# Développer une application informatique utilisant les algorithmes de graphes

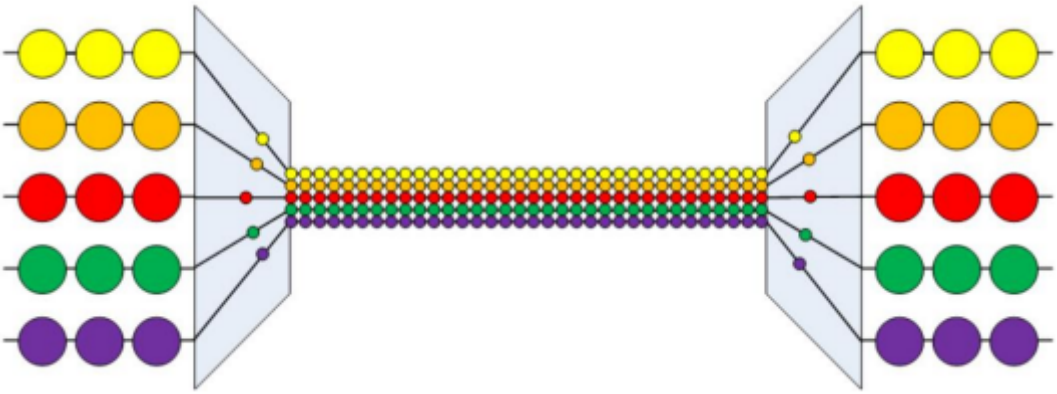
*BUT-RT-S3-SAE3.02*

## Sujet D

# Multiplexage fréquentiel

**Lohen CHENOLL, Antonin DAUTRIAT**

09/01/2023 – SAÉ3.02 – RT2A



# Sommaire

# I) Introduction

## 1.1 Qu'est-ce que le multiplexage fréquentiel ?

Le multiplexage fréquentiel (ou FDMA pour "Frequency Division Multiple Access" en anglais), est une technique de transmission de données qui permet de transmettre plusieurs signaux simultanément sur une même voie de transmission en utilisant des fréquences différentes pour chaque signal. Cela signifie que chaque signal est modulé en utilisant une fréquence spécifique, de sorte qu'il peut être transmis sur la même voie de transmission que d'autres signaux sans interférer avec eux.

Le multiplexage fréquentiel est utilisé dans divers secteurs d'activité, tels que :

- **Télécommunications** : Il est utilisé pour partager une bande passante limitée entre différents utilisateurs pour les appels téléphoniques, la téléphonie mobile, les communications radio et les systèmes de réseau de données.
- **Radiocommunication** : Il est utilisé pour partager une bande passante limitée entre différents utilisateurs pour les communications radio, les communications aériennes, les communications maritimes et les communications spatiales.
- **Télévision par câble** : Il est utilisé pour transmettre simultanément plusieurs chaînes de télévision sur une seule ligne de câble.
- **Radiodiffusion** : Il est utilisé pour transmettre simultanément plusieurs programmes de radio sur une seule fréquence.
- **Transports** : Il est utilisé pour la communication entre les différents équipements et systèmes dans les véhicules tels que les avions, les trains et les automobiles.
- **Applications industrielles** : Il est utilisé dans les réseaux de contrôle de processus industriels pour transmettre des données de capteurs et de commandes à des équipements de contrôle.
- **Surveillance** : Il est utilisé pour la surveillance des systèmes de caméras et de détection d'intrusion dans les bâtiments et les zones à protéger.

### 1.2 Présentation du sujet

Le sujet qui nous a été attribué se nomme “Multiplexage fréquentiel”. Le but de ce sujet est de concevoir et de développer un logiciel qui permet de déterminer le nombre minimum de fibres à utiliser pour multiplexer des signaux dont les gammes de fréquences sont connues.

Les gammes de fréquences des signaux seront lues dans un fichier, chaque ligne contenant le numéro du signal, sa fréquence minimale et sa fréquence maximale. Le fichier se présentera comme ceci :

| Signal | fréqu_min | fréqu_max |
|---|---|---|
| 1 | 100 | 500 |
| 2 | 300 | 700 |
| 3 | 750 | 1000 |

On peut donc déduire de ce fichier que le signal 1 a une gamme de fréquence de [100; 500], le signal 2 a une gamme de fréquence de [300; 700] et le signal 3 [750; 1000]. Ces 3 signaux ne pourront donc pas transiter au sein de la même fibre.

En sortie, le logiciel donnera le nombre de fibres à utiliser et les signaux cheminant dans chacune de ces fibres. De plus, nous avons ajouté en sortie le graphe d'incompatibilité des signaux afin d’avoir un aperçu des signaux qui ne pourront pas transiter dans la même fibre ainsi que le majorant du nombre de fibres. Nous avons aussi affiché un diagramme nous affichant

Cette question est liée à la notion de coloration de graphe, qui a été abordée lors d'un TD SAé 3.02 le lundi 12 décembre de 8h à 10h.

### 1.3 Partage du travail et environnement de travail

Pour la répartition du travail, nous avons établi un plan au début de la SAÉ afin de savoir qui allait effectuer quelle partie du code. Puis, une fois que la base du code a été réalisée, nous avons choisi des ajouts au code, un par un, afin de peaufiner notre travail et d’afficher un résultat en sortie qui soit le plus précis et pertinent possible.
Le code a été réalisé sur le logiciel Visual Studio Code, car cet outil est présent sur nos deux machines respectives et qu’il est très facile et pratique d’utilisation.
Pour le partage du travail, nous avons utilisé l’outil GitHub (lien de notre github) car cela nous permet d’avoir, l’un comme l’autre, le travail avec les dernières modifications de son camarade. Cela a donc été très pratique pour nous partager le code de la SAÉ, à jour.

# II) Fichier d'entrée

## 2.1 Création manuelle du fichier d'entrée

Le fichier d'entrée se nomme, pour notre groupe, "donnees.txt". Afin de le créer manuellement, il nous suffit juste de créer un fichier en .txt sur Visual Studio Code que nous renommons "donnees.txt", et comprenant les données inscrites de la même façon que dans la **présentation du sujet**, à savoir de la façon suivante :

```
1	100	500
2	300	700
3	750	1000
```

Dans cet extrait, on peut donc voir que le signal numéro 1 et le signal numéro 3 pourront transiter au sein de la même fibre, mais que le signal numéro 2 a une partie dans la même bande de fréquence que le signal numéro 1, il devra donc être dans une fibre différente.

## 2.2 Création automatique du fichier d'entrée

Afin de créer automatiquement le fichier d'entrée "donnees.txt", nous avons créé un fichier nommé "cree_donnees.py". Ce dernier est composé ainsi :

```
import os
from random import randint
```

Premièrement, nous importons les bibliothèques **os** et la sous bibliothèque **randint** de la bibliothèque **random**.
La bibliothèque **os** sert à pouvoir intéragir avec le système d'exploitation, afin de gérer l'arborescence des fichiers, de fournir des informations sur le système d'exploitation processus, variables systèmes, ainsi que de nombreuses fonctionnalités du système.
La sous bibliothèque **random.randint** sert à générer des nombres (integer) complètement aléatoires.

```
def cree_donnees_f(nb_signaux):
    try:
      # Envoie le fichier donnees.txt dans la corbeille
       os.remove('donnees2.txt') #Supprimer le fichier donnees2.txt s'il existe
    except FileNotFoundError:
       # Le fichier n'existe pas, rien à faire
       pass

    # Ouvrir le fichier donnees.txt en mode écriture
    fichier = open('donnees.txt', 'w')

    for i in range (nb_signaux):
       # Génère un nombre aléatoire entre 0 et 1000 pour freq_min
       freq_min = randint (0,1000)
       # Génère un nombre aléatoire entre 20 et 500 pour bp
       bp = randint (20,500)
       # Calcule la valeur de freq_max
       freq_max = freq_min + bp
       ligne = str(i+1) + ' ' + str(freq_min) + ' ' + str(freq_max) + '\n'
       # Ecrit les valeurs de i+1, freq_min, freq_max dans le fichier donnees.txt
       fichier.write(ligne)
    # Ferme le fichier donnees.txt
    fichier.close()
```

On peut voir, dans ce code, la fonction “cree_donnees_f”, qui crée le fichier “donnees.txt”, en effectuant les actions suivantes :

- Premièrement, il efface un fichier “donnees2.txt” si ce dernier existe déjà, sinon on passe à la suite;
- Dans un second temps, on ouvre un fichier nommé “donnees.txt” en mode écriture;
- Ensuite, on traverse les lignes du fichier et, pour chaque ligne, on ajoute les colonnes représentant le numéro du signal, les fréquences minimales et maximales de ce signal.
- Enfin, on ferme le fichier, et il peut enfin être exploité comme nous le souhaitons dans la **prochaine partie**.

# III) Exploitation du fichier d'entrée

Afin d'exploiter le fichier "donnees.txt", nous avons défini une fonction nommée "recup_donnees", récupérant le fichier que nous avons créé lors de la **partie précédente**. Nous allons donc voir, dans cette partie, comment récupérer et isoler chaque donnée présente dans ce fichier afin de pouvoir par la suite les traiter.

