Les réponses correctes sont exactement celles en gras. Des commentaires sont en italique.

Structures de données

1. La suite de bits 11101100 peut a priori représenter ...
   (a) Un entier naturel plus grand que 1000. *Au maximum sur 8 bits on représente les entiers de l'intervalle $[\![0, 2^8-1]\!] = [\![0, 255]\!]$. À moins d'introduire une façon très atypique de représenter les entiers...*
   (b) ►Un entier strictement négatif. *Représente $-2^7 + 2^6 + 2^5 + 2^3 + 2^2 = -20$ en complément à 2.*
   (c) ►Un rationnel. *Cette suite peut représenter un flottant, même si on ne peut pas savoir lequel exactement car on ne sait pas combien de bits sont utilisés pour l'exposant et combien pour la mantisse.*
   (d) ►Un caractère. *Oui, mais attention on est au delà de 127 donc ce n'est plus un caractère ASCII.*
2. Un pile est ...
   (a) ►Une structure FILO (First In, Last Out). *On peut aussi dire LIFO (Last In, First Out).*
   (b) Une structure FIFO (First In, First Out). *C'est une file.*
   (c) ►Une structure disponible nativement en OCaml. *On peut utiliser une liste pour implémenter une pile.*
   (d) ►Utile pour implémenter un parcours en profondeur. *C'est une alternative au DFS par fonction récursive : dans un while, on dépile, on traite le sommet, et on empile les voisins non visités.*
   (e) Utile pour implémenter un parcours en largeur.
3. Une file peut ...
   (a) ►Être implémentée par deux piles avec des opérations élémentaires en temps constant (en complexité amortie). *Le sommet de la pile 1 est la tête de file et celui de la pile 2 est sa queue. La création créé deux piles vides en temps constant, l'ajout est en temps constant, le retrait demande parfois de transvaser pile 2 dans pile 1.*
   (b) Être implémentée par une seule pile. *On a besoin de savoir où est la tête et où est la queue.*
   (c) Utile pour implémenter un parcours en profondeur.
   (d) ►Utile pour implémenter un parcours en largeur.
4. Une fonction de hachage peut être utilisée ...
   (a) ►En sécurité informatique.
   (b) ►Pour la reconnaissance de motif dans un texte. *Dans l'algorithme de Rabin-Karp.*
   (c) ►Pour la mémoïsation. *Pour implémenter un dictionnaire via table de hachage.*
   (d) Pour accélérer l'algorithme des k plus proches voisins. *Non, mais en revanche la recherche des k plus proches voisins d'un points peut être accélérée par l'utilisation d'arbres d-dimensionnels.*
5. Quelles structures sont persistantes ? *On rappelle que persistant/fonctionnel veut dire non-modifiable, contrairement à mutable/impératif*
   (a) ►Liste OCaml.
   (b) Tableau.
   (c) ►Arbre binaire OCaml.
   (d) Tas binaire.*L'implémentation classique utilise un tableau*
   (e) Chaîne de caractères en C.*Tableau de char*
   (f) ►Chaîne de caractères en OCaml.*Autre implémentation qu'en C*
6. Si on considère un arbre binaire strict et non vide à n noeuds internes, f feuilles et de hauteur h, quelles relations sont vérifiées parmi les suivantes ?
   (a) $f \leq n$. *Considérer un arbre à un noeud : $f = 1$ et $n = 0$.*
   (b) $f + n < 2^{h+1} - 1$. *Est faux sur une racine : $f + n = 1 + 0 = 1 = 2^{0+1} - 1$. L'inégalité large est vraie.*
   (c) ►$n \geq h$.
   (d) ►$f = n + 1$. *Se démontre par récurrence sur la taille par exemple.*
   (e) ►$f + n + h \neq 0$. *Puisque l'arbre est non vide.*
7. En ignorant la valeur des étiquettes des noeuds, combien y a-t-il d'arbres binaires de hauteur exactement 2 ?
   (a) 3.
   (b) 14.
   (c) ►21. *Il y a un arbre de hauteur -1, un de hauteur 0 et trois de hauteur 1. Dans un arbre de hauteur 2, l'un des fils doit être de hauteur 1 et l'autre est quelconque parmi les 5 possibles. L'ensemble G des arbres binaires de hauteur 2 dont le fils gauche est de hauteur 1 contient donc 15 éléments (donc plus que 14) et si on fixe le fils droit comme étant de hauteur 2, on a au maximum 15 arbres supplémentaires, et en fait moins, car certains sont déjà présents dans G. En tous cas, on en a moins que $15 + 15 = 30$. C'est donc la bonne réponse par élimination et on peut le vérifier en dessinant tous les arbres.*
   (d) 35.
8. Le parcours en largeur d'un arbre est ...
   (a) ►Linéaire en le nombre de noeuds.
   (b) ►Linéaire en le nombre d'arêtes. *Dans un arbre, $a = n - 1$ donc une quantité est linéaire en le nom-*

bre n de noeuds si et seulement si elles est linéaire en le nombre a d'arêtes.

(c) Linéaire en la hauteur. *Pas dans un arbre complet par exemple.*

(d) Jamais utilisé en pratique.

(e) L'ordre inverse du parcours en profondeur.

9. On note $r(a)$ la racine d'un arbre a. Soit a un arbre binaire non vide et de sous-arbres g et d. On suppose que les éléments de a sont tous différents. Quelle(s) proposition(s) suivante(s) sont équivalentes à « a est un arbre binaire de recherche (ABR) » ? *Il faut que tous les éléments de g soient inférieurs à $r(a)$, tous les éléments de d supérieurs à $r(a)$, g ABR et d ABR. On peut trouver des contre-exemples pour les autres propositions.*

(a) $r(g) < r(a)$, $r(d) < r(a)$, et g, d sont des ABR.

(b) $r(g) < r(a) < r(d)$, et g, d sont des ABR.

(c) Tous les éléments de g sont inférieurs à $r(a)$ et tous les éléments de d sont supérieurs à $r(a)$.

(d) ►Le parcours infixe de a est croissant.

(e) Le parcours préfixe de a est croissant.

10. Dans un arbre binaire de recherche de taille n et de hauteur h, on peut ...

(a) ►Trouver un élément en $O(n)$. *Puisque $h \leq n$.*

(b) ►Trouver un élément en $O(h)$.

(c) Trouver un élément en $O(\log n)$. *La recherche dans un ABR est linéaire en sa hauteur. Si l'arbre n'est pas équilibré, cette hauteur peut être de l'ordre de n donc cette opération n'est pas en $O(\log n)$.*

(d) Insérer un élément en $O(\log n)$. *Idem (C) car l'insertion est aussi en $O(h)$.*

(e) Supprimer un élément en $O(\log n)$. *Idem (C) car la suppression est aussi en $O(h)$.*

(f) ►Vérifier qu'il est bien ABR en $O(n)$. *Il suffit d'en faire le parcours infixe et de vérifier qu'il est trié. Attention, si on regarde naïvement si la racine est plus grande que le max à gauche, plus petite que le min à droite et que les deux fils sont ABR, l'analyse de complexité est plus délicate.*

11. Les arbres rouge-noir ... *Rappel : un arbre rouge-noir est un ABR strict dont la racine est noire (les feuilles vide aussi), les fils d'un noeud rouge sont noirs et tel que tous les chemins de la racine à une feuilles ont autant de noeuds noirs. L'insertion et la suppression utilisent des rotations.*

(a) ►Sont des ABR.

