Le QCM suivant est à choix et réponses multiples.

1 Structures et implémentations

1. Avec des représentations classiques, la suite de bits `10110010` peut représenter :
   (a) un entier naturel plus grand que 1000 ;
   (b) un entier strictement négatif ;
   (c) un rationnel ;
   (d) une lettre de l'alphabet courant.
2. Une pile est :
   (a) l'inverse d'une file ;
   (b) une structure FIFO ;
   (c) implémentable avec un tableau avec des opérations en temps constant ;
   (d) disponible nativement en C.
3. Une file peut être implémentée efficacement :
   (a) avec une pile ;
   (b) avec deux piles ;
   (c) avec un tableau ;
   (d) avec une file de priorité.
4. Qu'est-ce qu'une collision ?
   (a) l'accès simultané de deux fils d'exécution à la section critique ;
   (b) l'ajout d'un appel à la pile d'appels récursifs alors qu'elle est pleine ;
   (c) l'obtention de la même valeur de hachage lors du hachage de deux objets différents ;
   (d) la modification d'un objet qui en affecte un autre à cause d'une liaison de données.
5. Un tableau circulaire :
   (a) permet d'implémenter une pile ;
   (b) permet d'implémenter une file ;
   (c) permet d'implémenter une liste chaînée circulaire ;
   (d) a une taille fixe.
6. Une fonction de hachage peut être utilisée :
   (a) en sécurité informatique ;
   (b) pour la recherche de motif dans un texte ;
   (c) pour la mémoïsation ;
   (d) pour le découpage d'un texte lors d'une communication réseau.
7. Une table de hachage :
   (a) permet d'implémenter un dictionnaire ;
   (b) permet d'implémenter une file de priorité ;
   (c) nécessite de connaître à l'avance le nombre d'éléments maximal à insérer ;
   (d) est nécessairement mutable.
8. Un arbre binaire de recherche :
   (a) permet d'implémenter un dictionnaire ;
   (b) permet d'implémenter une file de priorité ;
   (c) nécessite de connaître à l'avance le nombre d'éléments maximal à insérer ;
   (d) est nécessairement mutable.
9. La structure de file de priorité peut être utilisée dans les algorithmes :
   (a) de Kruskal ;
   (b) de Kosaraju ;
   (c) de Dijkstra ;
   (d) de Berry-Sethi.
10. La structure Union-Find peut s'implémenter :
    (a) en temps constant pour l'opération unir ;
    (b) en temps constant pour l'opération trouver ;
    (c) en temps constant pour les opérations unir et trouver ;
    (d) en temps constant amorti pour les opérations unir et trouver.

2 Arbres

11. Dans un arbre binaire strict de taille n, à f feuilles et de hauteur h :
    (a) $f \leqslant n - f$;
    (b) $n < 2^{h+1} - 1$;
    (c) $h \leqslant n - f$;
    (d) $f = n - f + 1$.
12. Si on ignore les étiquettes, combien y a-t-il d'arbres binaires de hauteur 2 ?
    (a) 5 ;
    (b) 14 ;
    (c) 21 ;
    (d) 35.
13. Le parcours en largeur d'un arbre est :
    (a) linéaire en temps en la taille de l'arbre ;
    (b) constant en espace ;
    (c) linéaire en espace en la hauteur de l'arbre ;
    (d) l'inverse du parcours en profondeur.
14. Pour un arbre binaire de recherche de taille n et de hauteur h, on peut :
    (a) trouver un élément en $\mathcal{O}(h)$;
    (b) insérer un élément en $\mathcal{O}(\log n)$;
    (c) supprimer un élément en $\mathcal{O}(h)$;
    (d) vérifier que c'est un ABR en $\mathcal{O}(n)$.
15. Le nombre d'ABR de taille n dont les étiquettes sont les éléments de $[\![1, n]\!]$ est :
    (a) $\frac{1}{n+1}\binom{2n}{n}$;
    (b) $n!$;
    (c) 2^n ;
    (d) il n'y a pas de formule simple.

