Projets de programation

CPES2: Algorithmique et Applications

Léonard ASSOULINE Pierre POPINEAU Pierre SENELLART
Tatiana STARIKOVSKAYA

21 octobre 2022

Instructions

Ce document présente l'ensemble des sujets proposés pour les projets du cours d'Algorithmique et Applications. Vous devez sélectionner (en utilisant le sondage sur Moodle – premier arrivé, premier servi) le projet que vous voulez faire, par groupes de deux ou trois étudiants (de préférence trois). Au plus deux groupes peuvent choisir le même sujet (à l'exception du sujet *Jeu d'arcade*, plus ouvert, qui peut être choisi au plus quatre fois mais avec des types de jeux différents).

Les sujets sont de nature très variés, parfois très détaillés, parfois moins ; parfois précis, parfois plus ouverts. Dans tous les cas, vous devrez prendre contact par e-mail avec l'enseignant responsable du sujet que vous aurez choisi qui vous guidera sur votre projet.

Les dates importantes sont les suivantes :

11 novembre. Date limite pour choisir votre groupe et sujet (mais nous vous recommandons de le faire à l'avance).

11–13 janvier. Soutenances des projets (pendant votre créneau de TP).

Pour la soutenance vous devrez :

— Avoir un programme Python en état de fonctionnement.
— Fournir une archive de tout le code (ou un lien vers Github, voir ci-dessous), incluant un court fichier texte décrivant comment l'utiliser.
— Prévoir une présentation (dont la durée sera fixée ultérieurement, en fonction du nombre de groupes) de la manière dont vous avez conçu votre programme (en insistant sur les aspects intéressants d'algorithmique), avec une démonstration du programme lui-même.

Nous vous interrogerons également et vous demanderons en particulier de décrire les contributions respectives des différents membres du groupe.

Nous vous recommandons très fortement, pour travailler ensemble sur le projet, d'utiliser un système de versionnement de code, par exemple le système Git en créant votre projet sur `https://github.com/`. Cela vous évitera d'avoir à gérer vous même les différentes versions, vous permettra de travailler à plusieurs, de revenir en arrière, de ne pas perdre votre code… Un tutoriel Github est disponible sur `https://product.hubspot.com/blog/git-and-github-tutorial-for-beginners` et vous trouverez beaucoup de ressources sur Git et Github en ligne.

Certains des projets demandent de réaliser une interface graphique. Nous vous recommandons d'utiliser Tk (voir `https://docs.python.org/3/library/tk.html`), nous pourrons vous aider dans ce but.

N'hésitez pas à faire preuve de créativité et d'apporter votre touche personnelle aux sujets proposés !

A. Arbres de Merkle

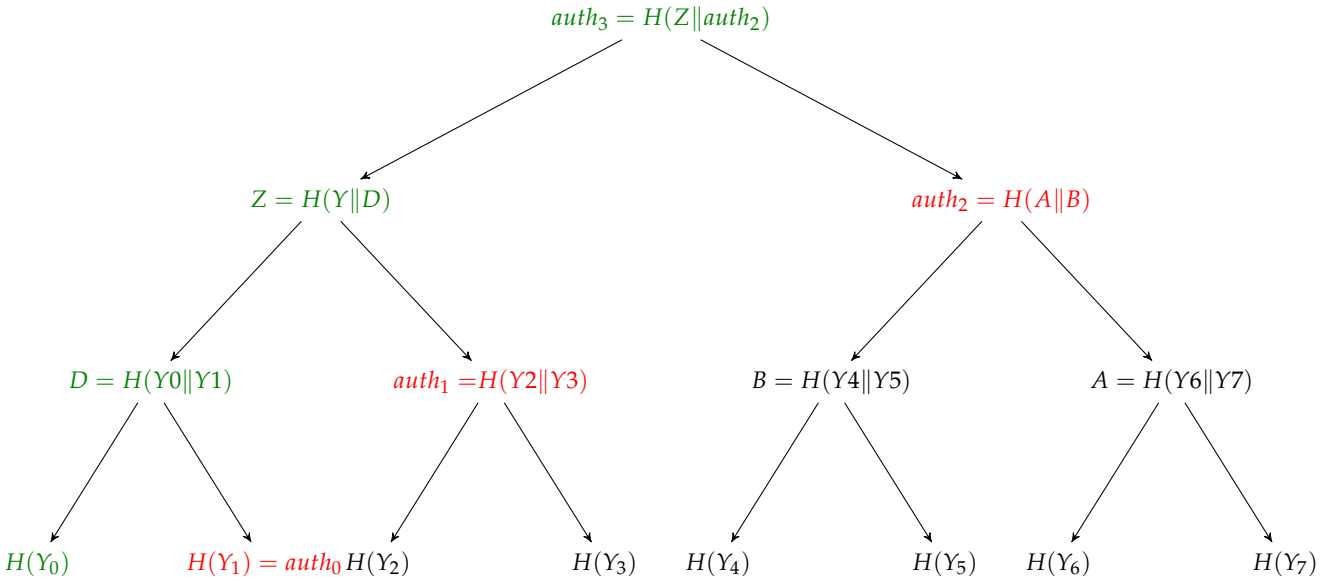
Responsable: Pierre POPINEAU (pierre.popineau@psl.eu)

