Génération d'acronymes

Épreuve pratique d'algorithmique et de programmation
Concours commun des Écoles normales supérieures

Durée de l'épreuve : 3 heures 30 minutes

Juin 2025

ATTENTION !

N'oubliez en aucun cas de recopier votre u_0
à l'emplacement prévu sur votre fiche réponse

Important.

Il vous a été donné un numéro u_0 qui identifie les fichiers d'entrée pour vos codes. Ceux-ci se trouvent dans un répertoire `data/u0/`, où `u0` est remplacé par votre numéro u_0. Les réponses attendues sont généralement courtes et doivent être données sur la fiche réponse fournie à la fin du sujet. À la fin du sujet, vous trouverez en fait deux fiches réponses. La première est un exemple des réponses attendues pour le numéro particulier $\widetilde{u_0}$. Cette fiche est destinée à vous aider à vérifier le résultat de vos programmes. Vous indiquerez vos réponses (correspondant à votre u_0) sur la seconde et vous la remettrez à l'examinateur à la fin de l'épreuve.

En ce qui concerne la partie orale de l'examen, lorsque la description d'un algorithme est demandée, vous devez présenter son fonctionnement de façon schématique, courte et précise. Vous ne devez en aucun cas recopier le code de vos procédures !

Quand on demande la complexité en temps ou en mémoire d'un algorithme en fonction d'un paramètre n, on demande l'ordre de grandeur en fonction du paramètre, par exemple : $O(n^2)$, $O(n \log n)$, etc. La lecture de l'instance par le code qui vous a été fourni ne sera pas prise en compte dans la complexité.

Il est recommandé de commencer par lancer vos programmes sur de petites valeurs des paramètres et de ***tester vos programmes sur des petits exemples que vous aurez résolus préalablement à la main ou bien à l'aide de la fiche réponse type fournie en annexe.***

Enfin, il est recommandé de lire l'intégralité du sujet avant de commencer afin d'effectuer les bons choix de structures de données dès le début.

La partie 1 doit être implémentée en C (à l'aide du fichier `base_1.c`), et les parties 2 et 3 doivent être implémentées en OCaml (à l'aide des fichiers `base_2.ml` et `base_3.ml`). Ces trois parties peuvent être traitées de manière indépendante.
Les fichiers susmentionnés vous sont fournis dans un dossier de base. Ce dossier contient également un sous-dossier `data/` dans lequel se trouvent les jeux de tests pour les différentes parties. Il est recommandé de garder une sauvegarde de tous les fichiers fournis, au cas où ceux-ci seraient modifiés par erreur.
Il est demandé de nous fournir sur votre clef USB vos fichiers sources. Ces consignes doivent impérativement être suivies.

Introduction

Dans ce sujet, nous souhaitons générer des acronymes à partir d'une séquence de mots. Plus précisément, étant donnés l'ensemble des mots du dictionnaire $\mathcal{D}$ et une séquence de mots p (ci-après dénommée *phrase*) constituée de mots de $\mathcal{D}$, nous souhaitons produire un acronyme $a \in \mathcal{D}$ correspondant à cette séquence de mots. Une phrase p contenant k mots est de la forme $m_1, \ldots, m_k$, et chacun des mot m_i est composé de lettre $\ell_1, \ldots, \ell_n$ (où n est la longueur de m_i). Si une lettre ℓ apparaît dans un mot m, nous notons $\ell \in m$.
Dans les différentes sections de ce sujet, vous devez implémenter plusieurs algorithmes permettant de générer des variantes d'acronymes.

1 Acronymes exacts

Dans cette première partie que vous traiterez en langage C , vous devez générer des acronymes que nous appellerons *exacts*. Il s'agit d'acronymes pour lesquels l'ordre des mots de la phrase considérée est respecté, et où chaque mot de la phrase est associé à exactement une lettre de l'acronyme, et inversement. Plus précisément, pour une phrase $p = m_1 \ldots m_n$, l'acronyme $a \in \mathcal{D}$ que vous devez identifier doit être de la forme $\ell_1 \ldots \ell_n$, avec $\ell_i \in m_i$ pour tout $i \in \{1, \ldots, n\}$.
Pour vous aider dans cette tâche, nous vous fournissons dans le fichier `base_1.c` la fonction `lireDictionnaire` qui vous permet de lire le dictionnaire $\mathcal{D}$ à partir d'un fichier texte. Cette fonction est donnée par :

```
int lireDictionnaire(char *nomFichier, char dico[MAX_MOTS][TAILLE_MAX_MOT]);
```

