Rapport de projet pour le cours Programmation impérative

Florian Lefebvre, Ennabili Younes

April 8, 2021

Contents

1 Cahier des charges du projet

Nous allons ici présenter la problématique du projet et donner les objectifs que nous nous sommes fixés.

1.1 Le sujet du projet : Grille d'anagrammes

Soit une grille d'anagrammes (c'est-à-dire de mots croisés dont on connaît non pas une définition mais la liste des lettres). On suppose l'existence d'un dictionnaire (qu'il faudra, en partie, construire ou récupérer). Écrire un programme qui trouve la solution de ces anagrammes. Dans un premier temps le dictionnaire sera limité et la taille de la grille aussi. Dans un second

temps ces éléments pourront grandir jusqu'à être en mesure de résoudre les anagrammes des journaux spécialisés. Il serait bon d'utiliser un fichier pour gérer le dictionnaire.

1.2 La définition d'un anagramme selon Wikipédia

Une anagramme est une sorte de jeu sur les mots, qui permute les lettres d'un mot pour en extraire un mot nouveau, ou d'un groupe de mots pour en extraire un sens nouveau. Dans la plupart des cas, il est sous-entendu que ce nouveau mot existe dans un dictionnaire, ou que la nouvelle expression signifie quelque chose. Mais la méthode est aussi pratiquée pour créer des mots qui n'existent pas (ou des pseudonymes), voir pour cacher le sens d'un texte comme en cryptographie.

2 Contexte du projet

2.1 Notre démarche globale

Nous voulons au départ tester des anagrammes sur des exemples simples sans avoir recours à des grilles. La première version de notre projet est d'être capable à partir d'un mot et d'un dictionnaire de générer tous les anagrammes de ce mot. La deuxième étape est la gestion assez complexe d'une grille afin de pouvoir reconnaître les anagrammes et de résoudre la grille. Nous avions l'intention de proposer une interface graphique mais nous avons vite compris que ce n'était pas cela le plus important, mais plutôt de pouvoir définir la démarche permettant de découvrir les solutions d'une grille.

2.2 Les outils utilisés

Le projet est développé en langage C sous Linux. Rapidement, nous avons eu la nécessité de chercher un dictionnaire nous permettant de former des anagrammes à partir d'un mot existant. Le premier ditionnaire (`https://www.pallier.org/extra/liste.de.mots.francais.frgut.txt`) est un dictionnaire de liste de mots avec conjugaisons, pluriels, accents, mots-composés, mais sans noms propres (UTF8). Nous avons trouvé le même dictionnaire sans accents, sans tirets, tous les mots en majuscules. C'est celui que nous avons choisi pour réduire la difficulté de traitement. Il est constitué de plus de 320000 mots.

En ce qui concerne les grilles, nous avons utilisé le site (`https://www.notretemps.com/jeux/jeux-en-ligne/anagrammes-juillet-2020-gratuits-5`,

