, en utilisant le type suivant :

```
type graphe = int list array
```

Si la formule contient n variables, les sommets de 0 à $n-1$ sont les variables, et ceux de n à $2n-1$ leur négation.

Le tableau contient $2n$ valeurs : l'élément d'indice i correspond aux sommets voisins du sommet i.

II) Résolution

II.1) Utilisation du graphe pour la satisfiabilité

Q1) *Que peut-on en déduire sur la formule si, pour une variable x, le graphe contient une arrête $x \to \neg x$? La formule est-elle alors satisfiable ?*

Q2) *Que peut-on alors en déduire sur la formule si, pour une variable x, le graphe contient un chemin entre x et $\neg x$?*

Q3) *En déduire une condition nécéssaire sur le graphe pour que la formule associée soit satisfiable.*

On admet dans un premier temps que cette condition est également suffisante.

II.2) Graphe en OCaml

Q4) *Écrire un code OCaml pour créer le graphe d'implication correspondant à une formule en 2-CNF.*

Q5) *Écrire un code OCaml pour créer le graphe d'implication correspondant à une formule en 2-CNF.*

II.3) Résolution sur le graphe

Définition 4 [Composantes fortement connexes]

> Pour G un graphe, on appelle *composante fortement connexe* de G tout sous-ensembles des sommets $C \in S$ tel que, pour tout $p, q \in C$, il existe un chemin de p à q.

Q6) *Montrez que la condition énoncée en **II.1)** est équivalente à "pour toute variable x, x et $\neg x$ ne sont pas dans la même composante fortement connexe.*

On va utiliser un algorithme classique pour calculer les composantes fortement connexes de notre graphe :

Algorithme 5 [de Kosaraju]

> - Effectuer un parcours en profondeur sur G, et noter le post-ordre ω, c'est à dire l'ordre de remontée du parcours. À noter que, lorsqu'on termine un parcours, si on a pas visité tous les sommets, il faudra reprendre depuis un sommet non-visité arbitraire.
> - Transposer le graphe G en G^T (c'est à dire inverser toutes ses arrêtes), et inverser le post-ordre ω en ω^T.
> - Effectuer un parcours en profondeur sur G^T, en suivant l'ordre indiqué par ω^T, c'est-à-dire que, lorsqu'un parcours est terminé, on reprend depuis le premier sommet de ω^T non visité.

Q7) *Écrire une fonction qui permet de faire un parcours en profondeur d'un graphe, en notant l'ordre de remontée. Vous pouvez vous aider d'un nouveau tableau de booléen pour identifier les sommets déjà visités.*

Q8) *Écrire des fonctions permettant d'inverser une liste, et de transposer un graphe.*

Q9) *En déduire une fonction qui execute l'algorithme de Kosaraju.*

Q10) *En déduire une fonction qui permet de determiner si une formule en 2-CNF est satisfiable.*

Q11) *Comment retrouver une valuation des variables telle que la formule soit vraie ?*