(b) ►Peuvent être utilisés pour implémenter un dictionnaire. *C'est l'un des intérêts d'un ABR.*

(c) N'assurent aucune garantie de complexité. *Au contraire puisqu'un arbre rouge-noir est équilibré.*

(d) N'ont pas d'utilité pratique.

(e) ►Nécessitent un bit d'information supplémentaire par noeud par rapport à un arbre binaire. *À savoir la couleur du noeud.*

(f) Ne peuvent pas stocker de valeurs.

12. La structure de tas peut être utilisée dans les algorithmes suivants ...

(a) ►Algorithme de Kruskal. *Un peu abusif : si on utilise un tri par tas pour trier les arêtes.*

(b) Algorithme de parcours en profondeur.

(c) Algorithme de Kosaraju.

(d) ►Algorithme du tri par tas. *Évidemment.*

(e) ►Algorithme de Dijkstra. *Via la file de priorité qui gère quel sommet étudier.*

13. Il existe une implémentation de la structure unir et trouver ayant une complexité ...

(a) ►Constante pour l'opération unir. *On représente les classes par les composantes connexes d'un graphe représenté par matrice d'adjacence. Unir revient à ajouter une arête ce qui se fait en temps constant. Trouver consiste (par exemple) à parcourir la composante pour en trouver le plus petit élément.*

(b) ►Constante pour l'opération trouver. *Via un tableau stockant le numéro de partition de i en case i.*

(c) Constante pour l'opération trouver et l'opération unir.

(d) Constante en complexité amortie pour l'opération trouver et l'opération unir. *Même avec union par rang et compression de chemin, la complexité amortie n'est pas constante mais en $O(\alpha(n))$. On peut juste dire "en pratique on peut la considérer constante".*

(e) ►En $O(\log n)$ dans le pire cas pour les opérations trouver et unir où n est le nombre d'éléments manipulés. *Via une forêt avec union par hauteur.*

14. Un graphe est régulier si tous ses sommets ont le même degré. Existe-t-il un graphe régulier ayant ...

(a) ►9 sommets. *Avec un graphe à 9 sommets sans arêtes par exemple.*

(b) ►Degré 5 avec 3 composantes connexes. *Juxtaposer 3 fois le graphe complet à 6 sommets.*

(c) Degré 5 avec 9 sommets. *Impliquerait que $2|A| = \sum \deg(s) = 9 \times 5 = 45$ donc que 45 est pair.*

(d) Degré 6 avec 6 sommets. *Chaque sommet ne peut être relié qu'à 5 sommets dans un graphe à 6 sommets.*

(e) ►Un degré p et un nombre q de sommets avec des p et q premiers. *Le graphe complet à 3 sommets convient ($p = 2$). On ne demande pas ici si on a un graphe régulier pour tous p et q premiers.*

15. L'ensemble des graphes orientés à n sommets ... *Il y a $n(n-1)$ arcs possibles dans un graphe orienté à n sommets puisqu'un arc est une paire de sommets distincts. Chaque partie de cet ensemble à $n(n-1)$ éléments donne naissance à exactement un graphe orienté : il y en a donc $2^{n(n-1)}$.*

    (a) Est en bijection avec l'ensemble des graphes non orientés à n sommets. *Dans un graphe non orienté il y a $k = n(n-1)/2$ arêtes possibles donc 2^k graphes possibles.*

    (b) Est en bijection avec l'ensemble des graphes non orientés à $\lceil n/2 \rceil$ sommets. *Avec les raisonnements précédents, il y a $2^{\lceil n/2 \rceil (\lceil n/2 \rceil - 1)/2}$ tels graphes.*

    (c) Est en bijection avec le groupe symétrique S_n. *Cet ensemble est de cardinal $n!$.*

    (d) ►Est en bijection avec l'ensemble des mots de $\{a,b\}^{n^2-n}$. *Correct puisque pour construire un mot sur $\{a,b\}$ de longueur $n^2 - n$, on a deux possibilité pour chacune des $n^2 - n$ positions.*

    (e) Est en bijection avec l'ensemble des langages qui contiennent n mots sur $\{a,b\}$. *Cet ensemble n'est pas fini puisque les langages $L_i = \{a^i, a^{i+1}, ..., a^{i+n-1}\}$ en sont et sont tous différents.*

16. Un graphe non orienté à m arêtes et n sommets est un arbre si et seulement si ...

    (a) Il est acyclique. *Contre-exemple : un graphe acyclique non connexe.*

    (b) ►Il est acyclique avec $m = n - 1$.

    (c) Il possède une racine.

    (d) ►Toute paire de sommets est reliée par un unique chemin.

    (e) ►Il est connexe avec $m < n$. *Un arbre vérifie cette propriété. Réciproquement, tout graphe connexe vérifie $m \geq n - 1$. Comme $m < n$, on a donc un graphe connexe avec $m = n - 1$ ce qui caractérise bien un arbre.*

17. La complexité d'un parcours en profondeur du graphe $G = (S, A)$ est ...

    (a) En $O(|S|^2)$ quelle que soit la représentation du graphe. *Sauf peut-être si on s'autorise un précalcul en $O(|S|^2)$ pour transformer la représentation choisie en une autre plus adéquate.*

    (b) En $O(|A|)$ si G représenté par listes d'adjacence. *$O(|S| + |A|)$: il faut initialiser le tableau des vus.*

    (c) ►En $O(|S|^2)$ si G est représenté par matrice d'adjacence.

    (d) En $O(|S|+|A|)$ si G est représenté par matrice d'adjacence. *Car le calcul des voisins d'un sommet s demande de l'ordre de $|S|$ opérations à la place de $\deg s$ avec listes d'adjacence.*

    (e) En $O(|S| + |A|)$ si G est représenté par liste d'arêtes. *Techniquement oui car on peut convertir une représentation par liste d'arêtes en une représentation par listes d'adjacence en temps $O(|S| + |A|)$.*

Algorithmique

18. Un invariant de boucle ...

    (a) ►Aide à prouver la correction d'un algorithme.

    (b) Est vérifié si et seulement si la propriété reste vraie après une itération si on la suppose vraie avant. *Il faut aussi que la propriété soit vraie avant la toute première itération de la boucle (initialisation).*

    (c) ►Est vérifié en sortie de boucle. *On entend ici "invariant" au sens de "propriété dont on a montré qu'elle est bien un invariant de boucle" et pas "propriété dont il faut encore montrer qu'elle est invariante.*

    (d) Correspond à une formulation de la correction de l'algorithme sur des sous-problèmes.

    (e) Aucune des propositions ci-dessus.