16. Un arbre rouge-noir :
    (a) est un tas binaire ;
    (b) permet d'insérer un élément en temps logarithmique en la taille ;
    (c) n'est jamais utilisé en pratique ;
    (d) est pénible à implémenter.
17. Un tas binaire de taille n et de hauteur h :
    (a) peut s'implémenter efficacement dans un tableau ;
    (b) peut s'implémenter efficacement avec une structure immuable d'arbres ;
    (c) permet la recherche d'un élement en temps $\mathcal{O}(\log n)$;
    (d) permet l'insertion d'un élément en temps $\mathcal{O}(h)$.

3 Graphes

18. Quelles sont les affirmations correctes ?
    (a) un graphe bipart admet un couplage parfait ;
    (b) un graphe qui admet un couplage parfait est biparti ;
    (c) un couplage maximal n'admet pas de chemin augmentant ;
    (d) un couplage n'ayant pas de chemin augmentant est maximal.
19. Soient C et C' deux couplages d'un graphe $G = (S, A)$. Alors :
    (a) $C \cup C'$ est un couplage de G ;
    (b) $C \cap C'$ eest un couplage de G ;
    (c) $C \Delta C'$ est un couplage de G ;
    (d) $C \setminus C'$ est un couplage de G.
20. Le nombre de couplages parfaits dans le graphe biparti complet $K_{n,n}$ est :
    (a) 2^n ;
    (b) $n!$;
    (c) $\binom{n}{2}$;
    (d) $\binom{n^2}{n}$.
21. Le nombre de graphes non orientés dont les sommets sont $\{1, 2, \ldots, n\}$ est :
    (a) 2^n ;
    (b) $n!$;
    (c) 2^{n^2} ;
    (d) $\binom{n}{2}$
22. Quelles sont les affirmations correctes ?
    (a) un graphe non orienté possède un arbre couvrant ;
    (b) un graphe pondéré connexe possède un unique arbre couvrant minimal ;
    (c) une arête de poids maximal n'appartient jamais à un arbre couvrant minimal ;
    (d) si tous les poids sont distincts, l'arête de poids minimal appartient à tous les arbres couvrants minimaux.
23. Un graphe non orienté à n sommets et p arêtes est un arbre si et seulement si par :
    (a) il est connexe et biparti ;
    (b) il est sans cycle et $n = p - 1$;
    (c) il est connexe et $p < n$;
    (d) il possède une racine et est binaire.
24. Un parcours en profondeur d'un graphe à n sommets et p arêtes est en complexité :
    (a) $\mathcal{O}(n^2)$ en temps et $\mathcal{O}(n)$ en espace pour une représentation par listes d'adjacence ;
    (b) $\mathcal{O}(n^2)$ en temps et $\mathcal{O}(n)$ en espace pour une représentation par matrice d'adjacence ;
    (c) $\mathcal{O}(n+p)$ en temps et $\mathcal{O}(p)$ en espace pour une représentation par listes d'adjacence ;
    (d) $\mathcal{O}(n+p)$ en temps et $\mathcal{O}(n)$ en espace pour une représentation par matrice d'adjacence.
25. L'algorithme A^* :
    (a) est correct avec une heuristique admissible ;
    (b) est correct avec une heuristique monotone ;
    (c) a la même complexité que l'algorithme de Dijkstra avec une heuristique admissible ;
    (d) a la même complexité que l'algorithme de Dijkstra avec une heuristique monotone.
26. Un ordre topologique d'un graphe $G = (S, A)$:
    (a) est unique s'il existe ;
    (b) existe pour tout graphe orienté ;
    (c) peut être calculé avec un parcours en largeur ;
    (d) peut être calculé en temps $\mathcal{O}(|S| + |A|)$.
27. L'algorithme de Kosaraju appliqué à un graphe $G = (S, A)$:
    (a) passe par le calcul d'un ordre topologique de G ;
    (b) permet de calculer les composantes connexes d'un graphe ;
    (c) permet de résoudre efficacement le problème 3SAT ;
    (d) nécessite de calculer le graphe transposé G^T.
28. Pour savoir si un graphe non orienté est une forêt, il suffit de connaître :
    (a) le nombre de sommets et d'arêtes ;
    (b) le nombre de sommets, le nombre d'arêtes et les degrés des sommets ;
    (c) le nombre de sommets, le nombre d'arêtes et le nombre de composantes connexes ;
    (d) aucune des réponses précédentes.
29. L'algorithme de Kruskal :
    (a) nécessite l'utilisation d'une file de priorité ;
    (b) construit un arbre couvrant petit à petit en conservant l'acyclicité ;
    (c) construit un arbre couvrant petit à petit en conservant la connexité ;
    (d) est optimal.