Introduction

Une *fonction de hachage* est une fonction $H : S \to E$ telle que pour tout $s \in S$, $H(s)$ est facile à calculer, et pour tout $e \in E$, $H^{-1}(e)$ est dificile à calculer. De plus, on cherche à avoir presque sûrement $H(s) \neq H(s')$ dès que $s \neq s'$. Ces fonctions sont très utilisées en cryptographie afin de générer des signatures uniques et fortes qui soient difficiles à inverser. On utilise également les fonctions de hachage pour créer des clés dans les tables de hachage. Dans ce projet, nous allons explorer une autre structure de donnée utilisant les fonctions de hachage : les arbres de Merkle (ou arbres de hachage).

Le principe des arbres de hachage est le suivant : soit $Y_0, \ldots, Y_{k-1}$ une liste d'entiers et H une fonction de hachage, on construit l'arbre de bas en haut. On commence par appliquer notre fonction de hachage aux valeurs du vecteur Y, puis on crée le nœud r, parent des nœuds g (d'étiquette n_g) et d (d'étiquette n_d) en créant le nœud r d'étiquette $n_r = H(n_g \| n_d)$ où $\|$ est l'opération de concaténation. De cette façon, on crée des étiquettes uniques et difficilement déchiffrables à chaque niveau de l'arbre de hachage.

Signature de Merkle



Générer une signature de Merkle consiste à ensuite créer un chemin partant des feuilles et allant jusqu'à la racine de l'arbre (ce chemin est en vert sur le schéma). La signature va être générée en utilisant les nœuds voisins de chaque nœud du chemin. Les noeuds voisins du chemin remontant jusqu'à la racine sont appelés *nœuds d'authentification* (en rouge sur le schéma), et vont servir à chiffrer le message. La figure ci-dessus montre un exemple d'arbre de hachage pour $k = 8$ (un arbre à trois niveaux).

Considérons un message s que l'on cherche à encoder. On commence par le chiffrer en utilisant une fonction de hachage pour obtenir $s' = H(s)$. Puis, on prend une feuille de l'arbre $H(Y_i)$. On détermine le chemin jusqu'à la racine, puis on obtient les nœuds d'authentification $auth_0, \ldots, auth_l$. La signature obtenue pour notre message sera alors $sig = s'|H(Y_i)\|auth_0\| \ldots \|auth_k$.

Pour signer un second message, on se décale d'un cran vers la droite dans les feuilles de l'arbre vers la feuille $H(Y_{i+1})$ puis on actualise la liste des nœuds d'authentification pour signer le prochain message. Ce projet va se pencher sur deux algorithmes permettant de calculer de façon efficace les chemins d'authentification dans un arbre de Merkle de taille donnée, afin de générer des signatures de Merkle.

Une fois qu'une signature de Merkle est générée, pour s'assurer de sa validité, il suffit de vérifier si $s' = H(s)$, puis, de manière itérative, si pour tout i, $auth_{i+1} = H(A_i \| auth_i)$, avec $A_0 = s'$. À cause de l'injectivité de la fonction H, si le message original est altéré, la signature de Merkle obtenue sera différente.

But du projet

Ce projet se déroule en plusieurs phases :

— Concevoir et implémenter les classes nécessaires à la création des arbres de hachage. On portera notamment une attention aux types de données à utiliser et aux fonctions de hachage utilisées en Python.
— Définir des méthodes algorithmiques pour calculer les étiquettes de feuilles de l'arbre de hachage, ainsi que pour l'actualisation des chemins d'authentification de l'arbre.
— Rechercher et implémenter les algorithmes TREEHASH et TREETRAVERSAL, deux algorithmes optimisés pour le calcul de la signature de Merkle
— Définir une méthode pour vérifier si une signature de Merkle est correcte
— Créer une interface utilisateur permettant d'associer une signature de Merkle à un fichier donné, ou, prenant un fichier et une signature de Merkle, détermine si la signature est correcte.

B. Comparaison d'algorithmes de tri

Responsable: Pierre Senellart (pierre@senellart.com)

Il existe un grand nombre d'algorithmes de tri, dont certains vus en cours : tri par insertion, tri par sélection, tri fusion, tri à bulle, tri rapide, tri par paquets, tri en tas, tri par comptage…

Le but de ce projet est de mettre en place une méthodologie pour mesurer la performance de ces algorithmes de tri sur différentes entrée. Il s'agira d'implémenter un grand nombre d'algorithmes de tri, de les tester sur des données bien choisies, d'interpréter et de présenter les résultats.

On pourra aussi ajouter si le temps le permet une représentation graphique du tri en train de se faire, à des fins pédagogiques.

