Projets algorithmique et complexité
M1 informatique

2024

Contents

1 Introduction

Le rapport écrit doit être réalisé en Latex (via des outils comme texlive, MikTeX ou www.overleaf.com). Ce texte est rédigé en Latex. Le fichier de style Latex et le fichier bib (pour la bibliographie) sont disponibles sur plubel, ainsi qu'un fichier d'exemple.

Le rapport doit être écrit en français, et long de 10 à 12 pages (sans compter les sources de votre programme, la table des matières, etc). N'imprimez pas votre rapport sur papier, le PDF produit suffit.

Il doit comprendre :

1. une présentation du projet,
2. une explication du ou des algorithmes utilisés (algorigramme et pseudo-code devront être présents),
3. une analyse de la complexité (si elle existe par ailleurs, vous pouvez vous y référer),
4. les détails pertinents de votre implémentation (points durs, organisation, structures de données),
5. des résultats et leurs analyses (exactitude, stabilité, régularité, évolution itérations/temps...). Vous devez afficher les courbes de temps/itérations, une sortie de votre programme OCaml en fichier *gnuplot* ou *csv* est pratique pour l'insertion à la volée dans Latex.
6. une conclusion qui établit (ou non) en justifiant le lien entre votre implémentation et l'analyse théorique de l'algorithme.

Rédigez votre rapport avec soin. La langue naturelle est le premier langage de programmation et de modélisation, aussi un rapport confus et criblé d'erreurs d'orthographe ou de français augure-t-il mal de votre talent d'informaticien. Le programme `ispell` peut vérifier l'orthographe, comme LanguageTool (lien).

Donnez les références bibliographiques, les adresses des sites web que vous avez utilisés. Pensez à commenter vos sources, sans excès. Il n'est pas imposé d'utiliser `ocamldoc` (similaire à `javadoc`).

Vous créerez une archive de votre projet au format ZIP, contenant les sources, le `makefile`, les fichiers d'exemples, votre rapport, et éventuellement le fichier de votre présentation orale (vous êtes libre sur le format du support de votre présentation).

L'archive doit contenir les fichiers supports de votre présentation orale. Par contre, prévoyez un fichier `LISEZMOI.md` (au format Markdown), un `makefile`, et une commande `"make clean"`, `"make all"`, et éventuellement quelques programmes de démonstration, que vous appelerez `demo1`, `demo2`, etc.

2 Pourquoi Ocaml est il imposé ?

- une fois que vous maîtriserez Ocaml, vous apprendrez facilement tous les autres langages de programmation.
- si cela peut vous rassurer, rien ne vous interdit de programmer d'abord votre projet avec votre langage favori.

- vous pouvez aussi utiliser le style procédural, ou orienté objet, en Ocaml. Vous pouvez utiliser des "références" pour programmer avec des affectations.
- vous maîtrisez (ou vous devriez maîtriser) le C, C++, Java, ou d'autres langages procéduraux. C'est donc l'occasion d'apprendre un langage vraiment différent, et une façon différente de penser et de programmer.
- une fois que vous saurez faire, programmer en Ocaml est plus rapide, plus sûr (un programme Ocaml qui compile a plus de chances de fonctionner qu'un programme C++ qui compile...), plus simple (vous n'avez pas à gérer la mémoire avec des `delete` plus ou moins hasardeux), beaucoup plus concis qu'en Java ou C++ ; d'ailleurs les prouveurs actuels comme Coq sont développés en Ocaml, ainsi que des programmes critiques.
- il y a un seul compilateur de Ocaml, celui de l'INRIA (donc pas de problème d'incompatibilité).

Attention : il est inutile de prévoir une sortie graphique de votre programme, ce n'est pas l'objet de ces projets. La bibliothèque `Graphics` a un support incomplet, on privilégiera une sortie texte intégrable dans Latex, comme gnuplot, csv (l'import et la mise en forme de tableau sont possibles facilement lien), tex ou pstTricks.

3 La notation du projet

Un projet dont le programme ne fonctionne pas ne peut pas obtenir la moyenne. Un projet dont le rapport est mauvais ne peut pas obtenir la moyenne. En cas de plagiat, la note sera de 0 pour le ou les groupes.