## 3.1 Étapes réalisées

Les différentes étapes de l'exploitation du fichier d'entrée afin de le rendre utilisable par notre programme sont les suivantes :

```
#recuperation des donnees sous forme d'une liste de liste
def recup_donnees(fichier):
    lst_donnees = []  #On initialise la liste lst_donnees
    with open(fichier, 'r') as d:   #On ouvre le fichier
        for line in d:
            col1, col2, col3 = line.strip().split(' ')
            col1 = int(col1)
            col2 = int(col2)  #On convertit les valeurs du fichier.txt en nombres
            col3 = int(col3)
 # maintenant, on peut traiter les données de chaque colonne comme des variables

#On insère les valeurs dans la liste
           lst_donnees.append([col1, [col2, col3]])
    return(lst_donnees)    # On affiche la liste
```

Premièrement, nous définissons notre fonction "recup_donnees", qui va servir à récupérer individuellement chaque donnée dans le fichier d'entrée sous forme d'une liste de listes, qu'on initialise à vide au début de la fonction ;
Ensuite, on ouvre le fichier d'entrée et on parcourt chaque ligne de ce dernier en dissociant les colonnes grâce aux espaces qui y sont présents, puis on affecte chaque colonne à une variable (ici col1, col2 et col3) ;
Après avoir fait cela, les valeurs présentes dans chaque colonne du fichier d'entrée étant des chaînes de caractères, il faut les convertir en nombres (integer) grâce à la commande "int(x)" ;
On peut donc désormais traiter chaque valeur de chaque colonne comme des nombres. On va donc les insérer dans la liste de listes de la forme suivante "lst_donnees.append([col1, [col2, col3]])".
Enfin, on affiche la liste de listes afin de voir si elle a correctement été créée.

# IV) Traitement des données

Pour cette partie c'est la présentation du fichier main.py qui va être réalisé et des fonctions qui le composent.

## 4.1 Librairies/Environnement utilisé

Dans un premier temps au dans les premières lignes du fichier c'est l'importation des différentes bibliothèques qui est réalisé. Voici donc le code que l'on trouve pour importer cela.

```
import networkx as nx
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator
```

On a donc trois bibliothèques qui sont nécessaires à notre code pour s'effectuer correctement.

### Bibliothèque Networkx

NetworkX est une bibliothèque Python qui permet de créer, manipuler et étudier les graphes. Il fournit des fonctionnalités pour gérer les sommets et les arêtes d'un graphe, ajouter et supprimer des nœuds et des arêtes, parcourir les nœuds et les arêtes, et calculer des propriétés de graphes telles que les degrés des sommets, leurs ordres et leurs connexités. Cette bibliothèque permet également la manipulation d'arguments rattachés à des objets de graphes.

### Bibliothèque matplotlib.pyplot

matplotlib.pyplot est une sous-bibliothèque de la bibliothèque Python matplotlib qui permet de créer des visualisations graphiques telles que des courbes, des histogrammes, des diagrammes à barres, des diagrammes à secteurs, des images, etc. Il propose une interface de style Matlab pour tracer les données en utilisant des commandes similaires à celles de Matlab. Il est souvent utilisé pour visualiser les données en utilisant des graphiques 2D et 3D et pour créer des figures pour les publications scientifiques et les rapports. Cette bibliothèque permettra ainsi d'afficher nos graphes de façon à les observer et à les fournir en tant que résultat.

### Bibliothèque matplotlib.ticker

matplotlib.ticker est également une sous-bibliothèque de la bibliothèque Python matplotlib qui permet de contrôler les graduations et les étiquettes des axes d'un graphique. Il fournit des outils pour définir le

format des étiquettes, la précision des graduations, les limites de l'axe, etc. Il permet également de personnaliser les graduations en utilisant des fonctions de formatage personnalisées pour les étiquettes. Il est utilisé pour améliorer la lisibilité des graphiques en s'assurant que les étiquettes des axes sont claires et précises. On utilisera ainsi cette bibliothèque en complément de matplotlib.pyplot pour que notre affichage soit personnalisé et comme on le souhaite pour fournir le meilleur résultat possible.

Par la suite, dans un souci de lisibilité nous avons découpé notre programme en plusieurs fonctionnalités différentes qui seront à chaque fois une fonction que l'on appellera lorsque l'on en aura besoin.

## 4.2 Création du graphe des incompatibilités

Pour suivre notre raisonnement, on doit donc dans un premier temps avoir une fonction qui nous retourne le graphe d'incompatibilité de notre multiplexage. Voici donc le code de la fonction:

```
def cree_graph_incompatibilite(fichier_choisie):
    # On récupère les données à partir du fichier sélectionné
    lst_donnees = recup_donnees(fichier_choisie)
    # On initialise un un graph vide
    G = nx.Graph()
    # Pour chaque signal dans les données, on ajoute un noeud au graph
    for signal in lst_donnees :
        G.add_node(signal[0])
    # Tant qu'il reste des signaux dans la liste des données
    while lst_donnees :
        # On sélectionne le premier signal de la liste
        signal_selec = lst_donnees[0]
        # On retire le signal de la liste
        lst_donnees.pop(0)
        # Pour chaque signal restant à comparer
        for signal_compare in lst_donnees :
            # Si le signal sélectionné et le signal à comparer sont incompatibles,
on passe au suivant
            if signal_selec[1][0]-10 > signal_compare[1][1] or
signal_selec[1][1]+10 < signal_compare[1][0]:
                pass
            # Sinon, on ajoute une arête entre ces deux signaux dans le graph
            else:
                G.add_edge(signal_selec[0],signal_compare[0])
    # On retourne le graph final
    return G
```

Le code ci-dessus est une fonction donc qui crée le graphe d'incompatibilités à partir d'un fichier de données. La fonction prend en entrée un fichier sélectionné par l'utilisateur qui contient donc les données à traiter. On va détailler en plus des commentaires déjà placés dans le code le fonctionnement de la fonction étape par étape.

1. La fonction utilise d'abord une autre fonction "recup_donnees" pour récupérer les données à partir du fichier sélectionné. Les données sont stockées dans une liste variable "lst_donnees".
2. Elle initialise un objet graph vide "G" utilisant la bibliothèque NetworkX. Ce dernier sera donc le graph d'incompatibilité que l'on retournera.

3. Elle parcourt ensuite la liste des données et ajoute un nœud pour chaque signal dans le graph G, afin de construire progressivement le graph jusqu'à avoir tous les signaux comme nœud.
4. Ensuite, elle utilise une boucle "while" pour réaliser les différents liens entre les sommets qui seront les signaux incompatibles. Pour cela, elle continue à traiter les signaux restants dans la liste "lst_donnees" donc jusqu'à ce que cette dernière soit vide, elle sélectionne le premier signal de la liste dans un variable, puis retire ce signal de la liste, et parcourt les autres signaux restants pour les comparer avec celui sélectionné.
5. Pour chaque signal à comparer, elle vérifie si le signal sélectionné et celui à comparer sont incompatibles, en comparant leurs fréquence min et leurs fréquence max en veillant à un écart de 10 minimum entre eux . Si c'est le cas, elle passe au signal suivant. Sinon, elle ajoute une arête entre ces deux signaux dans le graph G ce qui veut dire que les deux sommets sont incompatibles et doivent être sur deux fibres différentes.
6. Enfin, une fois toutes les boucles finies et la liste des données vide, la fonction retourne le graph final G.

### 4.3 Attribution des couleurs au signaux

Ce code implémente l'algorithme de Welsh-Powell pour la coloration de graphes.

```
#fonction que utilise l'algorithme de welsh powell pour la coloration du graph
def algo_welsh (fichier_choisie):
    # Créer un graphique à partir du fichier de données d'incompatibilité
    G = cree_graph_incompatibilite(fichier_choisie)
    # Obtenir la liste des noeuds du graphique
    lst_noeuds = G.nodes()
    # Initialiser un dictionnaire pour enregistrer les couleurs choisies pour
      chaque noeud
    dico_couleurs_choisies = {}
    # Initialiser tous les noeuds à la couleur blanche
    for noeud in lst_noeuds:
        nx.set_node_attributes(G, {noeud: 'white'}, 'couleur')
    # Pour chaque noeud, obtenir la liste de ses voisins et les couleurs possibles
    for noeud in lst_noeuds:
        lst_voisins = list(G.neighbors(noeud))
        couleurs_possibles = {
            "white": 1,
```

```
            "red": 0,
            "orange": 0,
            "yellow": 0,
            "green": 0,
            "blue": 0,
            "purple": 0,
            "pink": 0,
            "brown": 0,
            "gray": 0,
            "beige": 0,
            "olive": 0,
            "navy": 0,
            "teal": 0,
            "lavender": 0,
            "turquoise": 0,
            "coral": 0,
            "magenta": 0,
            "crimson": 0,
            "violet": 0}
        # Marquer les couleurs utilisées par les voisins du noeud comme
          indisponibles
        for voisin in lst_voisins :
            couleur_du_voisin = G.nodes[voisin]['couleur']
            couleurs_possibles[couleur_du_voisin] = 1
        # Trouver la première couleur disponible dans la liste des couleurs
          possibles et l'attribuer au noeud
        for key,value in couleurs_possibles.items():
            if value == 0 :
                nx.set_node_attributes(G, {noeud: key}, 'couleur')
                dico_couleurs_choisies[noeud]=key
                break
    # Retourner le dictionnaire des couleurs choisies pour chaque noeud
    return dico_couleurs_choisies
```

Nous allons présenter le fonctionnement de cette fonction étape par étape.

1. Le code prend en entrée un fichier de données d'incompatibilité (fichier_choisie).
2. Il utilise ces données pour créer un graphe (G) en utilisant la fonction cree_graph_incompatibilite(fichier_choisie).
3. Il obtient la liste des nœuds du graphe (lst_noeuds) en utilisant la fonction nodes() de NetworkX.

