Logigraphe

Logigraphe est un jeu de réflexion célèbre. On considère une version simple unidimensionnelle de ce jeu. Dans ce jeu, on donne au joueur une ligne de n cellules. Les cellules sont numérotées de 0 à $n-1$ de gauche à droite. Le joueur doit peindre chaque cellule en noir ou en blanc. On utilise 'X' pour désigner les cellules noires et '_' pour désigner les cellules blanches.

On présente au joueur une séquence $c = [c_0, \ldots, c_{k-1}]$ de k entiers strictement positifs : les *indicateurs*. Il doit peindre les cellules de sorte que les cellules noires de la ligne forment exactement k blocs de cellules consécutives. De plus, le nombre de cellules noires dans l'i-ème bloc (numéroté à partir de 0) à partir de la gauche doit être égal à c_i. Par exemple, si les indicateurs sont $c = [3, 4]$, le jeu résolu doit contenir exactement deux blocs de cellules noires consécutives : une de longueur 3 et ensuite une autre de longueur 4. Par conséquent, si $n = 10$ et $c = [3, 4]$, une solution qui satisfait les indicateurs est "_XXX__XXXX". Notez que "XXXX_XXX__" ne vérifie pas les indicateurs car les blocs de cellules noires ne sont pas dans l'ordre correct. Aussi, "__XXXXXXX_" ne vérifie pas les indicateurs car il y a un seul bloc de cellules noires et non pas deux blocs séparés.

On vous donne un logigraphe partiellement résolu. C'est à dire que vous connaissez n et c, et vous savez de plus que certaines cellules doivent être noires et d'autres doivent être blanches. Votre tâche est de déduire des informations additionnelles à propos des cellules.

Plus précisément, une *solution valide* est une solution qui satisfait les indicateurs et qui correspond aux couleurs des cellules déjà connues.
Votre programme doit trouver les cellules peintes en noir dans toutes les solutions valides, et les cellules peintes en blanc dans toutes les solutions valides.

Vous pouvez supposer que l'entrée est telle qu'au moins une solution valide existe.

Détails de l'implémentation

Vous devez implémenter la fonction (méthode) suivante :

- `string solve_puzzle(string s, int[] c)`.
  - `s`: chaîne de caractères de longueur n. Pour chaque i ($0 \leq i \leq n-1$), le caractère i est :
    - 'X', si la cellule i doit être noire,
    - '_', si la cellule i doit être blanche,
    - '.', s'il n'y a aucune information sur la cellule i.
  - `c`: tableau de longueur k contenant les indicateurs, comme défini

auparavant,
    - la fonction doit retourner une chaîne de caractères de longueur n. Pour chaque i ($0 \leq i \leq n-1$) le caractère i de cette chaîne doit être :
        - 'X', si la cellule i est noire dans toute solution valide,
        - '_', si la cellule i est blanche dans toute solution valide,
        - '?', sinon (c-à-d., s'il existe deux solutions valides telles que cette cellule i est noire dans l'une d'elles et blanche dans l'autre).

Dans le langage C, la signature de la fonction est légèrement différente :

- `void solve_puzzle(int n, char* s, int k, int* c, char* result)`
    - `n` : longueur de la chaîne de caractères `s` (nombre de cellules),
    - `k` : longueur du tableau `c` (nombre d'indicateurs),
    - les autres paramètres sont les mêmes que précédemment,
    - au lieu de retourner une chaîne de n caractères, la fonction doit écrire la réponse dans la chaîne de caractères `result`.

Les codes ASCII des caractères employés dans ce problème sont :

- 'X' : 88,
- '_' : 95,
- '.' : 46,
- '?' : 63.

Veuillez utiliser le modèle des fichiers fourni pour les détails de l'implémentation dans votre langage de programmation.

Exemples

Exemple 1

```
solve_puzzle(".........", [3, 4])
```

Toutes les solutions valides possibles de ce puzzle sont :

- "XXX_XXXX__",
- "XXX__XXXX_",
- "XXX___XXXX",
- "_XXX_XXXX_",
- "_XXX__XXXX",
- "__XXX_XXXX".

On peut observer que les cellules ayant les indices (numérotés à partir de 0) 2, 6, et 7 sont noires dans chaque solution valide. Chacune des autres cellules peut être noire, mais ne l'est pas obligatoirement. Par conséquent la réponse correcte est "??X??? XX??".

Exemple 2

```
solve_puzzle(".......", [3, 4])
```

Dans cet exemple la solution est entièrement déterminée et unique. La réponse correcte est "XXX_XXXX".

Exemple 3

```
solve_puzzle("..._._....", [3])
```

Dans cet exemple on peut déduire que la cellule 4 doit être blanche aussi — il n'y a aucun moyen de mettre trois cellules noires consécutives entre les cellules blanches d'indices 3 et 5. Par conséquent, la réponse correcte est "`???___????`".

Exemple 4

```
solve_puzzle(".X........", [3])
```

Il y a uniquement deux solutions valides qui correspondent à la description ci-dessus :

- "`XXX_______`",
- "`_XXX______`".

Ainsi, la réponse correcte est "`?XX?______`".

Sous-tâches

Dans toutes les sous-tâches $1 \leq k \leq n$ et $1 \leq c_i \leq n$ pour chaque $0 \leq i \leq k-1$.

1. (7 points) $n \leq 20$, $k = 1$, s contient seulement '`.`' (jeu vide),
2. (3 points) $n \leq 20$, s contient seulement des '`.`',
3. (22 points) $n \leq 100$, s contient seulement des '`.`',
4. (27 points) $n \leq 100$, s contient seulement des '`.`' et des '`_`' (informations sur les cellules blanches uniquement),
5. (21 points) $n \leq 100$,
6. (10 points) $n \leq 5\,000$, $k \leq 100$,
7. (10 points) $n \leq 200\,000$, $k \leq 100$.

Évaluateur fourni (grader)

L'évaluateur fourni (grader) lit l'entrée selon le format suivant :

- ligne 1 : chaîne de caractères s,
- ligne 2 : entier k suivi par k entiers $c_0, \ldots, c_{k-1}$.