Cette fonction prend en paramètres le chemin du fichier contenant le dictionnaire, ainsi qu'un tableau de chaînes de caractères `dico` dans lequel les mots du dictionnaire seront stockés, sous la forme de chaînes de caractères terminés par le caractère "`\0`". La fonction retourne le nombre de mots lus dans le dictionnaire. Le dictionnaire contiendra au plus 10 000 mots, chacun comportant au plus 9 caractères de l'alphabet latin, en minuscules et sans accents. Les constantes `MAX_MOTS` et `TAILLE_MAX_MOT` définies dans le fichier `base_1.c` contiennent ces longueurs maximales respectives (en tenant compte du caractère "`\0`"). La fonction `main` fournie dans le fichier `base_1.c` appelle cette fonction pour vous, et fournit le dictionnaire obtenu aux fonctions que vous devez implémenter dans la suite de ce sujet. Vous devez modifier cette fonction `main` pour y spécifier la valeur de votre u_0.

1.1 Acronymes utilisant les initiales

Dans cette partie, nous allons nous limiter au cas où les acronymes sont constitués exclusivement à partir des initiales des mots de la phrase donnée en entrée, que l'on qualifie, par anglicisme,

d'*initialismes*. Par exemple, la phrase "`voici un exemple`" donnera pour acronyme "`vue`" (si ce mot est dans le dictionnaire fourni), alors que la phrase "`ici ca ne marche pas`" ne correspond à aucun acronyme (*a priori*, "`icnmp`" n'est pas un mot du dictionnaire).
Vous devez pour cela implémenter la fonction du fichier `base_1.c` définie par :

```
int trouverInitialisme(char phrase[5][TAILLE_MAX_MOT],
    int taillePhrase,
    char dico[MAX_MOTS][TAILLE_MAX_MOT],
    int tailleDico);
```

Cette fonction prend en paramètres le tableau des mots constituant la phrase (chacun terminé par le caractère "`\0`"), le nombre de mots de la phrase, le tableau des mots du dictionnaire, et le nombre de mots du dictionnaire, respectivement, et retourne l'index de l'initialisme trouvé dans le dictionnaire, ou -1 si aucun initialisme n'a été trouvé.

Question 1 *Pour chacune des phrases suivantes, déterminez si un initialisme existe ou non dans le dictionnaire fourni en implémentant* `trouverInitialisme`*, et renseignez le résultat de cette fonction dans votre fiche réponse. Nous garantissons qu'au plus un acronyme existe dans le dictionnaire fourni pour chacune des phrases ci-dessous, mais il peut aussi ne pas en exister.* ***Vous devez implémenter cette question en C, en utilisant les mots du dictionnaire fourni dans le fichier*** `data/u0/entree-q1.txt`**.**

a) `dru echappa palotte`

b) `absolve composte epinceter`

c) `merite endurable campera`

d) `place oppidums texturat`

e) `rejaillir otorragie ironisa`

Question à développer pendant l'oral 1 *Décrivez l'algorithme utilisé pour répondre à la question précédente. Quelle structure de données pourrait vous permettre de retrouver efficacement si l'acronyme existe dans le dictionnaire (on ne considère pas le temps nécessaire à la lecture du dictionnaire) ? Il n'est pas demandé de l'implémenter ici.*

1.2 Acronymes utilisant des lettres quelconques

Nous autorisons maintenant l'utilisation de n'importe quelle lettre constituant les différents mots de la phrase donnée en entrée. Par exemple, la phrase "`voici un nouvel exemple`" peut donner comme acronyme "`inox`" (si ce mot est dans le dictionnaire fourni) : en effet, si l'on met en évidence les lettres de l'acronyme dans la phrase, on obtient "`voIci uN nOuvel eXemple`". En revanche, si l'on considère la phrase "`la non`", il n'existe pas d'acronymes possibles, car on ne pourrait former que les mots "`ln`", "`lo`", "`an`" et "`ao`", et *a priori* aucun n'existe dans le dictionnaire.
Vous devez pour cela implémenter la fonction du fichier `base_1.c` définie par :

```
int trouverAcronyme(char phrase[5][TAILLE_MAX_MOT],
    int taillePhrase,
    char dico[MAX_MOTS][TAILLE_MAX_MOT],
    int tailleDico);
```