30. Pour calculer toutes les distances entre deux sommets dans un graphe, le plus efficace peut être d'utiliser l'algorithme de :
    (a) Floyd-Warshall ;
    (b) Dijkstra ;
    (c) parcours en profondeur ;
    (d) parcours en largeur.

4 Langages

31. Soient $u, v, w \in \Sigma^*$. Quelles affirmations sont vraies ?
    (a) Si u est préfixe de v, alors v est suffixe de u ;
    (b) si u est sous-mot de v, alors u est facteur de v ;
    (c) si u est suffixe de v, alors u est facteur de v ;
    (d) si u et v sont suffixes de w, alors u est sous-mot de v ou v est sous-mot de u.
32. Soient L, L' et L'' trois langages. Quelles affirmations sont vraies ?
    (a) Si L et L' sont fini, alors $|LL'| = |L| \times |L'|$;
    (b) L^* est infini ;
    (c) $L(L' \cap L'') = LL' \cap LL''$;
    (d) $L(L' \cup L'') = LL' \cup LL''$.
33. Soient $L \subseteq L'$ deux langages. Quelles affirmations sont vraies ?
    (a) Si L' est rationnel, alors L aussi ;
    (b) si L n'est pas rationnel, alors L' non plus ;
    (c) si L' est fini, alors L est rationnel ;
    (d)
34. Le langage $\{a^i b^j \mid i \equiv j \mod 2\}$ peut se décrire par l'expression régulière :
    (a) $(aa \mid bb)^*$;
    (b) $(aa \mid ab \mid ba \mid bb)^*$;
    (c) $(aa)^*(ab \mid \varepsilon)(bb)^*$;
    (d) $a^{2k+\delta}b^{2\ell+\delta}$, $\delta \in \{0, 1\}$.
35. L'interprétation d'une expression régulière e peut toujours être reconnu par un automate :
    (a) déterministe et complet ;
    (b) déterministe et émondé ;
    (c) déterministe de taille $\mathcal{O}(|e|)$;
    (d) non déterministe de taille $\mathcal{O}(|e|)$.
36. Parmi les conditions suivantes, lesquelles sont suffisantes pour que le langage L soit rationnel ?
    (a) L est dénombrable ;
    (b) L est le langage miroir d'un langage rationnel ;
    (c) L^* est rationnel ;
    (d) L est reconnaissable.
37. Parmi les langages suivants, lesquels sont rationnels ?
    (a) $\{a^n b^n \mid n \in \mathbb{N}\}$;
    (b) $\{a^{3i+k}b^{5j+k} \mid i, j, k \in \mathbb{N}\}$;
    (c) les noms de variables licites en OCaml ;
    (d) les mots bien parenthésés.
38. Tester si un mot u de taille n est reconnu par l'automate A à p états :
    (a) peut être en temps $\mathcal{O}(n)$ si l'automate est déterministe ;
    (b) peut être en temps $\mathcal{O}(n)$ si l'automate est non déterministe ;
    (c) peut être en temps $\mathcal{O}(n + p)$ si l'automate est non déterministe ;
    (d) peut nécessiter un temps exponentiel en p si l'automate n'est pas déterministe.
39. Si e est une expression régulière de taille p, on peut tester si un mot u de de taille n est dans $\mathcal{L}(e)$:
    (a) en temps $\mathcal{O}(n + p)$;
    (b) en temps $\mathcal{O}(n + p^2)$;
    (c) en temps $\mathcal{O}(n^2 + p^2)$;
    (d) peut nécessiter un temps exponentiel en n ou p.
40. Le lemme de pompage :
    (a) permet de montrer qu'un langage est rationnel ;
    (b) permet de montrer qu'un langage n'est pas rationnel ;
    (c) se prouve en utilisant des expressions régulières ;
    (d) donne une caractérisation pour **tous** les mots d'un langage rationnel.
41. Quels problèmes sur les langages rationnels et automates peuvent être résolus en temps polynomial ?
    (a) étant données deux expressions régulières, déterminer si elles ont le même langage interprété ;
    (b) étant donnée une expression régulière, déterminer si le langage interprété est vide ;
    (c) étant donné deux automates déterministes, déterminer s'ils reconnaissent le même langage ;
    (d) étant donné deux automates non déterministes, déterminer s'ils reconnaissent le même langage.
42. Les dérivations gauches d'une grammaire G :
    (a) sont en bijection avec les dérivations ;
    (b) sont en bijection avec les dérivations droites ;
    (c) sont en bijection avec les arbres de dérivation ;
    (d) sont unique pour chaque mot $u \in L(G)$.
43. Parmi les langages suivants, lesquels sont algébriques ?
    (a) $\{uu \mid u \in \{a, b\}^*\}$;
    (b) $\{uv \in \{a, b\}^* \mid |u| = |v| \text{ et } u \neq v\}$;
    (c) $\{a^p \mid p \in \mathbb{P}\}$;
    (d) $\{a^i b^j c^k \mid i + j = k\}$.

