MÉTA-PLANS POUR LES LEÇONS D'INFORMATIQUE POUR L'AGRÉGATION DE MATHÉMATIQUES OPTION D

Table des matières

Introduction

Voici la liste des mes méta-plans pour les leçons d'informatique de l'option D. Ils contiennent les grandes lignes de ce que je comptais mettre dans mes leçons le jour de l'oral, les références permettant de construire ce plan et deux développements référencés. J'indique pour certaines parties la référence plus ou moins précise d'où je la tire entre crochets. La leçon 920 était mon impasse ; aussi, pas de plan pour celle-ci. J'espère que ce document vous aidera à élaborer vos propres plans !

901. Structures de données. Exemples et applications.

Plan :

Motivation : Le choix d'une structure de données adaptée au problème algorithmique qu'on se pose peut permettre de faciliter la conception de l'algorithme ou d'améliorer sa complexité (temporelle ou spatiale).

I/Formalisme [BBC]

→ Définition d'un type + types de base : entiers, booléens, tableau, liste (doublement) chaînée
→ Définition d'un type de données abstrait
→ Comment décrire un type de données abstrait : signature (= sorte et opérations possibles) + préconditions (= domaine de définition des opérations) + axiomes (= prop des opérations)
→ Ex : Description d'un dictionnaire
→ Rq : Les types abstraits permettent de concevoir des algorithmes en dehors de tout langage de programmation
→ Définition : structure de données = type abstrait + implémentation
→ Ex : Reprendre celui avec le dictionnaire

II/Structures linéaires [FG]

→ Ex : Liste : description du type + avantages et inconvénients des différentes implémentations (tableau / liste chaînée)
→ Ex : Pile : description de type + implémentations + appli : parcours en profondeur d'un graphe
→ Ex : File : description du type + implémentations + appli : parcours en largeur d'un graphe

III/Structures arborescentes [FG] [Cor]

1) Graphes

→ Définition d'un graphe (orienté) + ex vie courante
→ Définition d'un graphe par son type + implémentations = matrice d'adjacence, liste d'adjacence, liste des arêtes
→ Applis : Floyd-Warshall (matrice d'adjacence), algos de plus courts chemins, Prim...

2) Arbres

→ Définition d'un arbre + ex : arbre généalogique, arborescence de fichiers
→ Définition arbre binaire par induction / par son type abstrait + implem
→ Définition d'un tas + appli : tri par tas DEV1

IV/Quelques structures adaptées à la recherche

1) ABR

→ Définition arbre binaire de recherche (ABR) + complexité d'un recherche
→ Pb de l'ABR = sa hauteur. D'où les AVL pour équilibrer
→ Définition ABR optimal + complexité de la recherche

2) Tables de hachage

→ Définition table de hachage, clé, univers
→ Définition collisions + résolution par chaînage
→ Définition hachage parfait + comment l'obtenir sans prendre trop de mémoire DEV2
→ Rq : On peut aussi utiliser une fonction de hachage pour vérifier l'intégrité de données (md5sum)
→ On peut aussi parler de la structure Union-find + appli à la recherche d'arbre couvrant minimal avec Kruskal

ANNEXE : faire plein de dessins !

Références : Cormen [Cor], Beauquier [BBC], Froidevaux-Gaudel [FG]

Développements :

→ Hachage parfait [Cor p 258-262]
→ Complexité du tri par tas [Cor p 142-148]

902. Diviser pour régner. Exemples et applications.

Plan :

I/Principe

1) Description

→ C'est un méthode de conception d'algos : on divise le problème en sous problèmes de même nature, on résout récursivement ces sous problèmes (directement s'ils sont suffisamment "petits") puis on combine les solutions

→ Ex : Exponentiation rapide

→ Ex : Calcul de la hauteur d'un arbre binaire, parcours infixe, préfixe, postfixe des sommets d'un arbre

→ Ex : Recherche dichotomique dans un tableau trié

2) Complexité

→ Théorème fondamental (= "master theorem") + attention il ne couvre pas tous les cas

→ Ex : $T(n) = 2T(\frac{n}{2}) + \Theta(n)$ donne un complexité en $\Theta(nln(n))$ (relation de récurrence courante)

→ Ex : Expression de la complexité + complexité pour exponentiation rapide, recherche dichotomique et parcours d'arbres

3) Limites

→ Ex : Calcul des termes de la suite de Fibonacci + arbre des appels en annexe. Mauvaises complexités

→ Généralisons : Quand les sous problèmes se recoupent, DPR n'est pas idéal ; on se tourne alors vers la programmation dynamique

II/Deux algorithmes de tri

→ Tri fusion : entrée, sortie, sous problèmes, algorithme, relation de récurrence pour la complexité et complexité + rq : c'est un tri optimal et il en existe une version en place

→ Tri rapide : la même chose mais plus vite, écrire l'algo + modification pour obtenir le tri rapide randomisé DEV1

→ Appli (des deux tris) : Visualisation de données, tri des clés dans Dijkstra ou Prim, gestion de files de priorité, utile dans les algos gloutons...

III/Multiplications en tous genres

1) Produit de nombres

→ Ex : Produit de nombres par Karatsuba [Das p 45]

→ Applis : Calcul de produit matriciel, RSA, algos de calcul formel

2) Produit de matrices

→ Ex : Algorithme de Strassen

→ Appli : Calcul de l'inverse d'une matrice (en utilisant son polynôme minimal), résolution de systèmes linéaire, affichage graphique, algorithme de Floyd-Warshall pour les plus courts chemins, calcul de la fermeture transitive d'un graphe (avec Floyd-Warshall aussi)

3) Produit de polynômes

→ Ex : Algorithme FFT DEV1

→ Appli : dans une certaines mesure, Berlekamp !