19. Un algorithme qui termine en temps probabiliste avec une réponse exacte est un algorithme de ...

    (a) Atlantic City.
    (b) Atlanta.
    (c) ►Las Vegas.
    (d) Macao.
    (e) Monte Carlo.

20. Un algorithme dont le temps de calcul est garanti mais dont le résultat peut être inexact avec une certaine probabilité est un algorithme de ...

    (a) Atlantic City.
    (b) Atlanta.
    (c) Las Vegas.
    (d) Macao.
    (e) ►Monte Carlo.

21. Si pour un problème $\mathcal{P}$ on dispose d'un algorithme $\mathcal{A}$ qui s'exécute en temps polynomial, sans faux négatif et avec probabilité de faux positif inférieure à $1/3$ ...

    (a) $\mathcal{P} \in P$. *Aucune raison.*

    (b) ►On peut résoudre $\mathcal{P}$ sans faux négatif et avec une probabilité de faux positif arbitrairement petite. *En amplifiant l'algorithme.*

    (c) ►Si $\mathcal{A}$ affirme qu'une instance de $\mathcal{P}$ est négative, c'est bien le cas. *Par définition de faux négatif.*

(d) Il existe un algorithme qui résout $\mathcal{P}$ en temps polynomial en moyenne. *Ce serait vrai si en plus de $\mathcal{P}$ on disposait d'un algorithme polynomial permettant de vérifier si une sortie de $\mathcal{P}$ est bien une solution de $\mathcal{A}$.*

22. L'algorithme du tri rapide ...

(a) A une complexité pire cas en $\Theta(n \log n)$ et une complexité moyenne en $\Theta(n)$.

(b) ►A une complexité pire cas en $\Theta(n^2)$ et une complexité moyenne en $\Theta(n \log n)$.

(c) Est un algorithme de type Las Vegas. *Le tri rapide randomisé, oui.*

(d) A la même complexité dans le pire cas et en moyenne.

(e) Peut terminer plus tard dans sa version probabiliste que dans le pire cas de la méthode déterministe.

23. Le retour sur trace (backtracking) ...

(a) Correspond à un parcours en largeur.

(b) ►Correspond à un parcours en profondeur.

(c) Peut entraîner des boucles.

(d) ►Nécessite d'organiser les données. *On organise les solutions partielles dans un arbre.*

(e) ►Est utilisé par l'algorithme de Quine.

24. Un algorithme de type branch-and-bound ...

(a) S'applique à un problème de décision. *Non : d'optimisation.*

(b) Est une variante d'algorithme diviser pour régner.

(c) ►Est une variante d'un algorithme de backtracking. *La différence étant : dans du backtracking on élague quand une solution partielle ne peut plus être complétée en une solution totale / dans B&B, on élague quand une solution partielle ne peut pas être complétée en une solution totale qui a un meilleur coût qu'une solution totale actuellement connue.*

(d) Renvoie une approximation de la solution. *Peut tout à fait renvoyer une solution exacte.*

(e) Aucune des réponses précédentes.

25. Un algorithme glouton ...

(a) N'est utile que s'il donne une solution exacte. *Non, peut être utile pour construire des approximations.*

(b) ►Peut être un algorithme d'approximation.

(c) Est forcément polynomial. *A priori non mais s'il ne l'est pas on ne va probablement pas l'utiliser.*

(d) Est un cas particulier de backtracking. *Au contraire, on ne revient jamais sur ses pas en glouton.*

(e) ►Ne revient jamais sur une décision prise à une étape précédente.

26. Quels sont les algorithmes gloutons parmi les suivants ?

(a) ►Algorithme de Huffman. *Choix des deux arbres ayant les sommes d'occurences les plus faibles.*

(b) ►Algorithme de Dijkstra. *Choix du sommet non traité avec la plus petite distance connue à l'origine.*

(c) ►Algorithme de Kruskal. *Choix des arêtes par ordre croissant des poids.*

(d) ►Algorithme pour sac à dos en prenant les objets par ordre croissant de valeur. *Immédiat.*

27. On note $(I_n)_{n\in\mathbb{N}}$ une suite d'instances d'un problème de minimisation $\mathcal{P}$, $\mathcal{A}$ un algorithme qui donne des solutions aux instances de $\mathcal{P}$, $C_n^\star$ le coût optimal pour I_n et C_n le coût de la solution trouvée par $\mathcal{A}$ sur I_n.

(a) Pour tout $n \in \mathbb{N}$, $C_n \leq C_n^\star$. *Dans un problème de minimisation, le coût optimal est au contraire plus petit que le coût de toute solution. L'inégalité est vraie si $\mathcal{A}$ est exact, puisque c'est alors une égalité.*

(b) S'il existe $\alpha > 0$ tel que $\forall n \in \mathbb{N}$, $C_n \leq \alpha C_n^\star$, alors $\mathcal{A}$ est une α-approximation pour $\mathcal{P}$. *C'est la réciproque qui est vraie. On pourrait ne pas avoir l'inégalité pour d'autres instances de $\mathcal{P}$ que les I_n.*

(c) Si $C_n^\star/C_n \to +\infty$ à l'infini, alors $\mathcal{A}$ n'est pas un algorithme d'approximation pour $\mathcal{P}$. *Techniquement c'est vrai. En effet, $C_n^\star/C_n \leq 1$ puisqu'on a un problème de minimisation. Donc $C_n^\star/C_n \to +\infty$ est un énoncé faux et peu importe la véracité de ce qui suit le "alors", la table de vérité de l'implication indique que (C) est vraie.*

(d) ►Si $C_n/C_n^\star \to +\infty$ à l'infini, alors $\mathcal{A}$ n'est pas un algorithme d'approximation pour $\mathcal{P}$. *Si $\mathcal{A}$ était une α-approximation, on aurait pour tout $n \in \mathbb{N}$ $C_n \leq \alpha C_n^\star$, donc $C_n/C_n^\star$ serait borné à l'infini.*

28. Un algorithme de programmation dynamique ...

(a) ►Nécessite de stocker des valeurs intermédiaires.

(b) Consiste exclusivement à mémoïser des résultats. *Pas dans les approches top-down.*

(c) Consiste à partitionner les solutions en sous-problèmes distincts. *N'a pas de sens !*

(d) ►Consiste à formuler la solution d'un problème en fonction de la solution à des sous problèmes.

(e) Est inadapté lorsque les sous problèmes se recoupent. *Au contraire c'est le cadre d'application privilégié.*

29. Quels algorithmes utilisent de la programmation dynamique ?

(a) Algorithme de Dijkstra.

(b) ►Algorithme de Floyd-Warshall.

(c) Algorithme de Kosaraju.

(d) ►Algorithme de calcul des attracteurs.

(e) Algorithme de Quine.

30. Quels algorithmes utilisent la méthode diviser pour régner ?

(a) ►Algorithme de tri rapide.

(b) ►Algorithme de tri fusion.

(c) ►Algorithme de recherche par dichotomie dans un tableau trié.

(d) ►Algorithme de construction d'un arbre k-d.