C. Couverture par sommets

Responsable: Pierre POPINEAU (pierre.popineau@psl.eu)

Introduction

Vous représentez une société de sécurité et un client vous demande d'installer un nouveau réseau de caméras de surveillance dans ses locaux. Vos caméras se placent dans les angles des couloirs et ont une vision parfaite sur toute la longueur des couloirs dans leur périmètre. Soucieux de proposer une installation bon marché, une question naturelle est : où positionner les caméras de surveillance pour voir tous les couloirs tout en positionnant le moins de caméras possible.

Pour formaliser ce problème, on peut représenter le réseau de couloirs par un graphe : chaque couloir est une arête du graphe, et chaque intersection est un sommet, où on cherche à placer nos caméras. Le problème est alors de trouver un sous-ensemble S de sommets de cardinal minimal tel que chaque arête du graphe a une extrémité dans S. Ce problème est connu sous le nom de *problème de couverture par sommets* (ou *vertex-covering problem*) et il s'agit d'un des 21 problèmes NP-complets historiques.

But du projet

Ce projet s'articule en plusieurs phases :

— Tout d'abord, il faudra trouver une façon convenable de représenter les graphes en Python (soit en concevant une classe sur mesure, soit en s'appuyant sur les classes déjà existantes en Python).

— Ensuite, on pourra commencer par formaliser le problème d'optimisation, et développer un algorithme glouton pour le résoudre. On pourra chercher à améliorer cet algorithme, ou étudier la littérature pour trouver des méthodes algorithmiques plus efficaces. Une attention particulière sera portée sur la complexité des fonctions implémentées.

— Enfin, on pourra utiliser des librairies graphiques de Python pour représenter les graphes et les solutions obtenues par les différentes méthodes choisies.

Une extension classique du problème consiste maintenant à mettre une distance maximale de vision pour les caméras de surveillance, au-delà de laquelle l'image devient trop floue. Ce problème s'appelle le problème de *couverture minimale par boules* (ou *minimal covering location problem*). Si la première partie du projet arrive à son terme, on pourra essayer d'adapter les méthodes précédentes à ce nouveau problème.

D. Échecs

Responsable: Léonard ASSOULINE (leonard.assouline@psl.eu)

Les échecs sont un jeu de stratégie qui se joue à deux. Ils existent depuis au moins un millénaire, leur implémentation en un programme informatique faisant partie des premières utilisations ludiques des ordinateurs. Pour plus d'informations : `https://fr.wikipedia.org/wiki/Echecs`.

Ici on vous demande un programme où deux joueurs humains peuvent s'affronter. Il faudra avoir les fonctionnalités suivantes :

— Les joueurs choisissent qui joue les blancs et les noirs. (Les blancs commencent aux échecs)
— Les joueurs jouent à tour de rôle leurs pièces, selon les règles décrites ici : `https://fr.wikipedia.org/wiki/R%C3%A8gles_du_jeu_d%27%C3%A9checs`
— Les joueurs ne doivent pas pouvoir faire de déplacements illégaux.
— Le programme doit dire si un joueur est en situation d'echec au roi, et mettre fin à la partie en cas d'échec et mat ou de pat.
— Les joueurs doivent pouvoir abandonner.

Toute partie d'échecs pouvant se jouer en vrai doit pouvoir se jouer sur votre programme. En particulier, on souhaite voir une recréation de la partie 1 du match opposant Gary Kasparov à Deep Blue en 1996.

E. Fourmi de Langton

Responsable: Pierre POPINEAU (pierre.popineau@psl.eu)

Introduction

Parmi les automates cellulaires simples, la fourmi de Langton est un exemple intéressant. On considère une fourmi qui se déplace sur une grille infinie à deux dimensions. Chaque case de la grille peut être coloriée soit en noir, soit en blanc. Les règles de déplacement de la fourmi, placée à l'origine du repère sont les suivantes :

— Si la fourmi est sur une case blanche, elle tourne de 90° vers la droite, colorie la case où elle est en noir et avance d'une case.
— Si la fourmi est sur une case noire, elle tourne de 90° vers la gauche, colorie la case en blanc et avance d'une case.

Le but est d'étudier le comportement de la fourmi de Langton sur un temps long. Comme beaucoup d'automates cellulaires, des motifs intéressants peuvent émerger de règles aussi simples.

Il existe beaucoup d'extensions de la fourmi de Langton, en fonction de la direction dans laquelle la fourmi tourne en fonction de la couleur de la case où elle se trouve. Avec deux couleurs, la fourmi définie précédemment peut être appelée "DG", car la couleur 1 fait tourner vers la droite et la couleur 2 fait trouner vers la gauche. On peut ainsi définir une infinité de fourmi de Langton différentes avec cette nomenclature.

But du projet

Dans un premier temps, on définira la fourmi de Langton originale, avec ses deux règles de déplacement définies précédemment, puis on trouvera une façon de représenter en temps réel les déplacements de la forumi en utilisant une interface utilisateur.