Références : Cormen [Cor], Dasgupta [Das]

Développements :

→ Analyse du tri rapide randomisé [Cor p 166-170 et 158-161]

→ FFT [Cor p 827]

903. Exemples d'algorithmes de tri. Correction et complexité.

Idée de défense de plan : Intérêts de la leçon : pratique (on retrouve un tri dans beaucoup d'algos) + théorique (les algos de tris sont nombreux et variés, utilisent différentes méthodes de conception d'algos)

Plan :

I/Le problème du tri

→ Déf : Enoncé du problème du tri : entrée = une famille $(x_i)_{i\leqslant n}$ et un ordre total $<$ / sortie = une famille $(x_{\sigma(i)})_i$ triée (cad tel que $i < j \Rightarrow x_i \leqslant x_j$)

→ Rq : Notation Ω, Θ et O + définition complexité temporelle et spatiale

→ Définition tri stable / tri en place

→ rq : Les deux derniers points permettent de comparer les tris

→ Applis du tri : tri des transactions bancaires (meilleure visualisation), recherche dichotomique, intervient dans des algos comme Kruskal (arbre couvrant minimal) ou le balayage de Graham (enveloppe convexe en 2D), tri des clés dans une file de priorité ou dans les algos gloutons...

II/Tris par comparaison

→ Définition d'un tri par comparaison + rq : la complexité est le nombre de comparaisons

→ Prop : la complexité pire cas d'un tri par comparaison est en $\Omega(nln(n))$

→ Ex : Tri insertion : algo, complexité en $O(n^2)$, en place, stable

→ Ex : Tri fusion : algo, utilise DPR, complexité en $O(nln(n))$, stable, parallélisable, il en existe une version en place (difficile)

→ Ex : Tri rapide : algo, complexité en $O(n^2)$ au pire cas et en $O(nln(n))$ en moyenne, en place, parallélisable, pas stable (donner un ct-ex) + version randomisée DEV2

→ Ex : Tri par tas : def d'un tas max, algo, complexité en $O(nln(n))$, en place, pas stable DEV2

III/Autres tris (plus d'hypothèses qu'un tri par comparaison)

→ Ex : Tri par dénombrement : on rajoute l'hp que les éléments à trier sont dans $[\![0, k]\!]$. Algo, complexité en $O(n + k)$, stable mais pas en place

→ Ex : Tri par base : utilise en sous routine un tri stable, permet de trier des mots selon l'ordre lexicographique

→ Ex un peu à part : tri topologique. On sort un peu du cadre du tri car l'ordre qu'on considère est partiel (on veut justement le compléter en ordre total)

Références : Cormen [Cor]

Développements :

→ Analyse du tri rapide randomisé [Cor p 166-170 et 158-161]

→ Complexité du tri par tas [Cor p 142-148]

906. Programmation dynamique. Exemples et applications.

Plan :

I/Programmation dynamique : application à un exemple et principe généraux
1) Les limites de diviser pour régner (DPR)
- → Ex : Calcul des coeffs binomiaux + complexité exponentielle + arbre des appels récursifs en annexe
- → Amélioration temporelle via le triangle de Pascal
- → Amélioration spatiale : on n'a pas besoin de garder tout le triangle

2) Idées générales
- → Rappel de DPR : Pour résoudre le problème P_n, on résout $P_{i_1}, ..., P_{i_k}$ puis on les combine
- → Définition programmation dynamique : tout pareil sauf que les problèmes se chevauchent, du coup on les résout "dans le bon ordre"
- → Schéma général de la programmation dynamique :

```
Fonction résoudre le problème P :
    Résoudre les sous problèmes triviaux de P
    Parcourir les sous problèmes dans l'ordre croissant
        Résoudre le sous problème grâce à ceux déjà résolus
    Donner la solution de P
```

- → Rq : Ici, on résout les problèmes de façon ascendante. On peut garder un démarche descendante avec la mémoïsation
- → Schéma de la mémoïsation :

```
Fonction résoudre le problème P :
    Si la solution de p est déjà calculée, la renvoyer
    Sinon, la calculer, la renvoyer et la stocker
```

- → Rq : Avantage de la mémoïsation : on ne résout que les sous problèmes nécessaires

II/Exemples en algorithmique des graphes
- → Ex : Plus court chemin dans un graphe acyclique orienté [Das p 156]
- → Ex : Bellman-Ford DEV1 + améliorations possibles
- → Ex : Floyd-Warshall + appli : calcul de la fermeture transitive d'un graphe (ça permet d'enlever les ε-transitions dans un automate fini)
- → Ex : Ensembles indépendants dans un arbre [Das p 175] + rq : ce problème est NP-complet dans un graphe quelconque

III/Exemples en algorithmique du texte
- → Ex : Distance d'édition
- → Ex : Plus longue sous séquence commune
- → Ex : Algo CYK DEV2

Références : Cormen [Cor], Crochemore [Cro] (pour les algos du texte), Carton [Car], Dasgupta [Das]

Développements :
- → Bellman-Ford [Cor p 602-605]
- → CYK [Car p 199]

907. Algorithmique du texte. Exemples et applications.

Idée de défense de plan : les trois parties sont relativement indépendantes et présentant diverses structures / méthodes de programmation utilisées dans les algos du texte. Divers axes sont explorés : recherche de motif (appli : le "rechercher" dans la partie rechercher/remplacer dans un fichier), comparaison de mots (appli : propositions des correcteurs orthographiques, différences entre fichiers avec la commande diff, différence entre brins d'ADN), analyse syntaxique, compression sans perte (stockage, transmission)

Plan :

I/Programmation dynamique et comparaison de mots

1) Distance d'édition
   - → Définition substitution, délétion, insertion
   - → On peut mettre des coûts à ces opérations d'où la définition de distance d'édition + rq : le coût ne dépend pas de la position dans le mot ce qui est peu crédible par exemple en génomique
   - → Prop : Sous les bonnes hypothèses, la distance d'édition est une distance [Cro p 226] + ex : avec les bons coûts, on retrouve la distance de Hamming
   - → Définition alignement optimal + non unicité + ex en annexe
   - → Algo : Calcul de la distance d'édition (et d'un alignement)

2) Plus longue sous séquence commune (PLSC)
   - → Définition de PLSC + ex
   - → Prop : Avec des coûts d'insertion/délétion égaux à 1 et un coût de substitution égal à ∞, on a distance d'édition(u,v) = |u| + |v| - 2(longueur de la PLSC entre u et v)
   - → Rq : On peut donc utiliser l'algorithme précédent pour trouver la longueur d'une PLSC
   - → Prop : Recherche naïve + complexité
   - → Algo spécifique pour une PLSC [Cro p 365]