(e) ►Algorithme ID3.

31. En posant $C(0) = C(1) = 1$, pour quelles formules de récurrence obtient-on $C(n) = \Theta(\lambda^n)$ avec $\lambda > 1$?

(a) ►$C(n) = 3C(n-1)$. *Donne* $C(n) = \Theta(3^n)$.

(b) ►$C(n) = \sum_{i=0}^{n-1} C(i)$. *On a* $C(n) = C(n-1) + \sum_{i=0}^{n-2} C(i) = 2 \times \sum_{i=0}^{n-2} C(i) = 2C(n-1)$ *donc* $C(n) = \Theta(2^n)$.

(c) $C(n) = 3C(n/2) + O(n^2 \log n)$. *Donne* $C(n) = O(n^2 \log n)$.

(d) $C(n) = nC(n/2)$. *Si on note* $u_n = \log C(n)$, *cette relation donne* $u_n = u_{n/2} + \log n$, *d'où on tire que* $u_n \sim (\log_2 n)/2$. *On en déduit que* $u_n = \Theta(\sqrt{n}^{\log_2(n)})$ *qui est trop petit pour être un* $\Theta(\lambda^n)$ *(on raisonne par l'absurde et on conclurait que* $(\log_2 n)/n$ *est minoré par un nombre strictement positif).*

(e) ►$C(n) = 2C(n/2) + \Theta(2^n)$. *Donne* $C(n) = \Theta(2^n)$.

32. Pour rechercher si un motif m apparaît dans un texte t on peut ...

(a) ►Utiliser un automate.

(b) Utiliser l'algorithme de Lempel-Ziv-Welch. *Non, celui-ci c'est pour la compression.*

(c) ►Utiliser l'algorithme de Rabin-Karp.

(d) ►Utiliser l'algorithme de Boyer-Moore.

(e) ►Conclure en temps $O(|m||t|)$. *Oui, même avec l'algorithme naïf.*

33. L'algorithme de Kosaraju appliqué à un graphe ...

(a) ►Calcule les composantes fortement connexes.

(b) ►Utilise un double parcours en profondeur. *Techniquement le deuxième parcours n'est pas obligatoirement un parcours en profondeur.*

(c) ►Utilise le graphe transposé.

(d) ►Permet de résoudre 2SAT en temps polynomial.

(e) ►Utilise une structure de pile. *Via le parcours en profondeur.*

34. Un ordre topologique d'un graphe orienté G :

(a) ►existe ssi G est acyclique.

(b) est unique s'il existe.

(c) ►peut être calculé avec un parcours en largeur.

(d) ►peut être calculé en temps $O(|S| + |A|)$

35. Lesquels de ces algorithmes permettent de calculer la plus petite distance entre deux sommets d'un graphe ?

(a) ►$A^\star$.

(b) L'algorithme alpha-beta.

(c) ►L'algorithme de Dijkstra.

(d) L'algorithme de Boyer-Moore.

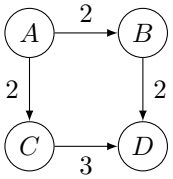
(e) ►L'algorithme de Floyd-Warshall.

36. L'algorithme $A^\star$ avec une heuristique h ... *Rappel : admissible* $= \forall v \in V, h(v) \leq \delta(v, but)$ */ monotone = cohérente* $= \forall (u,v) \in E, h(u) \leq p(u,v) + h(v)$.

(a) Correspond à l'algorithme de Floyd-Warshall si h est identiquement nulle. *Si* $h = 0$ *on retrouve Dijkstra.*

(b) ►Trouve un plus court chemin si h est admissible.

(c) Trouve un plus court chemin si h est monotone. *Contre-exemple avec* $h(A) = 0$, $h(B) = 5$, $h(C) = 0$, $h(D) = 5$ *(non admissible à cause de* B, *d'ailleurs) pour trouver un chemin de* A *à* D *dans :*



En revanche si $h(but) = 0$ *en plus de la monotonie, l'heuristique est admissible donc on trouve bien un plus court chemin.*

(d) Est polynomial en temps si h est admissible. *Non, potentiellement exponentiel, car un même sommet peut passer plusieurs fois dans la file de priorité avec des priorités différentes.*

(e) ►Est polynomial en temps si h est monotone. *Car tout noeud est extrait au plus une fois de la FdP.*

(f) N'est pas intéressant à appliquer sur un graphe non pondéré. *Trouver un plus court chemin en termes de nombre d'arêtes peut être pertinent dans certains contextes.*

37. Quelles propriétés sont justes sur les arbres couvrants parmi les suivantes ?

(a) Tout graphe possède un arbre couvrant. *Ce n'est pas le cas pour les graphes non connexes.*

(b) ►Dans un graphe G, tout arbre couvrant de poids minimum (ACM) contient au moins une arête de poids minimal de G. *"sinon, on prend* (s,t) *une des arêtes de poids minimal ; elles sont reliées dans*

l'ACM via un chemin qui n'emprunte aucune arête de poids minimal et il suffirait d'enlever une des arêtes de ce chemin et rajouter (s,t) pour avoir un arbre de poids strictement plus petit que celui d'un ACM.

(c) L'arête de poids maximal n'appartient à aucun ACM. *Considérer un graphe connexe à deux sommet : le seul ACM contient la seule arête qui est de poids maximal.*

(d) ►Si les poids de toutes les arêtes de G sont deux à deux distincts et que G est connexe alors G admet un unique ACM. *Montré en exercice.*

(e) Si G admet un unique ACM, les poids de ses arêtes sont deux à deux distincts. *Considérer un graphe connexe à 3 sommets et 2 arêtes de même poids.*

38. Lesquelles parmi les propriétés suivantes sur les couplages sont vraies ?

(a) Un couplage maximal est maximum. *Contre-exemple vu en cours.*

(b) ►Un couplage maximum est maximal.

(c) Un graphe admettant un couplage parfait est nécessairement biparti.

(d) ►Dans un graphe biparti, on peut calculer un couplage maximal en temps polynomial. *Oui puisqu'on peut calculer un couplage maximum via l'algorithme de Berge, qui est maximal.*

(e) ►Un couplage sans chemin augmentant dans un graphe biparti est maximum. *C'est même vrai dans les graphes qui ne sont pas bipartis : c'est le lemme de Berge.*

39. Soient C_1, C_2 deux couplages d'un graphe G. Alors :

(a) $C_1 \cup C_2$ est un couplage.

(b) $C_1 \cap C_2$ est un couplage.

(c) ►$C_1 \Delta C_2$ $(C_1 \cup C_2 \setminus C_1 \cap C_2)$ est un couplage. *On peut trouver des contre-exemples pour les autres*

(d) $C_1 \setminus C_2$ est un couplage.

40. L'algorithme des k plus proches voisins ...

(a) Est un algorithme d'apprentissage non supervisé. *Supervisé au contraire.*

(b) Gagne en précision lorsqu'on augmente k. *Pas nécessairement, surapprentissage.*

(c) Ne peut pas être acceléré par pré-traitement. *Faux car stocker les données d'entraînement dans un arbre d-dimensionnel permet de trouver plus rapidement les k voisins d'un point dont on cherche la classe.*

(d) ►Aucune des réponses ci-dessus.

