# Algorithmique et Structures de Données - Avancé

**Rappel :** Chaque projet comporte deux parties : la partie développement et la partie rapport. Même si certains projets peuvent être réalisés en binôme, ils seront soutenus individuellement et le rapport est donc propre à chaque étudiant. Pour la partie développement, quand aucun langage de programmation n'est spécifié, l'étudiant pourra choisir entre C ou JAVA. Concernant le rapport, il doit au minimum reprendre la problématique, la stratégie (la conception) envisagée pour la résoudre, le détail de la solution effective (+ si nécessaire, des extraits pertinents du code source) et enfin la conclusion et les références. Pour les projets réalisés en binôme, une section sur le travail en équipe est demandée.

## Liste des projets

**1. Interprète LISP** — (seul ou en binôme)

Écrire un programme capable d'exécuter du Lisp standard. Les programmes Lisp peuvent être :

- directement écrits en ligne de commande ;

  Le code est interprété une fois que toutes les parenthèses ouvertes ont été fermées. Ceci permettrait d'écrire des fonctions qui tiennent sur plusieurs lignes sans passer par un éditeur de texte.
- chargés et exécutés à partir d'un fichier.

L'interprète doit :

- avoir une gestion dynamique de la mémoire ;
- gérer les fonctions arithmétiques ;
- inclure `t, nil, undefined, car, cdr, cons, quote, print, set, if, cond, eq, atom, while` et le plus important, **defun** (ou **de**) ce qui permettrait de créer les fonctions manquantes directement en Lisp et les charger au démarrage de l'interprète (ex. : `append, length, caar, cadr,` ...).

**2. Calculatrice** — (seul ou en binôme)

Écrire en C une calculatrice scientifique en ligne de commande. Vous pouvez prendre pour exemple **bc** (saisir : `$ bc -l` dans un terminal). D'une part, cette calculatrice doit pouvoir parser toutes les expressions, gérer les opérations arithmétiques de base, le parenthésage, les priorités ainsi que les fonctions telles que : `sqrt, pow, cos, sin,` ... (voir les fonctions de `math.h`). Par ailleurs, elle doit pouvoir manipuler des variables ainsi que quelques instructions conditionnelles (`if, while,` ...).

**3. Rechercher - Remplacer** — (seul)

Nous souhaitons développer un outil (une commande `shell`) facilitant les opérations de rechercher/remplacer dans des fichiers texte. Cet outil prend plusieurs entrées possibles :

- -f fichier1 [fichier2 [...]] : pour rechercher dans les fichiers cités ;
- -d répertoire1 [répertoire2 [...]] : pour rechercher dans les fichiers présents dans les répertoires cités ;
- -R : (uniquement avec -d) pour effectuer une recherche récursive dans les

répertoires cités ;
- mask="un masque" : (uniquement avec -d) pour restreindre la recherche seulement aux noms de fichiers correspondant à "masque". Exemple : mask="*.c" demande de rechercher dans les fichiers `C` ;
- find="mot1" : rechercher le pattern "mot1" ;
- replace="mot2" : remplacer par le pattern "mot2".

**4. Métro** — (seul ou en binôme)

Créez un programme qui à partir d'une station de métro de départ, trouve le chemin permettant de se rendre à une station de métro d'arrivée. Ce programme doit utiliser des structures chaînées. On prendra comme exemple une partie du métro parisien (Lignes 1, 2, 3, 4, 5 et 6). La définition des lignes de métro est donnée dans un fichier texte ; chaque ligne est définie par un temps moyen entre chaque station (ex. 90s) et dans l'ordre, les noms des stations. Les intersections des lignes (les correspondances) seront calculées automatiquement ; chaque correspondance a un coût de 5mn. On pourra demander soit le chemin le moins coûteux en temps soit le chemin comportant le moins de correspondances.

**5. Le plus court chemin dans un labyrinthe** — (seul)

Écrire un programme qui génère aléatoirement des labyrinthes connexes (il existe au minimum un chemin reliant deux points) de taille quelconque. Trouver (utilisez l'algorithme de Dijkstra) le plus court chemin reliant deux points du labyrinthe.

**6. U.M.T.S.** — (seul)

Vous avez la possibilité de charger une image satellite de dimension quelconque (par ex. prendre une image PNG en $2^8$ ou en $2^{16}$ niveaux de gris où chaque pixel représente l'altitude du point à la coordonnée correspondante). Cette donnée représente donc le relief du terrain sur lequel une société de Télécoms souhaite placer $N$ antennes de relais. Sachant que ces antennes ne couvrent qu'un rayon de largeur $R$ et qu'elles n'émettent pas vers le haut (une demi sphère), écrire le programme qui calcule le meilleure emplacement pour ces $N$ antennes de façon à obtenir la meilleure couverture.

**7. Arbre Généalogique** — (seul)

Créer un programme qui gère un arbre généalogique. On pourra faire des recherches dans l'arbre généalogique, afficher les «`fils de, fille de, père de, mère de ...`» , ou bien effectuer des tris par noms ou par date de naissance par exemple. Les seules structures de données admises sont les listes chaînées.

**8. Arbre Binaire** — (seul)

Écrire en C, une strucutre générique de stockage de données. La structure prend la forme d'un arbre binaire préservant l'ordre et l'unicité des éléments (un pointeur vers une fonction de comparaison doît être donné). Ajouter les fonctionnalités d'insertion, de recherche/récupération et de suppression d'élément. Enfin, comparer (pour l'ensemble de ces fonctionnalités) votre implémentation à celle proposée en JAVA (ex. TreeSet).

**9. Huffman** — (seul)

Implémenter le codage de Huffman et en faire un outil de compression/décompression de fichiers.