Une fois que cette première partie a été terminée, on pourra modifier le code pour intégrer plus de couleurs et de règles de déplacement, en utilisant la nomenclature de Turk et Propp (définie plus haut).

F. Arbres généalogiques

Responsable: Tatiana STARIKOVSKAYA (tatiana.starikovskaya@ens.psl.eu)

La mythologie grecque est très compliquée :
`http://www.veritablehokum.com/comic/the-greek-god-family-tree/`
Il est par exemple souvent difficile de savoir si un dieu est l'ancètre d'un autre ! Le but du projet va être de développer un outil facilitant la compréhension des arbres généalogiques, tels que :

— la généalogie des dieux grecs : `https://pierre.senellart.com/tmp/greek_gods.txt` (dans un simple format texte avec les liens de parenté)
— la généalogie des familles royales d'Europe : `https://pierre.senellart.com/tmp/royal.gedml` (dans un format XML plus complexe avec l'ensemble des liens familiaux)
— d'autres arbres généalogiques, par exemple récupérables depuis le site de généalogie `https://www.geneanet.org/`

L'outil développé devra permettre a minima :

— de traiter au moins deux arbres généalogiques différents
— de visualiser des parties de l'arbre généalogique sous forme graphique, de naviguer dans l'arbre
— de déterminer efficacement si deux personnes sont ancêtres l'un de l'autre (une façon de faire est de calculer la liste de tous les ancêtres de l'un et de vérifier que l'autre est dans cette liste)

D'autres fonctionnalités pourront être ajoutées à cet outil, suivant votre imagination.

G. Générateur de charabia

Responsable: Pierre POPINEAU (pierre.popineau@psl.eu)

Introduction

Un générateur de charabia est un algorithme dont le but est de générer des mots nouveaux, mais crédibles dans une langue donnée. Une première idée serait de tirer des lettres au hasard dans l'alphabet, mais cette méthode ne respecte pas les spécificités de chaque langue.

Pour générer ces mots, les chaînes de Markov sont un bon outil mathématique : plutôt que de tirer les lettres au hasard, on va s'intéresser aux probabilités de transition entre les lettres d'une langue donnée, afin de respecter les règles existant. Par exemple, en français, l'enchaînement « qu » est presque certain, tandis que l'enchaînement « zw » n'existe pas.

Pour utiliser une chaîne de Markov algorithmiquement, on peut définir un *automate à état fini* : il s'agit d'un objet A auquel on associe un ensemble d'états S, qui correspond aux caractères de la langue choisie et une matrice de transition P, qui correspond aux probabilités de transition entre les caractères. On devra également prendre en compte la longueur du mot en question (c.-à-d., la probabilité qu'une lettre soit en début ou en fin de mot).

On peut également appliquer cette technique à la fonction des mots (article, déterminant, adjectif, adverbe) pour générer des phrase grammaticalement correctes.

Pour pouvoir générer ces matrices de transition, une quantité importante de mots sont nécessaires. Des corpus de mots existent sur internet afin de pouvoir faire des statistiques significatives (comme le projet Gutenberg (`https://www.gutenberg.org/`), qui numérise de grandes quantités de données issues de corpus littéraires).

But du projet

Ce projet s'articule en trois phases :

— La génération de la matrice de probabilités. Il va falloir créer une fonction qui lit un fichier textuel contenant des mots et qui génère les probabilités de transition entre chaque caractère (en prenant en compte la distribution des tailles de caractères).

— La création de l'automate, qui va générer une liste de mots de taille donnée en utilisant les probabilités de transition calculées précédemment.

— Le générateur de charabia en lui-même, qui va prendre un texte et remplacer certains mots par du charabia généré par notre algorithme. La fréquence/choix des mots à remplacer vous est laissé, on pourra essayer plusieurs méthodes (replacer par le mot le plus proche, par un mot au hasard).

Si le générateur de charabia est fini, on pourra ensuite essayer de l'adapter pour d'autres langues et voir les résultats obtenus.

H. Itinéraires dans les transports parisiens

Responsable: Pierre Senellart (pierre@senellart.com)

Le STIF met à disposition sur `https://data.iledefrance-mobilites.fr/explore/dataset/offre-horaires-tc-gtfs-idfm/information/` l'ensemble des informations sur les lignes et horaires de transport en commun en Île-de-France. Ces informations sont disponibles au format GTFS, qui est décrit sur `https://developers.google.com/transit/gtfs/reference/`.

Le but de ce projet est de réaliser un moteur de recherche d'itinéraires entre deux stations, gare ou arrêts de bus en Île-de-France. On pourra utiliser pour cela l'algorithme de Dijkstra, qu'il faudra rechercher.

Pour simplifier la tâche, il est possible (au moins dans un premier temps) :

— de se concentrer sur les lignes de métro, en oubliant RER et lignes de bus ;

— de ne pas considérer les horaires, mais faire l'hypothèse que les stations sont toujours desservies.