41. Un arbre de décision ...

(a) ►Peut être construit par un algorithme d'apprentissage supervisé. *ID3 par exemple.*

(b) Classe toujours correctement tous les éléments de l'ensemble d'apprentissage. *ID3 peut parfois créer des feuilles en considérant la classe majoritaire des exemples correspondants. Si cette classe majoritaire est "oui", les exemples d'entraînement pour lesquels la réponse est "non" seront mal classés.*

(c) Peut être amélioré à l'aide d'arbres k-d. *A priori, non, ça c'est pour les k plus proches voisins.*

(d) Peut être calculé sans calcul d'entropie par l'algorithme ID3. *ID3 utilise justement l'entropie. Mais on pourrait cocher cette réponse car : on peut faire ID3 sans utiliser le critère d'entropie pour séparer les exemples, et on peut construire un arbre de décision sans ID3.*

(e) ►A une hauteur bornée par la dimension des données. *On sépare au plus sur chacun des attributs.*

42. L'algorithme des k moyennes ...

(a) Repose sur l'entropie de Shannon. *C'est ID3 qui fait ça.*

(b) Converge vers une réponse optimale. *On peut avoir convergence vers un optimum local non global.*

(c) Ne converge pas nécessairement. *Il y a en revanche toujours convergence.*

(d) ►Ne reconnaît pas les classes non convexes. *Exemple : deux classes concentriques.*

(e) Nécessite un calcul de médioïde. *Le bon terme est "centroïde".*

43. La classification obtenue par un algorithme de regroupement hiérarchique ascendant ...

(a) Est indépendante de la distance choisie.

(b) ►Permet de choisir le nombre de clusters. *En coupant à la hauteur voulue.*

(c) Ne reconnaît pas les classes non convexes.

(d) Nécessite un calcul de centroïde.

(e) ►A une complexité dépendante de la distance considérée. *Ne serait-ce que parce que la complexité du calcul de cette distance intervient dans la complexité de l'algorithme complet.*

44. Un algorithme reposant sur une heuristique ...

(a) Ne peut pas donner une réponse optimale à coup sûr. *Considérer $A^\star$ avec une heuristique admissible.*

(b) Ne peut généralement pas garantir le temps d'exécution et l'optimalité du résultat. *$A^\star$ avec une heuristique monotone et admissible fournit un contre-exemple. On peut légitimement cocher cette réponse à cause du "généralement".*

(c) Ne peut pas être exécuté avec une heuristique différente.

(d) ►Peut être ajusté pour une utilisation pratique précise en adaptant l'heuristique considérée.

(e) Aucune des réponses ci-dessus.

45. Les positions gagnantes pour un joueur J_1 dans un jeu à deux joueurs ...

(a) ►Peuvent être calculées en temps polynomial en la taille du jeu. *Avec l'algorithme des attracteurs, avec programmation dynamique.*

(b) Dépendent des réponses de l'adversaire. *Justement non.*

(c) Sont nécessairement des sommets contrôlés par le joueur J_1. *Pas obligatoirement.*

(d) ►Peuvent mener à une défaite de J_1. *Il suffit que J_1 joue autre chose qu'une stratégie gagnante !*

(e) Aucune des réponses ci-dessus.

Logique

46. La formule « $(A \to B) \leftrightarrow (\neg A \vee B)$ » :

(a) ►est vraie au sens de la logique propositionnelle

(b) est prouvable dans la logique minimale*on a besoin du tiers exclu pour montrer $A \to B \vdash \neg A \vee B$*

(c) ►est prouvable dans la logique classique (en ajoutant tiers exclu ou raa à la logique minimale)

47. Quelles formules sont des tautologies ?

(a) $((A \Rightarrow B) \Rightarrow C) \Leftrightarrow (A \Rightarrow (B \Rightarrow C))$. *Rendue fausse par $v(A) = v(B) = v(C) = 0$.*

(b) $(A \vee B) \wedge (\neg A \vee \neg B)$. *Est rendue fausse par $v(A) = v(B) = 0$.*

(c) ►$\neg(A \wedge (A \vee B)) \Leftrightarrow \neg A$. *Il suffit de faire une disjonction de cas selon $v(A)$.*

(d) $((A \vee B) \wedge (C \vee D)) \Leftrightarrow (A \wedge B) \vee (B \wedge D)$. *Aucune chance.*

(e) ►$(A \Rightarrow B) \Leftrightarrow (\neg A \vee B)$. *Vrai vu l'équivalence classique $A \Rightarrow B \equiv \neg A \vee B$.*

48. La taille d'une formule est majorée par sa hauteur.

(a) Vrai.

(b) ►Faux. *C'est la hauteur d'un arbre qui est majorée par sa taille.*

49. Une variable libre ...

(a) ►Peut être liée à un autre endroit dans la formule. *Techniquement c'est plutôt une occurrence de cette variable qui peut être liée à un autre endroit.*

(b) A une portée. *Ce sont les quantificateurs qui ont une portée.*

(c) N'influe pas la sémantique d'une formule.

(d) Est toujours associée à une variable liée.

50. Soit φ et ψ deux formules telles que $\psi \models \varphi$. Cela signifie ...

(a) ►Que φ est conséquence de ψ.

(b) Que ψ est conséquence de φ.

(c) Que les valuations satisfaisant φ satisfont aussi ψ. *C'est l'inverse.*

(d) ►Si φ est une antilogie, alors ψ aussi. *Si une valuation satisfaisait ψ, elle satisferait φ ce qui n'est pas.*

(e) Cela n'a pas de sens, la notation $\models$ n'existe que sous la forme $v \models \varphi$ avec v une valuation.

51. Une équivalence entre formules du calcul propositionnel ...

(a) Signifie qu'elles ont le même arbre de dérivation.

(b) ►Peut se prouver à l'aide de tables de vérité.

(c) ►Peut se prouver à l'aide de substitutions dans des formules qu'on sait déjà être équivalentes.

(d) Ne peut pas être démontrée sans la déduction naturelle. *Si, avec un raisonnement sémantique.*

(e) Peut se prouver par une étude de la matrice de confusion.

52. Pour toute formule du calcul du calcul propositionnel il existe une formule équivalente ...

(a) ►Sous forme normale conjonctive.

(b) ►Sous forme normale disjonctive.

(c) Sous forme normale de Quine. *L'algorithme de Quine calcule la satisfiabilité d'une formule.*

(d) Sous forme normale littérale.

(e) Sous forme normale de Chomsky. *ça c'est pour les grammaires algébriques.*

(f) Aucune des réponses précédentes.

53. On sait résoudre le problème SAT en temps polynomial sur des instances ...

(a) ►Sous 2-CNF. *Via Kosaraju ou DPLL.*

(b) Sous 3-CNF. *A priori non par NP-complétude de `3-SAT`.*

(c) ►Sous 2-DNF. *La satisfiabilité d'un formule sous DNF se décide toujours polynomialement : il suffit de parcourir chaque clause et d'observer si l'une au moins ne contient pas un littéral et sa négation.*

(d) ►Sous 3-DNF.

