TD 4 – Calculabilité et complexité

Exercice 1. *Problème indécidable*

✎ Montrer qu'il n'existe pas d'algorithme pour déterminer si un algorithme donné termine sur toute entrée. *Montrer comment résoudre le problème de l'arrêt si on dispose d'un tel algorithme.*

Exercice 2. *Machines de Turing*

1. Donner la table de transition d'une machine de Turing qui teste si son entrée, interprétée comme un entier en base 2, est paire.
2. Donner la table de transition d'une machine de Turing qui teste si son entrée est un *palindrome* [1]

Exercice 3. *Simulateur de machine de Turing*

On veut écrire un simulateur de machine de Turing en Python. Pour représenter le ruban, on utilise une liste de `0`, `1` et `-1` (pour le symbole □). On représente les états par des chaînes de caractères, avec en particulier l'état `"init"` et l'état `"vrai"`. La table de transition est un dictionnaire, dont les clefs sont des couples `(etat_courant, symbole_lu)` et les valeurs des triplets `(symbole_ecrit, deplacement, nouvel_etat)`. On représente les déplacements gauche et droit par les entiers -1 et 1. Le ruban de la machine de Turing est potentiellement infini : pour simuler ce comportement, on doit agrandir le ruban (à droite ou à gauche) quand on sort de la liste qui le représente.

1. Écrire le dictionnaire correspondant à la table de l'exemple précédent.
2. Écrire une fonction `transition` qui prend en entrée une table (dictionnaire), et la configuration de la machine (état courant, ruban et position de la tête), et effectue la transition en mettant à jour le ruban, et en renvoyant le nouvel état et la nouvelle position de la tête. Si aucune transition n'est possible, la fonction renvoie `None`.
3. Écrire une fonction `simulateur` qui prend en entrée une table représentant une fonction f et un ruban initial contenant un mot w, et renvoie `True` si $f(w) = 1$ et `False` sinon.
4. *(bonus)* Encapsuler les fonctions précédentes dans une classe `MachineTuring`, et rajouter des fonctions d'affichage pour représenter le calcul en train de se faire par des images animées.

Exercice 4. *Problèmes dans* NP

1. Une *clique* dans un graphe est un sous-ensemble de ses sommets qui sont tous reliés deux à deux par une arête. Le problème Clique prend en entrée un graphe G et un entier k, et sa sortie est oui s'il existe une clique de taille k dans G, et non sinon. Montrer que Clique est dans NP.
2. Un *indépendant* dans un graphe est sous-ensemble des sommets tel qu'il n'existe aucune arêtes entre eux : quels que soient u et v dans un indépendant, il n'y a pas d'arête entre u et v. Le problème Indépendant prend en entrée un graphe G et un entier k, et sa sortie est vrai si et seulement si G possède un indépendant de taille k.
   i. Dessiner un graphe connexe ayant un indépendant de taille 4.
   ii. Montrer que le problème Indépendant est dans NP.
   iii. Montrer qu'il existe une réduction de Indépendant à Clique. *Indication. On peut considérer le complémentaire $G^\star$ d'un graphe G, qui possède une arête entre deux sommets u et v si et seulement s'il n'en existe pas dans G.*
3. Tester si un entier divise un autre entier se fait en temps polynomial. Pour factoriser un entier N, on peut tester pour tout entier $m \leq N$ si m divise N : est-ce un algorithme polynomial ?
4. Un entier est *composé* s'il n'est pas premier, c'est-à-dire s'il est divisible par un entier différent de 1 et lui-même. Montrer que tester si un entier est composé est un problème de NP.

1. Un palindrome est un mot qui se lit de la même façon de gauche à droite et de droite à gauche, comme `LAVAL` ou 01010, mais pas (()).

5. *(très difficile*[2]*)* Montrer que le problème précédent est dans P.

Exercice 5. *Exercice à rendre*

1. Donner la table de transition d'une machine de Turing qui teste si son entrée est de la forme $0\cdots01\cdots10\cdots0$, où le nombre de 0 dans la première partie, de 1 au centre et de 0 à droite sont quelconques, éventuellement nuls. *La machine doit répondre* VRAI *sur les entrées* 001000*,* 1*,* 000*,* 0111000 *et* FAUX *sur les entrées* 0101 *ou* 101.
2. Montrer qu'il n'existe pas d'algorithme qui, étant donné le code d'un algorithme A et une entrée x, détermine si sur l'entrée x, A renvoie 1. *Montrer qu'avec un tel algorithme, on pourrait résoudre le problème de l'arrêt.*

2. Ce résultat est dû à M. Agrawal, N. Kayal et N. Saxena en 2002. Il est déconseillé d'essayer de répondre à cette question…