TP1 : Algorithmes gloutons

Le but de ce TP est d'implanter des algorithmes gloutons vus en cours. On fournit une trame de code en Python pour chaque exercice dans les fichiers `TP1Exo1ChoixCours.py` *et* `TP1Exo2Emetteur.py`.

Coder en Python pour les tps :

- Le langage n'est pas imposé, mais Python est proposé (et les trames de codes sont fournies en Python) et c'est une bonne opportunité d'apprendre ce langage qui sera aussi le langage de programmation pour le module d'algo de L3. C'est la version 3 du langage (python3) qui est à utiliser.
- Un IDE pratique disponible sur les machines du parc de la FdS est `VSCodium` (version libre de `Visual Studio Code`). Il est assez intuitif, il y a de la doc lisible sur la page de `Visual Studio`. Il est possible d'utiliser un autre IDE, bien entendu, voire de compiler vos fichier *.py* sur un terminal (par `> python3 MonProg.py`).
- Pour se former à Python, il y a de nombreux tutoriels accessible. Le tutorial officiel est complet et permet vite d'apprendre sur des points précis. Le cours d'OpenClassroom est très complet et plus pédagogique. On trouve aussi des mémentos sur le langage, qui permettent d'être vite opérationnels : comme celui-ci ou celui-là. De plus, dans la suite du sujet de TP, il y a des commentaires sur le code à utiliser/produire, indiqués par ▷.
  Bon courage, et pas d'inquiétude, ça vient vite !

Exercice 1. *Choix optimal de cours*

Le but de l'exercice est d'implémenter l'algorithme glouton de choix de cours. Les dates de débuts et de fin de cours sont des entiers de 1 à 100. On stocke un ensemble de n cours dans un tableau `Cours`, où pour $i = 0, \dots, n-1$ la variable `Cours[i][0]` contient la date de début du cours i et `Cours[i][1]` sa date de fin.
Le fichier `TP1Exo1ChoixCours.py` contient une trame du code à produire.

1. Ouvrez le fichier `TP1Exo1ChoixCours.py` avec `VSCodium`. Essayer de comprendre un peu la structure et syntaxe du fichier, notamment :

   ▷Regarder les 'import' en début de fichier. En laissant son pointeur de souris dessus, `Vscodium` donne des infos sur leur utilité.
   ▷Les définitions de fonctions se font par `def`. Noter bien les `:` en fin de ligne et les indentations. Cela marque les blocs en Python.
   ▷Les variables sont typées automatiquement en Python, à leur initialisation.
   ▷Les 9 dernières lignes du fichiers sont hors de toute fonction, elles forment le 'main' du programme.
   ▷Noter la structure des boucles `for`. Aller chercher dans les mémentos ou autres à quoi correspond exactement `range(0,n)`. Noter la structure des test (`if`) et la gestion des entrées/sorties (`input` et `print`).
   ▷Expliquer la nécessité de l'usage de `int(...)` autour des `input(...)` utilisés.

   Exécuter le programme et observer le résultat produit.
2. Compléter la fonction `CoursAuHasard(n)` qui doit remplir le tableau `cours` en choisissant pour le cours i avec $i = 0, \dots, n-1$ une date de début choisie au hasard entre 1 et 90 et une durée du cours choisie au hasard entre 1 et 10.

   ▷La fonction `randint()` permet de renvoyer un entier au hasard. Trouver son fonctionnement exact dans les mémentos ou cours.
   ▷Noter la déclaration de tableau (ou liste) et la façon de rajouter un élément par `append()`.
3. Compléter la fonction `TriBulles(Cours)` qui trie le tableau `Cours` par dates de fin de cours croissantes en utilisant un tri à bulles. Le pseudo-code d'un tri à bulle est disponible dans la fiche de TD 1. **Bien tester** votre algorithme sur un petit nombre de cours.
   Remarque : dans un second temps on pourra utiliser un tri plus rapide si le temps le permet.

   ▷Pour échanger le contenu de deux variables a et b, on peut utiliser l'instruction `a,b=b,a`. Et cela fonctionne même si a et b sont des tableaux ou des listes...

4. Compléter la fonction `ChoixCoursGlouton(Cours)` qui implante l'algorithme CHOIXCOURSGLOUTON du cours. Le tableau `L[]` sera intialisé à vide et contiendra les cours choisis après l'appel de la fonction `ChoixCoursGlouton`. De plus, celle-ci affichera le nombre de ces cours choisis. **Tester et vérifier** votre algorithme. On pourra notamment essayer l'exemple pré-rempli (choix 1) pour obtenir l'affichage suivant :

```
Ensemble de cours disponibles:
[[76,78],[12,17],[13,15],[19,28],[12,20],[43,45],[44,45],[1,8],[68,78],[85,88]]
Cours triés par dates de fin croissantes :
[[1,8],[13,15],[12,17],[12,20],[19,28],[43,45],[44,45],[76,78],[68,78],[85,88]]
Nombre de cours choisis : 6
Choix de cours effectués :
[[1,8],[13,15],[19,28],[43,45],[76,78],[85,88]]
```