(e) Ne faisant intervenir que les connecteurs $\wedge$ et $\neg$.

54. Quels séquents sont prouvables en logique classique ? *Il suffit de regarder si le séquent est valide : si oui, il est prouvable et sinon non. Il n'y a donc pas besoin de chercher des arbres de preuve pour cette question.*

    (a) ►$\neg\neg A \vdash A$.
    (b) ►$\neg\neg\neg\neg A \vdash \neg A$.
    (c) ►$A \wedge B \vdash A \vee B$.
    (d) $A \vee B \vdash A \wedge B$.
    (e) ►$A, A \to B \vdash B$.

55. Pour la déduction naturelle en logique propositionnelle, $\Gamma \vdash F$ si et seulement si $\Gamma \models F$.

    (a) ►Vrai. *La déduction naturelle est correcte et complète (HP).*
    (b) Faux.

Théorie des langages

56. Sur l'aphabet $\Sigma = \{a, b\}$:

    (a) $\emptyset$ est une lettre.
    (b) $\emptyset$ est un mot.
    (c) ►$\emptyset$ est un langage.
    (d) ε est une lettre.
    (e) ►ε est un mot.
    (f) ►ε est un langage.*Le langage contenant seulement le mot ε*

57. Si u et v sont deux mots, quelles propriétés parmi les suivantes sont vraies ?

    (a) $|u| > 0$. *On pourrait avoir $u = \varepsilon$.*
    (b) u est préfixe de v ou v est préfixe de u. *Contre-exemple avec $u = a$ et $v = b$.*
    (c) Si u est préfixe de v alors v est suffixe de u. *Contre-exemple : $u = a$ est préfixe de $v = ab$.*
    (d) ►Si $u = vw$ alors v est un sous-mot de u. *C'en est même un préfixe.*
    (e) ►Si $u = vw$ alors v est un facteur de u. *C'en est même un préfixe.*

58. Si L, L' et L'' sont des langages alors ...

    (a) Si L et L' sont finis, $|LL'| = |L| \times |L'|$. *Contre-exemple : $L = \{\varepsilon, a\}$ et $L' = \{b, ab\}$. On a $LL' = \{b, ab, aab\}$ qui n'est pas de taille 4. On peut obtenir le même mot avec plusieurs concaténations différentes.*
    (b) $L^\star$ est infini. *Pas si $L = \{\varepsilon\}$.*
    (c) $L \subset LL$. *Contre-exemple : $L = \{a\}$; on a alors $LL = \{aa\}$. Fonctionne ssi L contienne ε.*
    (d) $L(L' \cap L'') = LL' \cap LL''$. *Contre-exemple : $L = \{a, ab\}$, $L' = \{\varepsilon\}$, $L'' = \{b\}$ car $\emptyset \neq \{ab\}$.*
    (e) ►Aucune des réponses précédentes.

59. Si L et L' sont deux langages tels que $L \subset L'$, quelles affirmations sont vraies ?

    (a) Si L' est reconnaissable alors L aussi. *Contre-exemple : $L = \{a^n b^n \mid n \in \mathbb{N}\}$ et $L' = (a + b)^\star$.*
    (b) Si L' n'est pas rationnel, L non plus. *Contre-exemple : $L' = \{a^n b^n \mid n \in \mathbb{N}\}$ et $L = \{\varepsilon\}$.*
    (c) ►Si L' est fini, L est rationnel. *Car alors L est fini donc rationnel.*
    (d) ►Si L est reconnaissable, il l'est par un automate déterministe et complet. *Si L est reconnaissable par un automate, on peut déterminiser et compléter ce dernier.*
    (e) Si L est reconnaissable, il l'est par un automate émondé et complet. *On ne peut pas garantir le caractère à la fois émondé et complet, par exemple avec $L = \{a\}$ sur l'alphabet $\{a, b\}$.*

60. Le langage $L = \{a^n b^m \mid n \equiv m \bmod 2\}$ est dénoté par l'expression régulière ... *Ce langage est bien rationnel et est dénoté entre autres par $(a^2)^\star(b^2)^\star + a(a^2)^\star b(b^2)^\star$: soit les exposants sont pairs tous les deux, soit impairs.*

    (a) $(aa+bb)^\star$. *aabbaa est dans ce langage mais pas dans L.*
    (b) $(aa + ab + ba + bb)^\star$. *abba est dans ce langage mais pas dans L.*
    (c) $((a(a + b)^\star b)^\star + (b(a + b)^\star a)^\star)^\star$. *abab est dans ce langage mais pas dans L.*
    (d) $a^n b^m$. *N'a pas de sens : qui sont m et n ?*
    (e) ►Aucune des réponses précédentes.

61. Les langages rationnels sont stables par ...

    (a) ►Préfixe. *Émonder un automate qui reconnaît L et rendre tous ses états finaux.*
    (b) ►Miroir. *Changer le sens des transitions d'un automate qui reconnaît L, les initiaux deviennent finaux et les finaux initiaux.*
    (c) ►Complémentaire. *On calcule un automate déterministe et complet (les deux sont importants) pour L puis les finaux deviennent non finaux et les non finaux deviennent finaux.*
    (d) ►Intersection. *Via la construction de l'automate produit.*

62. Un automate fini déterministe et complet ...

    (a) Est nécessairement émondé. *Contre-exemple : Un automate qui reconnaît $\{a\}$ sur $\{a, b\}$ avec un puits.*
    (b) Est unique pour chaque langage à renommage près des états. *C'est vrai dans le cadre d'un automate minimal. Pour un automate quelconque, on peut toujours rajouter des états inutiles.*

(c) ►Permet de tester l'appartenance d'un mot à un langage en temps linéaire. *Il suffit de lire toutes les lettres du mot et il n'y a qu'un seul chemin à explorer par déterminisme.*

(d) ►Donne aisément un automate reconnaissant le complémentaire du langage. *En inversant finaux et non finaux. Attention cette transformation ne marche pas si l'automate n'est pas déterministe et complet.*

(e) Est de taille exponentielle en la taille d'un automate non déterministe reconnaissant le même langage. *Pas nécessairement, même si au pire cas ça peut arriver.*

63. Si e est une expression rationnelle, le problème consistant à savoir si un mot u appartient à $L(e)$ ...

(a) ►Nécessite au moins une complexité linéaire en $|u|$. *Il faut a priori lire le mot. On peut ne pas cocher cette réponse à cause du "nécessite" : si $e = \emptyset$ par exemple, il n'y a pas besoin de lire le mot.*

(b) ►Peut être résolu en temps polynomial en $|u| + |e|$. *Construire un automate A non déterministe qui reconnaît $L(e)$ se fait polynomialement en $|e|$ via l'algorithme de Berry-Sethi. Cet automate a au plus de l'ordre de $|e|$ états et il y a au plus $|e|$ transitions par état. La lecture de u dans A est aussi polynomiale en $|e|$ et $|u|$: il suffit de garder en mémoire l'ensemble d'états (au plus n) dans lequel on peut être après lecture de chacune des au plus $|u|$ lettres de u, et lire une lettre consiste à examiner au plus $|e|$ transitions pour un coût majoré par $O(|u||e|^2)$.*

(c) Est un problème indécidable. *Non : cf (B).*

(d) N'a pas d'application pratique. *Ne serait-ce que la reconnaissance de motif.*

(e) Aucune des réponses précédentes.