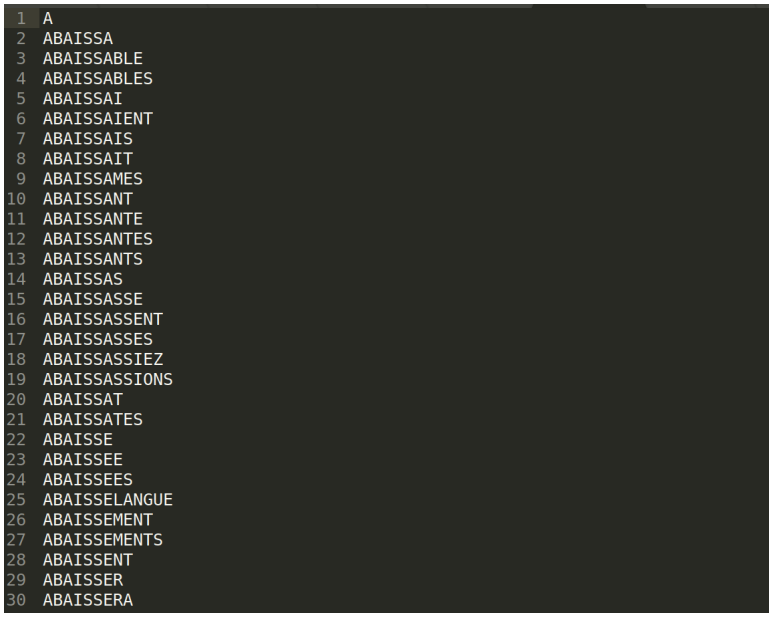

Figure 1: Les trente premiers mots du dictionnaire

`i221525`.

3 Implémentation du projet

3.1 Génération d'anagrammes

Pour effectuer cette première version de l'application, visant à générer tous les anagrammes d'un mot, nous avons tout d'abord charger le dictionnaire en mémoire. Voici ce que donne un exemple sur quelques mots.

Pour cela, nous avons utilisé un buffer de la taille du dictionnaire et un tableau contenant autant de mot que d'entrée dans le dictionnaire. Le but de ce tableau de mot étant d'associé à chaque mot la position de celui-ci dans le buffer. L'analyse du fichier du dictionnaire avec l'outil Unix od nous à permis de voir la frontière des mots, comme le montre la figure suivante.

On s'aperçoit que la frontière d'un mot est le caractère spéciale ''̊. Nous avons pu ainsi associer à chaque mot du dictionnaire, sa position dans le buffer et ainsi par la suite, retrouver facilement un mot grâce à sa position. Ensuite, pour pouvoir générer les anagrammes d'un mot, nous demandons à l'utilisateur un mot, puis nous parcourons le dictionnaire des mots, et pour chaque couple, nous vérifions si les deux mots sont des anagrammes possibles. L'algorithme permet de véfier que les deux mots ont des fréquences de lettres

```
Entrez une chaîne
ancre
Les chaînes ANCRE et CANER sont des anagrammes.
Les chaînes ANCRE et CARNE sont des anagrammes.
Les chaînes ANCRE et CERNA sont des anagrammes.
Les chaînes ANCRE et CRANE sont des anagrammes.
Les chaînes ANCRE et CRENA sont des anagrammes.
Les chaînes ANCRE et ECRAN sont des anagrammes.
Les chaînes ANCRE et ENCRA sont des anagrammes.
Les chaînes ANCRE et NACRE sont des anagrammes.
Les chaînes ANCRE et RANCE sont des anagrammes.
Entrez une chaîne
chien
Les chaînes CHIEN et CHINE sont des anagrammes.
Les chaînes CHIEN et NICHE sont des anagrammes.
Entrez une chaîne
sapin
Les chaînes SAPIN et PAINS sont des anagrammes.
Les chaînes SAPIN et PIANS sont des anagrammes.
Entrez une chaîne
salive
Les chaînes SALIVE et LEVAIS sont des anagrammes.
Les chaînes SALIVE et VALISE sont des anagrammes.
Les chaînes SALIVE et VELAIS sont des anagrammes.
Entrez une chaîne
```