3) CYK
   - → Contexte de l'analyse syntaxique
   - → Algo : CYK DEV1

II/Utilisation d'automates finis et recherche de motifs [Cor]
   - → Présentation du problème de la recherche de motif
   - → Algo naïf et complexité
   - → Automates des occurrences DEV2 + complexité du prétraitement (= construction de l'automate) et de la recherche + ex en annexe + lien avec l'analyse lexicale
   - → Amélioration avec l'algorithme KMP (s'inspire de l'automate)

III/Les apports de la structure d'arbre

1) ABRO
   - → Définition ABR optimal
   - → Appli : Traduction ou le remplacer dans les rechercher/remplacer

2) Compression
   - → Problème : compresser sans perte de données
   - → Idée de l'encodage différentiel + ex
   - → Idée du codage de Huffman + arbre associé à un codage
   - → Définition d'un codage préfixe + son importance
   - → Algo de Huffman pour calculer un arbre minimisant le coût (= longueur) du codage + complexité

Références : Cormen [Cor] (principalement), Crochemore [Cro], Carton [Car] (développement)

Développements :
   - → CYK [Car p 199]
   - → Automate des occurrences [Cor p 915-921]

909. Langages rationnels et automates finis. Exemples et applications.

Plan :

Cadre : Prérequis = alphabet, mot, langage. Σ est un alphabet non vide fini.

I/Langages rationnels, langages reconnaissables

1) Langages rationnels

→ Définition union, concaténation, étoile de langages
→ Définition : Rat(Σ) est la plus petite famille stable par ces opérations + ex
→ Définition expression rationnelle + prop : non ambiguïté + ex
→ Prop : Lien entre expression rationnelle et langage rationnel

2) Automates finis

→ Définition d'un automate fini (ε-transitions autorisées)
→ Définition mot reconnu, langage reconnu + ex + définition de Rec(Σ)
→ Définition d'un automate complet / déterministe (pour lequel les ε-transitions ne sont plus autorisées) + prop : on peut compléter / déterminiser tout automate fini

3) Lien entre les deux

→ Prop : Si A et A' reconnaissent L et L', on peut construire un automate reconnaissant $LL4$, $L+L'$ et L^*
→ Lemme d'Arden
→ Théorème de Kleene : Rat(Σ) = Rec(Σ)
→ Appli : Décidabilité de Presburger DEV1

II/Propriétés et caractérisation des langages rationnels

1) Stabilité et décidabilité

→ Prop : Rec(Σ) est stable par union, concaténation, étoile, intersection, complémentaire (mais pas par sous langage car il existe des langages non rationnels)
→ Prop : Pour les langages rationnels L, on peut décider si $u \in L$, $L = \varnothing$, si $L = L'$, si L est infini ou pas...
→ Rq : De façon générale, c'est difficile de trouver un problème indécidable sur les langages rationnels

2) Résiduels

→ Définition d'un résiduel
→ Prop : $L \in$ Rat(Σ) ssi L admet un nombre fini de résiduels
→ Définition de l'automate des résiduels = minimal + ex
→ Appli : Automate des occurrences DEV2
→ Congruence de Nérode pour calculer l'automate minimal + algo de Hopcroft

3) Lemmes de l'étoile

→ Les trois lemmes de l'étoile + ex d'application
→ Réciproque du lemme de l'étoile (admis)

Références : Carton [Car], Cormen [Cor] (pour le développement)

Développements :

→ Automate des occurrences [Cor p 915-921]
→ Décidabilité de Presburger [Car p 178]

912. Fonctions récursives primitives et non primitives. Exemples et applications.

Plan :

But : Formaliser le concept de calculabilité pour une fonction, cad caractériser les fonctions pour lesquelles on peut trouver l'image de tout argument par un procédé systématique. Ici on considère les fonction de $\mathbb{N}^p$ dans $\mathbb{N}$

I/Fonctions primitives récursives

1) Construction

→ Définition des fonctions de base, schéma de composition et de récursion primitive
→ Définition des fonctions primitives récursives
→ Ex : +, ×, prédécesseur, signe, - ... [Car p 182]
→ Appli : Schéma de la boucle for : si le corps de la boucle est primitif récursif alors n tours de cette boucle aussi

2) Prédicats récursifs primitifs

→ Définition prédicat primitif récursif
→ Ex : m>n, égalité à zéro
→ Prop : Si P et Q sont primitifs récursifs, P^c, $P \vee Q$ et $P \wedge Q$ aussi
→ Ex : L'égalité et l'inégalité large sont des prédicats primitifs récursifs
→ Appli : Schéma if..then..else : Si la condition P_{if} est un prédicat primitif récursif et f_{then} et f_{else} sont primitives récursives alors le schéma if..then..else aussi

3) Minimisation bornée

→ Définition minimisation bornée + prop : la minimisation bornée par rapport à un prédicat primitif récursif est primitive récursive
→ Ex : La division euclidienne, le min et le modulo sont primitifs récursifs + m|n et le fait d'être premier sont des prédicats primitifs récursifs

4) Cela ne suffit pas

→ Prop : L'ensemble des fonctions primitives récursives est dénombrable
→ Csqc : Il y a des fonctions de $\mathbb{N}$ dans $\mathbb{N}$ qui ne sont pas primitives récursives
→ Ex : La fonction d'Ackermann n'est pas primitive récursive DEV1

II/Fonctions récursives

1) Construction

→ Définition prédicat sûr
→ Définition minimisation non bornée
→ Définition fonction récursive totale
→ Rq : Savoir si un prédicat est sûr est indécidable
→ Ex : $n \longmapsto$ le plus petit p premier tel que $p > n$ est récursive totale
→ Appli : Schéma de la boucle while : si la condition C d'arrêt est un prédicat sûr et h est récursive totale alors le schéma "tant que C faire h" est récursif total
→ Définition fonction récursive partielle + ex

2) Lien avec les machines de Turing

→ Définition fonction calculable par machine de Turing au sens strict
→ Prop : Récursive totale implique calculable au sens strict + réciproque vraie DEV2
→ Définition fonction calculable par machine de Turing au sens large
→ Prop : Récursive partielle équivaut à être Turing calculable au sens large
→ Rq : il existe des fonctions qui ne sont même pas récursives partielles (castor affairé)

Références : Wolper [Wol] (principalement), Cori-Lascar [CL2], Lassaigne-Rougemont [LdR]

Développements :

→ Fonction d'Ackermann [CL2 p 18]
→ Une fonction Turing calculable est primitive récursive [LdR p 148] + [Wol p 135]