4. Il initialise un dictionnaire (dico_couleurs_choisies) pour enregistrer les couleurs choisies pour chaque nœud.
5. Il initialise tous les nœuds à la couleur blanche en utilisant la fonction set_node_attributes de NetworkX, couleur par défaut.
6. Pour chaque nœud dans lst_noeuds, il obtient la liste de ses voisins (lst_voisins) en utilisant la fonction neighbors() de NetworkX.
7. Il initialise une liste de couleurs possibles (couleurs_possibles) qui contient toutes les couleurs disponibles.
8. Il parcourt la liste de voisins et marque les couleurs utilisées par les voisins comme indisponibles dans couleurs_possibles.
9. Il parcourt la liste de couleurs possibles (couleurs_possibles) et trouve la première couleur disponible, dans le but de prendre le moins de couleurs différentes possibles. Il attribue cette couleur au nœud en cours en utilisant la fonction set_node_attributes de NetworkX. Il enregistre également cette couleur dans le dictionnaire des couleurs choisies pour ce nœud.
10. Il retourne le dictionnaire des couleurs choisies pour chaque nœud (dico_couleurs_choisies) en fin de l'algorithme.

Enfin, il retourne le dictionnaire des couleurs choisies pour chaque nœud en fin de l'algorithme. Voici un exemple de résultat en sortie de la fonction.

```
lohen@MacBook-Air-de-lohen SAE 3.02 % python3 main.py
{1: 'red', 2: 'red', 3: 'orange', 4: 'red', 5: 'orange', 6: 'yellow', 7: 'orange', 8: 'green', 9: 'blue', 10: 'green'}
```

### 4.4 Dessin du graphe d'incompatibilité

La seconde fonction "dessiner_graph" est celle qui permet d'afficher le graphe d'incompatibilité à l'aide de la bibliothèque matplotlib.pyplot.

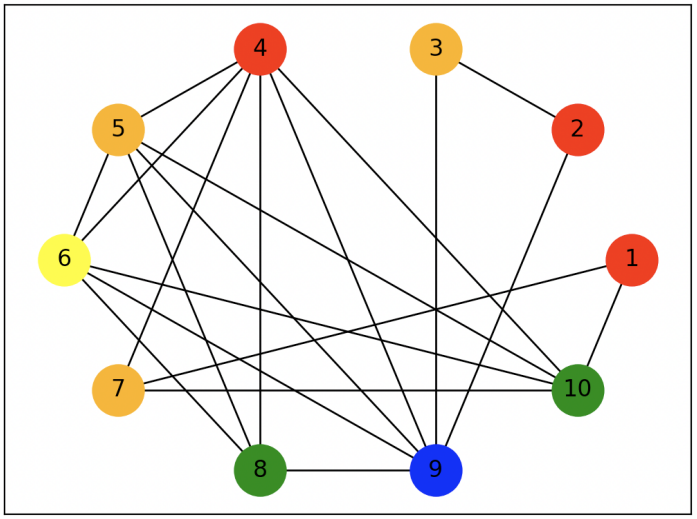
```
# Fonction que dessine le graphe d'incompatibilité
def dessiner_graph(fichier_choisie):
    # On crée un graph à partir du fichier sélectionné
    G = cree_graph_incompatibilite(fichier_choisie)
    # On définit une disposition circulaire pour les noeuds du graph
    pos = nx.circular_layout(G)
    # On récupère les couleurs choisies pour chaque signal dans un dictionnaire
    dico_couleurs_choisies = algo_welsh(fichier_choisie)
    # Pour chaque signal et sa couleur dans le dictionnaire
    for signal,couleur in dico_couleurs_choisies.items():
        # On dessine chaque noeud du graph en utilisant la couleur choisie
        nx.draw_networkx_nodes(G,pos,node_color=couleur,node_size=700,
        nodelist=[signal])
    # On dessine les arêtes du graph
    nx.draw_networkx_edges(G,pos)
    # On ajoute les labels des noeuds du graph
    nx.draw_networkx_labels(G,pos)
    # On affiche le graph
    plt.show()
```

Le code ci-dessus est une fonction qui réalise exclusivement le dessin du graph d'incompatibilité à partir d'un fichier de données. La fonction prend en entrée un fichier sélectionné par l'utilisateur. Nous allons la présenter étape par étape.

1. La fonction utilise d'abord la fonction "cree_graph_incompatibilite" pour créer un graph d'incompatibilité à partir du fichier sélectionné. Ce graph est ensuite stocké dans la variable "G".
2. Elle utilise alors la fonction "nx.circular_layout" pour définir une disposition circulaire pour les nœuds du graph G.
3. Ensuite, elle utilise une autre fonction "algo_welsh" pour récupérer les couleurs choisies pour chaque signal dans un dictionnaire "dico_couleurs_choisies".
4. Elle utilise ensuite une boucle "for" pour parcourir chaque signal et sa couleur dans le dictionnaire, et utilise la fonction "nx.draw_networkx_nodes" pour dessiner chaque nœud du graph G en utilisant la couleur choisie.
5. Elle utilise la fonction "nx.draw_networkx_edges" pour dessiner les arêtes du graph G.
6. Elle utilise la fonction "nx.draw_networkx_labels" pour ajouter les labels des nœuds du graph G.

7. Enfin, elle utilise "plt.show()" pour afficher le graph G.

Ce code nous permet de visualiser les relations de conflits entre les signaux en utilisant un graphe où les signaux sont des nœuds qui sont colorier suivant les conflits qui sont des arêtes. Voici un exemple de résultat en sortie de la fonction.




## 4.5 Trouve la clique du graph d'incompatibilité

La fonction "trouve_clique_max" ci-dessous renvoie la taille de la plus grosse clique du graphe d'incompatibilités. Voici le code de celle fonction:

```
def trouve_clique_max(fichier_choisie):
    # On crée un graph à partir du fichier sélectionné
    graph = cree_graph_incompatibilite(fichier_choisie)
    # On initialise une liste vide pour la clique maximale
    max_clique = []
    # Pour chaque noeud du graph
    for v in graph:
        # On initialise une liste vide pour la clique actuelle
        clique = []
        # On ajoute le noeud au début de la clique
        clique.append(v)
        # Pour chaque autre noeud du graph
        for u in graph:
            # Si tous les noeuds de la clique actuelle sont voisins du noeud u, on
              ajoute u à la clique
            if all(x in graph[u] for x in clique):
                clique.append(u)
        # Si la clique actuelle est plus grande que la clique maximale précédente,
          on met à jour la clique maximale
        if len(clique) > len(max_clique):
```

```
            max_clique = clique
    # On retourne la taille de la clique maximale
    return len(max_clique)
```

Ce code définit une fonction appelée "trouve_clique_min" qui prend en argument "fichier_choisie".

1. Il crée un graphe à partir du fichier choisi en utilisant la fonction "cree_graph_incompatibilite"
2. Il initialise une liste vide pour la clique maximale
3. Il parcourt tous les noeuds du graphe
   - 3.1. Il initialise une liste vide pour la clique courante
   - 3.2. Il ajoute le noeud courant à la clique courante
   - 3.3. Il parcourt tous les autres nœuds du graphe. Si tous les noeuds de la clique courante sont voisins du noeud courant, il ajoute ce noeud à la clique courante
   - 3.4. Si la clique courante est plus grande que la clique maximale précédente, il met à jour la clique maximale
4. Il retourne la taille de la clique maximale.

Ainsi cette fonction nous renvoie la taille du plus grand graph complet (clique) qu'elle a trouvé dans notre graph d'incompatibilité.
Voici un exemple de résultat en sortie de la fonction.

```
lohen@MacBook-Air-de-lohen SAE 3.02 % python3 main.py
5
```

## 4.6 Nombres de fibres nécessaires

Cette fonction nous permet de retourner le résultat du sujet c'est-à-dire le nombre de fibres minimum qui sont nécessaire pour transmettre l'ensemble des signaux du fichier.

```
def nb_fibre (fichier_choisie):
    # On récupère le dictionnaire des couleurs choisies pour chaque signal
    lst = algo_welsh(fichier_choisie)
    # On calcule le nombre de couleurs différentes utilisées par l'algorithme de
      Welsh et Powell
    result_welsh = int(len(set(lst.values())))
    # On calcule la taille de la clique maximale dans le graph
    result_clique = int(trouve_clique_max(fichier_choisie))
    # Si le nombre de fibres nécessaires selon l'algorithme de Welsh et Powell est
      égal à la taille de la clique maximale
    if result_clique == result_welsh:
        # On retourne le nombre de fibres nécessaires au minimum
        return f"Un total de {result_welsh} fibres sont nécessaire au minimum "
    # Sinon, on retourne le nombre de fibres théoriquement nécessaire, compris
```

```
    entre la taille de la clique maximale
  # et le nombre de couleurs utilisées par l'algorithme de Welsh et Powell
  else:
      return f"En théorie, entre {result_clique} et {result_welsh} fibres seront
      nécessaires"
```

Ce code définit une fonction appelée "nb_fibre" qui prend en argument "fichier_choisie".

1. Il récupère un dictionnaire des couleurs choisies pour chaque signal en utilisant la fonction "algo_welsh".
2. Il calcule le nombre de couleurs différentes utilisées par l'algorithme de Welsh et Powell en utilisant la fonction set() pour obtenir les éléments uniques et len() pour compter le nombre d'éléments uniques.
3. Il calcule la taille de la clique maximale dans le graphe en utilisant la fonction "trouve_clique_max".
4. Il vérifie si le nombre de fibres nécessaires selon l'algorithme de Welsh et Powell est égal à la taille de la clique maximale.
   4.1. Si c'est le cas, il retourne le nombre de fibres nécessaires au minimum
   4.2. Sinon, il retourne une phrase qui indique que le nombre de fibres nécessaires se trouve entre la taille de la clique maximale et le nombre de couleurs utilisées par l'algorithme de Welsh et Powell.