4 Eternity II. 2 étudiant

Vous trouverez sur Internet la description du jeu Eternity II. Vous générerez des puzzles aléatoires, vous en mélangerez les pièces, et vous résoudrez par une méthode de recherche arborescente avec retour en arrière (*backtrack*). Votre programme doit pouvoir résoudre des puzzles de taille 12 par 12, avec une dizaine de couleurs.

5 Eternity II et SAT. 2 étudiants

Vous trouverez sur Internet la description du jeu Eternity II. Vous récupérerez sur internet un programme de satisfaction de contraintes booléennes (le problème s'appelle SAT, ou 3-SAT ; des solveurs sont minisat ou picosat). Vous chercherez sur internet "eternity II SAT" pour trouver des articles présentant comment formuler le puzzle d'Eternity II comme un problème de satisfaction de contraintes booléennes ([Heu08, ABFM08] par exemple).

Vous générerez des puzzles aléatoires du jeu Eternity II. Vous générerez le problème de contraintes booléennes correspondant, que vous sauverez dans un fichier ; vous appelerez le solveur de contraintes booléennes ; vous lirez le

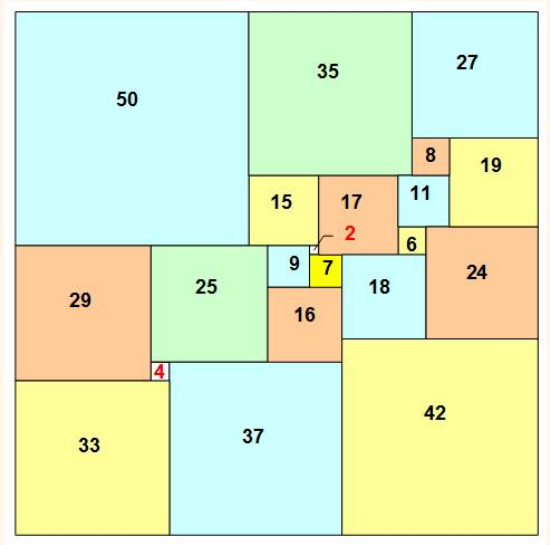
fichier solution. Vous testerez quelles tailles de puzzle sont solubles en un temps raisonnable (moins d'une 1/2 heure). Vous pouvez essayer plusieurs représentations du problème sous forme de contraintes booléennes, ainsi que plusieurs solveurs booléens.

Il existe un logiciel en Ocaml qui résout le problème SAT : SAT-MICRO, "petit mais costaud", qui est dû à Sylvain Conchon, Johannes Kanig, Stéphane Lescuyer. Il est téléchargeable sur internet.

6 Pavage en carrés. 2 étudiants

Soit un carré de coté n unités. Pour certaines valeurs de n, il est possible de paver le carré par des carrés plus petits, de côtés entiers et tous différents. Vous chercherez d'abord si n^2 peut s'écrire comme une somme de carrés d'entiers tous distincts. On sait que ce problème ne peut pas se généraliser en 3D et au delà.

Voir ce lien, d'où vient cette image :



7 Taquin. 2 étudiants

Faire des mouvements aléatoires sur un taquin. Puis résolvez le par programme (et sans tricher). Jusqu'à quelle taille de taquin pouvez vous aller avant que votre programme ne devienne trop lent ?

Liens : outre wikipedia, un autre lien

8 Interpolation. 2 étudiants

Vous calculerez la courbe algébrique $f(x, y) = \sum_i \sum_j a_{ij} x^i y^j = 0$, avec $i+j \leq d$ de degré d donné qui passe, ou approche, un ensemble donnés de points dans le plan. Vous utiliserez une résolution aux moindres carrés [CLRS01, PTVF92].

Vous enregistrerez la courbe et les points (issus d'un fichier).

Vous pouvez aussi générer des points sur une courbe connue (cercle, ellipse), pour vérifier que vous retrouviez bien la courbe en question.

Pour 2 étudiants : vous chercherez ensuite un autre type d'interpolation, par exemple par noyau gaussien (gaussian kernel). Mots clefs pour la recherche sur internet : gaussian kernel, radial basis function.

