Projet Pluridisciplinaire d'Informatique Intégrative 2
Wordle
1A TELECOM Nancy - 2021-2022

Solveur Wordle

Document de conception

Table des matières

1 Préambule

Ce document fait suite à la conception de l'application WEB Wordle. Seules les caractéristiques du solveur seront présentées ici. La présentation générale du jeu Wordle ne sera pas rappelée.

2 Introduction au solveur Wordle

2.1 Rappel des consignes

Nous devons réaliser un solveur, c'est-à-dire une application qui résout des parties du jeu Wordle. Celui-ci doit utiliser des structures de données avancées et efficaces en mémoire et en temps. De plus, cette application doit être interactive. En effet, après avoir lu la longueur des mots à trouver dans un fichier, le solveur doit, à chaque étape, afficher le mot qu'il propose en sortie standard. L'utilisateur lui renvoie une chaîne de caractère composée de 0 (si la lettre n'est pas dans le mot), de 1 (si la lettre n'est pas à la bonne place) et de 2 (si la lettre est à la bonne place). L'application arrête de s'exécuter lorsque les mots cherchés ont été trouvés, ou que l'utilisateur stoppe la partie en tapant "-1" en entrée standard.

2.2 Les langages utilisés

Le solveur sera exclusivement implémenté en C.

2.3 Sources utilisées

Pour concevoir notre solveur, nous nous sommes basés sur les vidéos de 3Blue1Brown **[1] [2]** et de ScienceEtonnante **[3]**. Elles traitent de ce problème en utilisant la théorie de l'information.

3 Description du solveur

3.1 Les structures de données

Toutes les structures de données partagent une même variable globale `nb_letters` indiquant le nombre de lettres du mot à trouver.

3.1.1 L'Arbre

Cette structure va nous permettre de gagner du temps lors du calcul de l'entropie en élaguant rapidement les mots ne correspondant pas à un pattern donné.

Type Abr :

Opérations internes :

Creat_abr	:	Entier	$\longrightarrow$	Abr
Enleve_branche	:	Abr $\times$ Lettre	$\longrightarrow$	Abr
Ajout_branche	:	Abr $\times$ Lettre	$\longrightarrow$	Abr

Observateurs :

Empty	:	Abr	$\longrightarrow$	Mot
Get_branche	:	Abr $\times$ Lettre	$\longrightarrow$	Entier
Nb_mots	:	Abr	$\longrightarrow$	Entier
Profondeur	:	Abr	$\longrightarrow$	Entier

Préconditions :

Creat_abr(i) est défini	$\Leftrightarrow$	$0 \leq i \leq$ nb_letters
Get_branche(A,a) est défini	$\Leftrightarrow$	a $\in [\![$'A','Z'$]\!]$ et $\neg$(Empty(A))
Enleve_branche(A,a) est défini	$\Leftrightarrow$	a $\in [\![$'A','Z'$]\!]$ et $\neg$(Empty(Get_branche(A,a)))
Ajout_branche(A,a) est défini	$\Leftrightarrow$	Profondeur(A) $<$ nb_letters et $\neg$(Empty(A))

Axiomes :

Notations compactes :

- n = nb_letters
- Rm_b(A,a) $\leftrightarrow$ Enleve_branche(A,a)
- Add_b(A,a) $\leftrightarrow$ Ajout_branche(A,a)
- Get_b(A,a) $\leftrightarrow$ Get_branche(A,a)
- P(A) $\leftrightarrow$ Profondeur(A)

Empty() :

Empty(Creat_abr(i)) = Vrai
Empty(Get_b(Creat_abr(i)),a) = Vrai $\quad \forall a \in [\![\text{'A','Z'}]\!]$
Empty(Rm_b(A,a)) = Faux
Empty(Add_b(A,a)) = Faux

Profondeur() :

Profondeur(Creat_abr(i)) = i
Profondeur(Rm_b(A,a)) = P(A)
Profondeur(Add_b(A,a)) = P(A)