44. Quels problèmes sur les grammaires sont indécidables ?
    (a) déterminer si une grammaire est ambiguë ;
    (b) déterminer si le langage de deux grammaires sont égaux ;
    (c) déterminer si le langage d'une grammaire est vide ;
    (d) déterminer si une grammaire est en forme normale de Chomsky.
45. L'analyse syntaxique descendante d'un mot u pour une grammaire G :
    (a) permet de déterminer si $u \in L(G)$;
    (b) ne fonctionne que si G n'est pas récursive ;
    (c) procède par retour sur trace ;
    (d) construit un arbre de dérivation à partir des feuilles.
46. Si une dérivation $\alpha \Rightarrow^* \beta$ est immédiate, alors :
    (a) α ne contient qu'une seule variable ;
    (b) β contient une variable de moins que α ;
    (c) la variable la plus à gauche de α a été dérivée ;
    (d) α peut être égal à β.

5 Calculabilité

47. Pour montrer l'indécidabilité du problème de l'arrêt, on suppose qu'il existe une fonction `arrêt` qui le résout, et :
    (a) on montre que `arrêt` appelé sur son propre code source ne termine pas ;
    (b) on dit que `arrêt <f, x>` pour `f` une fonction qui ne termine pas revient à simuler `f x`, donc ne termine pas ;
    (c) on construit une fonction `paradoxe <f>` qui contient un appel à `arrêt <paradoxe, paradoxe>` ;
    (d) on construit une fonction `paradoxe <f>` qui contient un appel à `arrêt <f, f>`.
48. Quelles propositions sont vraies ?
    (a) il existe $A \leqslant_m B \leqslant_m C$ tels que A et C sont indécidables et B est décidable ;
    (b) il existe $A \subseteq B \subseteq C$ tels que A et C sont indécidables et B est décidable ;
    (c) si $A \leqslant_m B$ et A est semi-décidable, alors B est semi-décidable ;
    (d) si $A \leqslant_m B \leqslant_m C$ et C est décidable, alors A est décidable.
49. Quels problèmes sont indécidables ?
    (a) déterminer si l'appel à une fonction sans argument termine ;
    (b) déterminer si l'appel à une fonction termine pour tout argument ;
    (c) déterminer s'il existe un argument pour lequel l'appel à une fonction termine ;
    (d) déterminer s'il existe un argument pour lequel l'appel à une fonction ne termine pas.
50. Pour tester si un entier n est premier, on teste s'il est divisible par chaque entier compris entre 2 et $\lceil \sqrt{n} \rceil$. Cet algorithme est de complexité :
    (a) sous-linéaire ;
    (b) linéaire ;
    (c) quadratique ;
    (d) exponentielle.
51. La complexité d'un problème de décision :
    (a) correspond à la complexité meilleur cas du meilleur algorithme qui le résout ;
    (b) correspond à la complexité meilleur cas du pire algorithme qui le résout ;
    (c) correspond à la complexité pire cas du meilleur algorithme qui le résout ;
    (d) dépend de la manière dont sont représentées les entrées.
52. Quels problèmes sont NP-difficiles (en supposant $\mathsf{P} \neq \mathsf{NP}$) ?
    (a) `3SAT` ;
    (b) le problème de l'arrêt ;
    (c) `2COLORATION` ;
    (d) l'existence d'un chemin eulérien.
53. Le problème du sac à dos :
    (a) peut être résolu en temps polynomial en la taille de l'entrée ;
    (b) peut être résolu en temps polynomial en le nombre d'objets et le poids maximal ;
    (c) possède une $\frac{1}{2}$-approximation polynomiale ;
    (d) possède une α-approximation polynomiale avec α arbitrairement proche de 1.
54. Soient $A \leqslant_{\mathsf{m}}^{\mathsf{p}} B$ deux problèmes, φ la fonction de réduction et x une instance de A. Alors :
    (a) $\varphi(x)$ est une instance ;
    (b) $\varphi(x)$ est un certificat ;
    (c) $\varphi(x)$ se calcule en temps polynomial en $|x|$;
    (d) si $\varphi(x) \in B$, alors $x \in A$;
55. Soit A un problème d'optimisation et B le problème de décision au seuil associé. Alors :
    (a) si A peut être résolu en temps polynomial, B aussi ;
    (b) si B peut être résolution en temps polynomial, A aussi ;
    (c) si $B \in \mathsf{P}$, alors A possède une approximation en temps polynomial ;
    (d) si $B \in \mathsf{NP}$, alors $A \in \mathsf{NP}$.

