A savoir faire

Ce document détaille quelques points à savoir faire / expliquer / implémenter pour structurer vos révisions.

Algorithmes à savoir écrire en C ou en OCaml en moins de 5 minutes

1. Recherche du minimum d'un tableau.
2. Algorithme d'Euclide.
3. Évaluation de polynôme via la méthode de Horner.
4. Exponentiation rapide.
5. Décomposition en base b.
6. Suite récurrente linéaire avec autant de variables de sauvegarde que l'ordre de la relation récurrente (exemple : calcul du n-ème terme de la suite de Fibonacci en temps linéaire et espace constant).
7. Insertion d'un élément dans une liste triée.
8. Recherche par dichotomie dans un tableau trié.
9. Calculer le nombre de noeuds et la hauteur d'un arbre (pas forcément binaire).
10. Parcours en largeur / en profondeur préfixe, infixe, postfixe d'un arbre binaire.
11. Recherche et insertion dans un ABR.
12. Produit matriciel naïf.
13. Calculer le miroir d'une liste en temps linéaire en sa taille.

Algorithmes classiques à savoir implémenter en 10 minutes

1. Tri rapide (de préférence en place).
2. Tri fusion en OCaml.
3. Parcours en largeur d'un graphe en OCaml.
4. Parcours en profondeur d'un graphe en OCaml.
5. Toutes les opérations sur une file ou pile implémentée par liste chaînée en C.
6. Toutes les opérations sur une file implémentée par tableau circulaire en OCaml.
7. Toutes les opérations sur une structure union-find implémentée via une structure arborescente avec union par hauteur.
8. Suppression dans un ABR.
9. Extraction et ajout dans un tas min.

Algorithmes dont il faut connaître par coeur le principe et savoir les appliquer à un exemple

1. Algorithme de Floyd-Warshall.
2. Algorithme de Kruskal.
3. Algorithme de Kosaraju.
4. Algorithme de Dijkstra.
5. Algorithme A*.
6. Tri topologique.
7. Détection de cycles dans un graphe orienté.
8. Calcul de couplage maximum dans graphe biparti.
9. Tri par tas.
10. Algorithme de Quine.
11. Algorithme de Huffman.
12. Algorithme de Boyer Moore.
13. Algorithme de Rabin Karp.
14. Algorithme LZW.
15. Algorithme de Berry-Sethi.
16. Algorithme de Brzozowski-McCluskey.
17. Algorithme de Peterson.
18. Algorithme des k plus proches voisins.
19. Algorithme ID3.
20. Algorithme des k moyennes.
21. Classification hiérarchique ascendante.
22. Algorithme minmax.

Techniques algorithmiques

1. Savoir calculer la complexité d'un programme récursif (et en particulier DPR), c'est-à-dire trouver la relation de récurrence vérifiée par la complexité et la résoudre. En particulier, savoir résoudre immédiatement les cas typiques : $C(n) = \alpha C(n-1)$, $C(n) = C(n/2) + O(1)$, $C(n) = 2C(n/2) + O(n)$.
2. Retravailler quelques exemples de programmation dynamique (retrouver la relation de récurrence et l'ordre dans lequel faire les calculs). Par exemple : plus longue sous séquence commune, distance d'édition, sac à dos.
3. Savoir montrer qu'un algorithme est Las Vegas / Monte Carlo. Connaître l'exemple du tri rapide randomisé.
4. Savoir montrer qu'un algorithme est / n'est pas une α-approximation.

