Checklist de révisions

Avertissement

Ce document a pour but de vous aider à faire le point pendant vos révisions pour les épreuves écrites. Il ne constitue pas un planning de révisions; un planning de révisions doit être conçu de manière individuelle en tenant compte de vos forces, de vos faiblesses et de vos besoins en matière de révisions. Ces listes ne prétendent pas décrire l'intégralité du programme : consulter le programme officiel qui seul fait foi.

1 Bases de données relationnelles

- [ ] Schéma relationnel d'une base de données relationnelles : tables, attributs, enregistrements, clefs primaires, clefs étrangères.
- [ ] Modèle entité-associations et traduction en schéma relationnel.
- [ ] Filtrage des lignes avec le mot-clef `WHERE`.
- [ ] Jointure de tables avec la syntaxe `T1 JOIN T2 JOIN ... JOIN Tn ON condition`.
- [ ] Opérations ensemblistes sur les tables `UNION`, `INTERSECT`, `EXCEPT`.
- [ ] Utilisation des mots-clefs : `DISTINCT`, `LIMIT`, `OFFSET`, et `ORDER BY` pour ré-arranger les résultats.
- [ ] Calculs sur les colonnes. Opérations statistiques `SUM, AVG, MIN, MAX, COUNT`.
- [ ] Regroupement de lignes avec `GROUP BY`.
- [ ] Filtrage des agrégats avec le mot-clef `HAVING` (à ne pas confondre avec `WHERE`).

2 Algorithmes

Pour chaque algorithme il faut le connaître, savoir le programmer, et pour beaucoup savoir justifier qu'il est correct et évaluer sa complexité.

2.1 Étude des algorithmes

- [ ] Complexité en temps ou en espace d'un algorithme mesuré sur une entrée de taille n.
- [ ] Notations $O(f(n))$ et $\Theta(f(n))$ pour les complexités asymptotiques.
- [ ] Complexité dans le pire cas, complexité moyenne, complexité amortie.

- [ ] Preuve de terminaison d'un algorithme. Utilisation d'un **variant**.
- [ ] **Preuve par récurrence** : en posant systématiquement la propriété $P(n)$ avec n un entier. Récurrence simple, double, forte.
- [ ] **Preuve par invariant** : en posant systématiquement la propriété invariante I.
- [ ] **Preuve par induction** : en posant systématiquement la propriété $P(x)$ où x est un objet défini inductivement; par exemple $P(a)$ avec a un arbre, $P(F)$ où F est une formule, $P(e)$ où e est une expression régulière, $P(L)$ où L est un langage régulier, ...
- [ ] Correction partielle et totale d'un algorithme.

2.2 Tableaux

- [ ] Tri par sélection. Complexité en $\Theta(n^2)$.
- [ ] Tri par insertion. Complexité pire cas en $O(n^2)$.
- [ ] Tri rapide. Complexité pire cas en $O(n^2)$.
- [ ] Tri par tas. Complexité pire cas en $O(n\log(n))$.
- [ ] Recherche par dichotomie dans un tableau trié.
- [ ] Recherche de la valeur ou de la position d'un maximum.

2.3 Listes

- [ ] Tri par sélection. Complexité en $\Theta(n^2)$.
- [ ] Tri par insertion. Complexité pire cas en $O(n^2)$.
- [ ] Tri par partition-fusion. Stratégie diviser pour régner. Complexité en $O(n\log(n))$.
- [ ] Recherche de la valeur maximale.

2.4 Arbres

- [ ] Taille et hauteur d'un arbre.
- [ ] Parcours d'un arbre en largeur à l'aide d'une file. Complexité linéaire.
- [ ] Parcours d'un arbre en profondeur par récursivité (ou avec une pile). Complexité linaire.
- [ ] Ordre de parcours préfixe, postfixe, infixe (pour un arbre binaire).

- [ ] **Arbres binaires de recherche :** recherche, insertion, suppression. Complexité en $O(h)$ où h est la hauteur de l'arbre.
- [ ] **Arbres binaires équilibrés rouge-noir :** définition, hauteur noire, recherche et insertion. L'opération de suppression dans un arbre rouge noire est (trop ?) compliquée et peut être mise de côté pendant les révisions.
- [ ] **Tas binaires :** création ($O(n)$), extraction ($O(\log(n))$), insertion ($O(\log(n))$), modification ($O(\log(n))$) d'une clef (en utilisant des percolations vers le haut ou le bas)
- [ ] Arbres n-aires (aussi appelés arbres généralisés).