913. Machines de Turing. Applications.

Plan :

I/Machines de Turing (= MT)
1) Formalisation
→ Définition (choix : un seul état initial, et une tête de lecture qui doit bouger à chaque étape)
→ Rq : Dessin pour illustrer ce que sont le ruban et la tête de lecture
→ Définition MT déterministe
→ Définition d'une configuration, étape de calcul, calcul, calcul acceptant
→ Définition d'un langage accepté / décidé par une MT

2) Extensions
→ Prop : Normalisation
→ Prop : Machines à ruban bi-infini, à plusieurs rubans = ça n'apporte rien de plus
→ Prop : On peut toujours déterminiser une MT
→ Enoncé de la thèse de Church-Turing

II/Calculabilité
1) R et RE
→ Rq : Un problème de décision est associé au langage des codages des instances positives
→ Définition de R, RE, co-RE + prop : RE $\neq \varnothing$
→ Définition d'un énumérateur + prop : $L \in RE$ ssi il existe un énumérateur pour L
→ Prop : Stabilités de R et RE

2) Décidabilité, indécidabilité
→ Ex : $L_\in = \{< M, w > | w \in L(M)\}$ est dans RE mais n'est pas décidable
→ Problème de l'arrêt
→ Définition fonction calculable
→ Définition d'une réduction (calculable) + prop : réduction et décidabilité
→ Théorème de Rice DEV1 + ex

III/Complexité en temps
1) Modèle
→ Définition temps de calcul + comparaison MT déterministes et non déterministes
→ Définition de TIME(f(n)) et NTIME(f(n)), de P, NP, EXP...
→ Ex de problèmes dans P et NP
→ Définition vérificateur en temps polynomial + prop : être dans NP équivaut à avoir un vérificateur polynomial

2) NP complétude
→ Définition réduction polynomiale
→ Définition NP-difficile, NP-complet
→ Théorème de Cook DEV2

Références : Carton [Car], Wolper [Wol]

Développements :
→ Théorème de Rice [Wol p 150]
→ Théorème de Cook [Car p 203]

914. Décidabilité et indécidabilité. Exemples.

Idée de défense de plan : 10ème problème de Hilbert : existe-t-il un algo qui permette de dire si une équation diophantienne admet une solution ou pas → Généralisation : si on a un énoncé mathématique, peut on *décider* (notion qu'on précise ici) s'il est vrai ou pas ? Dans la suite, on remarque que plus une structure est riche, moins on sait dire de choses dessus.

Plan :

I/Formalisme [Car p 139-160]

1) Machine de Turing (MT)

→ Définition MT, langage accepté, langage décidé
→ Définition de R et de RE
→ Prop : $R \subset RE$ + il existe des langages pas dans RE + stabilités de R et RE

2) Problème

→ Définition d'un problème, codage d'un problème, langage associé à un problème = langage des codages des instances positives du problème
→ Définition d'un problème décidable / indécidable + rq : ça ne dépend pas du codage
→ Définition fonction calculable + réduction
→ Prop : Si A se réduit à B et A est indécidable alors B aussi

II/Indécidabilité et MT

→ Prop : Indécidabilité problème de l'arrêt, problème du mot
→ Théorème de Rice DEV1
→ Appli : Langage vide est indécidable

III/Décidabilité et langages rationnels

→ Liste des problèmes décidables sur les langages rationnels : problème du mot, problème du langage vide, du langage universel, égalité de langage...
→ Rq : C'est difficile de trouver un problème indécidable sur les langages rationnels

IV/Décidabilité et indécidabilité : langages algébriques

→ Prop : Pour les langages algébriques, le problème du langage vide et du mot sont décidables
→ Def : Problème de correspondance de POST + POST modifié
→ Prop : Ils ont la même classe de décidabilité et POST modifié est indécidable
→ Appli : Le problème de l'intersection vide, du langage universel, de l'égalité et de l'ambiguïté sont indécidables sur les langages algébriques [Car]

V/Décidabilité et logique [DNR]

1) Vocabulaire

→ On se donne un langage du premier ordre et $\vdash$ un système de déduction correct et complet
→ Définition théorie, théorie consistante, théorie complète
→ Définition théorie récursive, théorie décidable
→ Prop : Une théorie complète, récursive sur un langage dénombrable est décidable

2) Presburger

→ Langage de l'arithmétique de Presburger
→ Axiomes de la théorie de Presburger
→ Décidabilité de Presburger DEV2

3) Peano

→ Langage et axiomes de Peano (on rajoute la multiplication à Presburger=
→ Théorèmes d'indécidabilité autour de Peano [DNR p 126-127]

Références : Carton [Car], Wolper [Wol], Autebert [Aut], [DNR]

Développements :

→ Théorème de Rice [Wol p 150]
→ Décidabilité de Presburger [Car p 178]

915. Classes de complexité. Exemples.

Plan :

Intro : S'intéresser à la décidabilité d'un problème ne suffit pas : en pratique un algo n'est utilisable que lorsqu'il s'exécute en temps raisonnable et utilise un espace limité. Rq : On peut passer d'un problème de décision à un problème d'optimisation et inversement : on s'autorise donc à utiliser le terme "problème" pour les deux. A tout problème de décision on associe le langage des instances positives du problème. Cette association permet de confondre problème et langage

I/Machines de Turing et complexité

1) Définitions

→ Rappel : Langage décidé par MT
→ Définition temps / espace de calcul d'une MT M sur ω
→ Définition complexité en temps / espace de M + complexité en temps (n) $\geqslant$ n
→ Rq : Parfois on ne considère pas l'espce pris par l'entrée dans la complexité en espace
→ Prop : Lien entre les deux complexités [Car p 194]

2) Influence du modèle sur la complexité temporelle

→ Théorème d'accélération = permet de négliger les constantes donc d'exprimer la complexité temporelle avec des O
→ Prop : Complexité en temps : influence d'un ruban bi-infini / plusieurs bandes

3) Influence du modèle sur la complexité spatiale

→ Prop : Complexité spatiale et ruban bi-infini / plusieurs bandes
→ Théorème de Savitch

II/Classes de complexité en temps

1) Définitions

→ Définition de TIME(f(n)) et NTIME(f(n))
→ Définition de P, NP, EXP, NEXP
→ Prop : Les diverses inclusions entre ces classe + attention il y en a plein pour lesquelles on ne sait pas si c'est strict ou pas

