Option Info MP/MP* : Conseils pour les concours

Conseils d'ordre général

1. Faire attention aux types des éléments considérés, toujours renvoyer un élément du type demandé !
2. Même si ce n'est pas obligatoire comme en Python, indenter les programmes pour les rendre plus lisibles, ne pas hésiter à insérer des commentaires.
3. Ne pas confondre liste et vecteur !! (et ne pas confondre les listes Caml et les listes Python) En particulier, les listes sont immuables et on ne peut pas accéder en temps constant au ième élément d'une liste (seulement à sa tête et à sa queue), il faut donc en général faire un filtrage quand on demande d'écrire une fonction sur les listes ! Pour les vecteurs, on peut accéder et modifier le ième élément mais la taille du vecteur est fixée dès le début.
4. Les noms de variables et de fonctions commencent toujours par une minuscule, les majuscules sont pour les constructeurs et les exceptions.
5. Ne pas oublier les `;` entre les différentes instructions, le `rec` après le `let` quand on écrit une fonction récursive, les `;;` à la fin du programme, les `done` à la fin des boucles `for` et `while`.
6. Quand on introduit une variable locale dans une fonction : `let a = ... in ...`.
7. Les objets introduits en Caml sont par défaut non modifiables ! Si l'on veut créer une variable on doit utiliser des références : `let a = ref ...`. Pour la modifier : `a := ...` et pour accéder à son contenu : `!a`. En général les références sont plutôt adaptées aux algorithmes itératifs (avec des boucles `for`, `while`) et pas aux récursifs.
8. L'énoncé impose parfois des restrictions (n'écrire que des fonctions récursives, pas de références...), en particulier les sujets CCP.
9. Bien justifier les complexités : ne pas oublier d'étape (en particulier penser aux coûts des fonctions auxiliaires), pour les fonctions récursives, on a souvent une relation de récurrence qui apparaît. Connaître la solution sous forme d'un O des récurrences suivantes : $C(n) = C(n-1) + 1$, $C(n) = C(n-1) + n$, $C(n) = C(n/2) + 1$, $C(n) = 2 * C(n/2) + n$.

Arbres binaires - Tas

1. Connaître la définition d'un arbre binaire de recherche et son implémentation en Caml.
2. Connaître les bornes de la hauteur d'un arbre binaire en fonction du nombre de nœuds et les cas extrémaux.
3. Connaître le principe d'une preuve par induction et savoir en rédiger une (on peut aussi la rédiger sous forme d'une récurrence sur la hauteur). Quand on demande de prouver un résultat sur un arbre binaire, avoir le réflexe d'y penser.

4. Savoir coder les fonctions usuelles sur les ABR : hauteur, nombre de nœuds/feuilles, recherche d'un élément, insertion aux feuilles/à la racine, suppression, parcours infixe/ préfixe/ postfixe, et connaître leurs complexités.

5. Savoir trier à l'aide d'un ABR (et la complexité dans le pire/meilleur des cas).

6. Savoir implémenter un dictionnaire à l'aide d'un ABR et le coût des opérations (renvoyer la valeur associée à une clé, ajouter/supprimer un couple).

7. Connaître la définition d'un tas (en particulier, un tas est équilibré et sa hauteur est logarithmique en le nombre de nœuds). On implémente les tas par des tableaux (la taille est donc fixée). Savoir où se situe le minimum d'un tas min.

8. Savoir passer de la représentation d'un tas sous forme de tableau à celle sous forme d'arbre. Savoir déterminer l'indice du père d'un sommet et les indices de ses fils. Savoir déterminer le chemin de la racine à un élément dont on connaît l'indice.

9. Connaître le principe du tri par tas, savoir l'exécuter sur un exemple et coder les différentes étapes (descente/montée d'un élément, construction du tas...) et leurs complexités.

10. Savoir représenter une file de priorité à l'aide d'un tas, et le coût des différentes opérations (insertion, suppression...).

Logique

1. Connaître la signification des connecteurs logiques $\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow$ et les tables de vérité associées. Savoir écrire un $\Rightarrow$ à l'aide des connecteurs $\wedge, \vee, \neg$.

2. Connaître la définition inductive des formules logiques et leur implémentation en Caml.

3. Connaître les définitions suivantes : formules équivalentes, tautologie, contradiction, instanciation, formule satisfiable.