Figure 2: Première version du projet

```
0000000   A  \r  \n   A   B   A   I   S   S   A  \r  \n   A   B   A   I
0000020   S   S   A   B   L   E  \r  \n   A   B   A   I   S   S   A   B
0000040   L   E   S  \r  \n   A   B   A   I   S   S   A   I  \r  \n   A
0000060   B   A   I   S   S   A   I   E   N   T  \r  \n   A   B   A   I
0000100   S   S   A   I   S  \r  \n   A   B   A   I   S   S   A   I   T
0000120  \r  \n   A   B   A   I   S   S   A   M   E   S  \r  \n   A   B
0000140   A   I   S   S   A   N   T  \r  \n   A   B   A   I   S   S   A
0000160   N   T   E  \r  \n   A   B   A   I   S   S   A   N   T   E   S
0000200  \r  \n   A   B   A   I   S   S   A   N   T   S  \r  \n   A   B
0000220   A   I   S   S   A   S  \r  \n   A   B   A   I   S   S   A   S
0000240   S   E  \r  \n   A   B   A   I   S   S   A   S   S   E   N   T
0000260  \r  \n   A   B   A   I   S   S   A   S   S   E   S  \r  \n   A
0000300   B   A   I   S   S   A   S   S   I   E   Z  \r  \n   A   B   A
0000320   I   S   S   A   S   S   I   O   N   S  \r  \n   A   B   A   I
0000340   S   S   A   T  \r  \n   A   B   A   I   S   S   A   T   E   S
0000360  \r  \n   A   B   A   I   S   S   E  \r  \n   A   B   A   I   S
0000400   S   E   E  \r  \n   A   B   A   I   S   S   E   E   S  \r  \n
0000420   A   B   A   I   S   S   E   L   A   N   G   U   E  \r  \n   A
0000440   B   A   I   S   S   E   M   E   N   T  \r  \n   A   B   A   I
0000460   S   S   E   M   E   N   T   S  \r  \n   A   B   A   I   S   S
0000500   E   N   T  \r  \n   A   B   A   I   S   S   E   R  \r  \n   A
0000520   B   A   I   S   S   E   R   A  \r  \n   A   B   A   I   S   S
0000540   E   R   A   I  \r  \n   A   B   A   I   S   S   E   R   A   I
0000560   E   N   T  \r  \n   A   B   A   I   S   S   E   R   A   I   S
0000600  \r  \n   A   B   A   I   S   S   E   R   A   I   T  \r  \n   A
0000620   B   A   I   S   S   E   R   A   S  \r  \n   A   B   A   I   S
0000640   S   E   R   E   N   T  \r  \n   A   B   A   I   S   S   E   R
0000660   E   Z  \r  \n   A   B   A   I   S   S   E   R   I   E   Z  \r
```

Figure 3: Exemple de dump du fichier du dictionnaire

identiques. Si c'est le cas, ils sont des anagrammes. Pour cela, deux tableaux sont utilisés, un pour chaque mot et lors du parcours de chaque mot, chaque tableau incréménte la lettre lue dans la position de celle-ci dans le tableau.

A la fin, si les deux tableaux sont identiques, nous avons bien à faire à deux anagrammes. Voici le code de cette algorithme crucial pour le projet.

```c
#include <stdio.h>

// Algorithme permettant de véfier si deux mots sont des anagrammes

int isAnagram(char str1[], char str2[])
{
  int tab1[26] = {0}, tab2[26] = {0}, i=0;

  for (i = 0; str1[i]!='\0'; i++) {
      if(str1[i] >= 'a' && str1[i] <= 'z') {
         str1[i] = str1[i] -32;
      }
   }
  i=0;
  while (str1[i] != '\0')
  {
    tab1[str1[i]-'A']++;
    i++;
  }

  i = 0;

  while (str2[i] != '\0')
  {
    tab2[str2[i]-'A']++;
    i++;
  }
  for (i = 0; i < 26; i++)
  {
    if (tab1[i] != tab2[i])
      return 0;
  }
  return 1;
}

```

Figure 4: Algorithme pour savoir si deux mots sont des anagrammes

3.2 Résolution d'une grille d'anagramme

La deuxième version de ce projet a pour but de faire résoudre à l'ordinateur les anagrammes d'une grille. Nous sommes partie d'une grille assez complexe pour faire nos tests qui est la suivante :