2) Exemples de problèmes dans P

→ Ex : Problème d'accessibilité dans un graphe (d'où, décider si le langage reconnu par un automate est vide)
→ Ex : Tout langage algébrique est dans P (CYK)
→ Ex : 2-color et 2-SAT sont dans P

3) Exemples de problèmes dans NP

→ Définition d'un vérificateur en temps polynomial
→ Prop : Appartenir à NP équivaut à avoir un vérificateur polynomial
→ Ex : PVC et cycle hamiltonien sont dans NP

4) NP-complétude

→ Définition réduction polynomiale + dessin + prop
→ Définition NP-difficile, NP-complet
→ Ex : SAT, 3SAT, PVC, 3-color, Cycle hamiltonien sont NP-complets
→ Théorème de Cook DEV2
→ Confronté à un problème NP-complet, on peut trouver un algo polynomial qui résout un sous problème de celui qu'on considère ou trouver une approximation polynomiale + ex PVCE DEV2

On peut aussi parler de complexité spatiale

Références : Carton [Car], Wolper [Wol], Cormen [Cor]

Développements :

→ Théorème de Cook [Car p 203]
→ Problème du voyageur de commerce euclidien [Cor p 1023]

916. Formules du calcul propositionnel : représentation, formes normales, satisfiabilité. Exemples.

Plan :

But : Formaliser les raisonnements mathématiques, encodage de problèmes (coloriage, emploi du temps)

I/Syntaxe

→ On se donne un ensemble dénombrables de variables + les symboles $(,)\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$
→ Définition des formules propositionnelles par induction + ex
→ Représentation des formules par un arbre ou un circuit électrique
→ Théorème de lecture unique d'où non ambiguïté
→ Rq : On peut enlever les parenthèses superflues + ex

II/Sémantique

1) Valuations

→ Définition valuation + extension aux formules + tables de vérité (exemple en annexe)
→ Définition formules équivalentes + ex $(p \Rightarrow q) \equiv (\neg q \Rightarrow \neg p)$
→ Appli : Système complet de connecteurs + ex $\{\wedge, \neg\}$
→ Définition de la satisfiablité

2) Formes normales

→ Définition littéral et clause + ex
→ Définition FNC, FND + ex
→ Prop : Toute formule est équivalente à une FND ou une FNC
→ Rq : Pas d'unicité pour la FNC + expliquer comment on opère la transformation + complexité
→ Prop : Transformation de Tseitin DEV1

III/Satisfiabilité

1) Compacité

→ Définition : formule valide, tautologie, contradiction, finiment satisfiable
→ Théorème de compacité pour la résolution en calcul propositionnel
→ Appli : k-coloriage (un graphe est 3-coloriable ssi tout sous graphe fini l'est)
→ On peut développer ce système de déduction

2) SAT

→ Le problème SAT + théorème de Cook DEV2
→ Un algo naïf pour SAT, mentionner les SAT solveurs
→ Rq : Si on a une FND, SAT se fait en temps linéaire ; si on a une FNC, c'est facile de vérifier la validité de la formule
→ Applis : réductions : choisir celle qu'on veut
→ Rq : 2SAT et Horn-SAT sont eux dans P

Références : Wolper [Wol], Carton [Car], Cori-Lascar [CL1], Lassaigne-Rougemont [LdR]

Développements :

→ Théorème de Cook [Wol p 185] ou [Car p 203]
→ Transformation de Tseitin [Car p 204]

918. Systèmes formels de preuve en logique du premier ordre. Exemples.

Plan :

But : Formaliser les preuves. On aimerait que cela permette de les automatiser.

I/Logique du premier ordre

1) Syntaxe

→ On se donne $\{v_i\}_i, \{c_i\}_i, \{f_i\}_i, \{p_i\}_i$ des ensembles dénombrables de variables / constantes / symboles de fonctions / symboles de prédicats
→ définition terme, formule atomique, formule + exs
→ Définition occurrences quantifiées d'une variable / variable libre + ex
→ Définition + notation d'une substitution

2) Sémantique

→ Définition structure + définition inductive de la valeur d'un terme dans une structure munie d'une valuation
→ Définition modèle + formule satisfiable ou valide

3) Théories et modèles

→ Définition d'une théorie, modèle d'une théorie + ex : Presburger avec $\mathbb{N}$
→ Définition F est conséquence sémantique d'une théorie T

4) Système de déduction

→ Définition d'un séquent + ex
→ Définition d'une règle de déduction + ex : modus ponens
→ Définition d'un système de déduction
→ Définition d'un séquent / formule prouvable
→ Définition théorie cohérente / complète

II/Déduction naturelle

→ Donner toutes les règles de déduction

2) Propriétés

→ Prop : La déduction naturelle est correcte ($T \vdash F \Rightarrow T \models F$)
→ Définition théorie de Henkin + prop : une théorie de Henkin cohérente et complète admet un modèle
→ Prop : Complétion d'une théorie cohérente en une théorie de Henkin + csqc : toute théorie cohérente admet un modèle
→ Prop : Complétude de la déduction naturelle ($T \models F \Rightarrow T \vdash F$)
→ Rq : Le calcul des séquents permet de symétriser les règles de la déduction naturelle

III/Résolution

1) Clauses

→ Définition d'une clause
→ Prop : Comment transformer une formules en clause + ex

2) Unification

→ Définition unificateur, unificateur principal + ex
→ Prop : Existence d'un unificateur principal dès qu'il existe un unificateur
→ Algo d'unification DEV1 + rq sur la complexité

3) Résolution

→ Règles de la résolution
→ Définition preuve par résolution / par réfutation
→ Prop : Correction de la résolution
→ Définition modèle de Herbrand
→ Prop : Complétude de la résolution DEV2 + appli : compacité de la résolution

Références : [DNR], Lassaigne-Rougemont [LdR]

Développements :

→ Complétude de la résolution [LdR p 102]
→ Etude d'un algorithme d'unification [DNR p 248]

919. Unification : algorithmes et applications.

Plan :

I/Syntaxe et unification
1) Termes et substitutions
→ Définition d'un langage du premier ordre
→ Définition terme + ex
→ Définition substitution + notation + résultat d'une substitution appliquée à un terme + ex
2) Unificateurs
→ Définition termes unifiables, unificateur + ex
→ Extension : unificateur d'un ensemble fini d'équations
→ Définition unificateur principal
→ Prop : Si on a un unificateur, on a un unificateur principal

