SENDTO:vendrame.vincent@gmail.com Nom de la méthode ———————— GlisserOVersDirection

Tâche

Cette fonction prend:

- un tableau de taille quelconque, ne pouvant contenir exclusivement les caractères:
  - '`O`',
  - '`#`',
  - et '' (espace),
- et une direction représentée par un booléen (`true`/`false`).

En fonction de la direction, elle doit , de façon répétée, faire glisser les 'O's vers la direction donnée jusqu'à atteindre un état stable ;

c'est à dire, les O sonts coincées soit:

- contre un bord (première/dernière position) du tableau
- un autre O ou #.

Les # ne glissent pas. Les espaces peuvent être remplacés et sonts considérés comme emplacements libres.

Exemple

L'appel de GlisserOVersDirection avec:

- Le tableau représentant `[O O  # #O# O ]`
- le booléen représentant la direction de glissement vers la droite

doivent donner le tableau `[   OO# #O#  O]`.

Préconditions

- Le tableau entré doit être un tableau de caractères (char[])
- Le tableau ne peut être de valeur null (!= null)
- Le tableau peut être vide, et alors rien ne doit être changé ({} donne {})
- Le tableau peut être dans un état déjà résolu / où rien n'est à changer (par exemple, [O] vers la gauche donne [O] malgré tout)
- La valeur de direction devrait être un booléen (boolean)

Postconditions

- Le tableau rendu doit contenir les mêmes nombres de ' ','#' et 'O' respectifs totaux.
- Les 'O's doivent être correctement glissés en fonction de la direction donnée.

- La position des '#' dans le tableau ne doit pas avoir changé.
- Si il n'y a rien à changer, alors le tableau ne doit pas avoir changé (voir dernier paragraphe de Tâche)
- Si , dans le cas d'une direction gauche (ou vice versa, droite), un O n'est ni:
  - en première/dernière position du tableau (contre un bord)
  - ou contre un '#' ou autre 'O' à sa gauche alors le tableau doit glisser ce O jusqu'à que cette condition soit satisfaite.

Plan de test

Cas normaux

- Entrée: `{'O','O',' ','#',' ','O',' '}` et direction vers la droite (`[OO # O ]`)
- Sortie: `{' ','O','O','#',' ',' ','O'}` (`[ OO#  O]`)
- Effet attendu: chaque caractère 'O' a glissé vers la droite et ne peut pas glisser plus vers la droite.
- Entrée: `{' ','#','O',' ','O','#','O',' ','#','O'}` et direction vers la droite (`[ O O#O #O]`)
- Sortie: `{' ','#','O','O',' ','#','O',' ','#','O'}` (`[ #OO #O #O]`)
- Effet attendu: chaque caractère 'O' a glissé vers la gauche et ne peut pas glisser encore plus vers la gauche.

Cas par branchement

Chaque cas nécessite un branchement pour déterminer, après chaque glissade (d'une case vers la direction choisie), si rien n'a changé et que l'on peut donc finir l'algorithme. Sinon, l'on continue à appliquer l'algorithme au problème jusqu'à que rien ne change.

Pour comparer, voici un cas sans branchement:

- Entrée: `{'O',' ','#','O','O',' ',' '}` vers la gauche (`[O #OO  ]`)
- Sortie: `{'O',' ','#','O','O',' ',' '}` (`[O #OO  ]`)
- Effet attendu: le tableau n'est pas altéré, car l'on ne peut plus rien glisser vers la gauche.

L'on détermine qu'aucune case ne doit être glissée ici, donc rien ne change.

Cas limites

- Entrée: {} , direction pas importante (`[]`)
- Sortie: {} (`[]`)

Cas simple

- Entrée: `{' ',' ',' ',' ','O'}` , direction vers la gauche (`[  O]`)
- Sortie: `{'O',' ',' ',' ',' '}` (`[O  ]`) ## Cas d'erreur
- Entrée: null, direction valide
- Sortie: null
- Entrée: `{'A','B',' ','#'}` , direction pas importante (`valeur invalide`)
- Sortie: null

Formalisation de tests

Incomplet * Étant donné … : * quand … , * alors … .

Comment

Le nom de cet algorithme m'est inconnu.

Cas simple

Limitons les paramètres à un tableau à 2 cases. (-> de taille 2)

Déjà, remarquons que lorsque l'on se retrouve avec certains tableaus, le résultat doit être identique:

- `[ ]` reste identique
- `[##]` reste identique
- `[# ]` reste identique
- `[ #]` reste identique

Ce ne sont pas tout les tableaus qui ne changent pas, mais ils sonts particuliers car ils ne contiennent pas un seul 'O'; car évidemment, seul les 'O' sonts capables de se mouvoir.