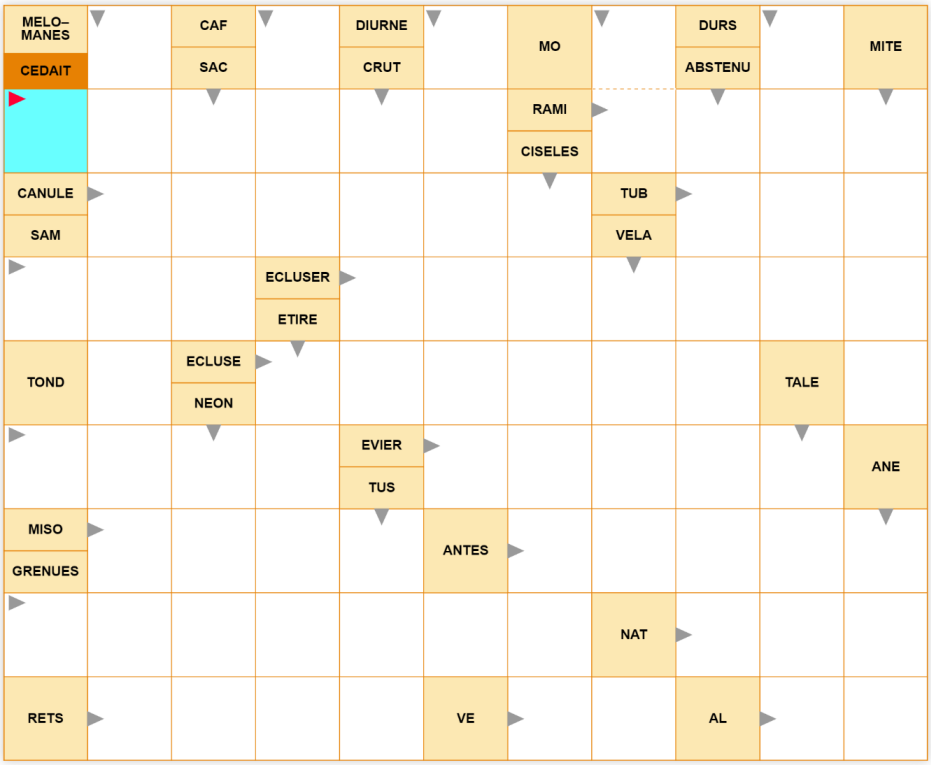

Figure 5: La grille de test

Au départ, il a fallut pouvoir trouver une structure de données permettant de représenter les contraintes définies par cette grille. Nous avons choisi d'utiliser un fichier avec un formalisme très simple qui est le suivant pour chaque mot de la grille : "mot colonne ligne H/V" (Horizontal ou vertical. Cette représentation permet de représenter de nombreuses grilles en étant assez simple à charger dans la mémoire de l'ordinateur.

Voici la représentation de la grille ci-dessus dans notre fichier :

L'étape suivante consiste à générer un dictionnaire à partir des mots que nous avons lus dans la structure décrivant la grille de test. Chaque mot doit être associé à tous ses anagrammes. L'avantage de cette représentation est de permettre un accès direct aux données. La clef de recherche subbit une transformation arithmétique grâce à une fonction de hachage pour fournir un indice appelé code de hachage. Ce dernier donne accès à la donnée située dans la table de hachage. Si le type de la clef est variable,le code de hachage est toujours un entier. Il indique le numéro de la case de la table de hachage où est localisée la donnée. N'ayant pas vu cette structure en cours, nous nous sommes documentés et nous avons utilisés et adapté à notre contexte, des