II/Algorithmes
1) Historique
→ Algorithme de Robinson
→ Prop : Il est non déterministe, exponentiel en temps et en place
2) Algo classique
→ Algo de Martelli-Montanari (code)
→ Prop : Il est correct et termine DEV1
→ Ex sur lequel il est exponentiel en temps et en place [DNR]
3) Améliorations
→ Première idée : Transformer les termes en graphes
→ Algo utilisant les graphes + ex + il est linéaire en place mais peut rester exponentiel en temps
→ Deuxième idée : Amélioration de l'algo précédent pour obtenir un temps quadratique
→ Rq : Avec Union-find, on peut avoir un algo pseudo-linéaire en temps

III/Application à la résolution
1) Clauses
→ Définition formule atomique + ex
→ Définition d'une clause
→ Prop : Transformation d'une formule en clauses + ex
2) Résolution
→ Règles de la résolution
→ Définition d'une preuve par résolution / par réfutation
→ Prop : Correction de la résolution
→ Définition d'un modèle de Herbrand
→ Prop :Complétude de la résolution DEV2

IV/Application à la réécriture
→ Définition de R un système de réécriture noté $\rightarrow$
→ Définition + notation de "u et v sont joignables"
→ Définition d'un système terminant, confluent, localement confluent
→ Définition de la position d'un sous terme + substitution d'un sous terme
→ Définition paire critique + ex en annexe
→ Lemmes des paires critiques + R est localement confluent ssi ses paires critiques sont joignables
→ Lemme de Newman
→ Th : Si R est terminant et a un nombre fini de règles, il est confluent ssi ses paires critiques sont joignables. On peut donc décider la confluence d'un tel système

Références : [DNR], Lassaigne-Rougemont [LdR], Baader [Baa]

Développements :

→ Complétude de la résolution [LdR p 102]
→ Etude d'un algorithme d'unification [DNR p 248]

920. Réécriture et formes normales.

IMPASSE

921. Algorithmes de recherche et structures de données associées.

Plan :

Intro : On cherche quelles structures utiliser pour manipuler un ensemble E pour lequel on souhaite rechercher, supprimer et insérer des éléments. Dans la suite, on considère que les clés sont deux à deux distinctes.

I/Recherche dans une structure "séquentielle"

→ Recherche dans une liste non triée + complexité des 3 opérations
→ Recherche dans un tableau trié (dichotomie) + complexité des 3 opérations
→ Cas particulier d'une liste de caractère = recherche de motif dans un texte : on peut le faire grâce à l'automate des occurrences DEV1

II/Tables de hachage [Cor]
1) Généralités

→ Définition univers et clés
→ Adressage direct + problème de mémoire si l'univers est grand
→ Principe du hachage + def : fonction de hachage
→ Définition d'une collision + gestion des collisions par chaînage
→ Définition d'un hachage uniforme + coûts des 3 opérations

2) Faire un bon hachage

→ Pb : Le pire cas d'une recherche peut être en $O(n)$ et en plus un utilisateur malveillant peut consciemment remplir une alvéole pour que ce pire cas arrive
→ Définition classe universelle de fonctions de hachage + ex
→ Prop : Hachage parfait DEV2

III/Structures arborescentes
1) ABR

→ Définition arbre binaire de recherche + prop : hauteur d'un tel arbre en fonction du nombre de noeuds
→ Algos pour la recherche, insertion, supprimer + complexités
→ Rq : Toutes ces complexités sont en O(hauteur de l'arbre) : on a un problème si l'arbre est en peigne

2) AVL

→ Définition d'un AVL
→ Rq : Les algos de recherche / insertion / suppression sont les mêmes qu'avec les ABR
→ Définition rotation à gauche et à droite + dessins en annexe
→ Complexité des 3 opérations incluant le rééquilibrage

3) On peut aussi parler des ABR optimaux

Conclusion : Tableau comparatif des structures / complexités des opérations [FG p 298]

Références : Cormen [Cor], Froidevaux-Gaudel [FG]

Développements :

→ Automate des occurrences [Cor p 915-921]
→ Hachage parfait [Cor p 258]

923. Analyse lexicale et syntaxique. Applications.

Plan :

Prérequis : Notions sur les automates finis

I/Analyse lexicale
1) Outils
- → Déf : C'est un procédé qui transforme une suite de caractère en une suite de mots
- → Définition unité lexicale, motif (représenté par une expression rationnelle), lexème + ex

2) Reconnaissance des lexèmes
- → Théorème de Kleene : d'où l'utilisation d'automates pour reconnaître des motifs
- → Prop : Tout automate fini est équivalent à un automate déterministe
- → Prop : Existence et construction de l'automate minimal (= des résiduels)
- → Automates des occurrences DEV1

3) Utilisation des automates
- → On créé un automate minimal pour chacun des motif puis on les lit simultanément jusqu'à ce qu'on ne puisse plus avancer dans aucun des automates. Juste avant cette étape il y avait donc au moins un des automates qui n'était pas bloqué. On prend le plus long préfixe du mot qui aboutissait dans un état final de cet automate pour savoir quel est le motif. Si il y a une égalité, on impose des priorités (de façon par exemple à ce que "12" devienne <nb> et non <chiffre><chiffre>)

II/Analyse syntaxique
But : Construire l'arbre représentant la structure du texte résultant de l'analyse lexicale
1) Grammaire, analyseur universel
- → Définition grammaire algébrique, dérivation, langage engendré + ex
- → Ici notre but est de savoir si la chaîne de noms d'unités lexicales peut être produite par la grammaire du langage source et si oui, de construire un arbre syntaxique
- → Définition grammaire de Chomsky
- → Algorithme CYK DEV2

2) Analyse descendante : cas de LL(1)
- → Définition de premier et suivant + algos
- → Table prédictive + exemple d'analyse descendante (de l'axiome vers le texte) sur grammaire LL(1)
- → Rq : Expliquer la dénomination LL(1)

3) Analyse ascendante LR(0)
- → Définition d'un item, règle de réduction
- → Automate des items pour analyse descendante (du texte vers l'axiome) sur grammaire LR(0)