6 Algorithmique

56. Un invariant de boucle :
    (a) aide à prouver la terminaison d'un algorithme ;
    (b) est vérifié en sortie de boucle ;
    (c) est vérifié à la fin d'un passage dans la boucle s'il était vérifié au début du passage dans la boucle ;
    (d) correspond à une formulation de la correction de l'algorithme sur des sous-problèmes.
57. Un algorithme probabiliste dont la réponse est toujours exacte, mais dont le temps de calcul est aléatoire est un algorithme de :
    (a) Atlantic City ;
    (b) Monte Carlo ;
    (c) Monaco ;
    (d) Las Vegas.
58. Si un problème de décision A peut être résolu par un algorithme dont le temps de calcul est d'espérance polynomiale, alors :
    (a) $A \in \mathsf{P}$;
    (b) A peut être résolu en temps toujours polynomial par un algorithme sans faux positif et une probabilité de faux négatif $< \frac{1}{3}$;
    (c) A peut être résolu en temps toujours polynomial par un algorithme sans faux positif et une probabilité de faux négatif arbitrairement petite ;
    (d) si l'algorithme répond qu'une instance est positive, elle l'est nécessairement.
59. L'algorithme de tri rapide randomisé sur un tableau de taille n :
    (a) a une complexité pire cas $\Theta(n \log n)$;
    (b) a une complexité moyenne $\Theta(n \log n)$;
    (c) est un algorithme de Monte Carlo ;
    (d) est plus efficace que le tri rapide déterministe.
60. Quelles techniques algorithmiques découpe un problème en sous-problèmes et résout les sous-problèmes indépendamment pour reconstruire une solution ?
    (a) programmation gloutonne ;
    (b) programmation dynamique ;
    (c) diviser pour régner ;
    (d) séparation et évaluation.
61. Un algorithme de type séparation et évaluation (*branch and bound*) :
    (a) est utile pour résoudre le problème du chemin hamiltonien ;
    (b) est utile pour résoudre le problème du voyageur de commerce ;
    (c) fournit une approximation de la solution ;
    (d) est une variante d'un algorithme de retour sur trace.
62. Un algorithme glouton :
    (a) n'est utile que s'il fournit une solution exacte ;
    (b) est polynomial ;
    (c) peut revenir sur une décision prise à une étape précédente ;
    (d) peut permettre de trouver facilement un couplage maximal pour l'inclusion.
63. Parmi les algorithmes suivants, lesquels peuvent être considérés comme gloutons ?
    (a) Kruskal ;
    (b) Kosaraju ;
    (c) Huffman ;
    (d) Quine.
64. On suppose que pour $n \leqslant 2$, $C(n) = \Theta(1)$. Pour quelles formules de récurrence peut-on conclure que $C(n) = \Theta(\lambda^n)$, avec un certain $\lambda > 1$?
    (a) $C(n) = 3C(n-1)$;
    (b) $C(n) = 2C(\sqrt{n})$;
    (c) $C(n) = \sum_{i=0}^{n-1} C(i)$;
    (d) $C(n) = 3C(n/2) + \Theta(n^3)$.
65. Quels algorithmes permettent de faire de la compression de données ?
    (a) Huffman ;
    (b) Rabin-Karp ;
    (c) Lempel-Ziv-Welch ;
    (d) Boyer-Moore.
66. Pour rechercher les occurrences d'une chaîne u de taille p dans un texte t de taille n :
    (a) l'algorithme naïf est efficace en pratique ;
    (b) on ne peut pas faire mieux que $\Theta(n \times p)$ dans le pire cas ;
    (c) on peut le faire en temps $\mathcal{O}(n/p)$ dans le meilleur cas ;
    (d) avec un précalcul sur u, on peut le faire en temps $\mathcal{O}(n)$ dans tous les cas.
67. L'algorithme LZW :
    (a) tire parti du fait que certaines lettres ont tendance à se suivre ;
    (b) tire parti du fait que certaines lettres sont plus fréquentes que d'autres ;
    (c) utilise une fonction de hachage ;
    (d) nécessite de parcourir tout le texte plus qu'une fois.
68. L'algorithme de Huffman appliqué à un texte de taille n :
    (a) correspond en fait à plusieurs algorithmes distincts ;
    (b) nécessite un espace mémoire logarithmique en n ;
    (c) s'exécute en temps linéaire en n ;
    (d) est optimal.