Cette fonction nous retourne donc une phrase réponse comportant le résultat, il ne reste plus qu'à l'afficher à l'utilisateur. Voici un exemple de résultat en sortie de la fonction.

```
lohen@MacBook-Air-de-lohen SAE 3.02 % python3 main.py
Un total de 5 fibres sont nécessaire au minimum
```

## 4.7 Attribuer des coordonnées x,y à un signal

Cette permet lorsqu'elle est appelée de fournir les coordonnées en x et y d'un signal pour l'afficher correctement .

```
def coordonnees_x(fichier_choisie,signal):
    # On récupère les données du fichier sélectionné
    lst_donnees = recup_donnees(fichier_choisie)
    # On récupère les coordonnées x1 et x2 du signal sélectionné dans la liste des
      données
    x1 = lst_donnees[signal-1][1][0]
    x2 = lst_donnees[signal-1][1][1]
    # On retourne les coordonnées x1 et x2
    return x1,x2
```

```
def coordonnees_y(fichier_choisie,signal):
    # On récupère le dictionnaire des couleurs choisies pour chaque signal
    dico_couleurs_choisies = algo_welsh(fichier_choisie)
    # On crée un dictionnaire associant à chaque couleur sa position sur l'axe y
    dico_placement = {
            "white": 0,
            "red": 1,
            "orange": 2,
            "yellow":3,
            "green": 4,
            "blue": 5,
            "purple": 6,
            "pink": 7,
            "brown": 8,
            "gray": 9,
            "beige": 10,
            "olive": 11,
            "navy": 12,
            "teal": 13,
            "lavender": 14,
            "turquoise": 15,
            "coral": 16,
            "magenta": 17,
            "crimson": 18,
            "violet": 19}
    # On récupère la couleur choisie pour le signal sélectionné
    couleur_choisie = dico_couleurs_choisies[signal]
    # On définit y1 et y2 comme étant égaux à la position de la couleur choisie
      sur l'axe y
    y1 = dico_placement[couleur_choisie]
    y2 = dico_placement[couleur_choisie]
    # On retourne les coordonnées y1 et y2 ainsi que la couleur choisie pour le
      signal sélectionné
    return y1,y2,couleur_choisie
```

Ce code définit deux fonctions "coordonnees_x" et "coordonnees_y" qui prennent toutes les deux en argument "fichier_choisie" et "signal".

La fonction “coordonnees_x” :

1. Il récupère les données du fichier choisi en utilisant la fonction "recup_donnees" ;
2. Il récupère les coordonnées x1 et x2 du signal choisi dans la liste des données ;
3. Il retourne les coordonnées x1 et x2.

La fonction “coordonnees_y” :

1. Il récupère le dictionnaire des couleurs choisies pour chaque signal en utilisant la fonction "algo_welsh" ;
2. Il crée un dictionnaire associant à chaque couleur sa position sur l'axe y ;
3. Il récupère la couleur choisie pour le signal sélectionné
4. Il définit y1 et y2 comme étant égaux à la position de la couleur choisie sur l'axe y ;
5. Il retourne les coordonnées y1 et y2 ainsi que la couleur choisie pour le signal sélectionné dans le but de pouvoir colorier le segment de la bonne couleur par la suite.

Ainsi ces deux fonctions travaillent ensembles pour fournir les coordonnées de placement du segment les représentant en ajoutant la couleur du signal que prendra également le segment.
Voici un exemple de résultat en sortie de la fonction. dans un premier temps les coordonnées de x puis celle et y avec la couleur :

```
lohen@MacBook-Air-de-lohen SAE 3.02 % python3 main.py
(405, 834)
lohen@MacBook-Air-de-lohen SAE 3.02 % python3 main.py
(2, 2, 'orange')
```

### 4.8 Trouve la clique du graphe d’incompatibilité

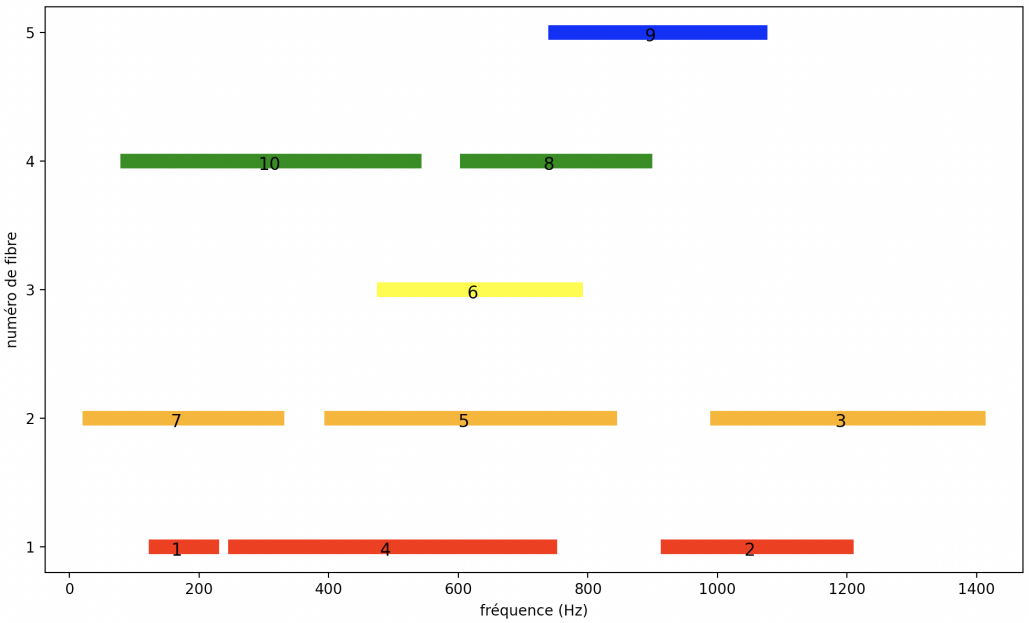
Cette fonction permet en lui fournissant le fichier de données de retourner un visuel de notre multiplexage plus simple pour notre compréhension. Pour cela elle fait appelle a tout les fonction du du main sauf “nb_fibre”.

```python
def visuel_multi(fichier_choisie):
    # On récupère le dictionnaire des couleurs choisies pour chaque signal
    dico_couleurs_choisies = algo_welsh(fichier_choisie)
    # On initialise la figure avec les dimensions 8,6 et l'axe
    fig, ax = plt.subplots(figsize=(12, 7))
    # Pour chaque signal et sa couleur dans le dictionnaire
    for signal,couleur in dico_couleurs_choisies.items():
        # On récupère les coordonnées x1, x2, y1, y2 et la couleur choisie pour le
          signal
        x1,x2 = coordonnees_x(fichier_choisie,signal)
        y1,y2,couleur_choisie = coordonnees_y(fichier_choisie,signal)
        # On dessine le segment sur l'axe
        ax.plot([x1, x2], [y1, y2], lw=10, color=couleur_choisie)
        # Ajout du numéro du signal sur le segment
```

```
            ax.text(((x1+x2)/2)-20, ((y1+y2)/2)-0.07, signal, fontsize=12)
        # On définit les labels des axes du graphique
        ax.set_xlabel('fréquence (Hz)')
        ax.set_ylabel('numéro de fibre')
        # On définit le format de l'axe y pour qu'il n'utilise que des nombres entiers
        ax.yaxis.set_major_locator(MaxNLocator(integer=True))
        # On affiche la figure
        plt.show()
```

1. La fonction "visuel_multi" prend en paramètre un fichier choisi, elle utilise cette variable pour créer un dictionnaire des couleurs choisies pour chaque signal en utilisant la fonction "algo_welsh".
2. Elle initialise une figure avec les dimensions 12,7 et un axe à l'aide de la bibliothèque matplotlib.
3. Pour chaque signal et sa couleur dans le dictionnaire:
    3.1. Elle récupère les coordonnées x1, x2, y1, y2 et la couleur choisie pour le signal en utilisant les fonctions "coordonnees_x" et "coordonnees_y" en lui passant en paramètre le fichier choisi et le signal.
    3.2. Elle dessine le segment sur l'axe en utilisant les coordonnées récupérées et la couleur choisie.
    3.3. Elle ajoute le numéro du signal sur le segment.
4. Elle définit les labels des axes du graphique en utilisant les fonctions "set_xlabel" et "set_ylabel"
5. Elle définit le format de l'axe y pour qu'il n'utilise que des nombres entiers.
6. Elle affiche la figure en utilisant la fonction "show" de la bibliothèque matplotlib.

Ainsi on a en sortie une figure représentative de la mise en place optimale du multiplexage des signaux fournis dans le fichier de données.
Voici un exemple de résultat en sortie de la fonction :



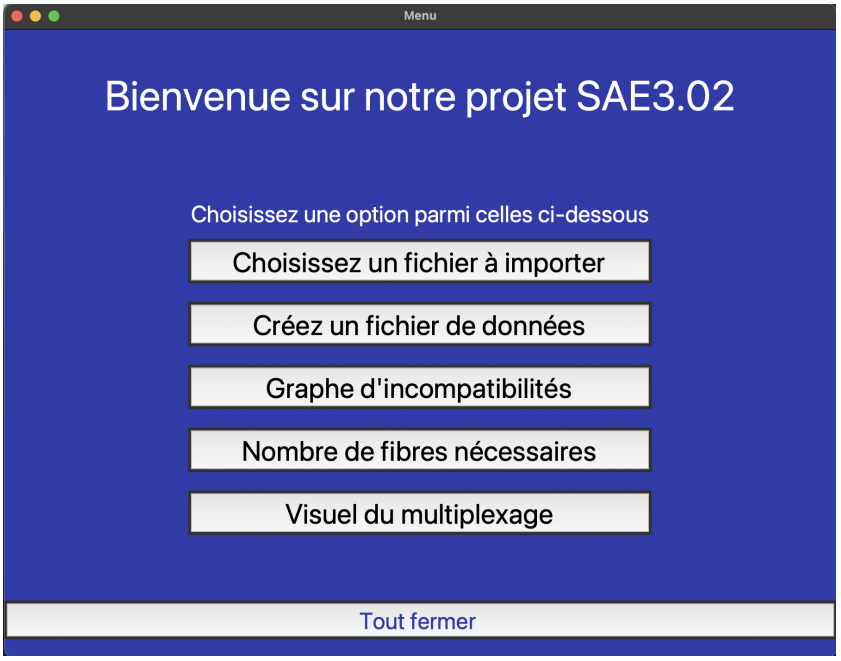

Nous pouvons donc apercevoir sur le diagramme ci-dessus que les 10 signaux qui ont été générés ont bien été répartis de façon à ce que l'on voit clairement la mise en place correcte du multiplexage.