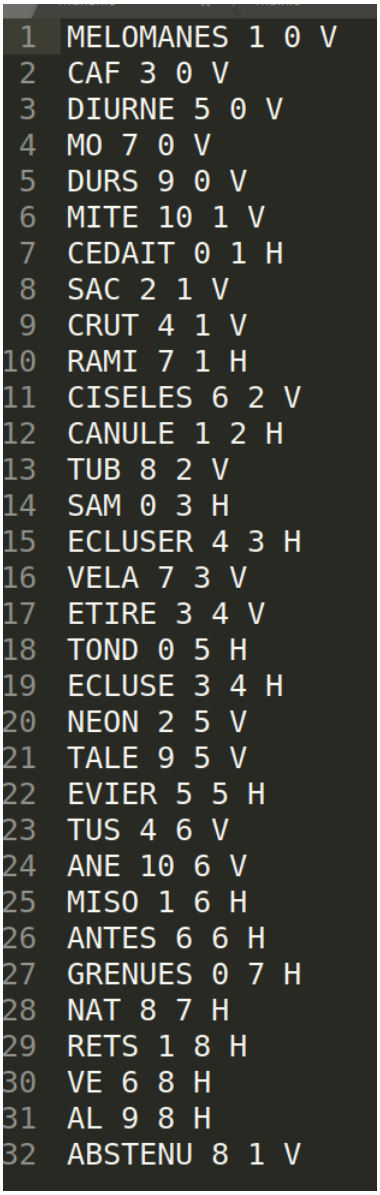

Figure 6: La représentation d'une grille

algorithmes décrits dans le livre : "Algorithmique en C, C++, Java, Python et Php" de Jean-Michel Léry dans la collection Ellipses. Voici ci-dessous un extrait de ce dictionnaire généré à partir des mots de la grille:

La prochaine étape est d'exploiter ce dictionnaire pour générer un produit cartésien de toutes les possibilités de notre dictionnaire. Par exemple, dans notre contexte où il y a 32 mots à découvrir et sachant que chaque mot

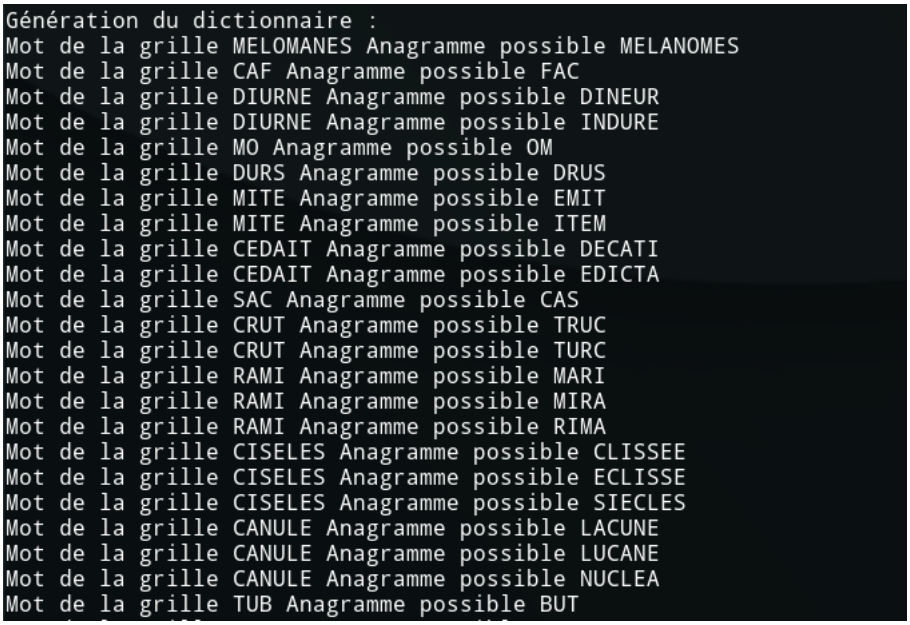

Figure 7: Extrait du dictionnaire

a des anagrammes, cela représentent 2654208 combinaisons possible. Cet algorithme a été difficile à implémenter car il a fallu générer dynamiquement des combinaisons en partant du dictionnaire. La question que l'on se posait était si le nombre de combinaisons n'allait pas engendrer un temps de réponse trop long. Dans le cas de figure que nous avons choisi, le résultat était obtenu progressivement au bout de quelques minutes. Chaque combinaison générée est testée dans la grille. Si un mot est en collision avec un autre mot, celui-ci est retirée en revenant à la grille précédente. A la fin du test de la combinaison, un nombre de mot s'intégrant dans la grille est trouvé et si celui-ci est supérieur à la valeur maximale d'une combinaison précédemment obtenue, la grille est affichée, et ce nombre devient la valeur maximale. Le résultat dépend de l'existence de chaque mot de la grille dans le dictionnaire que nous utilisons.

Voici ci-dessous la grille solution. Elle démontre l'intérêt d'utiliser un dictionnaire car la plupart des mots sont reconnus par celui-ci.

