Aix*Marseille université

Domaine Sciences et Technologies
L3 Info, parcours Maths-info

Algo. avancée et th. des graphes : TP 1
Code UE : S04IN5B2

Année 2023-24 (version du 19 septembre 2023)

2-SAT

Le travail se déroule en binôme. Sources partielles :

```
git clone https://gitlab.com/algographes/tp1.git
```

Langage à utiliser : `Java`.

Le problème 2-SAT est un problème de décision : étant donnée une formule 2-SAT P, il s'agit de répondre à la question "est-ce que la formule P est satisfiable ou non ? ". C'est un cas particulier du problème SAT qui est NP-complet, mais pour le problème 2-SAT il existe des approches *polynomiales*. Vous allez implémenter une approche de ce type. Les étapes pour la réalisation du projet se suivent et chacune dépends des étapes précédentes. Vous devez donc traiter les étapes les unes après les autres sans en sauter aucune et en commençant par la première.

Vous avez 4 semaines pour réaliser ce TP, qui sera evalué lors de la dernière séance. Les sources doivent être téléversés sur la page AMETICE du cours.

Le problème 2-SAT

Un problème 2-SAT est défini par un ensemble fini de clauses (on parle de *forme normale conjonctive*). Chaque clause est constituée de 2 littéraux et chaque littéral est une variable ou la négation d'une variable. Voici par exemple un problème 2-SAT constitué de 4 clauses, dans lequel apparaissent les 3 variables x, y et z :

$$\{x \vee \neg y, \neg x \vee z, x \vee z, \neg y \vee \neg z\}$$

La formule est satisfiable s'il existe une affectation des variables x, y et z avec les valeurs **V** (vrai) et **F** (faux) qui satisfait toutes les clauses (pour qu'une clause soit satisfaite il suffit qu'un de ses littéraux soit vrai). Par exemple l'affectation $x \leftarrow \mathbf{V}$, $y \leftarrow \mathbf{F}$, $z \leftarrow \mathbf{V}$, satisfait la formule ci-dessus : la première clause est satisfaite puisque x est vrai, la 2è clause parce que z est vrai, la 3è parce que x et z sont vrais, et la 4è parce que y a la valeur faux.

Déterminer si une formule SAT dans le cas général est satisfiable est très difficile (le problème est NP-complet). Dans le cas particulier où les clauses sont de longueur 2 il existe des approches efficaces (linéaires en le nombre de clauses). La plus connue est la suivante :

1. construire le *graphe des implications*,
2. calculer les composantes fortement connexes du graphe des implications,
3. si une composante contient à la fois un littéral et son opposé, la formule est insatisfiable ; dans le cas contraire la formule est satisfiable.

Vous allez implémenter cette approche.

Graphe des implications

Dans le problème 2-SAT chaque clause est de longueur 2, c'est à dire de la forme

$$l_1 \vee l_2$$

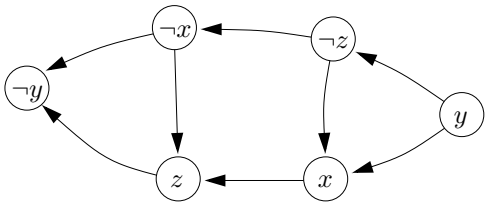
où l_1 et l_2 sont des littéraux. Remarquer que cette clause est équivalente à la formule

$$(\neg l_1 \Rightarrow l_2) \wedge (\neg l_2 \Rightarrow l_1)$$

Le graphe des implications représente les implications qui apparaissent dans cette formule. Plus formellement, si V est l'ensemble des variables figurant dans la formule, et C l'ensemble de ses clauses, le graphe des implications $G = (S, A)$ est un graphe orienté défini par :

$$S = \{x \mid x \in V\} \cup \{\neg x \mid x \in V\},$$
$$A = \{(\neg l_1, l_2), (\neg l_2, l_1) \mid (l_1 \vee l_2) \in C\}.$$

Notez donc que $|S| = 2 \times |V|$ et $|A| = 2 \times |C|$. Pour l'exemple précédent le graphe des implications serait :