Nb_mots() :

Nb_mots(Creat_abr(i)) = 0
Nb_mots(Rm_b(A,a)) = Nb_mots(A) − Nb_mots(Get_b(A,a))
$$\text{Nb\_mots(Add\_b(A,a))} = \begin{cases} \text{si P(A)} = n-1 & \rightarrow \text{ Nb\_mots(A)} + 1 \\ \text{sinon} & \rightarrow \text{ Nb\_mots(A)} \end{cases}$$

Get_b() :

Get_b(Creat_abr(i),a) $\rightarrow$ Non défini
$$\text{Get\_b(Rm\_b(A,}a_1\text{),}a_2\text{)} = \begin{cases} \text{si } a_1 \neq a_2 \rightarrow \text{Get\_b(A,}a_2\text{)} \\ \text{si } a_1 = a_2 \rightarrow \text{Non défini} \end{cases}$$
$$\text{Get\_b(Add\_b(A,}a_1\text{),}a_2\text{)} = \begin{cases} \text{si } a_1 \neq a_2 \rightarrow \text{Get\_b(A,}a_1\text{)} \\ \text{si } a_1 = a_2 \rightarrow \text{Creat\_abr(P(A)} + 1\text{)} \end{cases}$$

3.1.2 Le Dictionnaire

Le dictionnaire doit pouvoir être mis à jour tout au long de la partie. Des mots lui seront retirés au cours de celle-ci, tout en gardant une complexité d'accès en O(1). Nous avons donc opté pour une liste doublement chaînée contiguë.

Dans la spécification algébrique du dictionnaire ci-dessous, nous présenterons tout d'abord la structure de cellule. Celle-ci sert de base à la structure du dictionnaire.

Type C

Opérations internes :

Init_cell : Entier × Mot ⟶ C

Observateurs :

---

getMot : C ⟶ Mot
getInt : C ⟶ Entier

Préconditions du Type C :

Pas de pré-conditions particulières

Axiomes du Type C :

getMot() :

---

getMot(Init_cell(i,s)) = s

getInt() :

---

getInt(Init_cell(i,s)) = i

Type D<C,I>

Opérations internes :

Init_dico	:		$\longrightarrow$	D<C,I>
Suppr_dico	:	D<C,I> $\times$ Entier	$\longrightarrow$	D<C,I>

Observateurs :

Size	:	D<C,I>	$\longrightarrow$	Entier
getNbWord	:	D<C,I>	$\longrightarrow$	Entier
getWord	:	D<C,I> $\times$ Entier	$\longrightarrow$	Mot
getCell	:	D<C,I> $\times$ Entier	$\longrightarrow$	C
getIsSuppr	:	D<C,I> $\times$ Entier	$\longrightarrow$	Bool

Préconditions du Type D<C,I> :

Suppr_dico(D,i) est défini	$\Leftrightarrow$	$0 \leq i \leq Size(D)-1$ et $\neg$(getIsSuppr(D,i))
getWord(D,i) est défini	$\Leftrightarrow$	$0 \leq i \leq Size(D)-1$ et $\neg$(getIsSuppr(D,i))
getCell(D,i) est défini	$\Leftrightarrow$	$0 \leq i \leq Size(D)-1$ et $\neg$(getIsSuppr(D,i))
getIsSuppr(D,i) est défini	$\Leftrightarrow$	$0 \leq i \leq Size(D)-1$

Axiomes du Type D<C,I> :

Notations compactes :

- n = nb_letters
- m est la taille du dictionnaire chargé pour les mots de n lettres.

