Quelques propositions

1 Tests de connexité

— **Langage :** Python
— **Responsable du sujet :** Stéphane Gonnord
— **Sujets abordés :** Parcours de graphes, connexité.

1.1 Objet du projet

Pour savoir si un graphe (non orienté, non pondéré) est connexe, on peut utiliser différents algorithmes :
— si on dispose de la matrice d'adjacence, on peut utiliser un algorithme de type Warshall ;
— si le graphe est fourni par listes d'adjacences, on préférera un parcours du graphe en profondeur ;
— si le graphe est donné par une suite d'arêtes (éventuellement de façon dynamique), on peut également utiliser l'algorithme union/find qui permet de gérer à la volée le nombre de composantes connexes.

1.2 Les attendus

— Implémenter deux des trois points de vue présentés plus haut (mais les trois, c'est mieux !).
— Être capable d'exposer ces algorithme à l'oral, en particulier leur(s) complexité(s).
— Présenter des tests convaincants de petite taille.
— Tester le programme sur un grand nombre de graphes générés aléatoirement avec N sommets et $\frac{N \ln N}{2}$ arêtes. Estimer la proportion de graphes connexes.

1.3 En bonus

Voici divers points que vous pourrez choisir de développer :
— Une interface utilisateur (en console, ou graphique) permettant d'entrer un graphe.
— La possibilité de lire un graphe dans un fichier texte, sous un format que vous aurez spécifié.
— Comparer les différentes méthodes sur des tests à grande échelle.
— Après être allé consulter la littérature sur le sujet, faire des tests permettant d'illustrer les résultats d'Erdös et Rényi sur les seuils de connexité pour un graphe aléatoire : si on place $\frac{N(K+\ln N)}{2}$ arêtes dans un graphe à N sommets, alors la probabilité d'être connexe tend vers $e^{-e^{-K}}$ lorsque N tend vers $+\infty$.

2 Le compte est bon !

— **Langage :** Python, Caml, ...
— **Responsable du sujet :** Stéphane Gonnord
— **Sujets abordés :** Programmation dynamique, arbres binaires, ensembles et dictionnaires.

2.1 Objet du projet

Si vous ne connaissez pas « le compte est bon », vous pouvez googliser les règles... puis aller voir `http://www.youtube.com/watch?v=X1jwjiEQG9Y` pour vous détendre.

Il existe plusieurs méthodes pour obtenir des comptes approchants optimaux. On développera les deux suivantes :

— On calcule par programmation dynamique les ensembles de valeurs accessibles en utilisant seulement les entiers d'indices compris entre i et $j \geq i$. Les cas élémentaires sont clairs ; l'objectif correspond à $(i, j) = (0, 5)$ (ou $(1, 6)$...), et le calcul se fait à $j - i$ croissant.
— On engendre tous les arbres de calcul possibles (il y en a $C_1 + C_2 + \cdots + C_6$, avec C_n le n-ième nombre de Catalan), puis on habille les arbres avec des opérateurs et feuilles, et on évalue les arbres ainsi construits.

2.2 Les attendus

— Réfléchir aux structures de données (et langages) à utiliser pour chacun des algorithmes ; être capable de justifier les choix effectués.
— Implémenter les deux méthodes de résolution. Le résultat devra être d'une part le meilleur compte approchant... et d'autre part l'arbre de calcul permettant de l'obtenir (sous une forme à déterminer).
— Les programmes devront pouvoir récupérer les données du problème soit au clavier (à l'appel du programme, ou via une mini-interface textuelle), soit via un fichier texte (avec un format qui est laissé à votre convenance, mais devra être spécifié).
— Vous devez être capable d'exposer dans les grandes lignes les complexités des algorithmes en jeu, et discuter leur passage à l'échelle ($n = 10$?).

2.3 En bonus

Voici divers points que vous pourrez choisir de développer :
— Le test et la comparaison des deux méthodes à grande échelle sur des valeurs d'entrée générées aléatoirement (il faudra réfléchir à la façon de les générer !).
— Une interface graphique permettant d'une part d'entrer les données du problème, et d'autre part de présenter les résultats calculés.
— Voici une façon de décomposer 2016 :

$$2016 = 1 * ((2 + (3 + (4 + 5))) * (6 * ((7 + 8) + 9)))$$

En vous inspirant des idées de ce projet, écrivez un programme permettant de calculer ce type de décomposition, pour 2017 ou un peu au delà !