Cette fonction prend en paramètres le tableau des mots constituant la phrase (chacun terminé par le caractère "\0"), le nombre de mots de la phrase, le tableau des mots du dictionnaire, et le nombre de mots du dictionnaire, respectivement, et retourne l'index de l'acronyme trouvé dans le dictionnaire, ou -1 si aucun acronyme n'a été trouvé.

Question 2 *Pour chacune des phrases suivantes, déterminez si un acronyme existe ou non dans le dictionnaire fourni en implémentant* `trouverAcronyme`*, et renseignez le résultat de cette fonction dans votre fiche réponse. Nous garantissons qu'au plus un acronyme existe dans le dictionnaire fourni pour chacune des phrases ci-dessous, mais il peut aussi ne pas en exister.* ***Vous devez implémenter cette question en C, en utilisant les mots du dictionnaire fournis dans le fichier*** `data/u0/entree-q2.txt`**.**

a) `ravit alentit mimee lisible`

b) `logicisme fallut pagaye radouber`

c) `debraye regreffe parraine velums`

d) `anavenins javelots attirer`

e) `tartisse audition indelicat baclee stria`

Question à développer pendant l'oral 2 *Décrivez l'algorithme utilisé pour répondre à la question précédente, et indiquez sa complexité temporelle.*

2 Acronymes utilisant des lettres supplémentaires

Dans cette deuxième partie que vous traiterez en langage OCaml , vous devez maintenant générer des *quasi-acronymes*, c'est-à-dire des acronymes dans lesquels on autorise à ce que certaines lettres ne correspondent pas à un mot de la phrase. Néanmoins, pour éviter que les acronymes ne soient trop éloignés de la phrase d'origine, on limitera le nombre **total** de lettres ne correspondant à aucun mot à 2.
Plus précisément, pour une phrase $p = m_1 \dots m_n$, l'acronyme $a \in \mathcal{D}$ que vous devez identifier doit être de la forme $\ell_1 \dots \ell_k$ avec :

- $n \leq k \leq n + 2$;
- il existe une fonction $\sigma : \{1, \dots, n\} \to \{1, \dots, k\}$ telle que $\sigma(i) < \sigma(j)$ pour tout $i < j$, et $\ell_{\sigma(i)} \in m_i$ pour tout $i \in \{1, \dots, n\}$.

On parlera de *quasi-initialisme* lorsque, dans la définition ci-dessus, $\ell_{\sigma(i)}$ est l'initiale du mot m_i pour tout $i \in \{1, \dots, n\}$.
La phrase "`voici encore un exemple`" a ainsi pour quasi-initialisme le mot "`vecue`" ("`Voici Encore Un Exemple`"), et pour quasi-acronyme "`canaux`" ("`voiCi eNcore Un eXemple`"), si ces mots sont dans le dictionnaire. Notez ici que la lettre '`c`' dans le premier exemple, et les lettres '`a`' dans le second exemple, ne correspondent pas à des mots de la phrase.
Pour vous aider dans l'implémentation de votre solution, nous vous fournissons la fonction `lireDictionnaire` qui vous permet de lire le dictionnaire $\mathcal{D}$ à partir d'un fichier texte. Cette fonction est donnée par :

```
lireDictionnaire : string -> string list
```

Cette fonction prend en paramètre le chemin du fichier contenant le dictionnaire, et retourne la liste contenant les mots de ce dictionnaire. Le dictionnaire contiendra au plus 100 000 mots, chacun composés de caractères de l'alphabet latin, en minuscules et sans accents, dont la taille n'est pas limitée.