9 Stéganographie. 2 étudiants

Vous utiliserez le logiciel `convert`, ou `xv`, pour linux pour convertir les images dans des formats de fichiers facilement lisibles en Ocaml (PPM), en texte simple. Vous pouvez aussi utiliser la librairie `camlimages`.

Dans une image RVB, il est souvent possible de modifier, sans effet visible, le bit de poids faible de l'octet des composantes rouge, verte, bleue. Une méthode originale de stéganographie consiste à sacrifier ces bits et à les remplacer par ceux d'un texte texte encrypté (ou d'une image binaire encryptée) par la méthode RSA vue en cours de cryptographie.

Ocaml fournit une arithmétique sur des entiers (ou des rationnels) de longueur arbitraire : `nums.cma` ; vous l'utiliserez.

Ecrire le programme d'encryptage et de décryptage. Les clefs publiques et privées seront contenues dans des fichiers. Vous aurez besoin de calculer des grands entiers premiers p et q. La clef publique est le produit $n = pq$. La clef secrète est le couple (p, q). Vous utiliserez le test probabiliste de primalité donné en [CLRS01].

Erreur à ne pas commettre : encrypter chaque bit (ou chaque octet) séparément... Si le message à encrypter est court, il faut le compléter avec du bruit (des valeurs aléatoires).

10 Othello et optimisation. 3 étudiants

Vous utiliserez le *minimax* avec élagage alpha-beta fourni dans [CMP00, CMP98], disponible librement sur internet (ou à la BU).

Vous optimiserez les paramètres de la fonction évaluant la qualité d'une configuration : pour cela, vous ferez jouer entre eux des programmes avec différents paramètres, et vous sélectionnerez les programmes les meilleurs.

Vous devez lire quelques articles sur les algorithmes évolutionnaires, inspirés de la théorie Darwinienne de l'évolution (*swarm optimization*, *genetic algorithms*, *metaheuristics*) et proposer une implémentation.

11 Génération d'anagrammes de patronymes. 2 étudiants

A partir d'un prénom P et d'un nom N (en fait des lettres présentes dans le prénom et le nom : L), vous générerez un anagramme plausible : un autre prénom P' et un autre nom N'. Vous utiliserez un dictionnaire de prénoms, issu de insee.

La principale difficulté est de générer un nom plausible avec les lettres restantes (non utilisées dans le prénom). Jean Véronis proposait un tel logiciel sur son blog.

Pour générer un nom plausible, vous pouvez calculer, à partir de la liste de tous les prénoms, une chaîne de Markov : M_{ij} est la probabilité que la lettre i soit suivie de la lettre j. Vous pouvez ajouter 2 lettres virtuelles, pour le début et la fin du nom. Vous choisissez ensuite la permutation des lettres restantes pour le nom qui a la probabilité la plus grande.

Le programme de Jean Véronis générait, à partir de : "jean veronis", les anagrammes : "Jenna VOIRES", "Jonis AVENER, "Joane RIVENS", "Javier NONSE", "Joann VEISER", etc.

12 Ensembles de Julia et intervalles. 2 étudiants

Pour $c \in \mathbb{C}$, l'ensemble de Julia Jc est l'ensemble des points z du plan complexe tels que l'orbite de $z : z, f(z), f(f(z)), f(f(f(z))), ...$, où $f(z) = z^2 + c$, ne va pas à l'infini. Vous afficherez ces ensembles avec la méthode décrite par Afonso Paiva, Jorge Stolfi et Luis Henrique de Figueredo dans l'article « *Robust visualization of strange attractors using affine arithmetic* » Computers & Graphics 30 (6), 2006 pages 1020–1026 [PdFS06].

Vous n'êtes pas obligé d'utiliser une arithmétique affine, une arithmétique d'intervalles est suffisante. Le calcul des composantes fortement connexes d'un graphe orienté est expliqué dans le livre « *Introduction à l'algorithmique* » de Cormen, Leiserson, Rivest et Stein [CLRS01]. Des valeurs de c donnant de «beaux» ensembles de Julia sont mentionnés sur wikipedia.

13 SVD, GSVD, et compression d'images de visages. 2 étudiants

Vous programmerez la méthode présentée par Hervé Abdi dans « *Singular Value Decomposition (SVD) and Generalized Singular Value Decomposition (GSVD)* » pour compresser des fichiers d'images de visage. Vous pouvez utiliser la librairie Ocamlgsl, qui est une interface avec la GNU Scientific Library.