Références : Carton [Car], Schwarzentruber [Sch], Cormen [Cor] (pour développement)

Développements :
- → Automate des occurrences [Cor p 915-921]
- → CYK [Car p 199]

924. Théories et modèles en logique du premier ordre. Exemples.

Idée de défense de plan : Une théorie est un ensemble de formules : on aimerait savoir ce qu'on a défini avec, quelles structures → modèles

Plan :

I/Syntaxe et sémantique en logique du premier ordre

1) Syntaxe

→ Définition langage du premier ordre + ex : langage de la théorie des groupes

→ Définition terme (clos), formule atomique, formule + ex : quelques formules de la théorie des groupes

2) Interprétation

→ Définition structure, valuation, valeur d'une formule

→ Définition formule satisfiable, formule valide

3) Théories

→ Définition d'une théorie + ex : théorie des groupes

→ Définition modèle d'une théorie + ex pour la théorie des groupes

→ Définition formule T-valide, T-satisfiable +ex (toujours en théorie des groupes)

→ Définition théorie consistante, cohérente, contradictoire (avec $\vdash$ = la déduction naturelle)

→ Définition théorie de Henkin + prop : une théorie de Henkin cohérente et complète admet un modèle + complétion d'un théorie en une théorie de Henkin + csqc : toute théorie cohérente admet un modèle

II/Modèles de Herbrand et résolution

1) Clause

→ Définition clause

→ Prop : Transformation d'une formules en ensemble de clauses

2) Unification

→ Définition unificateur + algo d'unification

→ Règles de la résolution + définition d'une preuve par résolution / réfutation

3) Complétude de la résolution

→ Définition modèle de Herbrand

→ Prop : On a un modèle ssi on a un modèle de Herbrand

→ Appli 1 : Une version faible de Lowenheim-Skolem

→ Appli 2 : Complétude de la résolution DEV1

III/Théories récursives et décidables

→ Définition des deux notions

→ Recopier en gros le V de la leçon 914 + décidabilité de Presburger DEV2

Références : Carton [Car], [DNR], Lassaigne-Rougemont [LdR]

Développements :

→ Décidabilité de Presburger [Car p 178]

→ Complétude de la résolution [LdR p 102]

925. Graphes : représentation et algorithmes.

Plan :

I/Définitions

→ Définition graphe orienté, non orienté, pondéré
→ Définition sommet adjacent, chemin entre deux sommets, cycle, coût dans le cas pondéré
→ Définition : cas particulier des arbres + caractérisations
→ Représentations : matrice d'adjacence, liste d'adjacence, liste d'arêtes + comparaison coût mémoire et coûts des opérations classiques

II/Parcours

→ Parcours en largeur : algo + complexité
→ Appli : Prim, Dijkstra (même principe)
→ Parcours en profondeur : algo + complexité
→ Appli : Calcul des composantes fortement connexes
→ Appli des deux parcours : résoudre le problème d'accessibilité

III/Arbres couvrants minimaux

→ définition d'un ACM
→ Algo de Prim
→ Algo de Kruskal

IV/Plus courts chemins

→ A origine unique : Bellman-Ford DEV1
→ A originie unique : Dijkstra
→ Entre tout couple de sommets : Floyd-Warshall + appli : calcul de la fermeture transitive d'un graphe

V/Un peu de NP-complétude

→ Le problème k-color : il est NP-complet
→ Définition d'un cycle hamiltonien
→ Le problème du voyageur de commerce et du cycle hamiltonien sont NP-complets
→ Mais on peut quand même faire des choses : PVCE DEV2

Références : Cormen [Cor], Froidevaux-Gaudel [FG], Garey-Johnson [GJ]

Développements :

→ Bellman-Ford [Cor p 602-605]
→ Problème du voyageur de commerce euclidien [Cor p 1023]

926. Analyse des algorithmes, complexité. Exemples.

Plan :

Intro : Quand on a plusieurs algos corrects pour répondre à un problème, on souhaite les comparer. Une façon de faire est de s'intéresser à leurs complexités temporelles / spatiales

I/Généralités [FG p 5-27]

1) Notion de complexité

→ "Définition" d'un algorithme

→ Définition complexité spatiale (on ne compte pas la place prise par l'entrée) + rq : on pourrait formaliser cette notion avec les machines de Turing

→ Définition complexité temporelle = fonction c qui à une entrée x associe $c(x)$ le nombre d'opérations élémentaires effectuées sur l'entrée x + ex recherche de x dans la liste l : $c(x,l)$ = position de x dans l si $x \in l$ et $|l|$ sinon

→ Définition complexité pire cas $c_{max}(n) = max\{c(x)|x \text{ est de taille } n\}$ et de la complexité moyenne c_{moy} + prop $c_{moy} \leqslant c_{max}$ + ex recherche dans une liste $c_{moy}(n) = \frac{n}{2}$

→ Rq : Souvent on considère que les entrées sont équiprobables mais c'est souvent faux et ça peut influer

2) Mesurer la complexité

→ Rq : Un calcul exact est inutile car dépend de la machine, de l'implémentation

→ Notation O, Ω, Θ + ex

II/Quelques méthodes d'analyse de complexité

1) Méthodes directes

→ Algos itératifs = facile + ex CYK DEV1

→ Algo récursifs = plus compliqué + ex complexité du tri rapide randomisé DEV2

2) Relations de récurrence sur la complexité

→ Théorème fondamental de DPR + ex avec le tri fusion (ou Strassen)

→ Attention il ne couvre pas tous les cas

→ Utilisation de séries génératrices quand on a une relation de récurrence qui s'y prête + ex complexité moyenne du tri rapide [FG p 519]

3) Analyse amortie

→ Définition : La complexité amortie de l'opération o est le coût moyen de o dans une séquence de n opérations o

→ Ex : Méthode de l'agrégat + ex : compteur binaire [Cor] + ex ajout dans une table dynamique

→ Rq : Il existe d'autres méthodes : méthode comptable, potentiels...