À l'inverse, on pourra par exemple étendre le projet en :

— ajoutant un calcul d'itinéraire à partir d'une position GPS quelconque, ou d'une adresse quelconque (en utilisant une API de Geocoding telle que celle de Google Maps) ;

— ajoutant une représentation graphique de l'itinéraire.

I. Jeu d'arcade

Responsable: Léonard ASSOULINE (leonard.assouline@psl.eu)

L'objectif de ce projet est de faire un jeu rétro, par exemple Tetris, Snake, Space Invaders…

Si vous hésitez entre plusieurs projets, venez nous en parler. On peut essayer de vous orienter, en regardant quels types de techniques sont nécessaires dans chaque projet.

Plusieurs groupes (au plus quatre) peuvent prendre ce sujet, mais chaque groupe doit travailler sur un jeu différent.

Une possibilité pour l'implémentation (mais ce n'est pas la seule) est de s'appuyer sur la bibliothèque Pygame (`https://www.pygame.org/`) de fonctions simplifiant la programmation de certains jeux.

J. Jeu de la vie

Responsable: Léonard ASSOULINE (leonard.assouline@psl.eu)

Le jeu de la vie est ce qu'on appelle un automate cellulaire. Pour plus d'informations sur son histoire et son utilisation, voir `https://fr.wikipedia.org/wiki/Jeu_de_la_vie`.

Il y a donc des **cellules**, placées sur une grille à deux dimensions. (Chaque case de coordonnées $(i, j) \in \mathbb{Z}^2$ est une cellule). Chaque cellule est **vivante** ou **morte**.

Les cellules naissent et meurent selon les règles suivantes :

— Une cellule morte **devient vivante** si elle a **trois ou plus** voisines vivantes.

— Une cellule vivante **reste vivante** si elle a **deux ou trois** voisines vivantes.

— Une cellule vivante **meurt** sinon. En d'autres termes, une cellule meurt si elle a **strictement plus de trois voisines, ou strictement moins que deux**.

On y « joue » en choisissant avant le début de la partie quelles cellules sont vivantes, puis le jeu s'exécute jusqu'à ce qu'on décide de l'arrêter.

Ce « jeu » peut en réalité être vu comme un modèle de calcul : on peut écrire des algorithmes avec. Plus étonnant : il est **Turing-complet**, cela veut dire que ce modèle de calcul est aussi expressif que celui des ordinateurs : tout ce qu'un ordinateur peut faire, on peut le coder en jeu de la vie.

L'objectif de ce projet est de coder en Python un jeu de la vie, avec une interface graphique. L'utilisateur doit pouvoir choisir les cellules vivantes en cliquant sur les cases, ou bien en donnant en entrée un fichier texte contenant sur chaque ligne les coordonnées d'une case qui doit devenir vivante.

On doit pouvoir modifier la vitesse d'exécution du jeu, ou même le mettre en pause, et avancer étape par étape.

Dans des fichiers à part, mettez des exemples d'états initiaux que l'on appellera avec votre programme.

K. Logo

Responsable: Pierre Senellart (pierre@senellart.com)

Logo est une famille de langages de programmation, qui ont en particulier beaucoup été utilisées dans les écoles primaires et collèges pour initier les enfants à la programmation. Le langage, simple, contrôle une *tortue* qui traverse un écran tout en effectuant des dessins à l'aide de commandes élémentaires.

Pour une présentation générale de Logo, voir `http://ecoles.ac-rouen.fr/circdarnetal/new2/file/tice/tutoriel_XLOGO.pdf`

Des programmes Logo peuvent être testés dans un navigateur Web sur `http://lwh.free.fr/pages/prog/logo/logo.htm`

Le but de ce projet est de réaliser un *interpréteur Logo*, c'est-à-dire un programme Python qui prend en entrée un programme Logo et réalise dans une fenêtre graphique les différentes opérations de dessin du programme.

L. Poker : Texas Hold'em

Responsable: Léonard ASSOULINE (leonard.assouline@psl.eu)

Le Poker est un jeu d'argent qui se joue avec des cartes. Le jeu n'a sûrement pas besoin d'être présenté, mais pour voir toutes les règles : `https://fr.wikipedia.org/wiki/Poker`.

Le jeu se joue à plusieurs joueurs, qui possèdent des cartes. Chacun peut miser de l'argent, qui est mis en commun dans le *pot*. Celui qui a la meilleure main de cinq cartes repart avec le pot.