64. Tout automate est équivalent à...

(a) un automate déterministe, construit en complexité polynomiale.*complexité exponentielle car 2^n états dans l'automate des parties*

(b) ►un automate complet.

(c) ►un automate déterministe complet.*algorithme de déterminisation*

(d) un automate déterministe complet émondé.*rendre l'automate demande d'avoir un état « poubelle » qui n'est pas co-accessible*

(e) ►un automate sans ε-transition.

65. Un automate produit de deux automates A_1 et A_2 d'états Q_1 et Q_2...

(a) ►possède $Q_1 \times Q_2$ comme ensemble d'états.

(b) peut permettre de reconnaître $L(A_1)L(A_2)$.

(c) ►peut permettre de reconnaître $L(A_1) \cup L(A_2)$.

(d) ►peut permettre de reconnaître $L(A_1) \cap L(A_2)$.

(e) ►peut permettre de reconnaître $L(A_1) \setminus L(A_2)$.

(f) ►peut permettre de reconnaître $L(A_1)\Delta L(A_2)$.

66. On peut montrer que tout langage rationnel est reconnaissable par un automate fini ...

(a) En construisant l'automate des parties. *Permet de déterminiser un automate.*

(b) ►En utilisant les automates de Thompson.

(c) En utilisant la méthode d'élimination des états. *Permet de faire la réciproque, à savoir montrer qu'un langage reconnu est dénoté par un expression rationnelle.*

(d) Avec l'algorithme de McNaughton Yamada. *Idem (C).*

(e) ►En utilisant l'algorithme de Berry-Sethi.

67. Parmi les langages suivants, lesquels sont réguliers ?

(a) $\{a^n b^n \mid n \geq 0\}$. *Preuve par l'absurde avec le lemme de l'étoile.*

(b) Les mots bien parenthésés. *Preuve par l'absurde avec le lemme de l'étoile.*

(c) ►$\{a^n b^m \mid n \equiv m \bmod 2\}$. *Cf question 62. Ou faire un automate à 4 états.*

(d) ►Les mots sur $\{0, 1\}$ correspondant aux écritures binaires d'entiers divisibles par 3. *Cf exemple introductif du cours où les états indiquent les restes possibles modulos 3.*

(e) $\{a^p \mid p$ est premier$\}$. *Preuve par l'absurde avec le lemme de l'étoile.*

68. Un langage L est local ssi...

(a) ►L est reconnu par un automate local.

(b) L est inclus dans un langage régulier.

(c) ►$L\backslash\{\varepsilon\} = P(L)\Sigma^* \cap \Sigma^* S(L) \backslash \Sigma^* N(L)\Sigma^*$ où $N(L) = \Sigma^2 \backslash F(L)$.

(d) ►$u \in L \iff u_1 \in P(L) \wedge u_n \in S(L) \wedge \forall k, u_k u_{k+1} \in F(L)$

69. Parmi les conditions suivantes, lesquelles sont suffisantes pour que L soit rationnel ?

(a) L est dénombrable. *Contre-exemple : $L = \{a^n b^n \mid n \in \mathbb{N}\}$.*

(b) ►L est reconnu par un automate non déterministe.

(c) ►L est le complémentaire d'un langage rationnel. *Par stabilité des rationnels par complémentaire.*

(d) $L^\star$ est rationnel. *Contre-exemple : $L = \{a^p \mid p$ est premier$\}$ puisque $L^\star = a^\star$ est rationnel.*

(e) $L \subset \{a\}^\star$. *Contre-exemple : $L = \{a^p \mid p$ est premier$\}$.*

(f) ►L est local.

70. Si un langage vérifie le lemme de l'étoile, alors il est rationnel.

(a) Vrai.

(b) ►Faux. *Le lemme de l'étoile est une implication.*

71. Les langages algébriques ...

(a) Sont inclus dans les langages rationnels. *L'inclusion réciproque est vraie.*

(b) ►Sont stables par étoile de Kleene.

(c) Sont stables par complémentaire. *Si c'était le cas, la stabilité par union ferait qu'on a la stabilité par intersection ce qui n'est pas d'après le contre-exemple suivant.*

(d) Sont stables par intersection. $\{a^n b^n c^m \mid n, m \in \mathbb{N}\}$ *et* $\{a^m b^n c^n \mid n, m \in \mathbb{N}\}$ *(le premier est engendré par* $S \to ZC, C \to cC|\varepsilon, Z \to AZB\,\varepsilon, A \to a, B \to b$*) sont algébriques mais pas leur intersection* $\{a^n b^n c^n \mid n \in \mathbb{N}\}$.

(e) ►Sont stables par concaténation.

72. Quelles sont les grammaires ambiguës ?

(a) ►$S \to SS \mid a$. *Via le mot aaa.*

(b) $S \to ST \mid TS$, $T \to SS$. *Engendre* $\emptyset$ *donc il est vrai que tout mot engendré par cette grammaire admet un seul arbre de dérivation, d'après les merveilleuses propriétés de l'ensemble vide.*

(c) ►$S \to aSbS \mid bSaS \mid \varepsilon$. *Via le mot abab.*

(d) ►$S \to aSb \mid ab \mid \varepsilon$. *Via le mot ab.*

(e) $S \to aS \mid bS \mid \varepsilon$. *Car il n'y a jamais le choix sur comment dériver.*

73. Les dérivations droites sont en bijection avec les arbres de dérivation. *Au sens où si on se donne* G *et* $u \in L(G)$*, il y a bijection entre les arbres de dérivation pour* u *et les dérivations droites pour* u*. En effet, si on a un arbre, il donne une dérivation droite ; celle consistant à appliquer les règles telles que rencontrées dans un parcours en profondeur de l'arbre effectué de droite à gauche. Une dérivation droite donne immédiatement un arbre.*

(a) ►Vrai.

(b) Faux.

74. Un analyseur lexical ...

(a) ►Intervient avant l'analyseur syntaxique.

(b) ►Identifie les lexèmes.

(c) Permet de déterminer la structure d'un programme. *C'est le rôle de l'analyse syntaxique.*

(d) ►Peut utiliser un automate fini. *Pour reconnaître les mots qui sont des lexèmes.*

75. Un analyseur syntaxique ...

(a) Ne peut gérer que des grammaires non ambiguës. *Non, c'est juste plus délicat.*

(b) ►Considère les lexèmes comme des terminaux.

