N3 Informatique 27/11/2020
2020-2021

Rapport TP théorie des graphes

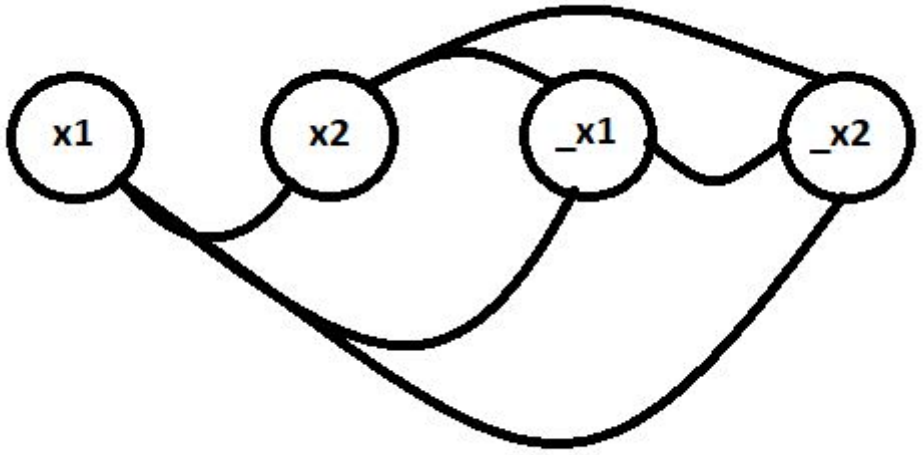
BEC Charly, PLESSIER Alexis, PUC Hugo.

Nombre total de clause d'une formule *(exercice 1)* :

(ATTENTION : lors de la rédaction de ce rapport, nous avons décidé d'utiliser la notation $\_x_n$ pour désigner la valeur NON x_n !)

Prenons par exemple 2 variables x_1 et x_2, donc on aura véritablement 4 variables: x_1, x_2, $\_x_1$ et $\_x_2$. Il faut savoir que les clauses $(x_1 \vee x_2)$ et $(x_2 \vee x_1)$ sont différentes.

En faisant un schéma, on peut se rendre compte qu'on va devoir utiliser la formule de la somme des entiers pour calculer le nombre de clauses peut prendre une formule F, 2-SAT avec n variables.



Ici, on peut faire les clauses suivantes: $(x_1 \vee x_2)$, $(x_1 \vee \_x_1)$, $(x_1 \vee \_x_2)$, $(x_2 \vee \_x_1)$, $(x_2 \vee \_x_2)$ et $(\_x_1 \vee \_x_2)$. Donc 6 clauses multiplié par deux car on peut intervertir les variables dans chaque parenthèses (en tout 12 clauses ici).

On se rend compte qu'on a 3 liaisons à partir de x_1, 2 liaisons à partir de x_2 et 1 liaison à partir de $\_x_1$. Cela donne: 3+2+1 = 6. On a ici la somme des nombres entier.

$$S(n) = \frac{n\,(n+1)}{2} = \frac{(n^2+n)}{2}$$

On applique à la formule la modification sur n, car si on prend deux variables (n=2), on doit multiplier n par 2 car on a aussi le négatif de chaque variable.
Mais on ne doit pas oublier que sur notre exemple, on fait la somme pour 3 chiffres (3+2+1) alors qu'on a 4 variables au total (x_1, x_2, $\_x_1$ et $\_x_2$). Donc n devrait se transformer en (n*2) - 1.

Notre formule passe de $\frac{(n^2+n)}{2}$ à $\frac{(n*2-1)^2+(n*2-1)}{2}$ qu'on multiplie par 2 car on peut inverser les variables dans chaque parenthèse:

$\frac{(n*2-1)^2+(n*2-1)}{2}$ * 2, la formule finale est donc: $(n*2 - 1)^2 + (n*2 - 1)$.

Essai numérique avec 2 variables comme l'exemple ci-dessus:
n = 2, $(2*2 - 1)^2 + (2*2 - 1) = 3^2 + 3 = 9 + 3 = 12$.