Size() :

Size(Init_dico()) = m
Size(Suppr_dico(D,i)) = m

getNbWord() :

---

getNbWord(Init_dico()) = m
getNbWord(Suppr_dico(D,i)) = getNbWord(D)−1

getWord() :

---

getWord(Init_dico(),i) = getMot(getCell(D,i))
getWord(Suppr_dico(D,i),j) = $\begin{cases} \text{si } i = j & \rightarrow \text{ Non défini} \\ \text{sinon} & \rightarrow \text{ getWord(D,j)} \end{cases}$

getCell() :

---

getCell(Init_dico(),i) = getCell(D,i)
getCell(Suppr_dico(D,i),j) = $\begin{cases} \text{si } i = j & \rightarrow \text{ Non défini} \\ \text{sinon} & \rightarrow \text{ getCell(D,j)} \end{cases}$

getIsSuppr() :

---

getIsSuppr(Init_dico(),i) = Faux
getIsSuppr(Suppr_dico(D,i),j) = $\begin{cases} \text{si } i = j & \rightarrow \text{ Vrai} \\ \text{sinon} & \rightarrow \text{ Faux} \end{cases}$

3.1.3 La Table d'occurrences

La réponse de l'utilisateur donne 2 types d'information :

— Une sur la position des lettres,

— Une sur la présence des lettres dans le mot.

L'arbre permet de filtrer les mots en raisonnant sur la position des lettres. Mais pour que le filtrage soit complet, il faut ajouter une structure qui raisonne sur le nombre d'occurrences de chaque lettre. C'est le rôle de la table d'occurrence.

Type Occ_table

Opérations internes :

Init_table	:		$\longrightarrow$	Occ_table
Change_min	:	Occ_table $\times$ Lettre $\times$ Entier	$\longrightarrow$	Occ_table
Change_max	:	Occ_table $\times$ Lettre $\times$ Entier	$\longrightarrow$	Occ_table
Fusion_table	:	Occ_table $\times$ Occ_table	$\longrightarrow$	Occ_table

Observateurs :

Get_min	:	Occ_table $\times$ Lettre	$\longrightarrow$	Entier
Get_max	:	Occ_table $\times$ Lettre	$\longrightarrow$	Entier
Trouver	:	Occ_table $\times$ Lettre	$\longrightarrow$	Bool
Sum_min	:	Occ_table	$\longrightarrow$	Entier

Préconditions :

Get_min(T,a) est défini	$\Leftrightarrow$	a $\in [\![$'A','Z'$]\!]$
Get_max(T,a) est défini	$\Leftrightarrow$	a $\in [\![$'A','Z'$]\!]$
Trouver(T,a) est défini	$\Leftrightarrow$	a $\in [\![$'A','Z'$]\!]$
Change_min(T,a,i) est défini	$\Leftrightarrow$	a $\in [\![$'A','Z'$]\!]$ et i $\geq$ (Get_min(T,a))
Change_min(T,a,i) est défini	$\Leftrightarrow$	a $\in [\![$'A','Z'$]\!]$ et i $\leq$ (Get_max(T,a))

Axiomes :

Notations compactes :

— n = nb_letters

— F(T1,T2) $\leftrightarrow$ Fusion_table(T_1,T_2)

— GetM(T,a) $\leftrightarrow$ Get_Max(T,a)

— Getm(T,a) $\leftrightarrow$ Get_Min(T,a)

Sum_min() :

Sum_min(Init_table(),a) = 0
Sum_min(Change_min(T,a,i) = i − Getm(T,a) + Sum_min(T)
Sum_min(Change_max(T,a,i) = Sum_min(T)
Sum_min(F(T_1,T_2)) = $\sum_{\text{a='A'}}^{\text{'Z'}}$ Getm(F(T_1,T_2),a)

Getm() :

Getm(Init_table(),a) = 0

$$\text{Getm(Change\_min(T,}a_1\text{,i)},a_2) = \begin{cases} \text{si } a_1 = a_2 \rightarrow i \\ \text{si } a_1 \neq a_2 \rightarrow \text{Getm(T,}a_2\text{)} \end{cases}$$

Getm(Change_max(T,a_1,i),a_2) = Getm(T,a_2)
Getm(F(T_1,T_2),a) = max(Getm(T_1,a),Getm(T_2,a))