Continuons:

- `[O ]` reste identique
- `[#O]` reste identique
- `[O#]` reste identique
- `[OO]` reste identique
- `[##]` reste identique

Maintenant, l'on a listé tout les tableaus qui ne changent pas: le seul tableau changeant est donc `[ O]`.

Donc, pour déduire qu'un tableau de taille 2 n'ait pas besoin de changer, il suffit de déterminer que `tableau[0]` soit égal à un espace et que `tableau[1]` à '`O`'.

Appelons-cela le **cas actif**, et lorsque l'on change le **cas actif** (c'est à dire `[ O]` vers `[O ]`), on appellera cela un **glissement**.

Lorsque le tableau ne nécessite plus de **glissements** , l'on dira qu'il est dans un **état stable**.

Le **prédicat d'état stable** rend vrai si le tableau est dans un état stable, et faux sinon; c'est-à-dire vrai si le tableau n'est pas `[ O]`. ## Cas plus complexe: tableau à 3 cases

Dans le cas d'un tableau à trois cases, l'on peut dire que celui-ci contient un sous-tableau de 2 cases au début:

- `[O  ]` contient le sous-tableau `[O ]` en position `0`
- `[ O#]` contient le sous-tableau `[ O]` en position `0`
- etc.

Donc ça signifie que les 2 première cases du tableau à 3 cases n'ont besoin d'être **glissées** que lorsque celles si sonts dans le **cas actif**.

L'on réapplique des glissements au tableaux jusqu'à que le tableau de taille 3 soit dans un **état stable**; l'on peut vérifier ça avec le **prédicat d'état stable** en l'appliquant successivement aux sous-tableaux du tableau 3 (c'est à dire `tableau[0],tableau[1]` et `tableau[1],tableau[2]`) pour vérifier que chaque sous-tableau de taille 2 est dans un état stable.

Exemples

Premier exemple: prenons le tableau de taille 3 `[ OO]`:

1. Vérifier si le tableau `[ OO]` satisfait le **prédicat d'état stable**:
    1. Le sous-tableau `[ O]` en `tableau[0],tableau[1]` ne le satisfait pas.
2. Ne le satisfiant pas, on sait que l'on doit continuer d'opérer sur le tableau `[ OO]`:
3. **Glisser** le sous-tableau `[ O]` vers `[O ]`. (Le tableau de taille 3 est maintenant dans l'état `[O O]`.)
4. Le sous-tableau `tableau[1],tableau[2]` est maintenant de valeur `[ O]`.
5. **Glisser** ce sous-tableau `[ O]` vers `[O ]`. (Le tableau de taille 3 est maintenant dans l'état `[OO ]`.)
6. Revérifier si le tableau `[OO ]` satisfait le **prédicat d'état stable**:
    1. Le sous-tableau `[OO]` en `tableau[0],tableau[1]` le satisfait.
    2. Le sous-tableau `[O ]` en `tableau[1],tableau[2]` le satisfait. (Dernier sous-tableau a valider.)
7. Le prédicat satisfait; on sort de l'algorithme.

Autre exemple: prenons le tableau de taille 3 `[ O#]`:

1. Vérifier si le tableau `[ O#]` satisfait le **prédicat d'état stable**:
    1. Le sous-tableau `[ O]` en `tableau[0],tableau[1]` ne le satisfait pas.

2. Ne le satisfiant pas, on sait que l'on doit continuer d'opérer sur le tableau `[ O#]`:
3. **Glisser** le sous-tableau `[ O]` vers `[O ]`. (Le tableau de taille 3 est maintenant dans l'état `[O #]`.)
4. Le sous-tableau `tableau[1],tableau[2]` est maintenant de valeur `[ #]`.
5. Ce sous-tableau est dans l'**état stable** `[ #]`; l'on ne le change pas.
6. Revérifier si le tableau `[O #]` satisfait le **prédicat d'état stable**:
    1. Le sous-tableau `[O ]` en `tableau[0],tableau[1]` le satisfait.
    2. Le sous-tableau `[ #]` en `tableau[1],tableau[2]` le satisfait. (Dernier sous-tableau a valider.)
7. Le prédicat satisfait; on sort de l'algorithme.

Cas généralisé

Étendre l'algorithme pour un tableau à N cases est simple:

Dans le cas d'un tableau à N cases, l'on peut dire que celui-ci contient un sous-tableau de 2 cases au début:

- `[ O# OO ...` contient le sous-tableau `[ O]` en position `0`
- `[ O# OO ...` contient le sous-tableau `[O#]` en position `1`
- `[ O# OO ...` contient le sous-tableau `[# ]` en position `2`
- `[ O# OO ...` contient le sous-tableau `[# ]` en position `3`
- etc.