Voici ci-dessous les trois dernières évolutions de notre programme.

4 Conclusions

Le programme que nous avons développé est pour l'instant un prototype qui nécessite plus de tests et des pistes d'améliorations sont prévues. L'une

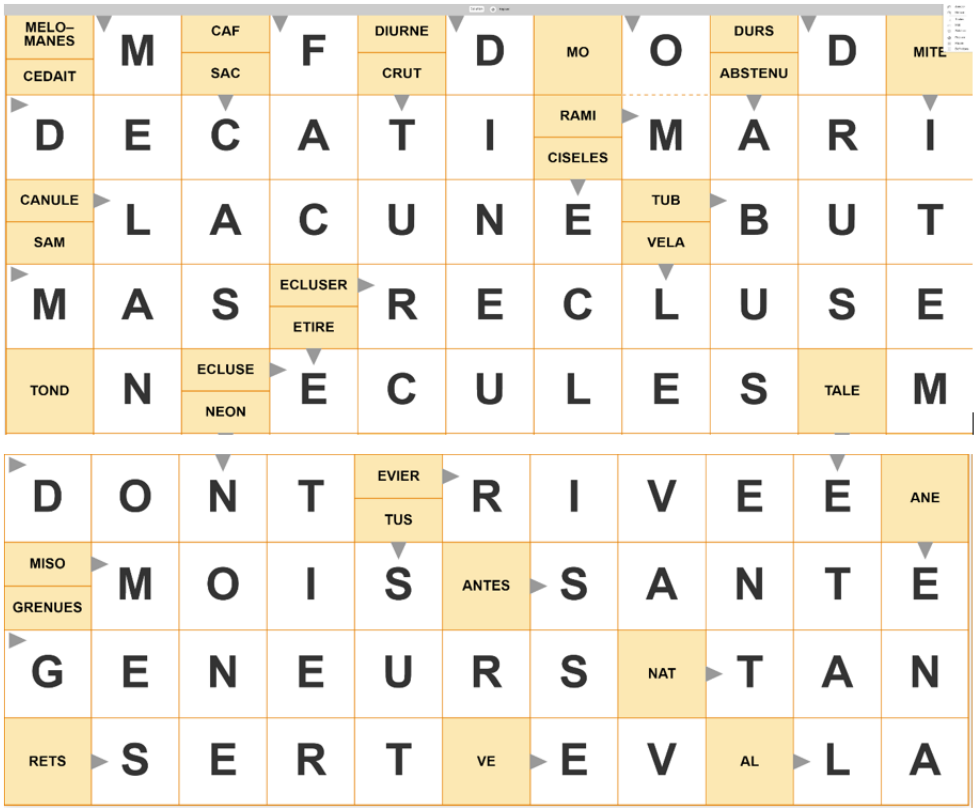

Figure 8: La grille solution

Figure 9: Evolution 1 de la grille

d'entre elles est de prendre en compte des mots qui ne sont pas dans le dictionnaire mais malgré tout de les exploiter en créant une combinatoire des lettres et d'essayer de les intégrer. Une autre possibilité serait de placer dans la grille dès le départ des mots à priori sûrs afin d'éviter d'accepter des mots

Figure 10: Evolution 2 de la grille

Figure 11: Evolution 3 de la grille

qui sont intégrables mais sans prendre en compte le contexte existant. Les interfaces en mode console sont très rudimentaires et rendent difficiles la lisibilité des informations. Nous avons commencé à développer une version en mode graphique avec sdl image 1.2. Voici ci-dessous une copie d'écran du résultat. Nous aurions aimé terminé cette interface mais ils nous a manqué malheureusement un peu de temps. Nous allons essayer de le faire prochainement. Ce projet nous amène à prendre conscience de l'importance de la démarche de résolution de problèmes et nous pensons que les choix que nous avons effectués sont perfectibles, très formateurs pour notre apprentissage et encourageant dans des projets à venir.

Figure 12: Une interface graphique pour le projet