Colle d'informatique 2

Programme

Grammaires hors contextes

- Vocabulaire : symnboles terminaux $\mathcal{V}_T$, non terminaux $\mathcal{V}_N$, initial (axiome) S, règles de production $\mathcal{R}$ avec la notation $\rightarrow$. Les notations ne sont pas figées.
- Dérivation immédiate $\Rightarrow$, dérivation $\Rightarrow^*$.
- Langage engendré par une grammaire, langage non contextuel.
- Arbre d'analyse (ou de dérivation), dérivation à gauche, à droite.
- Ambiguïté d'une grammaire.

Décidabilité

- Modèle de calcul (pas de machine de Turing mais un programme (=algorithme) C ou OCaml et une mémoire illimitée).
- Problème de l'arrêt. Notion d'indécidabilité.
- Problème de décision. Fonction calculable. Problème décidable.

Graphes

- Révisions de première année : définitions, implémentations, parcours, etc.
- Composantes fortement connexes, lien avec 2SAT vu en TP.
- Arbre couvrant de poids minimum par l'algorithme de Kruskal.

Commentaires

- Les formes normales de Chomsky (CNF) ont été présentées en travaux dirigés. Vous pouvez en parler si vous le souhaitez.
- Le programme des deux semaines à venir ne contient pas la partie sur les classes de complexité ; elle sera ajoutée au programme la semaine avant Noël.
- Le chapitre sur la décidabilité a été fait mais peu d'exercices ont été traités. Vous pouvez poser des questions d'application directe.
- L'algorithme A* n'a pas encore été vu.
- Vous pouvez proposer des sujets qui comportent d'autres structures de données : arbres, union-find (vu rapidement).

Extraits du programme officiel

Grammaires non contextuelles

Les grammaires formelles ont pour principal intérêt de définir des syntaxes structurées, en particulier celles des langages informatiques (langage de programmation, langage de requête, langage de balisage, etc.). On s'intéresse surtout à la manière dont les mots s'obtiennent par la grammaire et, de façon modeste, à la manière d'analyser un mot (un programme) en une structure de données qui le représente.

Notions	**Commentaires**
Grammaire non contextuelle. Vocabulaire : symbole initial, symbole non-terminal, symbole terminal, règle de production, dérivation immédiate, dérivation. Langage engendré par une grammaire, langage non contextuel. Non contextualité des langages réguliers.	Notations : règle de production $\rightarrow$, dérivation immédiate $\Rightarrow$, dérivation $\Rightarrow^*$. On montre comment définir une expression arithmétique ou une formule de la logique propositionnelle par une grammaire. On peut présenter comme exemple un mini-langage fictif de programmation ou un mini-langage de balisage. Sont hors programme : les automates à pile, les grammaires syntagmatiques générales, la hiérarchie de Chomsky.
Arbre d'analyse. Dérivation à gauche, à droite. Ambiguïté d'une grammaire. Équivalence faible.	On présente le problème du « sinon pendant » (*dangling else*).
Exemple d'algorithme d'analyse syntaxique.	On peut présenter au tableau un algorithme *ad hoc* d'analyse syntaxique par descente récursive (algorithme *top-down*) pour un langage de balisage fictif (par exemple, la grammaire de symbole initial S et de règles de production $S \rightarrow TS|c, T \rightarrow aSb$ sur l'alphabet $\{a, b, c\}$). On ne parle pas d'analyseur LL ou LR. On ne présente pas de théorie générale de l'analyse syntaxique.
Mise en œuvre	
On étudie surtout de petits exemples que l'on peut traiter à la main et qui modélisent des situations rencontrées couramment en informatique. On fait le lien avec la définition par induction de certaines structures de données (listes, arbres, formules de logique propositionnelle).	

Décidabilité et classes de complexité

On s'intéresse à la question de savoir ce qu'un algorithme peut ou ne peut pas faire, inconditionnellement ou sous condition de ressources en temps. Cette partie permet de justifier la construction, plus haut, d'algorithmes exhaustifs, approchés, probabilistes, etc. On s'appuie sur une compréhension pratique de ce qu'est un algorithme.

Notions	**Commentaires**
Problème de décision. Taille d'une instance. Complexité en ordre de grandeur en fonction de la taille d'une instance. Opération élémentaire. Complexité en temps d'un algorithme. Classe **P**.	Les opérations élémentaires sont les lectures et écritures en mémoire, les opérations arithmétiques, etc. La notion de machine de Turing est hors programme. On s'en tient à une présentation intuitive du modèle de calcul (code exécuté avec une machine à mémoire infinie). On insiste sur le fait que la classe **P** concerne des problèmes de décision.
Réduction polynomiale d'un problème de décision à un autre problème de décision.	On se limite à quelques exemples élémentaires.
Certificat. Classe **NP** comme la classe des problèmes que l'on peut vérifier en temps polynomial. Inclusion $\mathbf{P} \subseteq \mathbf{NP}$.	Les modèles de calcul non-déterministes sont hors programme.
NP-complétude. Théorème de Cook-Levin (admis) : SAT est NP-complet.	On présente des exemples de réduction de problèmes NP-complets à partir de SAT. La connaissance d'un catalogue de problèmes NP-complets n'est pas un objectif du programme.
Transformation d'un problème d'optimisation en un problème de décision à l'aide d'un seuil.	
Notion de machine universelle. Problème de l'arrêt.	
Mise en œuvre	

On prend soin de distinguer la notion de complexité d'un algorithme de la notion de classe de complexité d'un problème. Le modèle de calcul est une machine à mémoire infinie qui exécute un programme rédigé en OCaml ou en C. La maîtrise ou la technicité dans des formalismes avancés n'est pas un objectif du programme.

Algorithmique des graphes

Notions	**Commentaires**
Accessibilité. Tri topologique d'un graphe orienté acyclique à partir de parcours en profondeur. Recherche des composantes connexes d'un graphe non orienté.	On fait le lien entre accessibilité dans un graphe orienté acyclique et ordre.
Recherche des composantes fortement connexes d'un graphe orienté par l'algorithme de Kosaraju.	On fait le lien entre composantes fortement connexes et le problème 2-SAT.
Recherche d'un arbre couvrant de poids minimum par l'algorithme de Kruskal.	On peut mentionner l'adaptation au problème du chemin le plus large dans un graphe non-orienté.
Mise en œuvre	
Une attention particulière est portée sur le choix judicieux du mode de représentation d'un graphe en fonction de l'application et du problème considéré. On étudie en conséquence l'impact de la représentation sur la conception d'un algorithme et sur sa complexité (en temps et en espace). On se concentre sur l'approfondissement des algorithmes cités dans le programme et le ré-emploi de leurs idées afin de résoudre des problèmes similaires. La connaissance d'une bibliothèque d'algorithmes fonctionnant sur des principes différents mais résolvant un même problème n'est pas un objectif du programme.	