4. Savoir transcrire un énoncé en français sous forme de formules logiques (s'entraîner sur des sujets CCP). Attention à certains pièges, le "ou" logique est inclusif par exemple.

5. Savoir déterminer la table de vérité d'une formule et le coût de sa construction en fonction du nombre de variables.

6. Savoir représenter une formule sous forme d'arbre.

7. Connaître les règles de calcul : double négation, lois de De Morgan, tiers exclu, distributivité.

8. Savoir ce qu'est une formule sous forme normale conjonctive/disjonctive. Savoir trouver une formule équivalente sous FNC/FND (il y a plusieurs manières de procéder). Questions subsidiaires : est-ce que toute formule est équivalente à une formule sous FNC/sous FND ? Entre la FNC et la FND, laquelle est la plus pratique pour décider si une formule est satisfiable ?

Langages et automates

1. Connaître les définitions suivantes : mot, langage, concaténation de deux mots, préfixe, suffixe, facteur, sous-mot.

2. Etant donnés deux langages L_1 et L_2, savoir ce que signifient : $L_1 + L_2$, $L_1.L_2$, L_1^*, L_1^+.

3. Connaître la définition inductive des expressions rationnelles, et du langage associé à une expression rationnelle. Attention à ne pas confondre l'expression rationnelle et le langage associé. Savoir ce qu'est un langage rationnel.

4. Savoir décrire le langage associé à une expression rationnelle sur des exemples simples, et inversement donner une expression rationnelle représentant des langages simples (mots commençant par a, contenant au plus un b, un nombre pair de a etc...)

5. Connaître la définition d'un automate fini, ainsi que des mots suivants : état initial, terminal (ou final, ou encore acceptant), transition, calcul, langage reconnu par un automate, automates équivalents, automate déterministe, automate complet.

6. Savoir construire un automate reconnaissant des langages simples (s'entraîner sur des sujets CCP).

7. Savoir déterminiser un automate (et le coût de l'opération dans le pire des cas), le rendre complet.

8. Dans le cas d'un automate déterministe complet, savoir ce qu'est la fonction de transition.

9. Connaître le théorème de Kleene.

10. Savoir construire l'automate de Glushkov associé à une expression rationnelle en utilisant l'algorithme de Berry-Sethi. Bien connaître les définitions des différents outils utilisés : automates locaux, expressions rationnelles linéaires, ensembles P, S et F. Savoir déterminer les ensembles P, F et S d'une union, concaténation ou passage à l'étoile d'un langage (attention aux différents cas : si L_1 contient le mot vide...). Connaître les complexités des différentes étapes de la construction.

11. Savoir utiliser le théorème de Kleene pour montrer des propriétés de clôture des langages rationnels (passage au complémentaire, intersection, passage au miroir...). Attention, il faut souvent supposer les automates déterministes complets dans les preuves. Savoir déterminer un automate reconnaissant l'union/l'intersection de deux langages (automate produit).

Graphes

1. Connaître les définitions suivantes : graphe non orienté, graphe orienté, graphe pondéré, sommets adjacents, degré d'un sommet, degré entrant/sortant dans le cas orienté, chemin dans un graphe, sommets connectés, composantes connexes (cas non orienté)/fortement connexes (cas orienté).

2. Connaître les deux implémentations usuelles des graphes en Caml : listes d'adjacence, matrice d'adjacence. Savoir passer de l'une à l'autre et les avantages de chacune. Savoir déterminer laquelle adopter en fonction du problème à résoudre.

3. Pour chacune des deux représentations, savoir coder les fonctions suivantes : ajout/suppression d'une arête, d'un sommet, rendre non orienté un graphe orienté.

4. Savoir réaliser le parcours en largeur/profondeur d'un graphe. Savoir les coder en Caml en choisissant l'implémentation la plus adaptée. Connaître leur complexité.

5. Savoir déterminer les composantes connexes/fortement connexes à l'aide d'un parcours.

6. Dans le cas d'un graphe pondéré, connaître les définitions suivantes : distance entre deux sommets, chemin de poids minimal. Savoir que si le graphe ne comporte pas de cycle de poids négatif, il y a toujours un chemin de poids minimal.

7. Connaître les algorithmes de Floyd-Warshall et de Dijkstra, savoir les appliquer à des exemples, les coder en choisissant la meilleure implémentation, connaître leurs complexités. Pour l'algorithme de Dijkstra, il est bien de savoir qu'on peut l'améliorer en utilisant un tas et la complexité dans ce cas.