5. (*Bonus*) Écrire une fonction `Alarmes` qui permet de résoudre le problème des alarmes incendies : on veut faire sonner le moins possible d'alarmes incendies, tout en garantissant qu'au moins une alarme retentit pendant chaque cours. *Pour cette question, vous devez étudier le problème (et trouver l'algorithme glouton), déterminer les structures de données à utiliser, implanter votre solution, et* ***effectuer des tests****. Aucune trame de code n'est fournie, même pour les tests...*
Comparer le nombre d'alarmes minimales et le nombre maximal de cours trouvé par CHOIXCOURSGLOUTON. Expliquer (voire démontrer !) ce que vous observez.

Exercice 2. SETCOVER *en dimension 2*

1. Compiler et exécuter le fichier `TP1Exo2Emetteurs.py`. Noter l'usage de `mathplotlib` pour obtenir une sortie graphique. Le programme tourne avec $n = 100$ maisons et un rayon de couverture $rayon = 120$.

   ▷La fonction `len` permet de retourner la taille d'un tableau intialisé.

2. Implémenter la fonction `GenererMaison(Maison, n)` qui remplit le tableau `Maison` avec des coordonnées pour les maisons choisies au hasard dans $[1,1000] \times [1,1000]$.

   ▷La fonction `pass` ne fait rien, mais Python n'accepte pas de blocs vides. Elle empêche ici d'avoir une erreur de syntaxe. Vous pouvez l'enlever une fois votre fonction remplie.

3. Implanter la fonction `Couvre(Maison,i,j)` qui retourne vrai si, et seulement si, les maisons i et j se situent à moins de `rayon` l'une de l'autre.

   ▷Trouver commet mettre une valeur au carré et utiliser la racine carrée en Python...

4. Implanter la fonction `choixMaison(Maison,MaisonsRestantes)` qui retourne l'indice de la prochaine maison sur laquelle placer un émetteur. Le tableau `MaisonsRestantes` contient pour chaque i de 0 à $n-1$ un marqueur 0 ou 1 pour indiquer si la $i^{\text{ème}}$ maison est déjà couverte. Le tableau `MaisonsRestantes` est initialisé dans la fonction `choixEmetteurGlouton`.
5. Implanter la fonction `choixEmetteurGlouton` qui trouve un ensemble d'émetteurs qui couvre toutes les maisons, avec l'algorithme vu en cours. Le tableau Emetteur contient 1 à l'indice i si un émetteur est placé sur la maison i et 0 sinon. Vous devrez obtenir des résultats graphiques semblables à ceux donnés ci-après.
6. (*Bonus*) Implanter la fonction `ChoixEmetteursOpt` qui effectue le même travail que `ChoixEmetteurs` mais avec un algorithme qui renvoie une solution optimale.

   ▷L'instruction `for perm in permutations(range(n)):` permet de parcourir toutes les permutations de l'ensemble $\{1,\dots n\}$.

7. (*Bonus*) Rechercher un exemple où l'algorithme glouton donne un résultat non optimal.

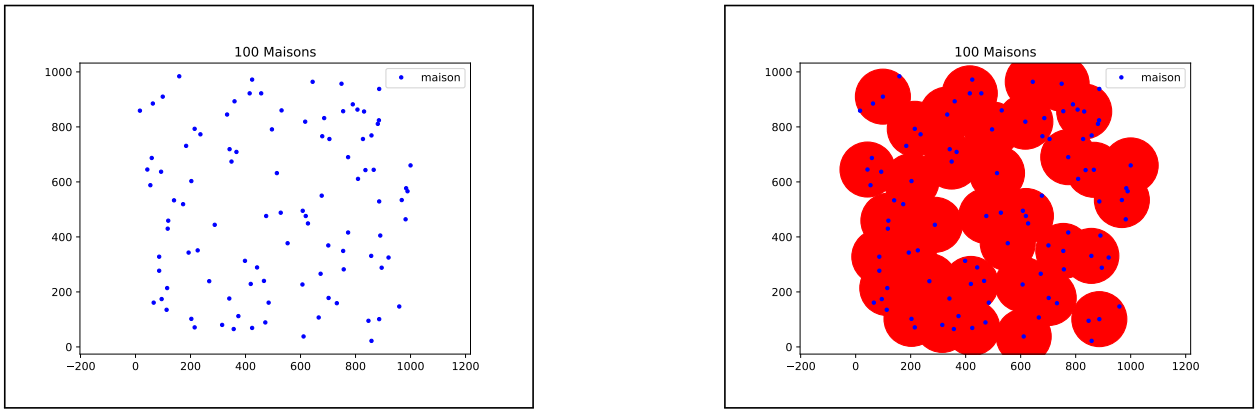

FIGURE 1 – À gauche, les maisons (en bleu) dans le plan, à droite les couvertures des émetteurs choisis (en rouge).