# Programme de colle - Semaine 8 (colle du 5 novembre)

*La démonstration des énoncés marqués d'une étoile est exigible*

## 1 Langages réguliers

- Mots : alphabet, mot sur un alphabet, concaténation, puissance d'un mot, structure de monoïde.
- Langages : opérations ensemblistes sur les langages (union, intersection, différence, complémentaire), concaténation de deux langages, étoile de Kleene d'un langage, **l'union d'une famille finie de langages réguliers est un langage régulier (*)**
- Langages réguliers (aussi appelés langages rationnels): définition inductive de la classe des langages réguliers, **tout langage fini est régulier (*)**
- Expressions régulières : définition inductive, langage dénoté par une expression régulière (notation $\mu(e)$)
- **Un langage est régulier si et seulement si il existe une expression régulière qui le dénote (*)** [NB : interroger sur un seul sens uniquement]
- Expressions régulières étendues : aucune connaissance spécifique n'est exigible mais les étudiants doivent savoir trouver une expression régulière équivalente pour chacune des opérations introduites.

## 2 Parcours de graphes

**Tous les algorithmes ci-dessous doivent pouvoir être écrits aisément par l'élève en pseudo-code**

- Définitions usuelles : graphe orienté, non orienté, sommet/nœud, arête/arc, voisins (entrants/sortants), degré, chemins, cycles, graphe pondéré, etc.
- Représentation par matrice d'adjacence ou listes d'adjacence.
- Généralités sur les parcours : notion de nœud ouvert (découvert mais non encore exploré), de nœud fermé (exploré, ce qui découvre ou redécouvre les voisins), arborescence d'un parcours
- Parcours en largeur : avec une file pour enregistrer les sommets ouverts
- Parcours en profondeur : avec une pile pour enregistrer les sommets ouverts, ou par récursivité
- Algorithme de Dijkstra : sur des graphes pondérés avec des **poids positifs**. **La valeur $g(n)$ attribuée à chaque nœud fermé est le poids d'un chemin optimal menant de la source à $n$ (*)**. Algorithme exprimé à l'aide d'une file de priorité.
- Algorithme A* : heuristique, heuristique admissible, heuristique monotone; **une heuristique monotone est admissible(*)**. Si l'heuristique est admissible alors A* construit des chemins optimaux. Si de plus l'heuristique est monotone, il n'est pas nécessaire de ré-ouvrir les sommets et la complexité est équivalente à Dijkstra.

## 3 Automates finis

Les étudiants doivent savoir appliquer les constructions et algorithmes sur les automates.

- Automate fini déterministe (afd) : définition et calcul d'un afd, notion de calcul dans un afd,
- Langage reconnu (ou accepté) par automate : calcul réussi, mots reconnus (ou acceptés). Classe des langages reconnaissables.
- Accessibilité et co-accessibilité; afd émondé
- Afd complet; complétion d'un afd
- **Automate complémentaire : Si $L$ est reconnaissable alors $\bar{L}$ l'est (*)**
- Automate fini non déterministe (afnd) : définition et calcul d'un afnd, mots reconnus, langage accepté
- Déterminisation d'un afnd

- Automate fini non déterministe avec transitions instantanées ($\varepsilon$-afnd) : définition, $\varepsilon$-clôture, calculs d'un $\varepsilon$-afnd, mots reconnus, langage accepté
- Déterminisation d'un $\varepsilon$-afnd (automate des parties)

*NB : le lien entre langages réguliers et langages reconnaissables n'a pas encore été étudié*

## 4 Union-Find et algorithme de Kruskal

- Structure Union-Find : implémentation naïve favorisant l'opération `FIND` avec un tableau dans lequel chaque case $i$ contient le représentant de l'élément numéro $i$. Complexités des opérations.
- Structure Union-Find : implémentation à l'aide d'une forêt favorisant l'opération `UNION`.

  1. Optimisation 1 : en plaçant systématiquement l'arbre de plus petite hauteur comme fils lors de l'opération union. **Dans ce cas, la hauteur de chaque arbre ne dépasse pas $\log_2(m)$ avec $m$ la taille de l'arbre considéré (*)**
  2. Optimisation 2 : en compressant les chemins lors de l'opération `FIND`. La complexité amortie obtenue est hors programme.
- Notion d'arbre couvrant d'un graphe non orienté. Algorithme générique de construction d'un arbre couvrant : utilisation de Union-Find dans ce cadre.
- Arbre couvrant de poids minimal d'un graphe non orienté pondéré. Existence. **Algorithme de Kruskal** : savoir décrire l'algorithme en pseudo-code avec la structure Union-Find, savoir l'appliquer sur un exemple, la preuve n'est pas au programme de colle.