```
#!/usr/bin/env python

#Code par Solene BELLISSARD en CMI informatique Ã  l'USMB

############################
# Definition de la machine #
############################

# Les rubans sont consideres infinis a gauche et a droite et sont "blancs" par
defaut
# Le symbole associee au blanc (0, '', 'B', None, etc) est a definir pour chaque
alphabet de travail
# Il n'est utile de representer que les parties non infiniment blanches des
rubans.
# Par exemple le ruban d'entree:
# -------+---+---+---+---+---+---+---+---+---+---+---+---+---+---+-------
# <-inf- |   |   |   | 0 | 1 | 1 |   | 0 |   |   | 1 |   |   |   | -inf->
# -------+---+---+---+---+---+---+---+---+---+---+---+---+---+---+-------
# peut etre represente ainsi:
# ruban = [0, 1, 1, None, 0, None, None, 1]

# Structure d'une table d'action
"""
table_actions = {
    etat_1: {
        symbole_1: action,
        symbole_2: action,
        ...
    },
    etat_2: {
        ...
    },
    ...
}
"""
# Structure d'une action
"""
action = [ecriture_ruban_sortie, deplacement_ruban_sortie,
deplacement_ruban_entree, etat_suivant]
"""
# Note: les couples (etat+symbole) qui n'ont pas d'actions associees entraine le
passage a l'etat REJET.

# Etats de fin imposes
ETAT_ACCEPTATION = 0
ETAT_REJET = 1


def bouger_tete(position_tete, ruban, deplacement, symbole_blanc=None):
    """
    Simule le deplacement d'une tete de lecture/ecriture sur un ruban
    :param position_tete: int
    :param ruban: list[]
    :param deplacement: str
    :param symbole_blanc: any
    :return:
    """
    if deplacement == "droite":
        position_tete += 1
        if position_tete >= len(ruban):  # Gestion depassement positif
allocation tableau ruban
            ruban.append(symbole_blanc)
    elif deplacement == "gauche":
        position_tete -= 1
```

```
        if position_tete < 0:  # Gestion depassement negatif allocation tableau
ruban
            ruban.insert(0, symbole_blanc)
            position_tete += 1
    # else: None, pas de deplacement
    return position_tete


def machine_2_rubans(ruban_entree, etat_initial, table_actions,
symbole_blanc=None):
    """
    Simule le fonctionnement d'une machine de Turing Ã  deux rubans (un
d'entree, un de sortie)
    :param ruban_entree: list[]
    :param etat_initial: int
    :param table_actions: dict[]
    :param symbole_blanc: any
    :return:
    """
    ruban_sortie = [symbole_blanc]  # Ruban infini alloue dynamiquement
    etat_courant = etat_initial
    tete_ruban_entree = 0  # Representation de la position de la tete sur le
ruban d'entree
    tete_ruban_sortie = 0  # Representation de la position de la tete sur le
ruban de sortie

    while etat_courant not in (ETAT_ACCEPTATION, ETAT_REJET):
        symbole_lu = ruban_entree[tete_ruban_entree]

        # Verification de la prise en charge de cet etat dans la table d'action
        if etat_courant not in table_actions.keys():
            # C'est un cas d'erreur, on stoppe la machine
            etat_courant = ETAT_REJET
            break

        actions_pour_etat_courant = table_actions[etat_courant]

        # Verification de la prise en charge de ce cas (etat+symbole) dans la
table d'action
        if symbole_lu not in actions_pour_etat_courant.keys():
            # C'est un cas d'erreur, on stoppe la machine
            etat_courant = ETAT_REJET
            break

        action = actions_pour_etat_courant[symbole_lu]

        # Realisation de l'action
        # - Ecriture ruban de sortie
        if action[0] is not None:
            ruban_sortie[tete_ruban_sortie] = action[0]
        # - Deplacement tete sur ruban de sortie
        tete_ruban_sortie = bouger_tete(tete_ruban_sortie, ruban_sortie,
action[1], symbole_blanc)
        # - Deplacement tete sur ruban d'entree
        tete_ruban_entree = bouger_tete(tete_ruban_entree, ruban_entree,
action[2], symbole_blanc)
        # - Changement de l'etat courant
        etat_courant = action[3]

    # Retirer les symboles blancs aloues inutilement en debut et fin de ruban
    while len(ruban_sortie) > 0 and ruban_sortie[0] == symbole_blanc:
        ruban_sortie.pop(0)
    while len(ruban_sortie) > 0 and ruban_sortie[-1] == symbole_blanc:
        ruban_sortie.pop(-1)
```

```
    return etat_courant, ruban_sortie


############################
# Affichage de la machine #
############################
def afficher_ruban(ruban, symbole_blanc=None):
    """
    Affiche joli :)
    :param ruban: list[]
    :param symbole_blanc: any
    """
    taille_ruban = len(ruban)
    ligne_jolie = "-" * 8 + "+"
    ligne_jolie += "---+" * taille_ruban
    ligne_jolie += "-" * 8

    milieu = " " * 8 + "|"
    for lettre in ruban:
        if lettre != symbole_blanc and lettre != "":
            milieu += " " + str(lettre)[:1] + " |"
        else:
            milieu += " " * 3 + "|"
    milieu += " " * 8

    print(ligne_jolie)
    print(milieu)
    print(ligne_jolie)


def afficher_machine(ruban_entree, ruban_sortie, etat_final):
    """
    Affiche joli :)
    :param ruban_entree: list[]
    :param ruban_sortie: list[]
    :param etat_final: int
    """
    print("Ruban d'entree :")
    afficher_ruban(ruban_entree)

    if etat_final == ETAT_ACCEPTATION:
        print("Ruban de sortie :")
        afficher_ruban(ruban_sortie)
    else:
        print("REJET DU RUBAN D'ENTREE")


##############################
# Utilisation de la machine #
##############################

# ## Machine de turing a deux rubans qui double le nombre de 1 sur le ruban
d'entree
# ## jusqu'au premier 0 rencontre.
# On utilise l'alphabet de travail {0, 1, None} (None est le symbole blanc)

# Definition des etats
ETAT_LECTURE = 2
ETAT_AJOUT = 3

# Definition des actions
table_actions_doubler = {
    ETAT_LECTURE: {
```

```
        0: [None, None, None, ETAT_ACCEPTATION],  # Zero de fin de ruban
d'entree
        1: [1, "droite", None, ETAT_AJOUT],  # Recopie du 1 lu
    },
    ETAT_AJOUT: {
        1: [1, "droite", "droite", ETAT_LECTURE],  # Ajout d'un 1 supplementaire
    },
}

if __name__ == "__main__":
    # Exemples de rubans d'entree
    # On suppose qu'un ruban d'entree est infini a droite et gauche
    # Ici on ne represente que la partie qui n'est pas blanche (None)
    r1 = [1, 1, 0]
    r2 = [1, 1, 1, 1, 0]
    r3 = [1, 1, 0, 1, 0]
    r4 = [1, 0]
    r5 = [0, 1, 1]
    r6 = [1, 1]

    # Changer que le ruban d'entree
    ruban_entree = r2
    etat_final, ruban_sortie = machine_2_rubans(ruban_entree, ETAT_LECTURE,
table_actions_doubler)
    afficher_machine(ruban_entree, ruban_sortie, etat_final)
```