# V) Affichage graphique des résultats

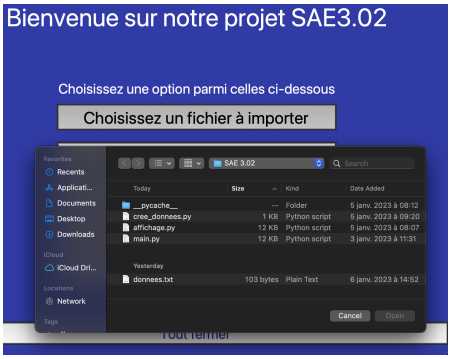
La dernière partie de notre projet consistait à présenter nos résultats de façon explicite et intuitive, à l'aide de boutons notamment, et on a opté pour la bibliothèque tkinter pour cette réalisation graphique.

Pour mener à bien cette partie du projet, nous avons dû apprendre à utiliser l'entièreté de la bibliothèque tkinter, déjà abordée lors de l'année précédente mais il a fallu pousser son apprentissage. On a tout d'abord créé une interface visuelle, mais ne possèdent aucun bouton. Puis on s'est rapidement heurté au premier problème, le premier était pour les boutons notre programme n'était pas du tout fait pour être utilisé sur une interface graphique, on a du donc le revoir et le découper en plus de fonctions indépendantes que prévue à la base, ainsi ces dernières qui n'auront par la suite plus qu'à être appelé.

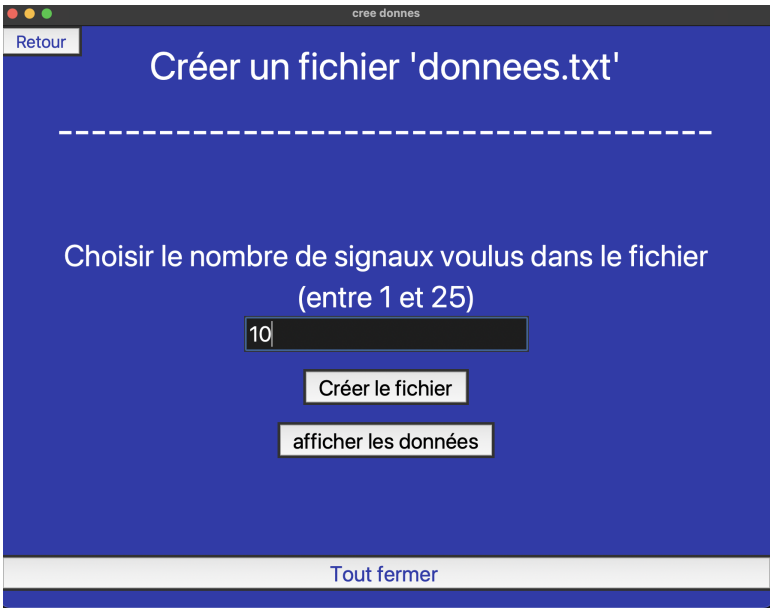
On est arrivé à un premier résultat qui sera par la suite notre menu principal, voici le résultat obtenu.




Le premier bouton “Choisissez un fichier à importer” nous renvoie sur une seconde fenêtre qui ouvre notre répertoire et nous permet d'importer un fichier. quo sera donc traiter par notre programme

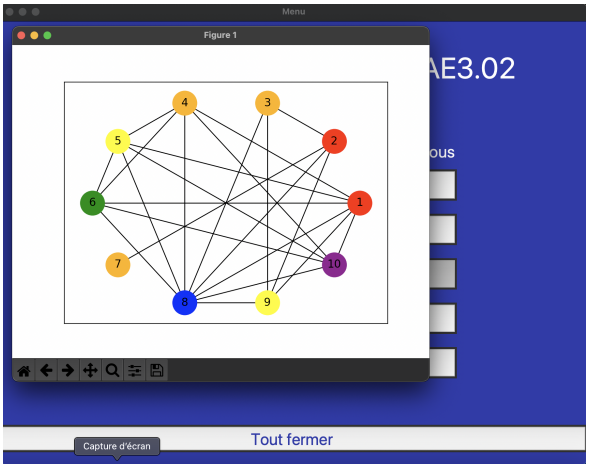


Le deuxième bouton “Créez un fichier de données” nous renvoie sur une seconde fenêtre également que voici et elle nous permet de créer un fichier constitué du nombre de signal que l'on souhaite puis un second bouton nous permet d'afficher les données créées aléatoirement de ce fichier.

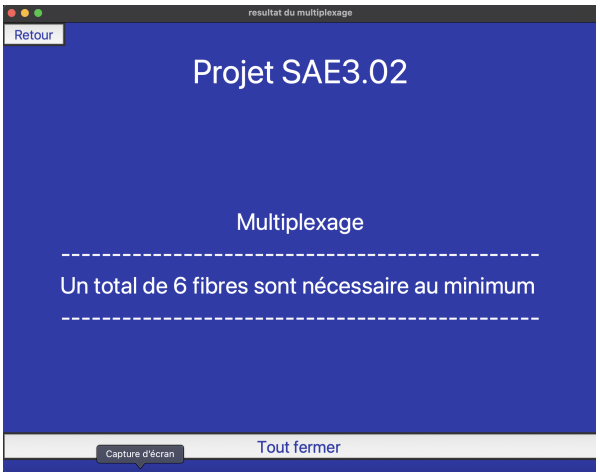



Voici comment sont affichés les données après la création du fichier


Les trois boutons suivants sont des résultats visuels du traitement du fichier fourni ci-dessus par code. Voici les résultats de ces trois boutons ci-dessous.

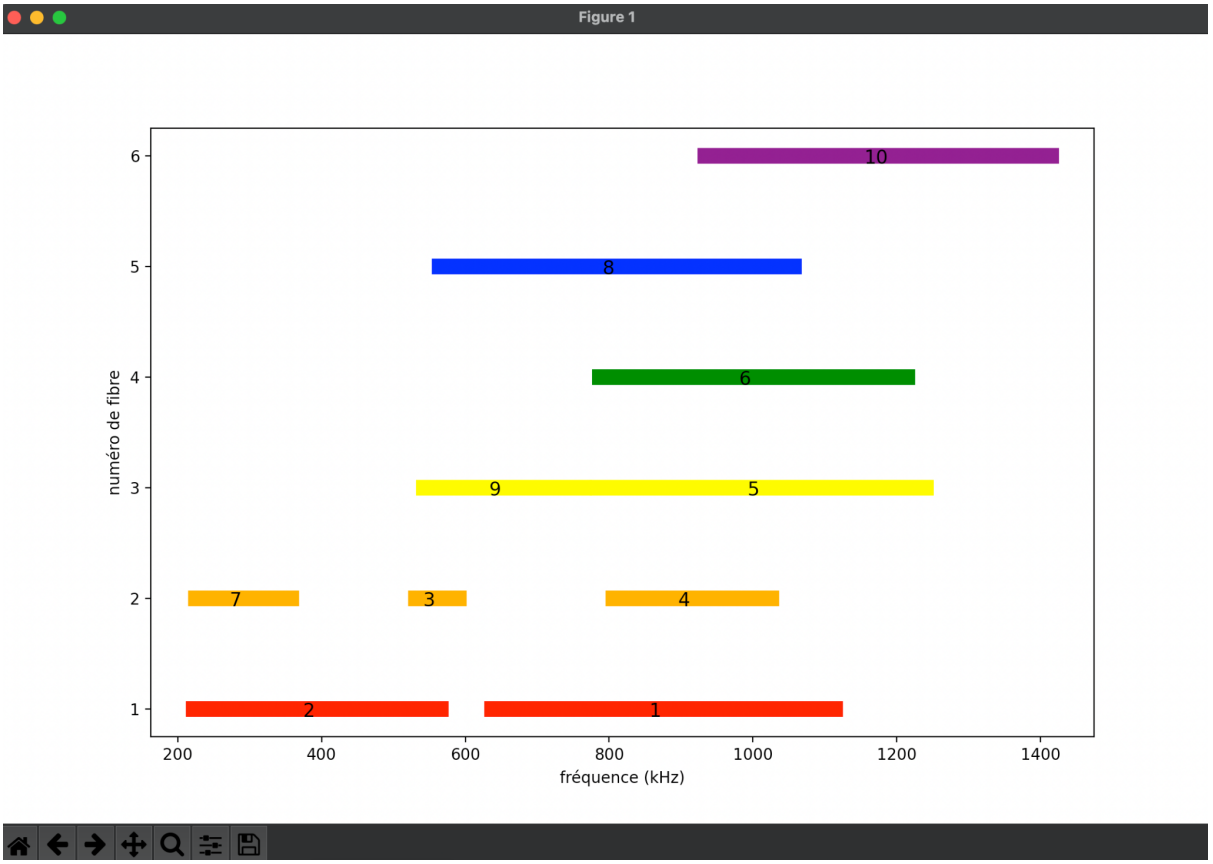



Voici ce que nous affiche le bouton “Graphe d’incompatibilité” lorsqu’il est cliqué. C’est donc le graphe d’incompatibilité créé en fonction des signaux du fichier.