Ici on s'intéresse au *Texas Hold'em*, qui est la variante du Poker la plus jouée. Pour toutes les informations : `https://fr.wikipedia.org/wiki/Texas_hold%27em`. Les règles du programme seront les suivantes :

— Les cartes proviennent d'**un seul paquet**, elles peuvent avoir les faces, par ordre croissant de valeur : **2 à 10, Valet, Dame, Roi, As**, et les couleurs **Pic, Cœur, Trefle, Carreau**.
— Il y a au moins **deux joueurs**, chacun possède **deux cartes**, tirées aléatoirement du paquet.
— Les joueurs ont chacun une somme d'argent qui lui est propre, que l'on renseignera au programme.
— Quand un joueur n'a plus d'argent, il est éliminé du jeu.
— Voir la page Wikipedia du Hold'em pour le reste. Pour ce qui est des mains : `https://fr.wikipedia.org/wiki/Main_au_poker`
(On rappelle que l'As peut intervenir dans une suite As - 2 - 3 - 4 - 5 ou dans 10 - Valet - Dame - Roi - As, mais pas dans d'autres cas.)

On attend du programme que toute partie de poker qui puisse se jouer en vrai, puisse se jouer dessus. En particulier, on souhaite voir une recréation de l'affrontement final entre Le Chiffre et James Bond, dans Casino Royale (2006).

M. Satisfiabilité du Sudoku

Responsable: Pierre POPINEAU (pierre.popineau@psl.eu)

Introduction

On considère $(v_i)_{1 \geqslant i \geqslant n}$ une suite de variable propositionnelles, qui peuvent prendre les valeurs VRAI ou FAUX. Un *littéral* l est soit égal à une variable propositionnelle, soit la négation d'une variable. Enfin, on définit une *clause* comme étant la disjonction d'un nombre fini de littéraux, i.e. $c = l_1 \wedge l_2 \wedge \cdots \wedge l_k$ où $\wedge$ est l'opération logique OU.

Une question centrale en informatique est de savoir, étant donné une clause quelconque dépendant de n variables propositionnelles, s'il existe une assignation pour les variables $v_1, \ldots, v_n$ telle que la clause c aie la valeur logique VRAI. Dans ce cas, on dit que la clause c est *satisfiabile*. Ce problème est la problème de satisfiabilité, aussi appelé SAT.

Le problème de satisfiabilité fait partie de la classe des problèmes NP-complets, pour lesquels on ne connaît pas d'algorithme déterministe donnant une solution en temps polynomial mais pour lesquels ont peut vérifier la validité d'une solution simplement. Beaucoup de variantes existent au problème SAT, comme le problème 2-SAT, où on se resreint à des clauses à 2 littéraux ou HORN-SAT, qui utilise des clauses possédant au plus une négation.

Un exemple concret d'application du problème de satisfiabilité concerne les grilles de Sudoku. Soit une grille de sudoku de 9x9 chiffres. On cherche à trouver les chiffres manquants tels que chaque ligne, chaque colonne et chaque sous-carré de 3x3 contienne tous les chiffres de 1 à 9 une seule fois. Si à chaque case on associe une clause contenant 9 variables propositionnelles, telles que celle prenant la valeur logique VRAI correspond à la valeur de la case. On peut ainsi définir une clause pour chaque grille de sudoku, et satisfaire cette clause résoudra la grille de sudoku.

But du projet

Ce projet va s'intéresser à la résolution de SAT et à la manipulation de littéraux en Python. On commencera par implémenter la recherche exhaustive pour résoudre une formule propositionnelle pour voir les types de données à utiliser.

Dans un second temps, on s'intéressera à l'algorithme DPLL (Davis-Putnam-Logemann-Loveland), qui est un algorithme récursif permettant de résoudre SAT de manière plus efficace que la recherche exhaustive.

Une fois qu'un solveur aura été codé, on pourra s'intéresser au problème de la résolution de Sudoku, en associant à une grille donnée de sudoku une clause, la résolvant et affichant la solution ainsi trouvée sous la forme d'une grille 9x9.

N. Séries sur IMDB

Responsable: Pierre Senellart (pierre@senellart.com)

IMDB (`https://www.imdb.com/`) est un site Web présentant un grand nombre d'informations sur les films et séries télévisées. Une partie importante des données d'IMDB peut être récupérée depuis `https://datasets.imdbws.com/`.

Le but de ce projet est d'exploiter ces données pour afficher des matrices des notes obtenues par les épisodes d'une série, saison par saison, à l'image de la matrice suivante obtenue pour *The Big Bang Theory* :