7 Jeux et apprentissage

69. Un arbre de décision :
    (a) a une hauteur majorée par la dimension des données ;
    (b) peut être construit par un algorithme d'apprentissage supervisé ;
    (c) classe correctement les données d'apprentissage ;
    (d) améliore la recherche des plus proches voisins.
70. Le clustering hiérarchique ascendant :
    (a) permet de choisir a posteriori le nombre de clusters ;
    (b) a un résultat qui ne dépend pas de la distance choisie ;
    (c) a une complexité qui ne dépend pas de la distance choisie ;
    (d) commence par un cluster contenant toutes les données et le découpe au fur et à mesure.
71. L'algorithme des k-moyennes :
    (a) ne converge pas nécessairement ;
    (b) converge vers un optimum global s'il converge ;
    (c) est un algorithme d'apprentissage supervisé ;
    (d) ne fonctionne pas avec des clusters non convexes.
72. L'algorithme des k-plus proches voisins :
    (a) est plus précis si on augmente k ;
    (b) est plus précis si on augmente la dimension des données d'apprentissage ;
    (c) est plus précis si on augmente le nombre de données d'apprentissage ;
    (d) permet de faire de la régression linéaire.
73. Un arbre k-dimensionnel pour n données d'apprentissage :
    (a) est un arbre de décision ;
    (b) peut être construit en temps $\mathcal{O}(n \log n)$;
    (c) permet la recherche du plus proche voisin en temps $\mathcal{O}(\log n)$;
    (d) n'est efficace qu'en petite dimension.
74. Une stratégie gagnante pour le joueur 1 :
    (a) est une série de coups menant à la victoire ;
    (b) dépend de la stratégie du joueur 2 ;
    (c) n'a pas de mémoire des coups précédents ;
    (d) permet de garantir d'obtenir la victoire du joueur 1.
75. Dans un jeu biparti, les positions gagnantes pour le joueur 1 :
    (a) peuvent être calculées en temps linéaire en la taille du jeu ;
    (b) dépendent des coups choisis par le joueur 2 ;
    (c) sont des sommets où c'est au joueur 1 de jouer ;
    (d) mènent nécessairement à une partie gagnée par le joueur 1.
76. L'algorithme Min-Max peut :
    (a) permettre le calcul de l'attracteur d'un joueur ;
    (b) être de complexité temporelle exponentielle en la profondeur maximale ;
    (c) être de complexité temporelle exponentielle en le nombre maximal de coups possibles depuis une position ;
    (d) être de complexité spatiale linéaire en la profondeur maximale.
77. L'algorithme Alpha-Beta :
    (a) garantit de trouver un coup gagnant ;
    (b) garantit de trouver le même résultat que l'algorithme Min-Max ;
    (c) nécessite de connaître à l'avance les valeurs de l'heuristique pour toutes les positions à profondeur maximale ;
    (d) permet de gagner en complexité spatiale.