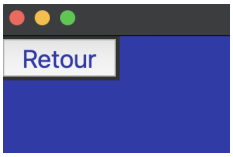

Voici ce que nous affiche le bouton “Nombres de fibres nécessaires”. Cela représente le résultat qui est la consigne de notre sujet de SAE. C'est-à-dire le nombre de fibres nécessaires pour la mise en place de du multiplexage.

Le dernier bouton “Visuel du multiplexage” nous permet d'avoir un résultat global de notre multiplexage avec quelque chose de facile à lire et très visuel.




On y voit donc la mise en place que donnerait le multiplexage. On y lit donc rapidement le nombre de fibres nécessaire dans l’exemple suivant un total de 6 fibres sont nécessaires, et chacune représentait par une couleur différentes. On lit également les fréquences en abscisses de chaque signal dont le label est écrit directement sur le dernier.
On pourrait donc donner cela comme un résultat global final du traitement du fichier choisi par notre programme.

On a également deux boutons dont voici les explications de leurs utilités




Le premier est le bouton “Retour” situé en haut à gauche qui permet lorsque l’on a ouvert une fenêtre retourner sur la précédente ( exemple fenêtre menu )

Le second est le bouton “Tout fermer” situé en bas au centre, ce dernier permet de fermer les fenêtres tkinter et de mettre fin au programme qui serait en cours d’exécution.

# VI) Journal de bord

La SAÉ3.02 devait être terminée dans un délai de 4 semaines de cours. Nous avons donc réparti notre travail sur ces 4 semaines comme ci-dessous :

| Semaine | Travail Réalisé |
|---|---|
| 50 | ☐ Participation au TD sur la coloration des graphes, qui nous va nous servir pour notre sujet ;<br>☐ Mise en place un cahier des charges des tâches à réaliser puis se partager le travail et les étapes de la SAÉ ;<br>☐ Début du code concernant l'exploitation du fichier d'entrée. |
| 01 | ☐ Mise en place de notre solution pour le partage du travail (GitHub), ainsi que notre environnement de travail (Visual Studio Code) ;<br>☐ Création des différents fichiers python pour l'exploitation du fichier d'entrée et de l'affichage en sortie ;<br>☐ Définition des fonctions qui ont été établies dans le cahier des charges ;<br>☐ Exploitation et lecture d'un fichier d'entrée type ;<br>☐ Mise en place de la fonction permettant la création d'un graphe d'incompatibilité ;<br>☐ Mise en place de la fonction permettant la coloration avec Welsh et Powell ;<br>☐ Ajout d'une fonction retournant la taille de la clique maximale<br>☐ Commencer la réflexion sur une possible interface graphique. |
| 02 | ☐ Crée la fonction permettant de générer un fichier de données au hasard ;<br>☐ Créé un programme qui permet d'importer un fichier d'entrée quelconque pour l'analyser ;<br>☐ Travail sur l'interface graphique qui doit être finalisé avant la fin de semaine ;<br>☐ Codé le diagramme dans lequel les fibres sont superposées dans lesquels les signaux sont répartis ;<br>☐ Créé les différents boutons pour l'affichage en sortie sur l'interface graphique ;<br>☐ Commencé à rédiger le compte rendu de la SAÉ pour notre sujet. |
| 03 | ☐ Terminé et déposé la rédaction du compte rendu ;<br>☐ Préparé notre oral avec support afin d'être prêts pour le passage;<br>☑ ~~Rendre l'ensemble du travail en temps et en heure.~~ |

# VII) Annexes (code python)

## 7.1 cree_donnnees.py

Ce fichier permet de créer un nombre voulu de signaux de fréquences minimales et maximales aléatoire et de les mettre dans un fichier donnees.txt que l'on va exploiter.

```
import os
from random import randint

#-------------------------------------------------------------------------------
#----CRÉATION-DU-FICHIER-DONNEES.TXT-------------------------------------------
#-------------------------------------------------------------------------------

def cree_donnees_f(nb_signaux):
  try:
    # Envoie le fichier donnees.txt dans la corbeille
     os.remove('donnees2.txt') #Supprimer le fichier donnees2.txt s'il existe
  except FileNotFoundError:
     # Le fichier n'existe pas, rien à faire
     pass

  # Ouvrir le fichier donnees.txt en mode écriture
  fichier = open('donnees.txt', 'w')

  for i in range (nb_signaux):
     # Génère un nombre aléatoire entre 0 et 1000 pour freq_min
     freq_min = randint (0,1000)
     # Génère un nombre aléatoire entre 20 et 500 pour bp
     bp = randint (20,500)
     # Calcule la valeur de freq_max
     freq_max = freq_min + bp
     ligne = str(i+1) + ' ' + str(freq_min) + ' ' + str(freq_max) + '\n'
     # Ecrit les valeurs de i+1, freq_min, freq_max dans le fichier donnees.txt
     fichier.write(ligne)
  # Ferme le fichier donnees.txt
  fichier.close()
```