	1	2	3	4	5	6	7	8	9	10	11	12
0	6.7									7.7		
1	8.3	8.3	8.4	8.8	8.1	7.7	7.9	7.2	6.8	7.7	7.6	7.2
2	8.4	8.2	7.9	8.5	8.1	7.7	8.1	7.6	7.4	7.4	7.3	7.6
3	7.8	8.8	8.2	8.1	7.8	8.1	8.8	7.0	7.7	7.3	7.0	7.2
4	8.2	8.0	8.1	8.1	8.0	8.3	8.0	7.2	7.5	8.0	7.1	7.1
5	8.0	8.4	8.3	7.8	8.1	8.1	7.7	7.5	7.5	7.5	7.4	6.8
6	8.5	8.3	7.6	8.1	7.7	7.9	8.5	7.2	7.2	7.4	7.2	8.3
7	8.2	8.7	7.9	8.1	8.6	7.7	7.8	7.8	8.0	8.0	7.4	7.4
8	8.3	8.2	9.0	8.2	7.7	8.5	7.6	8.0	8.0	8.1	7.2	7.4
9	8.1	8.2	8.4	8.5	8.2	8.3	8.9	7.5	7.9	7.5	7.8	7.3
10	8.1	7.8	8.6	7.8	8.0	8.1	7.7	7.0	8.4	7.4	7.4	7.5
11	8.3	9.2	8.4	8.7	7.4	7.9	8.2	7.4	9.1	7.6	7.1	7.3
12	8.0	8.0	8.4	7.7	7.8	8.5	7.5	7.3	7.5	7.3	7.5	7.2
13	8.4	8.1	8.1	8.3	8.2	8.5	7.6	7.5	7.5	7.0	7.3	7.1
14	8.2	8.1	8.5	7.9	8.3	7.9	8.0	8.0	7.6	7.3	7.3	7.1
15	8.3	8.9	8.1	7.8	7.8	8.3	8.3	8.1	7.5	7.3	7.5	7.5
16	8.4	8.3	8.4	8.0	7.7	8.0	7.7	7.9	7.4	7.2	7.1	8.0
17	8.6	8.2	8.5	8.0	7.9	7.7	7.5	7.6	7.5	7.5	6.8	7.1
18		8.2	8.5	8.2	8.3	7.8	7.5	7.4	7.6	7.2	7.2	7.2
19		8.3	8.0	8.1	8.3	8.1	8.2	7.6	7.5	7.1	7.3	7.0
20		8.0	8.2	8.3	8.2	8.1	8.1	7.5	7.5	7.2	7.2	7.2
21		8.6	8.2	8.3	8.6	8.1	7.6	7.1	7.5	7.3	7.5	7.3
22		8.3	9.1	7.9	7.8	8.0	7.7	7.3	7.5	6.8	7.3	8.0
23		8.4	8.6	8.1	8.2	8.1	7.9	7.5	7.4	7.1	7.7	9.2
24				8.6	8.6	7.8	8.1	8.0	7.8	8.6	9.0	9.6

La documentation des fichiers de données est consultable sur `https://www.imdb.com/interfaces/`. Le programme devra prendre en entrée le nom d'une série télé et produire en sortie une telle matrice de note (par exemple sous la forme d'une table dans le langage HTML, qui puisse être affichée dans un navigateur Web).

O. Tessellation de Voronoï

Responsable: Pierre POPINEAU (pierre.popineau@psl.eu)

Introduction

Soit $P = \{p_1, \ldots, p_n\}$ un ensemble de points de $\mathbb{R}^2$ et d une distance de $\mathbb{R}^2$. On peut associer à P un diagramme (ou tessellation) de Voronoï $V(P) = \{V_1, \ldots, V_n\}$, qui est une partition du plan telle que chaque sous-ensemble V_k contient l'ensemble des points plus proches de p_k que de n'importe quel autre point de P, c.-à-d., $V_k = \{x \in \mathbb{R}^2 \mid d(x, p_k) < d(x, p_i) \quad \forall i \neq k\}$.

FIGURE 1 – Une tessellation de Voronoi avec 20 points (CC-BY-SA 4.0, Baru Ertl)

Il existe beaucoup d'algorithmes pouvant calculer la tessellation de Voronoï, avec des complexités variables (comme ceux utilisant la triangulation de Delaunay, ou l'algorithme de Fortune). On peut également définir des tessellations de Voronoï avec des distances autres que la distance euclidienne, comme la distance de Manhattan.

But du projet

Le but de ce projet est d'explorer différentes méthodes algorithmiques permettant de calculer le diagramme de Voronoï associé à un ensemble de points donné, et de comparer leurs complexités. On pourra chercher du côté de l'agorithme de Fortune, de la méthode de Bowyer–Watson, ou des méthodes « diviser pour régner ».

Une fois que les diagrammes de Voronoï ont été calculés pour la distance euclidienne, on pourra étudier ces mêmes diagrammes avec la distance de Manhattan (ou distance « taxi ») ou la distance de Mahalanobis afin de voir comment les diagrammes changent.

P. Transformée de Burrows–Wheeler et encodage de Huffman

Responsable: Pierre POPINEAU (pierre.popineau@psl.eu)

Introduction

La transformée de Burrows–Wheeler (souvent notée par son acronyme anglais BWT) est une méthode algorithmique qui permet de préparer des données en vue d'une compression. Cette méthode se base sur des permutations des chaînes de caractères afin de rapprocher des caractères similaires pour faciliter une compression. Le code de Huffman quant à lui est une méthode de compression sans perte d'un message textuel. Ce projet va s'intéresser à ces méthodes de compression.