2.5 Textes

- [ ] Recherche naïve d'un motif à l'aide d'une fenêtre glissante. Complexité en $\Theta(np)$.
- [ ] **Boyer-Moore :** Amélioration de l'algorithme naïf avec des sauts de fenêtre.
- [ ] Algorithme de recherche de motif de **Rabin-Karp** utilisant une empreinte.
- [ ] Compression de texte avec un code préfixe généré par l'algorithme de **Huffman**.
- [ ] Compression de texte avec l'algorithme de **Lempel-Ziv-Welch** : codage en construisant progressivement le dictionnaire, décodage en reconstituant à la volée le dictionnaire utilisé.

2.6 Graphes

- [ ] Parcours en largeur à l'aide d'une file. Complexité linéaire.
- [ ] Parcours en profondeur par récursivité (ou avec une pile). Complexité linéaire.
- [ ] Calcul des **composantes connexes** d'un graphe non orienté : à l'aide d'un parcours.
- [ ] **Tri topologique** d'un graphe orienté acyclique.
- [ ] Calcul des **composantes fortement connexes** d'un graphe orienté : à l'aide de l'algorithme de **Kosaraju**.
- [ ] **Algorithme de Dijkstra** : calcul des chemins optimaux depuis un sommet unique (uniquement avec des poids positifs).
- [ ] **Algorithme de Floyd-Warshall** : calcul des chemins optimaux entre chaque paire de sommets (sans cycle de poids négatifs).
- [ ] **Algorithme A*** : Similaire à l'algorithme de Dijkstra en utilisant une heuristique. Heuristique admissible. Heuristique monotone. Si l'heuristique est admissible l'algorithme est correct. Si de plus elle est monotone il n'y a pas besoin de revisiter les sommets noirs.

- [ ] **Arbre couvrant de poids minimal** : en utilisant la structure de donnée Union-Find.
- [ ] **Couplage maximal** : recherche d'un couplage maximal avec les chemins augmentants.

2.7 Algorithmes probabilistes

- [ ] Algorithmes de Las Vegas : le but est d'accélérer les choses grâce au hasard.
- [ ] Algorithmes de Monte-Carlo : le but est d'avoir une bonne solution rapidement (temps déterministe) mais qui ne sera pas toujours exacte.

2.8 Intelligence artificielle

- [ ] Apprentissage supervisé : algorithme des **k-plus proches voisins**. Lien avec les arbres k-dimensionnels.
- [ ] Apprentissage supervisé : algorithme **ID3**.
- [ ] Évaluation des résultats avec la matrice de confusion.
- [ ] Apprentissage non supervisé : algorithme des **k-moyennes**.
- [ ] Apprentissage non supervisé : **clusterisation hiérachique ascendante**.
- [ ] Jeu d'accessibilité à deux joueurs : arène, partie, stratégies.
- [ ] Détermination des stratégiques gagnantes par le calcul des attracteurs pour les petits jeux.
- [ ] **Algorithme min-max**. Élagage $\alpha - \beta$.

3 Structures de données

Pour chaque structure de données, il faut savoir la liste des opérations implémentées (structure abstraite) et aussi les manières classiques de les implémenter (structures concrètes).

- [ ] **Liste** : implémentations à l'aide d'un tableau, à l'aide de maillons simplements chaînés
- [ ] **Pile** : implémentations à l'aide d'une liste, d'un tableau.
- [ ] **File** : implémentations à l'aide d'un tableau (file bornée), d'une liste ou de deux listes (complexité amortie linéaire des opérations).
- [ ] **Dictionnaire** : implémentations à l'aide d'une liste associative, d'un arbre binaire de recherche, d'un arbre k-dimensionnel, d'une table de hachage.

- [ ] **File de priorité** : implémentation à l'aide d'une liste triée, d'un tas binaire codé dans un tableau.
- [ ] **Union-Find** : implémentations à l'aide d'un tableau, à l'aide d'une forêt (avec les deux optimisations vues).

4 Méthodes algorithmiques

Pour chacune de ces méthodes algorithmiques, il faut savoir identifier les situations dans lesquelles elle peut être utilisée et savoir la mettre en œuvre. Les cas les plus difficiles devraient être guidés.

