On zombie road again

1 le background

Devant l'échec commercial de son premier film « Le bison zombie », le dieu du calembour foireux décide de réaliser une suite, bien plus ambitieuse et totalement différente. Le pitch : un cow-boy avec un très mauvais sens de l'orientation remonte une ligne de chemin de fer pour rejoindre le lieu de son prochain contrat. Malheureusement pour lui, des poussins zombie, qui éclosent un peu au hasard, suivent la même voie en sens inverse. Pourquoi ? On s'en fout toujours, car c'est seulement le film numéro 2. Le cow-boy pourra-t-il survivre à son voyage en se cachant dans les gares le long de la voie ? Telle est de nouveau la question.

2 l'énoncé

Pour tester différent scénarios, le dieu fait appel directement à vous car la SSII lui a pris des frais intermédiaires indécents. Les différentes situations sont décrites par une chaîne de caractères de longueur N (variable selon les situations, mais $N \leq 10000$) composée de points, de caractères C pour les cachettes, d'un unique caractère H représentant l'humain, et de 0 à $N-1$ caractères Z représentant les zombies. Les ., C, H et Z peuvent se trouver n'importe où dans cette chaîne mais **jamais** au même endroit. Par exemple : `..H.CZC..Z..` ou encore `Z.ZH.Z.C`. Pour simuler le déplacement des intervenants, on utilise un pas de temps plutôt qu'un déplacement en continu. A chaque pas de temps, on suppose que les Z se déplacent de 1 caractère vers la gauche, H de 1 caractère vers la droite ou 0 s'il se cache. L'objectif de H est d'atteindre la fin de chaîne qui représente l'arrivée. La simulation doit déterminer si H va :

— croiser un zombie sans être caché, et donc mourir,
— croiser des zombies en étant chaque fois caché, et donc survivre,
— atteindre la fin de chaîne sans jamais avoir besoin de se cacher.

Important : les déplacements étant concurrents, on considère que l'humain peut se cacher s'il atteint une cachette exactement en même temps que le zombie. Il peut également atteindre la cachette avant et s'y cacher pendant plusieurs tours, c'est-à-dire le temps que le zombie atteigne lui-même la cachette. En revanche, si le zombie est juste après l'humain et que ce dernier bouge, alors il meurt forcément. Ces situations sont illustrées par les exemples suivants.

— `H.C.Z` : après 2 déplacements, H et Z arrivent au même moment sur C. H peut donc se cacher pendant tout un tour et survivre.

— `HC.Z` : après 1 déplacement, H arrive sur C et Z juste après. Le tour suivant, si H sort de la cachette, il rencontre Z et meurt. H doit donc se cacher pendant deux tours pour survivre.
— `H.CZ` : après 1 déplacement, Z arrive sur C, et H juste avant. Lors du déplacement suivant, H pourrait arriver sur C. Mais comme Z avance aussi, H n'a pas le temps de se cacher et meurt.

Pour tester les situations, votre programme doit lire sur l'entrée standard :

1. un entier M représentant le nombre de situations à tester
2. M chaînes de caractères au format décrit ci-dessus (avec ., C, H, Z), représentant chacune une situation.

Ensuite, votre programme doit déterminer pour chaque situation, le sort de l'humain et écrire sur la sortie standard :
— `dead` si l'humain meurt,
— `X` s'il peut atteindre l'arrivée en se cachant, `X` étant le nombre **minimal** de fois où il doit se cacher afin de survivre.
Le tableau 1 donne un exemple d'entrée et la sortie associée

entrée	sortie
`3`	`0`
`...C..Z..Z..H..C..`	`dead`
`...H.CZ.CC..Z`	`1`
`....H.C.ZCC...Z..`	

Table 1 – Exemple d'entrée et de sortie associée

3 les ressources

Pour vous aider dans la réalisation du programme, vous trouverez sur
`http://cours-info.iut-bm.univ-fcomte.fr`
un article dans la section `hackathon` portant le même titre que l'exercice. Il contient un lien permettant de télécharger un canevas de code, ainsi que le fichier d'entrée donné ci-dessus.

Bien entendu, vous êtes libres d'utiliser ou non ce canevas, mais c'est un gain de temps que de s'en servir comme base.