De plus, les phrases que vous aurez à traiter dans cette partie étant relativement longues, nous vous fournissons la fonction `lirePhrase` définie par :

```
lirePhrase : string -> string list
```

Cette fonction prend en paramètre le chemin du fichier contenant une phrase d'entrée pour votre programme, et retourne la liste des mots de cette phrase.
Ces deux fonctions sont appelées pour vous dans le fichier `base_2.ml`, et fournissent les dictionnaires et les phrases lues en paramètres des fonctions que vous allez implémenter ci-après. N'oubliez pas de renseigner la valeur de votre u_0 dans ce fichier.

Question 3 *Pour chacune des phrases contenues dans les fichiers suivants, déterminez combien de quasi-initialismes existent dans le dictionnaire fourni. Pour cela, vous devez implémenter la fonction* `compterInitialismes` *du fichier* `base_2.ml`*. Inscrivez ensuite les résultats obtenus dans votre fiche réponse. On rappelle qu'ici seules les initiales des mots de la phrase sont à considérer.* ***Vous devez implémenter cette question en OCaml, en utilisant les mots du dictionnaire fournis dans le fichier*** `data/u0/entree-q3.txt`**.**

- ***a)*** `data/u0/entree-q3-a.txt`
- ***b)*** `data/u0/entree-q3-b.txt`
- ***c)*** `data/u0/entree-q3-c.txt`
- ***d)*** `data/u0/entree-q3-d.txt`
- ***e)*** `data/u0/entree-q3-e.txt`

Question 4 *Pour chacune des phrases contenues dans les fichiers suivants, déterminez combien de quasi-acronymes existent dans le dictionnaire fourni, en appliquant la nouvelle approche décrite ici. Pour cela, vous devez implémenter la fonction* `compterAcronymes` *du fichier* `base_2.ml`*. Inscrivez ensuite les résultats obtenus dans votre fiche réponse. On rappelle qu'ici toutes les lettres des mots de la phrase peuvent être utilisées.* ***Vous devez implémenter cette question en OCaml, en utilisant les mots du dictionnaire fournis dans le fichier*** `data/u0/entree-q4.txt`**.**

- ***a)*** `data/u0/entree-q4-a.txt`
- ***b)*** `data/u0/entree-q4-b.txt`
- ***c)*** `data/u0/entree-q4-c.txt`
- ***d)*** `data/u0/entree-q4-d.txt`
- ***e)*** `data/u0/entree-q4-e.txt`

Question à développer pendant l'oral 3 *Décrivez les algorithmes utilisés pour répondre aux questions précédentes, en précisant la différence entre les deux questions. Donnez la complexité temporelle et la complexité spatiale de ces algorithmes.*

Question à développer pendant l'oral 4 *Quel formalisme pourrait vous permettre de représenter l'existence de lettres intermédiaires quelconques entre les lettres de la phrase pour former un acronyme, et ainsi résoudre efficacement le problème ? Vous présenterez brièvement la représentation qu'il faudrait utiliser dans ce cas. Il n'est pas demandé de l'implémenter.*

3 Acronymes non dépendants de l'ordre des mots

Dans cette troisième partie que vous traiterez en langage OCaml, vous devez maintenant générer des *quasi-acronymes désordonnés*, c'est-à-dire des quasi-acronymes dans lesquels l'ordre des mots de la phrase n'est pas nécessairement préservé. Comme dans la partie précédente, on autorisera aussi à ce qu'au plus deux lettres de l'acronyme ne correspondent pas à un mot de la phrase.
Plus précisément, pour une phrase $p = m_1 \dots m_n$, l'acronyme $a \in \mathcal{D}$ que vous devez identifier doit être de la forme $\ell_1 \dots \ell_k$ avec :

- $n \leq k \leq n+2$;
- il existe une fonction $\sigma : \{1, \dots, n\} \to \{1, \dots, k\}$ telle que $\sigma(i) \neq \sigma(j)$ pour tout $i \neq j$, et $\ell_{\sigma(i)} \in m_i$ pour tout $i \in \{1, \dots, n\}$.