On retombe bien sur les 6 possibilités de clauses (multipliées par 2 car on peut interchanger les variables dans chaque parenthèses) donc 12 clauses possibles.

Formule 2-SAT non satisfaisable *(exercice 2)* :

$F = (x_1 \vee x_2) \wedge (\_x_1 \vee x_2) \wedge (\_x_1 \vee \_x_2) \wedge (x_1 \vee \_x_2)$

Peu importe la valeur de x1 ou x2 il existe une clause qui est égale à 0.
Par conséquent la formule n'est pas satisfaisable.

Evaluation d'une formule 2-SAT *(exercice 3)* :

Pour évaluer le résultat d'une formule 2-SAT, nous avons décidé, dans un premier temps, de demander à l'utilisateur le nombre de valeurs positives dans la formule et ensuite de renseigner sa valeur (0 pour FAUX et 1 pour VRAI). Ces valeurs vont être mise dans une liste et les valeurs négatives ($\_x_n$) vont être déduite et mise également dans cette liste. Cette étape terminée, l'utilisateur peut renseigner le nombre de clauses qu'il souhaite ainsi que les variables de ces clauses grâce à un système expliqué très simple. Ces dernières sont alors aussi entrées dans une liste avec la valeur de l'union des deux variables directement rentrée à l'index de la clause. Il ne nous reste plus qu'à comparer (ET binaire, intersection) le résultat de toutes les clauses pour savoir si le résultat rend VRAI ou FAUX?
Le code de cette partie précise sera disponible ci-joint ce rapport (*exo3.sage.py*).

Construction d'un graphe à partir d'une formule 2-SAT *(exercice 4)*:

Pour créer le graphe orienté de la formule rentrée préalablement, nous nous sommes appuyé du document annexe et de la définition suivante :

> Soit une clause (u,v), le graphe se bâtit en construisant les arcs orientés $\bar{u} \rightarrow v$ et $\bar{v} \rightarrow u$.

Nous allons donc définir un graphe orienté g dans un premier temps grâce à la fonction *DiGraph()* et construire chaque arc à la guise des choix de l'utilisateur en choisissant le système de choix suivant, Soit n variables positives :

- Le choix de 1 à n correspond à une variable positive.
- Le choix de n+1 à 2n correspond à une variable négative.

Chaque arc est créé à l'aide de la fonction *add_edge(u,v)* construisant un arc orienté de deux sommets u vers v et nous affichons le graphe entier en fin de programme à l'aide de la fonction *show()* pour laisser le programme s'exécuter avant.

Validité d'une formule 2-SAT et attribution de valeur *(exercice 5)*:

Jusqu'ici nous avons donc l'attribution de valeurs d'une formule, le construction du graphe impliqué et le résultat de la formule suivant les valeurs entrées.

Dans un premier temps nous allons déterminer la validité de la formule, nous allons faire ceci en partant du principe que si x_i et $\_x_i$ sont dans la même Composante Fortement Connexe (CFC), la formule est non satisfaisable. Ceci peut s'implémenter en sage grâce à la fonction *strongly_connected_component_containing_vertex()* prenant en paramètre un graphe et un sommet. Cette fonction retourne une liste pour tous les sommets, il ne nous reste plus qu'à tester tous ces derniers et de voir si, dans la liste, $\_x_n$ est présent. Si c'est le cas nous renverrons chaque sommets problématique ainsi qu'un message d'erreur.

Maintenant, si la formule est satisfaisable mais que le résultat avec les valeurs entrées par l'utilisateur est faux, le programme va pouvoir attribuer des valeurs nécessaires à l'obtention d'un résultat VRAI pour cette formule. Le principe est simple, nous récupérons les clauses dans deux listes *attr1* et *attr2* puis nous avons fait le choix de donner des valeurs aléatoires (entre 0 et 1) aux variables positives (et donc négatives indirectement). Ceci étant fait, et sachant qu'il existe une solution, nous testons à chaque fois les clauses entre elles jusqu'à obtenir un résultat VRAI. Nous pouvons alors renvoyer les valeurs de variable.

Le code de cette partie précise sera disponible ci-joint ce rapport (*exo5.sage.py*).