Dans la transformée de Burrows–Wheeler, on choisit un message à encoder, ici « SALUT ». On génère ensuite la matrice de toutes les rotations possibles du message : « SALUT, TSALU, UTSAL, LUTSA, ALUTS », qu'on trie par ordre alphabétique pour obtenir la matrice suivante :

$$\begin{pmatrix} A & L & U & T & S \\ L & U & T & S & A \\ S & A & L & U & T \\ T & S & A & L & U \\ U & T & S & A & L \end{pmatrix}$$

On garde en mémoire l'indice de la chaîne originale dans la nouvelle matrice (ici, 3), ainsi que la dernière colonne de la matrice, « SATUL ». Pour compresser la chaîne, on utilise ensuite l'algorithme « Move-To-Front ».

Cet algorithme consiste à associer à chaque caractère entre A et Z un indice entre 0 et 25. Lorsqu'un caractère est lu, on retient son indice, puis on le place en tête du tableau et tous les autres caractères. Par exemple, la chaîne « GGGGGGGBBA » va être codée par la chaîne « 6000000202 » : G a pour indice 6, après décalage, il passe à l'indice 0. B une fois décalé a l'indice 2, et A, décalé de deux crans vers la gauche, se retrouve à l'indice 2. Pour décoder la chaîne ainsi obtenue, il suffit d'inverser la procédure en partant du tableau initial.

Enfin, une fois que la chaîne a été générée par Move-To-Front, on utilise un arbre de Huffman pour compresser le texte obtenu. Pour ce faire, on détermine le nombre d'occurrences de chaque caractère dans la chaîne obtenue, puis on initialise des arbres pour chaque caractère, dont le poids est égal à son nombre d'occurrences. On construit alors l'arbre de Huffman récursivement en prenant les deux arbres de poids minimal. On crée ensuite un arbre ayant pour fils droit celui des deux de poids minimal et fils gauche celui de poids maximal, et dont le poids est la somme des deux. On continue ainsi de suite jusqu'à obtenir l'arbre complet. Une fois que l'arbre est construit, pour générer le code de Huffman, on descend l'arbre depuis la racine jusqu'aux feuilles en ajoutant un « 0 » si on part à gauche d'un nœud et un « 1 » sinon.

But du projet

Le but de ce projet va être d'implémenter une méthode de compression utilisant les trois algorithmes décrits précédemment. On commencera par la transformée de Burrows–Wheeler, puis Move-To-Front et enfin l'encodage de Huffman.

On implémentera d'abord des fonctions permettant d'encoder un message textuel en utilisant ces méthodes, puis les fonctions inverses permettant de décoder un message encodé.

On pourra également étudier la performance de cette méthode de compression ainsi que l'efficacité des méthodes définies.

Q. Web scraping

Responsable: Pierre Senellart (pierre@senellart.com)

Le *Web scraping* consiste à récupérer et extraire de l'information structurée depuis des sites Web. Il s'agit dans ce projet de choisir un site Web approprié, de récupérer toute ou partie des pages Web qu'il héberge, d'extraire des informations utiles de ces pages Web, et de les exploiter pour une application.

Par exemple :

— À partir du site de généalogie des mathématiciens `https://genealogy.math.ndsu.nodak.edu/` on peut extraire toutes les données de quel mathématicien a fait sa thèse avec quel directeur de thèse et ensuite construire un immense arbre généalogique correspondant, que l'on peut naviguer.

— À partir du site `http://ethnologue.com/` d'information sur les langues du monde, on peut récupérer toutes les fiches d'information sur les langues et les présenter d'une autre manière ou calculer des statistiques des langues les plus parlées.

N'importe quel site de ce type contenant des informations structurés peut être une cible intéressante. Attention cependant :

— Certains sites interdisent via leur fichier `robots.txt` l'acquisition automatique d'informations. Voir `https://www.robotstxt.org/orig.html` pour les détails sur ce fichier.

— Certains sites mettent en place des stratégies techniques de bloquage quand trop de requêtes leur parviennent.

R. Wikipedia : Pages importantes

Responsable: Pierre Senellart (pierre@senellart.com)

L'intégralité du contenu de Wikipedia peut être librement téléchargé depuis `https://dumps.wikimedia.org/`. Cela représente une quantité importante de données. Pour simplifier, on pourra se concentrer dans ce projet sur `https://dumps.wikimedia.org/simplewiki/`, une version en anglais simplifié de l'encyclopédie, beaucoup plus petite que la version en anlais (ou même que celle en français).

On s'intéresse à identifier les pages *importantes* de l'encyclopédie. On peut voir Wikipedia comme un graphe, avec un lien d'une page vers une autre si la première pointe vers la seconde. Une manière de mesurer l'importance des pages dans un tel graphe est via l'algorithme PageRank dont une description détaillée se trouve... sur Wikipedia `https://en.wikipedia.org/wiki/PageRank`

Plus le score de PageRank d'une page est grand, plus cette page a de l'importance.

Ce projet consiste à rechercher et comprendre l'algorithme PageRank, extraire des données de Wikipedia le graphe de l'encyclopédie, et implémenter le calcul du PageRank sur ce graphe pour extraire les pages les plus importantes.