8 Logique

78. Le problème SAT peut être résolu en temps polynomial pour une formule :
    (a) en 2-FNC ;
    (b) en 3-FNC ;
    (c) en FND ;
    (d) ne contenant pas de symbole $\neg$.
79. Quel symbole mathématique peut remplacer $\bigstar$ dans « $(A \to B) \bigstar (\neg A \vee B)$ » pour qu'elle représente une affirmation correcte ?
    (a) $=$
    (b) $\equiv$
    (c) $\leftrightarrow$
    (d) $\vdash$
80. Pour la sémantique booléenne usuelle :
    (a) si $\Gamma \vdash_m A$, alors $\Gamma \vDash A$;
    (b) si $\Gamma \vDash A$, alors $\Gamma \vdash_m A$;
    (c) si $\Gamma \vdash_c A$, alors $\Gamma \vDash A$;
    (d) si $\Gamma \vDash A$, alors $\Gamma \vdash_c A$.
81. En logique minimale, si $\Gamma \vdash \neg A \to \neg B$ est prouvable, quels séquents sont prouvables ?
    (a) $\Gamma, \neg A \vdash \neg B$;
    (b) $\Gamma \vdash B \to A$;
    (c) $\Gamma \vdash \neg\neg B \to \neg\neg A$;
    (d) $\Gamma \vdash \neg\neg A \vee \neg B$.
82. Quelles formules sont des théorèmes en logique classique ?
    (a) $((A \to B) \to C) \leftrightarrow (A \to (B \to C))$;
    (b) $(A \vee B) \wedge (\neg A \vee \neg B)$;
    (c) $((A \to B) \to A) \to A$;
    (d) $(A \vee B) \wedge (C \vee D) \leftrightarrow (A \wedge C) \vee (B \wedge D)$.

83. Quels séquents sont prouvables en logique du premier ordre ?
    (a) $A \vdash \forall x\, A$;
    (b) $A \vdash \exists x\, A$;
    (c) $\exists x \forall y\, x \to y \vdash \forall y \exists x\, x \to y$;
    (d) $\neg(\exists x\, A \wedge B) \vdash \forall x(\neg A \vee \neg B)$.
84. Pour effectuer la substitution $A[x := t]$, on peut être amené à remplacer toutes les occurrences de x :
    (a) libres dans A ;
    (b) liées dans A ;
    (c) libres dans t ;
    (d) liées dans t.
85. Si A est une formule close et B est une formule non close :
    (a) A n'a pas de variable liée ;
    (b) A n'a pas de variable libre ;
    (c) $A \vdash B$ peut être prouvable ;
    (d) $B \vdash A$ peut être prouvable.