Par exemple, la phrase "`ici l ordre importe peu`" peut donner comme acronyme "`enclot`" (si ce mot est dans le dictionnaire). Dans ce cas, en mettant en évidence les lettres de l'acronyme et en mettant les mots dans l'ordre approprié, la phrase devient : "`pEu iCi L Ordre imporTe`" (la lettre '`n`' n'est pas utilisé).
Pour résoudre ce problème, on se propose de le voir comme un problème de couplage biparti maximum dans un graphe. Un graphe orienté $\mathcal{G} = (V, E)$ est un couple où V est un ensemble de nœuds et E est un ensemble d'arcs $(u, v) \in E^2$. Un graphe non-orienté $\mathcal{G} = (V, E)$ est un couple où V est un ensemble de nœuds et E est un ensemble d'arêtes $\{u, v\} \subset V$, avec $u \neq v$. Un graphe non-orienté est dit *biparti* lorsqu'il existe une partition $\{U, V\}$ de ses sommets telle que, pour toute arête $\{v, v'\} \in E$, ($v \in U$ et $v' \in V$) ou ($v' \in U$ et $v \in V$). Pour décrire un graphe biparti, nous noterons alors $\mathcal{G} = (U, V, E)$. Un *couplage* dans un graphe biparti $\mathcal{G} = (U, V, E)$ est un sous-ensemble C de E telle que, pour toute paire d'arêtes e et e' dans C, $e \cap e' = \emptyset$, c'est à dire que e et e' n'ont pas de nœuds en commun. Un nœud f de $\mathcal{G}$ est dit *libre* pour C si il n'est l'extrémité d'aucune arête de C, c'est-à-dire, pour toute arête $\{u, v\} \in C$, $f \neq u$ et $f \neq v$. Un couplage C est dit *maximum* s'il n'existe aucun couplage C' dont le cardinal est strictement supérieur à celui de C.
Afin de construire un couplage maximum dans un graphe biparti, nous allons utiliser la méthode des *chemins augmentants*. Plus précisément, soit $\mathcal{G} = (U, V, E)$ ce graphe, et soit C un couplage de $\mathcal{G}$. Un chemin $c = (v_0, v_1, \dots, v_{2p+1})$ de longueur impaire est augmentant pour C si :

- v_0 et v_{2p+1} sont libres pour C,
- pour tout $i \in \{0, \dots, p\}$, $\{v_{2i}, v_{2i+1}\} \in E \backslash C$,
- pour tout $i \in \{0, \dots, p-1\}$, $\{v_{2i+1}, v_{2i+2}\} \in C$, et
- c ne contient pas deux fois le même sommet.

Autrement dit, un chemin augmentant est un chemin de longueur impaire, alternant les arêtes de C et de $E \backslash C$, et reliant deux sommets libres distincts.
Afin d'identifier un chemin augmentant dans un graphe biparti $\mathcal{G} = (U, V, E)$ pour un couplage C, on construit le graphe orienté $\mathcal{G}_C = (U \cup V, \tilde{E})$, où $\tilde{E}$ est donné par :

$$\begin{aligned} \tilde{E} = &\{(v, u) \mid \{u, v\} \in C, u \in U, v \in V\} \\ &\cup \{(u, v) \mid \{u, v\} \in E \backslash C, u \in U, v \in V\} \end{aligned}$$

Afin d'identifier un chemin augmentant, il suffit alors de trouver un chemin P dans $\mathcal{G}_C$ entre un nœud $u_f \in U$ et un autre nœud $v_{f'} \in V$ tous les deux libres pour C. Le résultat de l'augmentation du couplage C par le chemin augmentant P est défini comme la différence symétrique entre C et P. Concrètement, les arêtes du chemin P qui sont dans le couplage sont enlevées, alors que les arêtes du chemin P qui ne sont pas dans le couplage sont ajoutés au nouveau couplage.
La figure 1 illustre les différentes étapes permettant, à partir d'un couplage existant, identifier un chemin augmentant pour ensuite construire un couplage plus grand.