L'on réapplique des glissements au tableaux jusqu'à que le tableau de taille N soit dans un **état stable**; l'on peut vérifier ça avec le **prédicat d'état stable** en l'appliquant successivement aux sous-tableaux du tableau N (c'est à dire `tableau[0],tableau[1]`,`tableau[1],tableau[2]`,`tableau[2],tableau[3]`,...,jusqu'à `tableau[N-2],tableau[N-1]`) pour vérifier que chaque sous-tableau de taille 2 est dans un état stable. Si ça n'est pas le cas, on réapplique l'algorithme jusqu'à que le tableau de taille N se trouve dans un **état stable**.

Exemple: 15 cases

Le tableau à 15 cases suivant: `[  O## O #  OOOO]`

1. Le sous-tableau `[ O]` en `tableau[1],tableau[2]` n'est pas dans un **état stable**, donc le tableau n'est pas dans un **état stable**; appliquons l'algorithme.
2. Le sous-tableau en `tableau[0],tableau[1]` est dans un **état stable** et n'a pas besoin de **glisser**.
3. **Glisser** le sous-tableau `tableau[1],tableau[2]` (vers `[O ]`, donc) -> `[ O ## O #  OOOO]`
4. Les sous-tableaux en `tableau[2],tableau[3]`,`tableau[3],tableau[4]`,`tableau[4],tableau[5]` sonts dans des **états stables** et n'ont pas besoin de **glisser**.
5. **Glisser** le sous-tableau en `tableau[5],tableau[6]` -> `[ O ##O  # OOOO]`

6. Les sous-tableaux en `tableau[6],tableau[7]`,`tableau[7],tableau[8]`, `tableau[8],tableau[9]`, `tableau[9],tableau[10]` sonts dans des **états stables** et n'ont pas besoin de **glisser**.
7. **Glisser** le sous-tableau en `tableau[10],tableau[11]` ->`[ O ##O  # O OOO]`
8. **Glisser** le sous-tableau en `tableau[11],tableau[12]`-> `[ O ##O  # OO OO]`
9. **Glisser** le sous-tableau en `tableau[12],tableau[13]`-> `[ O ##O  # OOO O]`
10. **Glisser** le sous-tableau en `tableau[13],tableau[14]`->`[ O ##O  # OOOO ]`. **Fin du tableau (N=15, N-1=14.)**
11. Le sous-tableau `[ O]` en `tableau[0],tableau[1]` n'est pas dans un état stable, le prédicat d'état stable n'est donc pas satisfait. On répète:
12. `[0],[1]` est `[ O]`, glisser: `[O  ##O  # OOOO ]`.
13. `[10],[11]` est `[ O]`, glisser: `[O  ##O  #O OOO ]`.
14. `[11],[12]` est `[ O]`, glisser: `[O  ##O  #OO OO ]`.
15. `[12],[13]` est `[ O]`, glisser: `[O  ##O  #OOO O ]`.
16. `[13],[14]` est `[ O]`, glisser: `[O  ##O  #OOOO  ]`. **Fin du tableau (N=15, N-1=14.)**
17. Les sous-tableaux du tableau de taille 15 sonts tous dans des états stables; l'on sait donc que le tableau de taille 15 est dans un état stable. **Fini.**

Gérer la direction vers la droite?

Il suffit de faire mirroir; on remplace le **cas actif**, `[ O]`, par son opposée: `[O ]`.

Pseudocode

```
//Pseudocode pour l'algorithme:
/**
*Répond vrai si les deux cases d'un sous-tableau de taille 2 correspondent à un état stable
*/
fonction PrédicatÉtatStable(Caractère gauche, Caractère droite,Direction direction) {
    si (direction==Gauche) {
        retourne not (gauche==' ' et droite=='O')
        } sinon { //direction==Droite
        retourne not (gauche=='O' et droite==' ')
        }
}
/**
*Répond vrai si le tableau donné satisfait le cas actif.
*/
fonction PrédicatÉtatStable(Caractères tableau,Direction direction) {
         pour (Entier i = 0; i<tableau.Taille-1; i+=1) {
             si (not PrédicatÉtatStable(tableau[i],tableau[i+1],direction)) {
                 retourne false
```

```
                }
            }
            retourne true
}


fonction GlisserOVersDirection(Caractères tableau,Direction direction) {
    si (tableau == null)
        {retourne null} //Cas limite.
    si (tableau.Taille == 0)
        {retourne tableau} //Cas limite.
    pendant (not PrédicatÉtatStable(tableau,direction)) { //Pendant que le prédicat du cas d
        pour (Entier i = 0; i<tableau.Taille-1; i+=1) {
            Caractère soustableauGauche = tableau[i];
            Caractère soustableauDroite = tableau[i+1];
            si (not PrédicatÉtatStable(soustableauGauche,soustableauDroite,direction)) {
                tableau[i]= soustableauDroite;
                tableau[i+1] = soustableauGauche; //Inverser les caractères 'O' et ' ' (ou
                }
            }
        }
}
```