9 Programmation et système

86. Sur un ordinateur moderne, dans quels cas peut-on exécuter en quelques secondes un algorithme faisant $f(n)$ opérations élémentaires ?
    (a) $f(n) = 2^n$ et $n = 20$;
    (b) $f(n) = n!$ et $n = 20$;
    (c) $f(n) = n^3$ et $n = 10^4$;
    (d) $f(n) = n \log_2 n$ et $n = 10^7$.
87. Quelle commande permet d'afficher le contenu d'un dossier ?
    (a) `cd`
    (b) `ls`
    (c) `top`
    (d) `sudo rm -rf /`
88. L'utilisation de plusieurs fils d'exécution :
    (a) entraîne un non-déterminisme de l'exécution ;
    (b) n'est utile que si on dispose de plusieurs processeurs ;
    (c) rend nécessaire l'utilisation de mutex ;
    (d) accélère les calculs par rapport à un programme séquentiel.
89. Deux fils d'exécution exécutent en parallèle le programme `for (int i=0; i<10; i++) k++;` où `k` est une variable commune initialisée à zéro. Après l'exécution, la valeur de `k` peut être :
    (a) au minimum 1 ;
    (b) au minimum 2 ;
    (c) au maximum 20 ;
    (d) au maximum 65.
90. Un mutex :
    (a) signifie *mutable execution* ;
    (b) peut être verrouillé par deux fils simultanément ;
    (c) fonctionne toujours en FIFO ;
    (d) garantit l'absence d'interblocage.
91. L'algorithme de Peterson implémente un mutex qui vérifie :
    (a) l'exclusion mutuelle ;
    (b) l'absence de famine ;
    (c) l'alternance des fils d'exécution ;
    (d) une bijection entre les verrouillages de chacun des fils.
92. L'algorithme de la boulangerie de Lamport :
    (a) utilise de l'attente active ;
    (b) demande de pouvoir calculer le maximum d'un tableau de manière atomique ;
    (c) nécessite de connaître à l'avance le nombre maximal de fils d'exécution ;
    (d) garantit l'absence de famine.
93. Un sémaphore :
    (a) bloque un fil sans décrémenter si le compteur doit passer en négatif ;
    (b) libère le fil en attente depuis le plus longtemps lorsqu'il est incrémenté ;
    (c) libère un fil uniquement quand le compteur passe de négatif à positif ;
    (d) permet de connaître le nombre de fils en attente.
94. En compilant en C, l'option `-fsanitize=address` :
    (a) permet de détecter des fuites mémoire ;
    (b) permet de détecter un dépassement d'entiers ;
    (c) détecte les erreurs à la compilation ;
    (d) détecte les erreurs à l'exécution.
95. En C, c'est une erreur d'appeler `free` :
    (a) deux fois sur le même pointeur ;
    (b) sur un pointeur `NULL` ;
    (c) sur `&t[1]`, si `t` est un pointeur renvoyé par `malloc` ;
    (d) sur un pointeur d'un objet sur la pile.
96. En C, il peut être pertinent, dans une fonction, de renvoyer :
    (a) un objet défini par `struct s = {.x = 1, .y = 2};`
    (b) un tableau défini par `int t[3] = {1, 2, 3};`
    (c) un tableau défini par `int* t = malloc(3 * sizeof(*t));`
    (d) l'objet `&n`, où `n` est défini par `int n = 1;`
97. En OCaml, la fonction `List.copy` :
    (a) s'exécute en temps constant ;
    (b) s'exécute en temps linéaire en la taille de la liste ;
    (c) peut créer de la liaison de données ;
    (d) n'existe pas.

98. En OCaml, lorsqu'on fait `let lst = l1 @ l2` :
    (a) cela prend un temps proportionnel à $|\texttt{l1}|$;
    (b) cela prend un temps proportionnel à $|\texttt{l2}|$;
    (c) cela prend un temps proportionnel à $|\texttt{l1}|+|\texttt{l2}|$;
    (d) cela peut créer de la liaison de données entre `lst` et `l1`.

99. On considère le code suivant en OCaml :

```
type arbre = V | N of arbre * arbre

let rec cree n = function
    | 0 -> V
    | n ->
        let a = cree (n - 1) in
        N (a, a)

let rec taille = function
    | V -> 0
    | N (g, d) -> 1 + taille g + taille d
```

    Quelles affirmations sont correctes ?
    (a) l'appel `cree n` s'exécute en temps $\mathcal{O}(n)$;
    (b) l'appel `taille (cree n)` s'exécute en temps $\mathcal{O}(n)$;
    (c) l'objet renvoyé par `cree n` est mathématiquement un arbre ;
    (d) l'objet renvoyé par `cree n` est stocké en mémoire comme un arbre.

100. Dans une table `villes` qui contient, pour chaque ville de France, son `id`, son `nom` et son nombre d'`habitants`, déterminer la ville ayant le nombre maximal d'habitants peut se faire par la commande :

(a)
```
SELECT nom, MAX(habitants) FROM villes
```

(b)
```
SELECT nom FROM villes
ORDER BY habitants DESC LIMIT 1
```

(c)
```
SELECT nom FROM villes
WHERE habitants = MAX(habitants)
```

(d)
```
SELECT nom FROM villes
WHERE habitants =
  (SELECT MAX(habitants) FROM villes)
```