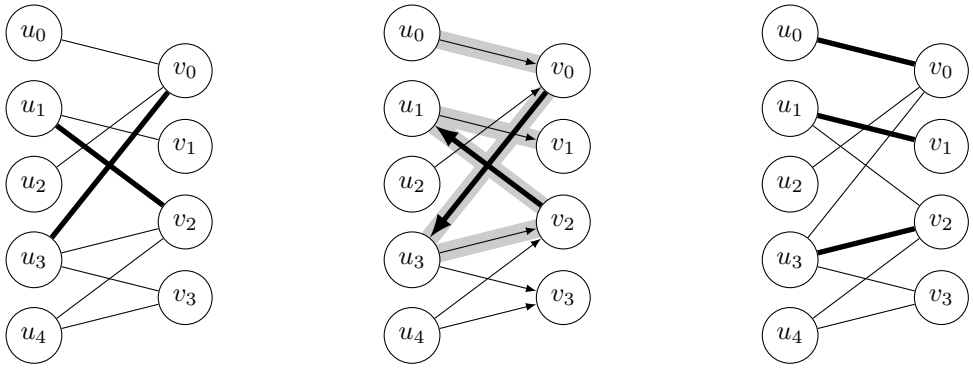

FIGURE 1 – Un exemple d'utilisation de la méthode des chemins augmentants pour déterminer un nouveau couplage dans un graphe biparti. À gauche, le couplage initial C du graphe $\mathcal{G}$ est représenté par les arêtes épaisses. Au centre, un chemin augmentant est mis en évidence sur le graphe orienté $\mathcal{G}_C$ construit à partir de C. À droite, le nouveau couplage obtenu est représenté par les arêtes épaisses.

La première étape de cette partie consiste à implémenter un algorithme permettant de déterminer si un chemin augmentant existe dans un graphe biparti $\mathcal{G} = (\{u_0, \ldots, u_{g-1}\}, \{v_0, \ldots, v_{d-1}\}, E)$ pour un couplage C donné. Par souci de simplicité, on appellera respectivement *partie gauche* et *partie droite* les deux ensembles de nœuds constituant le graphe biparti $\mathcal{G}$. Pour cela, une fonction `lireGrapheCouplages` est fournie dans le fichier `base_3.ml`, et est donnée par :

```
lireGrapheCouplages : string -> int * int * (int list array) * (int array list)
```

Cette fonction prend en paramètre le chemin du fichier contenant le graphe biparti et ses couplages associés, et retourne un quadruplet (g, d, E, C) où :

- g est le nombre de nœuds dans la partie gauche du graphe,
- d est le nombre de nœuds dans la partie droite du graphe,
- E est la représentation du graphe sous la forme d'une liste d'adjacence, c'est-à-dire un tableau de g listes (une liste par nœud de la partie gauche), où la liste d'indice i contient les voisins du nœud u_i situés dans la partie droite du graphe.
- C est une liste de couplages du graphe biparti, chaque couplage étant représenté à l'aide d'un tableau t d'entiers, dans lequel $t[j]$ est le nœud $u_{t[j]}$ de la partie gauche du graphe auquel est couplé le nœud de la partie droite v_j, ou -1 si le nœud v_j n'est pas couplé.

Cette fonction est appelée pour vous dans le fichier `base_3.ml`, et son résultat est donné en paramètre de la fonction `compterCheminsAugmentants` que vous devez implémenter ici. N'oubliez pas de renseigner la valeur de votre u_0 dans ce fichier. L'instance `data/u0/entree-q5-a.txt` de numéro $\widetilde{u_0} = 0$ correspond à l'exemple de la Figure 1. Vous pouvez vous en servir pour debogguer votre code.

Question 5 *Implémentez la fonction* `compterCheminsAugmentants` *déterminant combien de couplages admettent un chemin augmentant dans le graphe biparti donné parmi ceux donnés dans les fichiers ci-dessous. Renseignez ce nombre dans votre fiche réponse.* ***Vous devez implémenter cette question en OCaml.***

a) `data/u0/entree-q5-a.txt` ***b)*** `data/u0/entree-q5-b.txt`

c) `data/u0/entree-q5-c.txt` ***d)*** `data/u0/entree-q5-d.txt`

Question à développer pendant l'oral 5 *Quelle est la complexité temporelle de l'algorithme que vous avez implémenté pour répondre à la question précédente ?*