14 Bellmann-Ford, Floyd-Warshall et Flot maximal. 3 étudiants

On implémentera ces 2 méthodes en OCaml, pour des graphes dont les nœuds, la topologie et les poids sont tirés au hasard. Les poids des arcs seront utilisés comme les capacités du flot.

D'autres données peuvent être utilisés (voir ici). On donnera les distances entre sources et puits, en séparant clairement l'apport du précalcul de la matrice de Floyd-Warshall, sur de grands ensembles.

15 Programmation linéaire. 3 étudiants

De nombreux solveurs utilisant la programmation linéaire existent (`glpk`, `cplex`...), vous devez comprendre leurs fonctionnements et proposer une implémentation efficace et robuste en OCaml. Une source d'informations : Fast Linear Programming

Tester sur les exemples des TD et ou trouvés sur Internet.

16 Problème du voyageur de commerce. 2 étudiants

Vous résoudrez le problème du voyageur de commerce ("*Travelling salesman problem*") avec l'heuristique du recuit simulé ("*simulated annealing*"). Les villes seront données par leurs coordonnées 2D, et la distance entre deux villes est la distance euclidienne. Une des méthodes part d'une permutation aléatoire et remplace itérativement deux segments AB et CD par les deux segments AD et BC, ou par AC et BD, selon le plus court.

Vous trouverez de nombreux exposés de la méthode sur internet, par exemple wikipedia. Vous pouvez utiliser les données de villes françaises ou routes européennes ou encore John Burkardt.

17 Triangulation d'un ensemble de points 2D. 2 étudiants

En modifiant une méthode de calcul d'enveloppe convexe, vous triangulerez un ensemble donné de points 2D (issus d'un fichier texte). Puis, par échange d'arêtes, vous calculerez la triangulation de Delaunay. Lire : "An incremental algorithm based on edge swapping for constructing restricted Delaunay triangulations", par Marc Vigo Anglada, disponible sur internet, ou bien le polycopié de Franck Hétroy : Un petit peu de géométrie algorithmique lien. Utilisez des coordonnées entières pour éviter les difficultés dues à l'imprécision numérique.

18 Références

[ABFM08] Carlos Ansótegui, Ramón Béjar, César Fernàndez, and Carles Mateu.Edge matching puzzles as hard sat/csp benchmarks. In CP '08 : Proceedings of the 14th international conference on Principles and Practice of Constraint Programming, pages 560–565, Berlin, Heidelberg, 2008. Springer-Verlag. lien

[Aud13] Pierre Audibert. Géométrie des pavages. Lavoisier, 2013. cote BU Dijon : 516/1181.

[CJB07] A. Couturier and G. Jean-Baptiste. Programmation fonctionnelle appliquée aux calculs scientifiques : Objective CAML. Méthodes numériques & applications. Number vol. 1 in Les Cours de l'ICES. Cépaduès éd., 2007.

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to Algorithms. MIT Press, 2001.

[CMP98] Emmanuel Chailloux, Pascal Manoury, and Bruno Pagano. Développement d'applications avec Objective Caml. O'Reilly France, 1998. lien

[CMP00] Emmanuel Chailloux, Pascal Manoury, and Bruno Pagano. Developping Applications with OCAML. Oreilly book, 2000. lien

[Har07] Jon D. Harrop.OCaml for Scientists. May 2007. lien

[Heu08] Marijn J.H. Heule. Solving edge-matching problems with satisfiability solvers. pages 88–102. University of Leuven, 2008. lien

[Hic08] Jason Hickey. Introduction to Objective Caml. 2008.

[PdFS06] Afonso Paiva, Luiz Henrique de Figueiredo, and Jorge Stolfic. Robust visualization of strange attractors using affine arithmetic. Computers & Graphics, 30(6) :1020–1026, december 2006. lien

[PS98] C. H. Papadimitriou and K. Steiglitz. Combinatorial optimization: algorithms and complexity. Dover, 1998.

[PTVF92] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. Numerical Recipes in C, the Art of Scientific Computing. Cambridge University Press, 1992.