Arbres et graphes

1. Savoir montrer les encadrements classiques entre hauteur et taille d'un arbre binaire.
2. Savoir transformer un arbre d'arité quelconque en arbre binaire (transformation left-child right-sibling).
3. Connaître et savoir étudier la correction et la complexité de la recherche / insertion / suppression dans un arbre binaire de recherche. Savoir justifier l'intérêt de l'équilibrage.
4. Connaître la définition d'un arbre rouge noir et la preuve de la majoration de sa hauteur.
5. Connaître et savoir étudier la correction et la complexité de l'insertion, la suppression, la construction d'un tas implémenté par un tableau. Idem pour le tri par tas (en place).
6. Savoir comment utiliser un tas pour implémenter une file de priorité.
7. Bien connaître les différentes représentation des graphes et leurs intérêts.
8. Connaître le lien entre les puissances de la matrice d'adjacence et l'existence de chemins dans un graphe.
9. Connaître et savoir montrer les différentes caractérisations d'un arbre.

10. Connaître et savoir prouver et utiliser le lien entre nombre d'arêtes et somme des degrés.
11. Savoir prouver rigoureusement l'existence d'un arbre couvrant dans un graphe connexe.
12. Savoir prouver la validité du tri topologique.
13. Pour les algorithmes sur les graphes à connaître : voir dans les parties précédentes.

Logique

1. Savoir écrire un programme qui évalue une formule du calcul propositionnel étant donnée une valuation.
2. Bien connaître le vocabulaire et savoir l'utiliser avec précision : satisfiable, tautologie, antilogie, valuation, formules équivalentes, conséquence sémantique, séquent, prémisse, conclusion, règle d'inférence...
3. Savoir établir rapidement une table de vérité.
4. Connaître les équivalences classiques : lois de de Morgan, $A \Rightarrow B \equiv \neg A \vee B$...
5. Savoir mettre une formule sous forme normale disjonctive ou conjonctive.
6. Connaître les règles de la déduction naturelle et s'entraîner à les appliquer sur des exemples.

Langages formels

1. Connaître le vocabulaire générique sur les langages : mot, préfixe, suffixe, facteur, sous-mot, mot vide, langage, concaténation, étoile de Kleene...
2. Savoir définir ce qu'est un langage rationnel et faire la différence d'avec une expression rationnelle.
3. Savoir compléter ou émonder un automate.
4. Savoir construire l'automate pour l'union, le complémentaire, l'intersection, la concaténation.
5. Savoir éliminer les ε-transitions d'un automate.
6. Savoir déterminiser un automate via l'automate des parties.
7. Savoir appliquer la méthode d'élimination des états et l'algorithme de Berry-Sethi.
8. Connaître le théorème de Kleene et savoir comment le démontrer.
9. Savoir utiliser proprement le lemme de l'étoile pour montrer qu'un langage n'est pas rationnel.
10. Savoir montrer par double inclusion l'égalité entre L et $L(G)$, engendré par une grammaire algébrique G.
11. Savoir montrer qu'un langage rationnel est algébrique.
12. Savoir écrire un arbre syntaxique et utiliser cette notion pour montrer qu'une grammaire est ambiguë.

Calculabilité et complexité

1. Savoir montrer que le problème de l'arrêt est indécidable.
2. Savoir transformer un problème de décision en problème d'optimisation à seuil.
3. Savoir montrer qu'un problème est dans la classe NP.
4. Savoir montrer la NP-difficulté d'un problème par réduction depuis un problème NP-difficile.

IA et jeux

1. Savoir faire la distinction entre apprentissage supervisé et non supervisé.
2. Connaître la notion de matrice de confusion.
3. Savoir définir proprement les termes stratégie et stratégie gagnante.
4. Savoir calculer l'attracteur d'un joueur, en particulier avec l'algorithme minmax.
5. Connaître les variantes de l'algorithme minmax (avec élagage, à profondeur bornée).

Concurrence

1. Connaître le vocabulaire quant aux propriétés des programmes concurrents : famine, interblocage, exclusion mututelle.
2. Savoir établir une trace d'exécution et l'utiliser pour montrer une propriété.
3. Savoir utiliser un verrou ou un sémaphore pour garantir une propriété dans une situation de concurrence.
4. Connaître les idées de preuves de la correction de l'algorithme de Peterson et de Lamport, dans l'optique de pouvoir les réinvestir lorsqu'on demande d'établir une propriété dans une situation de concurrence.