## 7.2 main.py

Voici les différents programmes constituant le cœur de notre traitement des données, avec un découpage en fonctions comme présenté ci dessus.

```
import networkx as nx
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator

#variable globale
lst_couleurs_choisies = {}



#-------------------------------------------------------------------------------
#----RECUPERATION-DONNEES-EN-LISTE---------------------------------------------
#-------------------------------------------------------------------------------



#récupération des données sous forme d'une liste de liste
def recup_donnees(fichier):
   lst_donnees = []  #On initialise la liste lst_donnees
   with open(fichier, 'r') as d:   #On ouvre le fichier
       for line in d:
           col1, col2, col3 = line.strip().split(' ')
           col1 = int(col1)
              col2 = int(col2)       #On convertit les valeurs du fichier.txt en
nombres (integer)
           col3 = int(col3)
            # maintenant, on peut traiter les données de chaque colonne comme des
variables
            lst_donnees.append([col1, [col2, col3]])  #On insère les valeurs dans
la liste
   return(lst_donnees)



#-------------------------------------------------------------------------------
#----CREATION-GRAPH-INCOMPATIBILITE--------------------------------------------
#-------------------------------------------------------------------------------



def cree_graph_incompatibilite(fichier_choisie):
   # On récupère les données à partir du fichier sélectionné
   lst_donnees = recup_donnees(fichier_choisie)
   # On initialise un graph vide
   G = nx.Graph()
```

```
    # Pour chaque signal dans les données, on ajoute un noeud au graph
    for signal in lst_donnees :
        G.add_node(signal[0])
    # Tant qu'il reste des signaux dans la liste des données
    while lst_donnees :
        # On sélectionne le premier signal de la liste
        signal_selec = lst_donnees[0]
        # On retire le signal de la liste
        lst_donnees.pop(0)
        # Pour chaque signal restant à comparer
        for signal_compare in lst_donnees :
            # Si le signal sélectionné et le signal à comparer sont incompatibles, on passe au suivant
                        if  signal_selec[1][0]-10  >  signal_compare[1][1]  or signal_selec[1][1]+10 < signal_compare[1][0]:
                pass
            # Sinon, on ajoute une arête entre ces deux signaux dans le graph
            else:
                G.add_edge(signal_selec[0],signal_compare[0])
    # On retourne le graph final
    return G


#-------------------------------------------------------------------------------
#----ALGO-WELSH-POWELL-COLORATION-----------------------------------------------
#-------------------------------------------------------------------------------


#fonction que utilise l'algorithme de welsh powell pour la coloration du graph
def algo_welsh (fichier_choisie):
    # Créer un graphique à partir du fichier de données d'incompatibilité
    G = cree_graph_incompatibilite(fichier_choisie)
    # Obtenir la liste des noeuds du graphique
    lst_noeuds = G.nodes()
     # Initialiser un dictionnaire pour enregistrer les couleurs choisies pour chaque noeud
    dico_couleurs_choisies = {}
    # Initialiser tous les noeuds à la couleur blanche
    for noeud in lst_noeuds:
        nx.set_node_attributes(G, {noeud: 'white'}, 'couleur')
    # Pour chaque noeud, obtenir la liste de ses voisins et les couleurs possibles
    for noeud in lst_noeuds:
        lst_voisins = list(G.neighbors(noeud))
        couleurs_possibles = {
```

```
            "white": 1,
            "red": 0,
            "orange": 0,
            "yellow": 0,
            "green": 0,
            "blue": 0,
            "purple": 0,
            "pink": 0,
            "brown": 0,
            "gray": 0,
            "beige": 0,
            "olive": 0,
            "navy": 0,
            "teal": 0,
            "lavender": 0,
            "turquoise": 0,
            "coral": 0,
            "magenta": 0,
            "crimson": 0,
            "violet": 0}
             # Marquer les couleurs utilisées par les voisins du noeud comme
indisponibles
        for voisin in lst_voisins :
            couleur_du_voisin = G.nodes[voisin]['couleur']
            couleurs_possibles[couleur_du_voisin] = 1
           # Trouver la première couleur disponible dans la liste des couleurs
possibles et l'attribuer au noeud
        for key,value in couleurs_possibles.items():
            if value == 0 :
                nx.set_node_attributes(G, {noeud: key}, 'couleur')
                dico_couleurs_choisies[noeud]=key
                break
    # Retourner le dictionnaire des couleurs choisies pour chaque noeud
    return dico_couleurs_choisies


#-------------------------------------------------------------------------------
#----AFFICHAGE-DU-GRAPH---------------------------------------------------------
#-------------------------------------------------------------------------------


# Fonction que dessine le graphe d'incompatibilité
def dessiner_graph(fichier_choisie):
    # On crée un graph à partir du fichier sélectionné
```

```
    G = cree_graph_incompatibilite(fichier_choisie)
    # On définit une disposition circulaire pour les noeuds du graph
    pos = nx.circular_layout(G)
    # On récupère les couleurs choisies pour chaque signal dans un dictionnaire
    dico_couleurs_choisies = algo_welsh(fichier_choisie)
    # Pour chaque signal et sa couleur dans le dictionnaire
    for signal,couleur in dico_couleurs_choisies.items():
        # On dessine chaque noeud du graph en utilisant la couleur choisie

nx.draw_networkx_nodes(G,pos,node_color=couleur,node_size=700,nodelist=[signal])
    # On dessine les arêtes du graph
    nx.draw_networkx_edges(G,pos)
    # On ajoute les labels des noeuds du graph
    nx.draw_networkx_labels(G,pos)
    # On affiche le graph
    plt.show()


#-------------------------------------------------------------------------------
#----FONCTION-DE-LA-CLIQUE-----------------------------------------------------
#-------------------------------------------------------------------------------


def trouve_clique_max(fichier_choisie):
    # On crée un graph à partir du fichier sélectionné
    graph = cree_graph_incompatibilite(fichier_choisie)
    # On initialise une liste vide pour la clique maximale
    max_clique = []
    # Pour chaque noeud du graph
    for v in graph:
        # On initialise une liste vide pour la clique actuelle
        clique = []
        # On ajoute le noeud au début de la clique
        clique.append(v)
        # Pour chaque autre noeud du graph
        for u in graph:
            # Si tous les noeuds de la clique actuelle sont voisins du noeud u, on
ajoute u à la clique
            if all(x in graph[u] for x in clique):
                clique.append(u)
        # Si la clique actuelle est plus grande que la clique maximale précédente,
on met à jour la clique maximale
        if len(clique) > len(max_clique):
            max_clique = clique
```

```
    # On retourne la taille de la clique maximale
    return len(max_clique)


#-------------------------------------------------------------------------------
#----RESULTAT-NB-FIBRES-NECESSAIRE---------------------------------------------
#-------------------------------------------------------------------------------


def nb_fibre (fichier_choisie):
    # On récupère le dictionnaire des couleurs choisies pour chaque signal
    lst = algo_welsh(fichier_choisie)
     # On calcule le nombre de couleurs différentes utilisées par l'algorithme de
Welsh et Powell
    result_welsh = int(len(set(lst.values())))
    # On calcule la taille de la clique maximale dans le graph
    result_clique = int(trouve_clique_max(fichier_choisie))
     # Si le nombre de fibres nécessaires selon l'algorithme de Welsh et Powell est
égal à la taille de la clique maximale
    if result_clique == result_welsh:
        # On retourne le nombre de fibres nécessaires au minimum
        return f"Un total de {result_welsh} fibres sont nécessaire au minimum "
     # Sinon, on retourne le nombre de fibres théoriquement nécessaire, compris
entre la taille de la clique maximale
    # et le nombre de couleurs utilisées par l'algorithme de Welsh et Powell
    else:
         return f"En théorie, entre {result_clique} et {result_welsh} fibres seront
nécessaires"


#-------------------------------------------------------------------------------
#----ATTRIBUTION-COORDONNEES---------------------------------------------------
#-------------------------------------------------------------------------------


def coordonnees_x(fichier_choisie,signal):
    # On récupère les données du fichier sélectionné
    lst_donnees = recup_donnees(fichier_choisie)
     # On récupère les coordonnées x1 et x2 du signal sélectionné dans la liste des
données
    x1 = lst_donnees[signal-1][1][0]
    x2 = lst_donnees[signal-1][1][1]
    # On retourne les coordonnées x1 et x2
    return x1,x2
```

```
def coordonnees_y(fichier_choisie,signal):
   # On récupère le dictionnaire des couleurs choisies pour chaque signal
   dico_couleurs_choisies = algo_welsh(fichier_choisie)
   # On crée un dictionnaire associant à chaque couleur sa position sur l'axe y
   dico_placement = {
           "white": 0,
           "red": 1,
           "orange": 2,
           "yellow":3,
           "green": 4,
           "blue": 5,
           "purple": 6,
           "pink": 7,
           "brown": 8,
           "gray": 9,
           "beige": 10,
           "olive": 11,
           "navy": 12,
           "teal": 13,
           "lavender": 14,
           "turquoise": 15,
           "coral": 16,
           "magenta": 17,
           "crimson": 18,
           "violet": 19}
   # On récupère la couleur choisie pour le signal sélectionné
   couleur_choisie = dico_couleurs_choisies[signal]
    # On définit y1 et y2 comme étant égaux à la position de la couleur choisie
sur l'axe y
   y1 = dico_placement[couleur_choisie]
   y2 = dico_placement[couleur_choisie]
    # On retourne les coordonnées y1 et y2 ainsi que la couleur choisie pour le
signal sélectionné
   return y1,y2,couleur_choisie


#-------------------------------------------------------------------------------
#----VISUEL-DU-MULTIPLEXAGE-----------------------------------------------------
#-------------------------------------------------------------------------------


def visuel_multi(fichier_choisie):
   # On récupère le dictionnaire des couleurs choisies pour chaque signal
   dico_couleurs_choisies = algo_welsh(fichier_choisie)
```

```
    # On initialise la figure avec les dimensions 12,7 et l'axe
    fig, ax = plt.subplots(figsize=(12, 7))
    # Pour chaque signal et sa couleur dans le dictionnaire
    for signal,couleur in dico_couleurs_choisies.items():
        # On récupère les coordonnées x1, x2, y1, y2 et la couleur choisie pour le
signal
        x1,x2 = coordonnees_x(fichier_choisie,signal)
        y1,y2,couleur_choisie = coordonnees_y(fichier_choisie,signal)
        # On dessine le segment sur l'axe
        ax.plot([x1, x2], [y1, y2], lw=10, color=couleur_choisie)
        # Ajout du numéro du signal sur le segment
        ax.text(((x1+x2)/2)-20, ((y1+y2)/2)-0.07, signal, fontsize=12)
    # On définit les labels des axes du graphique
    ax.set_xlabel('fréquence (kHz)')
    ax.set_ylabel('numéro de fibre')
    # On définit le format de l'axe y pour qu'il n'utilise que des nombres entiers
    ax.yaxis.set_major_locator(MaxNLocator(integer=True))
    # On affiche la figure
    plt.show()
```

