1 Partie A

On considère le texte suivant : AABBBCDDDDDDEEF. Donner un codage des 6 premières lettres de l'alphabet obtenu avec l'algorithme d'Huffman qui permettrait de compresser ce texte de manière optimale.

Nous allons ici présenter une autre technique de compression qui repose aussi sur la fréquence des lettres utilisées. Voici son principe :

1. On ordonne les caractères suivant l'ordre croissant de leur fréquence dans un tableau.
2. On cherche ensuite à partitionner le tableau de caractères en deux sous-tableaux de manière à ce que les sommes des fréquences des caractères de chacun de ces sous-tableaux soient égales ou le plus proches possible.
3. Tous les codes des caractères du premier sous-tableau commencent alors par un 0 et ceux du deuxième sous-tableau par un 1.
4. Puis on continue à partitionner chacun des deux sous-tableaux en deux, en rajoutant un 0 aux codes des caractères des premiers sous-tableaux et en rajoutant un 1 aux codes des caractères des deuxièmes sous-tableaux. On continue ainsi les partitions jusqu'à ce qu'on aboutisse à un sous-tableau réduit à un seul caractère.

Donner un codage obtenu pour le même message que précédemment.

2 Partie B

L'objectif est d'implémenter le test de satisfiabilité d'une formule du calcul propositionnel sous forme 2SAT.

1. Rappeler la définition d'une formule sous forme 2SAT.

   Pour représenter une telle formule, on utilisera le type suivant :

```
type litteral =
| P of int (* positive occurrence *)
| N of int (* negative occurrence *)

type clause = litteral * litteral
type twocnf = clause list
```

   On supposera que les variables présentes dans une formule sont numérotées consécutivement à partir de zéro.

   Soit $\phi = \bigwedge_{i=1}^{r} F_i$ une formule en 2-CNF (chaque F_i est donc de la forme $l \vee l'$, où l et l' ne contiennent pas la même variable) et $x_0, \ldots, x_{n-1}$ les variables présentes dans ϕ. On définit le graphe $G_\phi = (V_\phi, E_\phi)$ comme suit :
   — il y a un sommet pour chaque littéral possible (autrement dit, $2n$ sommets $x_0, \neg x_0, \ldots, x_{n-1}, \neg x_{n-1}$) ;
   — pour chaque clause $l \vee l'$, il y a deux arcs $\neg l \to l'$ et $\neg l' \to l$.

2. On considère la formule suivante :

$$\phi = (x_0 \vee \neg x_2) \wedge (x_1 \vee x_3) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (x_3 \vee \neg x_0)$$

   Représenter son graphe G_ϕ.

3. Ecrire une fonction `max_var : twocnf->int` qui renvoie l'indice maximal d'une variable apparaissant dans la formule passée en entrée.
4. Écrire une fonction `graph_of_cnf: twocnf->graph` prenant en entrée une formule ϕ en 2-CNF et renvoyant le graphe G_ϕ associé. On pourra supposer qu'aucune clause ne contient deux fois le même littéral, ni un littéral et son complémenté. Pour le type graphe, on utilisera `type graph = int list array`. Tester votre fonction sur le graphe `ex` fourni. (on doit obtenir `[|[6];[5];[4];[6];[6;0];[3];[];[1,5,2]|]`)
5. Rappeler la condition nécessaire et suffisante sur le graphe G_ϕ vue en cours qui caractérise le fait qu'une formule ϕ sous forme 2SAT est satisfiable.
6. Afin d'utiliser cette caractérisation, on va implémenter l'algorithme de Kosaraju.
   (a) Écrire une fonction `transpose` qui prend en entrée un graphe G et renvoie son graphe transposé (ou miroir).
   (b) Écrire une fonction `post_order` qui effectue un parcours en profondeur complet d'un graphe et renvoie la liste de ses sommets par instant de fin de traitement décroissant.
   (c) Compléter le code fourni pour obtenir une fonction `kosaraju : graph-> int list list` qui renvoie une liste contenant chaque composante fortement connexe du graphe passé en entrée. Chaque CFC est elle même une liste.
   (d) Tester votre fonction en vérifiant que le fichier `arxiv.txt` représente un graphe à 21608 composantes fortement connexes (une fonction de lecture du fichier vous est fournie).
7. Écrire une fonction `satisfiable` déterminant si une formule ϕ en 2-CNF est satisfiable. On exige une complexité linéaire en la taille de la formule (nombre d'occurrences de variables).