- [ ] Dichotomie.
- [ ] Diviser pour régner.
- [ ] Retour sur trace (*backtracking*)
- [ ] Algorithmes gloutons.
- [ ] Programmation dynamique.
- [ ] Séparation et évaluation (*branch and bound*).

5 Gestion des ressources de la machine

- [ ] **Mémoire** : pile et tas.
- [ ] Allocation dynamique sur le tas avec `malloc`. Libération avec `free`.
- [ ] **Fichiers et flux** : savoir écrire ou lire dans un fichier ou un flux, notions de flux standards (`stdout, stdin, stderr`).
- [ ] Programmation concurrente avec plusieurs fils d'exécution.
- [ ] Synchronisation à l'aide de **Mutex** et de **Sémaphores**.
- [ ] **Algorithme de Peterson** pour 2 fils d'exécution.
- [ ] **Algorithme de la boulangerie de Lamport** pour n fils d'exécution.

6 Informatique théorique

6.1 Logique

- [ ] **Formules propositionnelles** : variables propositionnelles, connecteurs logiques.
- [ ] Sémantique des formules propositionnelles à l'aide des valuations.
- [ ] Formules satisfiables, tautologies.
- [ ] Tables de vérité, algorithme de Quine.
- [ ] Équivalence de formules. Équivalences usuelles.
- [ ] Formes normales conjonctives et disjonctives.
- [ ] **Formules de la logique des prédicats** : constantes, variables, fonctions, termes, prédicats, formules atomiques, quantificateurs.
- [ ] Déduction naturelle.

6.2 Expression régulières

- [ ] Mots et langages.
- [ ] Définition inductive des **langages réguliers**.
- [ ] Définition inductive des **expressions régulières**.
- [ ] Expression régulière étendue POSIX.
- [ ] Les expressions régulières dénotent les langages réguliers.

6.3 Automates

- [ ] Automate fini déterministe.
- [ ] Automate fini non déterministe avec ou sans ε-transitions.
- [ ] Langage reconnu par un automate.
- [ ] **Théorème de Klenne** : les langages réguliers et les langages reconnaissables sont les mêmes.
- [ ] Passer d'une expression régulière à un automate : algorithme de Berry-Sethi, langages locaux, automate de Glushkov.

- [ ] Passer d'une expression régulière à un automate : automates de Thomson.
- [ ] Passer d'un automate à une expression régulière : algorithme par élimination des états.
- [ ] Opérations sur les automates : automates produits, complétion d'un automate, complémentaire, déterminisation d'un automate non déterministe.
- [ ] Stabilités de la classe des langages réguliers.
- [ ] **Lemme de l'étoile**. Pour démontrer qu'un langage n'est pas régulier.

6.4 Grammaires non contextuelles

- [ ] **Grammaire non contextuelle** aussi appelée **Grammaire algébrique**. Symboles terminaux et non terminaux. Règles de production.
- [ ] Dérivation d'un mot. Dérivations immédiates. Dérivations gauches, droites.
- [ ] Langage engendré par une grammaire. Langages non contextuels. Les langages réguliers sont non contextuels.
- [ ] Arbre d'analyse, aussi appelé arbre de dérivation.
- [ ] Ambiguïté d'une grammaire.

6.5 Décidabilité et complexité

- [ ] Problèmes de décision.
- [ ] Classe **P** : problèmes de décision pouvant être **résolus** en temps polynomial par rapport à la taille de l'instance.
- [ ] Principe de réduction polynomiale $A \leq_P B$.
- [ ] Classe **NP** : problèmes de décision pouvant être **vérifiés** en temps polynomial par rapport à la taille de l'instance. Les instances positives possèdent des cerficats de taille polynomiale et il existe une machine vérifiant les certificats de temps d'exécution polynomial par rapport à la taille de l'instance.
- [ ] Problèmes **NP**-complets.
- [ ] **Théorème de Cook** : SAT est **NP**-complet. On prouve la NP-complétude d'autres problèmes par la méthode de réduction polynomiale.
- [ ] Problèmes indécidables. Problème de l'arrêt.
- [ ] Problèmes d'optimisation. Transformation d'un problème d'optimisation en un problème de décision à l'aide d'un seuil.