## 7.3 affichage.py

```
import networkx as nx
import matplotlib.pyplot as plt
from tkinter import *
from main import *
from cree_donnees import *
from tkinter import filedialog

fichier_choisie = 'donnees.txt'

#-------------------------------------------------------------------------------
----------------------
#----FENETRE-DU-MENU-----------------------------------------------------------
----------------------
#-------------------------------------------------------------------------------
----------------------

# Première fenêtre qui est le menu de notre interface
def w1():
    global fichier_choisie
    # On crée une fenêtre Tkinter
    w1 = Tk()
    # On définit le titre de la fenêtre et ses dimensions
    w1.title("Menu")
    w1.geometry("900x680")
    w1.minsize(480,300)
    # On définit la couleur d'arrière-plan de la fenêtre
    w1.config(background="#2F3CAD")

    # Fonction pour passer à la fenêtre 2
    def aller_w2():
        w1.destroy()
        w2()
    # Fonction pour passer à la fenêtre 3
    def aller_w3():
        w1.destroy()
        w3()
    # Fonction pour importer un fichier
    def import_file():
        global fichier_choisie
        # On utilise une boîte de dialogue pour sélectionner un fichier
        fichier_choisie = filedialog.askopenfilename()
```

```
    # On crée un frame pour les gros boutons
    frame1= Frame(w1,bg="#2F3CAD")
    # On crée un frame pour les autres boutons
    frame = Frame(w1,bg="#2F3CAD")

    # On ajoute du texte à la fenêtre
            title_     =     Label(frame1,text="Bienvenue    sur    notre    projet
SAE3.02",font=("Courrier",45),bg="#2F3CAD")
    title_.pack(side=TOP,) # On affiche le texte

    # On ajoute du texte à la fenêtre
          title_    =    Label(frame,text="Choisissez    une    option    parmi    celles
ci-dessous",font=("Courrier",25),bg="#2F3CAD")
    title_.pack()

    # On crée un bouton pour importer un fichier
          bouton_importerF    =    Button(frame,text="Choisissez    un    fichier    à
importer",font=("Courrier",30),command=import_file,bg="#2F3CAD",fg="black")
    bouton_importerF.pack(pady=10,fill=X)

    # On crée un bouton pour créer un fichier (renvoie vers la fenêtre 3)
           bouton_creefichier    =    Button(frame,text="Créez    un    fichier    de
données",font=("Courrier",30),command=aller_w3,bg="#2F3CAD",fg="black")
    bouton_creefichier.pack(pady=10,fill=X)

    # On crée un bouton pour afficher le graphe d'incompatibilité
                            bout_graph          =          Button(frame,text="Graphe
d'incompatibilités",font=("Courrier",30),command=lambda:
dessiner_graph(fichier_choisie),bg="#2F3CAD",fg="black")
    bout_graph.pack(pady=10,fill=X)

    # On crée un bouton pour afficher la fenêtre 2
                bout_result      =      Button(frame,text="Nombre      de      fibres
nécessaires",font=("Courrier",30),command=aller_w2,bg="#2F3CAD",fg="black")
    bout_result.pack(pady=10,fill=X)

    # On crée un bouton pour afficher le visuel du multiplexage
                        bout_result        =        Button(frame,text="Visuel        du
multiplexage",font=("Courrier",30),command=lambda:visuel_multi(fichier_choisie),b
g="#2F3CAD",fg="black")
    bout_result.pack(pady=10,fill=X)

    # On crée un bouton pour fermer toutes les fenêtres et arrêter le programme
```

```
    fermer = Button(w1, text="Tout fermer ", font=("Courrier", 25), command=exit,
bg="#2F3CAD", fg="#2F3CAD")
    fermer.pack(pady=20, fill=X, side=BOTTOM)

    # On affiche les frames
    frame1.pack(expand=True)
    frame.pack(expand=True)
    # On affiche la fenêtre
    w1.mainloop()



#-------------------------------------------------------------------------------
#----FENETRE-DU-RESULTAT-(NB-FIBRES)-------------------------------------------
#-------------------------------------------------------------------------------



# Fenetre de l'onglet 'nombre de fibres nécessaires'
def w2():
    # On crée la fenêtre 2
    w2 = Tk()
    w2.title("resultat du multiplexage")
    w2.geometry("900x680")
    w2.minsize(480, 300)
    w2.config(background="#2F3CAD")

    # On définit la fonction de retour à la fenêtre 1
    def retour_w2():
        w2.destroy()
        w1()

    # On crée les frames
    frame2= Frame(w2,bg="#2F3CAD")
    frame3= Frame(w2,bg="#2F3CAD")

    # On ajoute le titre de la fenêtre
        title_  =  Label(frame2,  text="Projet  SAE3.02",  font=("Courrier",  50),
bg="#2F3CAD")
    title_.pack(pady=40)

    # On ajoute le titre du résultat du multiplexage
        titlet_  =  Label(frame3,  text=  "Multiplexage",  font=("Courrier",  35),
bg="#2F3CAD")
    # On ajoute un séparateur
```

```
                titlea_          =          Label(frame3,          text=
"-----------------------------------------------",      font=("Courrier",      35),
bg="#2F3CAD")
    # On ajoute le résultat du multiplexage
     titlebis_ = Label(frame3, text= nb_fibre(fichier_choisie), font=("Courrier",
35), bg="#2F3CAD")
    # On ajoute un séparateur
                titleb_          =          Label(frame3,          text=
"-----------------------------------------------",      font=("Courrier",      35),
bg="#2F3CAD")
    # On affiche tous les éléments de la fenêtre
    titlet_.pack()
    titlea_.pack()
    title_.pack()
    titlebis_.pack()
    titleb_.pack()

    # On crée un bouton pour fermer toutes les fenêtres et le programme
     fermer = Button(w2, text="Tout fermer ", font=("Courrier", 25), command=exit,
bg="#2F3CAD", fg="#2F3CAD")
    fermer.pack(pady=20, fill=X, side=BOTTOM)
    # On crée un bouton pour retourner à la fenêtre précédente
       rtr  =  Button(w2,  text="Retour",font=("Courrier",  20),  command=retour_w2,
bg="#2F3CAD", fg="#2F3CAD" )
    rtr.place(x=0, y=0)
    # On affiche les frames
    frame2.pack(side=TOP)
    frame3.pack(expand=True)

    # On lance la boucle Tkinter pour afficher la fenêtre
    w2.mainloop()



#-------------------------------------------------------------------------------
#----FENETRE-POUR-CRÉER-DONNEES.TXT----------------------------------------------
#-------------------------------------------------------------------------------


def w3():
    # On crée une fenêtre Tk et définir son titre et ses dimensions
    w3 = Tk()
    w3.title("cree donnes")
    w3.geometry("900x680")
    w3.minsize(480, 300)
```

```
    w3.config(background="#2F3CAD")

    # On définie une fonction pour détruire w3 et ouvrir w1
    def retour_w3():
        w3.destroy()
        w1()
    # Fonction pour passer à la fenêtre 3
    def aller_w4():
        w3.destroy()
        w4()

    # On définie une fonction pour créer un fichier appelé "donnees.txt"
    # avec le nombre de signaux spécifié par l'utilisateur
    def donnes_f():
        nb_signaux = int(champ_saisie.get())
        cree_donnees_f(nb_signaux)
        afficher_donnees.pack()

    # On crée deux frames dans la fenêtre w3
    frame2 = Frame(w3, bg="#2F3CAD")
    frame3 = Frame(w3, bg="#2F3CAD")

    # On crée également un bouton pour créer le fichier "donnees.txt"
        title_ = Label(frame2, text="Créer un fichier 'donnees.txt'",
font=("Courrier", 45), bg="#2F3CAD")
      title_bb = Label(frame2, text="---------------------------------------",
font=("Courrier", 45), bg="#2F3CAD")
    # On affiche les titres que l'on a créé
    title_.pack(pady=15)
    title_bb.pack()

    # On crée un sous titres donnant les instructions
     title_ = Label(frame3, text= "Choisir le nombre de signaux voulus dans le
fichier", font=("Courrier", 35), bg="#2F3CAD")
     titlebis_ = Label(frame3, text= "(entre 1 et 25)", font=("Courrier", 35),
bg="#2F3CAD")
     # On crée un un champ de saisie permettant de saisir le nombre de signaux
voulue dans le fichier
    champ_saisie = Entry(frame3, font=("Courrier", 25))
     # On crée un bouton permettant d'executer le script qui crée le fichier
'donnees.txt'
      cree_D = Button(frame3, text="Créer le fichier", font=("Courrier", 25),
command=donnes_f, bg="#2F3CAD", fg="black")
```

```
    # On crée un bouton permettant d'exécuter le script qui crée le fichier 'donnees.txt'
        afficher_donnees = Button(frame3, text="afficher les données", font=("Courrier", 25), command=aller_w4, bg="#2F3CAD", fg="black")
    # On affiche tous les éléments de la fenêtre
    title_.pack()
    titlebis_.pack()
    champ_saisie.pack()
    cree_D.pack(pady=20)

    # On crée un bouton pour fermer toutes les fenêtres et le programme
    fermer = Button(w3, text="Tout fermer ", font=("Courrier", 25), command=exit, bg="#2F3CAD", fg="#2F3CAD")
    fermer.pack(pady=20, fill=X, side=BOTTOM)
    # On crée un bouton pour retourner à la fenêtre précédente
    rtr = Button(w3, text="Retour", font=("Courrier", 20), command=retour_w3, bg="#2F3CAD", fg="#2F3CAD" )
    rtr.place(x=0, y=0)

    # On affiche les frames
    frame2.pack(side=TOP)
    frame3.pack(expand=True)

    # On démarre la boucle principale
    w3.mainloop()



#-----------------------------------------------------------------------------
#----FENETRE-POUR-AFFICHER-DONNEES.TXT-----------------------------------------
#-----------------------------------------------------------------------------


def w4():
    # On crée une fenêtre Tk et définir son titre et ses dimensions
    w4 = Tk()
    w4.title("affichage donnees")
    w4.geometry("900x680")
    w4.minsize(480, 300)
    w4.config(background="#2F3CAD")

    # On récupère les données du fichier choisi
    donnees = recup_donnees(fichier_choisie)

    # On définie une fonction pour détruire w4 et ouvrir w1
```

```
    def retour_w4():
        w4.destroy()
        w1()

    # On crée deux frames dans la fenêtre w4
    frame2 = Frame(w4, bg="#2F3CAD")

    # On crée un titre et l'affiche sur notre fenêtre
    # On crée également un bouton pour créer le fichier "donnees.txt"
      title_ = Label(frame2, text="Voici le contenu du fichier 'donnees.txt'",
font=("Courrier", 45), bg="#2F3CAD")
       title_bb = Label(frame2, text="---------------------------------------",
font=("Courrier", 45), bg="#2F3CAD")
    # On affiche les titres que l'on a créé
    title_.pack(pady=15)
    title_bb.pack()

    # On crée les labels pour les titres de colonne
    label1 = Label(w4,text='signal',font=("Courrier", 15), bg="#2F3CAD")
    label1.place(x=350, y=160)
    label2 = Label(w4, text='fréq min',font=("Courrier", 15), bg="#2F3CAD")
    label2.place(x=430, y=160)
    label3 = Label(w4, text='fréq max',font=("Courrier", 15), bg="#2F3CAD")
    label3.place(x=520, y=160)
    # On initialise la coordonnée y de notre premier widget de données à 190
    y1 = 190
     # On parcourt les données récupérées et on affiche chaque élément dans un
label
    for k in donnees:
        label1 = Label(w4,text=k[0],font=("Courrier", 12), bg="#2F3CAD")
        label1.place(x=350, y=y1)
        label2 = Label(w4, text=k[1][0],font=("Courrier", 12), bg="#2F3CAD")
        label2.place(x=430, y=y1)
        label3 = Label(w4, text=k[1][1],font=("Courrier", 12), bg="#2F3CAD")
        label3.place(x=520, y=y1)
         # On incrémente la coordonnées y pour que la prochaine ligne soit écrite
en dessous
        # de la ligne précédente
        y1 += 16

    # On crée un bouton pour fermer toutes les fenêtres et le programme
     fermer = Button(w4, text="Tout fermer", font=("Courrier", 25), command=exit,
bg="#2F3CAD", fg="#2F3CAD")
    fermer.pack(pady=20, fill=X, side=BOTTOM)
```

```
    # On crée un bouton pour retourner à la fenêtre précédente
      rtr = Button(w4, text="Retour", font=("Courrier", 20), command=retour_w4,
bg="#2F3CAD", fg="#2F3CAD" )
    rtr.place(x=0, y=0)

    # On affiche les frames
    frame2.pack(side=TOP)

    # On démarre la boucle principale
    w4.mainloop()

# Lance l'affichage de la fenetre menu
w1()
```

### 7.4 Exemple du fichier donnees.txt

Voici un exemple possible du fichier de données que l'on portrait créer ou bien importer.

```
1 134 220
2 924 1199
3 1000 1403
4 256 741
5 405 834
6 486 781
7 31 320
8 614 888
9 750 1066
10 90 532
```