Calcul des composantes fortement connexes

Dans un graphe orienté, deux sommets u et v sont *fortement connectés* s'il existe un chemin orienté de u vers v et vice versa. C'est une relation d'équivalence et l'ensemble des sommets du graphe peut être partitionné en composantes fortement connectées, sous-ensembles de sommets dans lesquels entre deux sommets quelconques il existe un chemin orienté. L'algorithme de Kosaraju (linéaire en le nombre d'arcs) construit les composantes fortement connexes d'un graphe donné. Le calcul s'effectue en deux temps :

étape 1 : parcours en profondeur du graphe des implications G ; on mémorisera les dates de fin de traitement des sommets,

étape 2 : parcours en profondeur du graphe miroir de G noté G^{rev} (le graphe G dans lequel les arcs sont inversés) ; chaque fois, pour lancer l'exploration, on choisira le sommet non visité dont la date de fin de traitement à l'étape 1 est la plus grande. Les arbres construits lors de ce parcours sont les composantes fortement connexes du graphe G.

Ce que vous devez faire

Dans l'ordre :

1. lire les clauses depuis un fichier (téléchargez le fichier exemple formule-2-sat.txt),
2. construire le graphe des implications G,
3. calculer les composantes fortement connexes, et pour cela vous devrez construire le graphe transposé G^{rev}.
4. déterminer si aucune composante ne contient à la fois un littéral et son opposé.

Les formules que vous aurez à traiter sont de la forme suivante (c'est un exemple) :

```
c Commentaire
p cnf 4 3
1 -2 0
3 4 0
-3 -4 0
```

La première ligne (commençant par 'c') est un commentaire. La deuxième ligne indique le nombre de variables (ici il y a 4 variables, qui sont implicitement numérotées de 1 à 4) et le nombre de clauses (dans l'exemple il y a 3 clauses). Chaque clause est définie par ses littéraux (x ou $-x$ selon le signe du littéral) et se termine par la valeur 0.

Les graphes seront représentés avec des listes d'incidence (tableau indexé sur les sommets contenant, pour chaque sommet, la liste chainée de ses arcs sortants du sommet).

Pour débuter le projet, vous allez utiliser les définitions (classes et methodes) à votre disposition sur gitlab :

```
git clone https://gitlab.com/algographes/tp1.git
```

Éléments de bonne programmation, et autres

— Factoriser votre code : votre code sera divisé en plusieurs fichiers, chaque fichiers contiendra une seule classe. Les objectifs de chaque classe seront clairement identifiés et limités (vous pouvez les décrire dans le README). Par exemple : on pourra identifier les classes `Parser`, `Graph`, `Parcours` (pour les algorithmes de parcours de graphe) `Kosaraju` (pour le calcul des composantes fortement connexes), `GrapheImplication` (qui pourra hériter de la classe `Graphe`, par exemple), `Solver`, `Main` ...

— Respecter, dans le rendu, la structure arborescente des fichiers donnée. Cela est nécessaire pour tester et évaluer votre rendu.

— La compilation de votre rendu produira un fichier `2SAT.jar` que l'on pourra utiliser via le terminal de la façon suivante :

```
java -jar 2SAT.jar [nomfichier]
```

— Vous disposez, dans les sous-répertoires `formulas/testSet0` et `formulas/testSet1`, de 20 fichiers contenant de formules, avec en commentaire si elles sont ou non satisfaisables. Utilisez-les pour tester votre programme !

La prise en compte de ces éléments sera appréciée pour l'évaluation de votre rendu.

Mise(s) en garde

— Dans une implementation usuelle d'une classe `Graphe` (comme par exemple celle proposée sur le `git`) les sommets sont numerotés de 0 à $n-1$ (dans une grapge (V, E) tel que $|V| = n$). Par ailleurs, les sommets du graphe d'implication d'une formule 2-SAT avec k variables propositionnelles sont numérotés de $-k$ à -1 et de 1 jusqu'à k.
Il faudra donc réaliser des fonctions de codage et de décodage mettant en correspondance bijective les ensembles

$$\{-k, \dots, -1\} \cup \{1, \dots, k\} \qquad \text{et} \qquad \{0, \dots, 2k-1\}.$$

On vous laisse le choix de la classe où situer ces fonctions.

— Le projet se déroule par binômes. La note du projet prendra en compte (en le pénalisant) l'existence de code jugés trop similaire des binômes différents.