III/Comment améliorer la complexité

→ Changer de structure de données pour modifier le coût des opérations élémentaires + ex : matrice d'adjacence VS liste d'adjacence + ex Dijkstra, la complexité est différente si on utilise un tas de Fibonacci plutôt qu'un tas

→ Compromis temps/espace : on peut essayer d'augmenter la complexité spatiale pour diminuer la temporelle (et inversement) + ex : mémoïsation en général + rq : parfois, on peut même gagner sur les deux plans, exemple de Fibonacci en conservant uniquement les deux dernières valeurs (linéaire en temps, constant en espace)

→ Parfois, on a une borne min pour la complexité d'un algo, par exemple pour les tris par comparaison (tous en $\Omega(n)$) + rq : on peut quand même essayer d'améliorer les constantes des algorithmes optimaux (tri rapide > tri fusion)

Références : Cormen [Cor], Froidevaux-Gaudel [FG], Carton [Car] (pour CYK)

Développements :

→ CYK [Car p 199]

→ Complexité du tri rapide [Cor p 166-170]

927. Exemples de preuves d'algorithme. Correction, terminaison.

Plan :

Intro : On présente des méthodes permettant de montrer qu'un algorithme termine et est correct relativement à sa spécification + définition spécification (= une prop P_1 sur l'entrée et une prop P_2 sur la sortie) + ex avec un tri

I/Terminaison

1) Indécidabilité
   - → Définition : Un algo termine s'il s'exécute en un nombre fini d'étapes élémentaires sur toute entrée conforme à sa spécification
   - → Ex : fonction de Syracuse
   - → Prop : Indécidabilité du problème de l'arrêt

2) Ensemble bien fondé
   - → Définition ensemble bien fondé + ex ordre lexicographique [Alb]

3) Algos itératifs
   - → Définition d'un variant de boucle
   - → Prop : Si une boucle possède un variant, elle termine
   - → Ex : Dans une boucle for, (indice final) - (indice initial) est un variant
   - → Ex : Un algo des graphes

4) Algos récursifs
   - → Prop : Soit f une fonction récursive sur A et $\varphi : A \to E$ = ensemble bien fondé. Soit $B = \varphi^{-1}\{\text{éléments minimaux de } \varphi(A)\}$. Si $\forall b \in B$, $f(b)$ termine et si pour tout e, n'apparaissent qu'un nombre fini de $f(y)$ et tous tels que $\varphi(y) < \varphi(x)$ alors f termine pour tout élément de A
   - → Ex : Factorielle termine
   - → Ex : Ackermann termine (ordre lexicographique)

II/Correction

1) Définitions
   - → Définition correction partielle / totale
   - → Ex : Syracuse est partiellement correct + la non primalité de Fermat est partiellement correcte

2) Algos itératifs
   - → Définition d'un invariant de boucle
   - → Ex : Euclide et tri insertion sont corrects
   - → Ex : Le tri par tas est totalement correct DEV1
   - → EX : Unification est totalement correct DEV2

3) Algos récursifs
   - → Définition induction bien fondée
   - → Ex : Tri fusion, factorielle, calcul de la hauteur d'un arbre

Références : Cormen [Cor], [DNR], Wolper [Wol] (pour le pb de l'arrêt), Albert [Alb]

Développements :
- → Etude d'un algorithme d'unification [DNR p 248]
- → Correction du tri par tas [Cor p 142-148]

928. Problèmes NP-complets : exemples et réduction.

Plan :

Prérequis : Notions sur les machines de Turing

I/La classe NP [Car]

1) Complexité

→ Définition classe P, classe NP
→ Prop $P \subset NP$ + on ne sait pas si $P = NP$
→ Définition vérificateur en temps polynomial
→ Prop : Etre dans NP équivaut à avoir un vérificateur polynomial

2) NP-complétude

→ Définition réduction polynomiale
→ Définiton problème NP-dur, NP-complet
→ Prop : Si A est NP-complet et se réduit à B alors B est Np-complet

3) Complexité et décision

→ Définition problème de décision + son langage associé = {instances positives}
→ Définition problème d'optimisation
→ Lien entre les deux + ex + le problème d'optimisation est dit NP-complet quand le problème de décision associé l'est

II/Autour de la satisfiabilité [Car]

→ Le problème SAT + théorème de Cook DEV1
→ Problèmes CNF-SAT et 3SAT : ils ont NP complets car SAT $\leqslant$ CNF-SAT et CNF-SAT $\leqslant$ 3SAT
→ Les problèmes 2SAT, HORN-SAT et CND-SAT sont eux dans P

III/Problème du voyageur de commerce

→ Prop : Le problème du chemin hamiltonien est NP-complet (que le graphe soit orienté ou pas) car CNF-SAT $\leqslant$ chemin hamiltonien
→ Prop : Celui du cycle hamiltonien aussi
→ Prop : Le problème du voyageur de commerce est NP-complet car cycle hamiltonien $\leqslant$ PVC + rq sur la version optimisation du PVC
→ Déns le cadre euclidien : PVCE DEV2

IV/Programmation linéaire en nombres entiers [GJ]

→ Prop : PLNE est NP-complet car CNF-SAT $\leqslant$ PLNE
→ La relaxation de ce problème est dans P (admis)

Références : Carton [Car], Cormen [Cor], Garey-Johnson [GJ], Sipser [Sip] (pour certaines réductions)

Développements :

→ Théorème de Cook [Car p 203]
→ Problème du voyageur de commerce euclidien [Cor p 1023]

Bibliographie

[Alb] Albert, *Cours et exercices d'informatique*
[Aut] Autebert, *Théorie des langages et des automates*
[Baa] Baader, *Term rewriting and all that*
[BBC] Beauquier, Berstelle et Chrétienne, *Elements d'algorithmique*
[Car] Carton, *Langages formels*
[CL1] Cori et Lascar, *Logique mathématique, Tome 1*
[CL2] Cori et Lascar, *Logique mathématique, Tome 2*
[Cor] Cormen et alii, *Algorithmique*
[Cro] Crochemore, *Algorithmique du texte*
[Das] Dasgupta, Papadimitriou et Vazirani, *Algorithms*
[DNR] David, Nour et Raffalli, *Introduction à la logique*
[FG] Froidevaux et Gaudel, *Types de données et algorithmes*
[GJ] Garey et Johnson, *Computers and intractability : A guide to the theory of NP-completness*
[LdR] Lassaigne et de Rougemont, *Logique et fondements de l'informatique*
[Sch] Schwarzentruber, *Compilation : analyse lexicale et syntaxique*
[Sip] Spiser, *Introduction to the theory of computation*
[Wol] Wolper, *Introduction à la calculabilité*