Vous devez maintenant exploiter l'algorithme implémenté dans la question précédente pour construire un couplage maximum dans un graphe biparti $\mathcal{G} = (\{u_0, \ldots, u_{g-1}\}, \{v_0, \ldots, v_{d-1}\}, E)$. On admettra qu'un couplage C pour un graphe biparti $\mathcal{G}$ n'admettant pas de chemin augmentant est maximum. Pour cela, nous vous fournissons également une fonction `lireGraphe` dans le fichier `base_3.ml`, donnée par :

```
lireGraphe : string -> int * int * (int list array)
```

Cette fonction prend en paramètre le chemin du fichier contenant le graphe biparti, et retourne un triplet (g, d, E) dans le même format que celui décrit pour la fonction `lireGrapheCouplages` (sans la liste C des couplages existants). Elle est appelée pour vous dans le fichier `base_3.ml`, et son résultat est donné en paramètre de la fonction `trouverCouplage` que vous devez implémenter ici, afin de déterminer la taille d'un couplage maximum dans le graphe donné. L'instance `data/u0/entree-q6-a.txt` de numéro $\widetilde{u_0} = 0$ correspond à l'exemple de la Figure 1. Vous pouvez vous en servir pour debogguer votre code.

Question 6 *Implémentez la fonction* `trouverCouplage` *déterminant la taille d'un couplage maximum de chacun des graphes donnés dans les fichiers suivants. Renseignez la taille de ces couplages dans votre fiche réponse.* ***Vous devez implémenter cette question en OCaml.***

a) `data/u0/entree-q6-a.txt` ***b)*** `data/u0/entree-q6-b.txt`

c) `data/u0/entree-q6-c.txt` ***d)*** `data/u0/entree-q6-d.txt`

Question à développer pendant l'oral 6 *Quelle est la complexité temporelle de l'algorithme que vous avez implémenté pour répondre à la question précédente ?*

En exploitant les algorithmes que vous avez implémentés dans les questions précédentes, vous devriez maintenant pouvoir résoudre le problème initial, à savoir la génération d'acronymes désordonnés. Les fonctions `lireDictionnaire` et `lirePhrase` décrites dans la section précédente sont également définies dans le fichier `base_3.ml`, et sont appelées pour vous afin de vous fournir les dictionnaires et les phrases en paramètres de la fonction `compterAcronymesDesordonnes` que vous devez implémenter.

Question 7 *Pour chacune des phrases contenues dans les fichiers suivants, déterminez combien de quasi-acronymes désordonnés existent dans le dictionnaire fourni. Pour cela, vous devez implémenter la fonction* `compterAcronymesDesordonnes` *du fichier* `base_3.ml`*. Inscrivez ensuite les résultats obtenus dans votre fiche réponse. On rappelle qu'ici toutes les lettres des mots de la phrase peuvent être utilisées.* ***Vous devez implémenter cette question en OCaml, en utilisant les mots du dictionnaire fournis dans le fichier*** `data/u0/entree-q7.txt`***.***

a) `data/u0/entree-q7-a.txt` ***b)*** `data/u0/entree-q7-b.txt`

c) `data/u0/entree-q7-c.txt` ***d)*** `data/u0/entree-q7-d.txt`

e) `data/u0/entree-q7-e.txt`

Question à développer pendant l'oral 7 *Décrivez comment vous avez utilisé l'algorithme de couplage maximum pour répondre à la question précédente.*

Fiche réponse type : Génération d'acronymes

$\widetilde{u_0}$: 0

Question 1

a) 2668

b) 7057

c) 6703

d) 7941

e) 3843

Question 2

a) 1952

b) 1582

c) 1755

d) 4462

e) 8591

Question 3

a) 3

b) 2

c) 2

d) 3

e) 3

Question 4

a) 15

b) 12

c) 896

d) 0

e) 133

Question 5

a) 1

b) 344

c) 350

d) 313

Question 6

a)	4
b)	74
c)	93
d)	83

b)	56
c)	23 482
d)	683
e)	1275

Question 7

a)	2346

Fiche réponse : Génération d'acronymes

Nom, prénom, u_0 : .....

Question 1

a)

b)

c)

d)

e)

Question 2

a)

b)

c)

d)

e)

Question 3

a)

b)

c)

d)

e)

Question 4

a)

b)

c)

d)

e)

Question 5

a)

b)

c)

d)

Question 6

a)

b)

c)

d)

b)

c)

d)

e)

Question 7

a)