(c) N'est pas utilisé en pratique. *Si, dans les compilateurs par exemple.*

(d) Choisit la grammaire à considérer en fonction des données en entrée. *N'a pas de sens.*

(e) Aucune des réponses précédentes.

Complexité et calculabilité

76. Parmi les problèmes ci-dessous, lesquels sont des problèmes de décision ?

(a) La résolution d'une grille de Sudoku.

(b) ►La primalité d'un entier.

(c) Le plus court chemin entre deux points.

(d) ►Le problème de l'arrêt.

(e) ►L'égalité entre deux langages.

77. Parmi les problèmes ci-dessous, lesquels sont des problèmes d'optimisation ?

(a) La résolution d'une grille de Sudoku. *On pourrait argumenter que ce problème est celui consistant à minimiser le nombre d'incohérences (deux mêmes chiffres sur la même ligne, colonne, carré) dans une grille complète qui respete celle en entrée. Mais c'est un peu abusif car a priori, résoudre un Sudoku veut plutôt dire : trouver une solution correcte ou affirmer qu'il n'y en a pas.*

(b) La primalité d'un entier.

(c) ►Le plus court chemin entre deux points.

(d) Le problème de l'arrêt.

(e) L'égalité entre deux langages.

78. Si n est la taille de l'entrée, pour quelles complexités peut-on dire que l'algorithme considéré est polynomial ?

(a) ►$O(\log n!)$. *Car ceci est un* $O(n \log n) = O(n^2)$

(b) $2^{o(n)}$. *On a* $(\log n)^2 = o(n)$ *et si on avait* $2^{(\log n)^2} \leq n^k$ *pour un certain* k*, on aurait* $(\log n)^2 / \log n \leq k$.

(c) $O((\log n)^n)$. *S'il existe* k *tel que* $(\log n)^n \leq n^k$*, on aurait* $n \log\log n \leq k \log n$ *: contradiction à l'infini.*

(d) ►$O(\sqrt{n})$. *C'est par exemple un* $O(n)$.

(e) $O(2^n)$. *Est évidemment exponentiel en* n.

79. L'algorithme cherchant à déterminer si un entier n est premier en calculant son modulo par tous les entiers entre 2 et $\lceil \sqrt{n} \rceil$ est de complexité ...

(a) Quasi-linéaire.

(b) Polynomiale. *Non car la taille de l'entier n est $\log n$ donc $O(\sqrt{n}) = O(\sqrt{e^{\log n}})$.*

(c) Sous-linéaire.

(d) ►Exponentielle.

(e) Linéaire. *Cette réponse, ainsi que (A) et (C) sont fausses vu (B).*

80. La complexité d'un problème de décision correspond à ...

(a) ►La complexité pire cas du meilleur algorithme qui le résout. *On pourrait ne pas cocher cet item car la notion de "meilleur algorithme" est trop floue. Un algorithme peut être dans plusieurs classes de complexité différentes selon le type d'algorithme pour le résoudre considéré.*

(b) La complexité pire cas du pire algorithme qui le résout.

(c) La complexité meilleur cas du meilleur algorithme qui le résout.

(d) ►Dépend du format d'entrée des instances.

(e) N'est pas définie.

81. On suppose que A est NP-complet. Soit $B \in$ NP. Pour montrer que B est NP-complet, il suffit de montrer...

(a) ►A se réduit polynomialement à B.

(b) B se réduit polynomialement à A.

(c) ►$A \leq_p B$.

(d) $B \leq_p A$.

(e) ►À chaque instance I_A de A, on peut associer une instance I_B de B en temps polynomial telle que I_A est une instance positive de A ssi I_B est une instance positive de B.

(f) À chaque instance I_B de B, on peut associer une instance I_A de A en temps polynomial telle que I_A est une instance positive de A ssi I_B est une instance positive de B.

82. Si le problème A se réduit polynomialement au problème B alors ...

(a) Si A se résout en temps polynomial, B aussi. *Tout problème dans P est NP donc se réduit à `SAT`.*

(b) ►Si B se résout en temps polynomial, A aussi.

(c) A est considéré comme plus difficile que B. *Non, c'est l'inverse.*

(d) ►Une instance de A peut être transformée en une instance de B. *Dans le cadre de réductions many-one.*

(e) Si A est décidable alors B aussi.

83. Si un problème est dans NP alors ...

(a) On ne sait pas le résoudre en temps polynomial. *Tout problème P est dans NP.*

(b) ►Il peut être résolu en temps exponentiel.

(c) ►Il peut être résolu en espace polynomial. *Si A est dans NP, il existe un ensemble de certificats C et un problème $B \in P$ tel que x est une instance positive de A ssi il existe $c \in C$ de taille polynomiale en $|x|$ tel que (x, c) est positive pour B. Un algorithme pour savoir si x est positive pour A consiste donc à parcourir tous les certificats et utiliser un algo polynomial pour B. Pour chaque certifiat on utilise un temps donc un espace polynomial et on réutilise le même espace pour stocker le certificat suivant.*

(d) Il s'agit d'un problème fonctionnel.

(e) Pour toute instance du problème, on peut créer un certificat en temps polynomial. *Si on savait faire ça, on aurait $P = NP$. Pour un problème NP on sait juste qu'il existe un certificat pour toute instance positive et que la vérification qu'un certificat en est bien un se fait en temps polynomial*

84. Lesquels de problèmes suivants sont NP-complets ?

(a) ►`3SAT`.

(b) Le problème de l'arrêt. *Il n'est même pas décidable.*

(c) ►L'existence d'un chemin hamiltonien.

(d) L'appartenance d'un mot à un langage régulier. *Non, cf partie langages.*

(e) ►Le problème CLIQUE.

85. Les problèmes indécidables ...

(a) Ne sont pas les mêmes suivant le langage de programmation choisi. *Techniquement c'est vrai, dès lors qu'on considère un langage Turing-complet et un qui ne l'est pas.*

(b) Ne peuvent pas être réduits à un autre problème.

(c) Dépendent de la façon dont sont représentées les instances du problème. *En revanche la complexité d'un algorithme qui résout un problème décidable dépend de la façon dont sont représentées les instances.*

(d) Ne concernent que des problèmes se rapportant au fonctionnement d'algorithmes. *Non, par exemple, déterminer si une grammaire algébrique est ambiguë est indécidable.*

(e) ►Aucune des réponses précédentes.

86. Parmi les suivants, quels sont les problèmes indécidables ?

(a) ►Le problème de l'arrêt sur les entrées de taille inférieure à 5.

(b) Le problème de la terminaison en moins de n étapes sur toute instance de taille inférieure à n. *Il suffit d'exécuter n étapes de calcul (en $O(n)$) et de regarder si c'est fini.*

(c) ►Le problème de la terminaison en strictement plus de n étapes sur toute instance de taille inférieure à n.

(d) Le problème de l'équivalence de deux expressions régulières. *Il suffit de faire des automates pour les deux, d'en construire la différence symétrique et de vérifier si le langage reconnu est vide.*

(e) Le problème de l'équivalence entre deux formules du calcul propositionnel. *Il suffit de faire leurs tables de vérité.*