GetM() :

GetM(Init_table(),a) = n

$$\text{GetM(Change\_min(T,}a_1\text{,i)},a_2) = \begin{cases} \text{si } a_1 = a_2 \rightarrow \text{Getm(T,}a_2\text{)} \\ \text{si } a_1 \neq a_2 \rightarrow \text{Getm(T,}a_2\text{)} - (\text{Getm(T,}a_1\text{)} - i) \end{cases}$$

$$\text{GetM(Change\_min(T,}a_1\text{,i)},a_2) = \begin{cases} \text{si } a_1 = a_2 \rightarrow i \\ \text{si } a_1 \neq a_2 \rightarrow \text{Getm(T,}a_2\text{)} \end{cases}$$

$$\text{GetM(F(}T_1\text{,}T_2\text{),a)} = min \left\{ \begin{array}{c} \text{Getm}(T_1,a) \\ \text{Getm}(T_2,a) \\ n - \text{Getm(F}(T_1,T_2)\text{,a} + \text{Sum\_min(F}(T_1,T_2)) \end{array} \right\}$$

Trouver() :

Trouver(Init_table(),a) = Faux

$$\text{Trouver(Change\_min(T,}a_1\text{,i)},a_2) = \begin{cases} \text{si } a_1 = a_2 \rightarrow i == \text{GetM(T,}a_2\text{)} \\ \text{si } a_1 \neq a_2 \rightarrow \text{Getm(T,}a_2\text{)} == \text{GetM(T,}a_2\text{)} \end{cases}$$

$$\text{Trouver(Change\_max(T,}a_1\text{,i)},a_2) = \begin{cases} \text{si } a_1 = a_2 \rightarrow \text{Getm(T,}a_2\text{)} == i \\ \text{si } a_1 \neq a_2 \rightarrow \text{Getm(T,}a_2\text{)} == \text{GetM(T,}a_2\text{)} \end{cases}$$

Trouver(F(T_1,T_2),a) = Getm(F(T_1,T_2),a) == GetM(F(T_1,T_2),a)

3.2 Fonctionnement global du solveur

La fonction `main()` récupère tout d'abord le nombre de lettres de la partie en lisant le fichier `wsolf.txt`, ainsi que le premier mot que propose le solveur. Ce mot aura été pré-calculé pour les mots entre 1 et 10 lettres dans le fichier openers.txt. Au-delà le 10 lettres, on se contente du premier mot du dictionnaire pour des raisons de complexité.

Ce mot est affiché en sortie standard.

Puis,, les structures de données sont initialisées, et on peut ensuite entrer dans la boucle principale : tant que la solution n'est pas trouvée ou que l'utilisateur ne décide pas d'en sortir en tapant "-1", on lance le calcul du prochain mot à proposer.

On va chercher dans un premier temps à déterminer les possibilités restantes. Pour cela, on va :

1. Mettre à jour la table d'occurrence : à partir de la nouvelle information, on peut faire évoluer le nombre maximum et minimum d'occurrences de chaque lettre dans le mot à trouver.
2. Mettre à jour l'arbre : on supprime toutes les branches qui mènent vers un mot non-valide grâce à la nouvelle table d'occurrence.
3. Mettre à jour le dictionnaire : on supprime du dictionnaire les mots que l'on estime inutiles pour le calcul du meilleur mot du coup suivant.

Références

[1] 3Blue1Brown, "Solving Wordle using information theory." https://www.youtube.com/watch?v=v68zYyaEmEA.

[2] 3Blue1Brown, "Oh, wait, actually the best Wordle opener is not "crane"...." https://www.youtube.com/watch?v=fRed0Xmc2Wg.

[3] ScienceEtonnante, "JE CRAQUE WORDLE ! (grâce aux maths)." https://www.youtube.com/watch?v=iw4_7ioHWF4.