3 Exploration de chaînes de Markov

— **Langage :** Python
— **Responsable du sujet :** Stéphane Gonnord
— **Sujets abordés :** Parcours de graphe, calculs matriciels, simulation stochastique, chaînes de Markov.
— **Biblio :** Par exemple « Probabilités pour les non probabilistes », seconde édition ; *W. Appel.*

3.1 Objet du projet

Voici trois exemples de processus stochastiques que l'on peut étudier :
— Lors d'une série de lancers de pile/face indépendants (et équitables), quel est le temps moyen d'attente pour la première apparition de PFP ? et de FFP ? Et quelle est la probabilité pour que le premier PFP apparaisse avant le premier FFP ?
— Lors d'un jeu au tennis, si les joueurs ont des probabilités constantes (respectivement p et $1 - p$) de remporter chaque point (et ce de façon indépendante à chaque point), quelle est la probabilité pour le premier joueur de remporter le jeu ? Et quel est le temps moyen d'un jeu ?
— Au jeu des « petits chevaux », combien de temps- faut-il pour qu'un joueur gagne ? [1]

De tels exemples peuvent être traités soit via la simulation directe (permettant d'estimer à la fois des probabilités et des espérances) soit par une analyse probabiliste plus ou moins élémentaire... soit par

1. Oui, « trop », mais encore ?

une analyse de la matrice stochastique décrivant le processus (réduction, ou estimation de ses puissances élevées).

Plus généralement, quand une chaîne de Markov est décrite par sa matrice, on dispose de toute une artillerie théorique pour estimer diverses choses, ou pour classifier les différents états (transitoires, récurrents, absorbants) ou classes d'états. Le calcul effectif de la nature de ces états peut se faire par parcours de graphe (à vous d'inventer les solutions !) ou par analyse de la matrice.

3.2 Les attendus

- Traiter par la simulation les deux premiers exemples pour obtenir des résultats empiriques de probabilités et espérances (en fonction de p, dans le premier exemple).
- Utiliser un logiciel permettant de faire du calcul matriciel (par exemple Python+numpy, ou maple, ou matlab, ...) pour traiter les matrices stochastiques M des deux premiers exemples : d'une part en évaluant M^N pour N assez grand, et d'autre part en réduisant les matrices M, ce qui permet d'obtenir la limite de M^n lorsque n tend vers $+\infty$. Comparer les résultats issus des trois attaques.
- Écrire un programme permettant de prendre en compte une chaîne de Markov quelconque donnée par sa matrice de transition : on déterminera de trois façons les états transitoires/absorbants/récurrents.
- Discuter les complexités des différentes parties de vos algorithmes et programmes.

3.3 En bonus

Voici divers points que vous pourrez choisir de développer :

- Le premier exemple est en relation avec le « paradoxe de Penney » : le web vous fournira des précisions sur ce problème qui est bien connu et compris. Utilisez la simulation et les outils théoriques mis en place pour vérifier les (ou une partie des) résultats recensés dans la littérature.
- En précisant les règles, étendre le deuxième exemple au problème du gain du match (simulation et analyse de chaînes de Markov).
- Dans la classification théorique des états d'une chaîne de Markov, allez plus loin en déterminant les différentes classes d'états d'une chaînes de Markov : les récurrentes et les transitoires.
- Dans le problème des petits chevaux, simplifiez les règles en prenant un seul joueur avec un seul cheval, et évaluez de différentes façons le temps de jeu pour mener ce cheval à l'écurie ! Ensuite, compliquez les règles.

4 Problème du voyageur de commerce

- **Langage :** Python
- **Techniques :** complexité, arbre couvrant minimum
- **Responsable du sujet :** Fabien Rico

Le problème du voyageur de commerce est un problème d'algorithmique très connu où à partir d'un graphe des villes et de routes reliant ces villes, on recherche un chemin qui passe une et une seule fois par chacune des villes et dont la longueur est minimale.

Ce problème est NP-complet ce qui signifie qu'il ne peut sans doute pas être résolut efficacement en temps raisonable, mais (dans certains cas) il existe des moyens de calculer une approximation par exemple, si les distances entre les villes obéïssent à l'inégalité triangulaire, il est possible d'utiliser l'algorithme de Christofides qui donne une solution au pire égale à $3/2$ de la solution optimale.

Vous devez implémenter l'algorithme de Christofides. Votre logiciel doit partir d'un fichier contenant les villes et le coût des liaisons directes entre les ville et fournir le chemin trouvé.

Pour améliorer l'algorithme vous pouvez :

- Dévelloper un interface graphique pour présenter les résultats (et leur différentes étapes)
- Utiliser d'autres méthodes de résolution, par exemple la solution naive (pour calculée la solution optimale), l'utilisation des méthodes d'optimisation ...