TI301 – Algorithmique et Structures de données 2

Projet Informatique et Mathématiques

Ce projet est rédigé en commun avec le département de mathématiques

Etude de Graphes de Markov – PARTIE 1

Les graphes de Markov sont des représentations des chaînes de Markov à temps discret, un outil très utile en probabilités, et dont les applications sont très nombreuses. Le but de ce projet est de réaliser des traitements automatiques sur ces graphes afin d'en extraire quelques caractéristiques, et de les visualiser.

Dans ce projet ne seront présentées que les notions théoriques essentielles pour la partie informatique

Qu'est-ce qu'un graphe de Markov ?

Un graphe de Markov est un graphe d'états (simplement numérotés en général), avec des probabilités de passage d'un état à un autre.

Un graphe de Markov sera composé de :

- Un ensemble de sommets
  - Chaque sommet représente un état ;
  - Chaque sommet sera identifié par un numéro.
- Un ensemble d'arêtes
  - Chaque arête relie deux sommets ;
  - Chaque arête possède un poids, qui est une valeur réelle **strictement positive, et inférieure ou égale à 1**. Cette valeur représente la probabilité de passer d'un sommet à un autre (d'un état à un autre) ;
  - Une arête peut relier un sommet à lui-même.

Ce type de graphe doit de plus respecter la contrainte suivante :

La somme des probabilités 'sortantes' d'un sommet (de toutes les arêtes qui partent de ce sommet) doit être égale à 1.

Représentations

On représente classiquement ces graphes de deux manières.

Matrice

Par une matrice de probabilités M, où chaque valeur M_{ij} indique la probabilité de passer de l'état i à l'état j. Une ligne i correspond aux probabilités 'sortantes' de l'état (ou sommet) i

Exemple

$$M = \begin{pmatrix} 0.95 & 0.04 & 0.01 & 0 \\ 0 & 0.9 & 0.05 & 0.05 \\ 0 & 0 & 0.8 & 0.2 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

La ligne 1 indique, pour l'état 1, que :

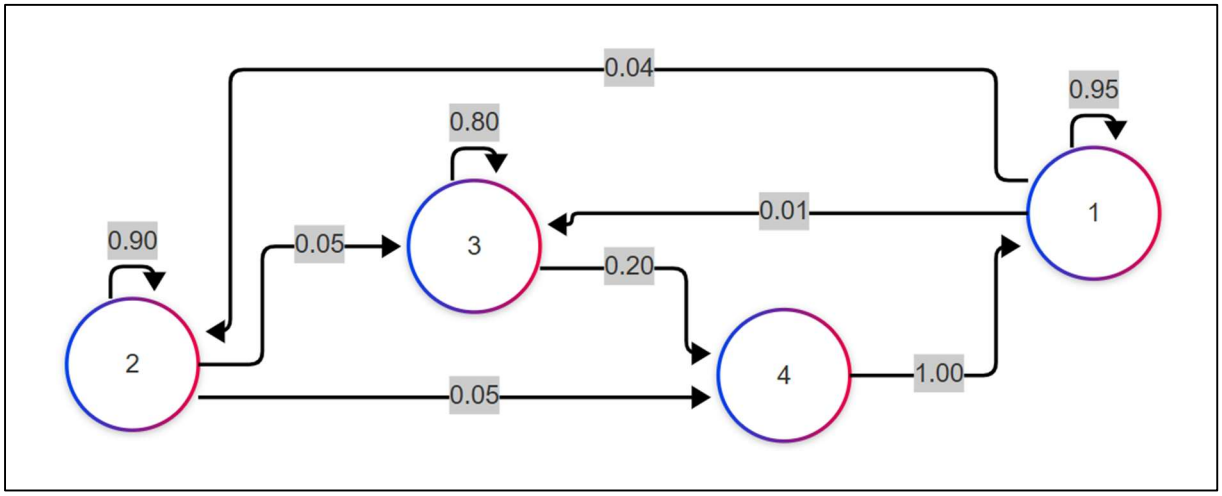
la probabilité de rester à l'état 1 est de 0,95
la probabilité de passer à l'état 2 est de 0,04
la probabilité de passer à l'état 3 est de 0,01
la probabilité de passer à l'état 4 est de 0

On constate également que la somme de tous les coefficients de cette ligne est bien égale à 1

Dessin du graphe

Par le dessin du graphe orienté correspondant, où ces mêmes probabilités sont écrites sur les arêtes.

Exemple de dessin du graphe associé à la matrice M ci-dessus



C'est le type de dessin que vous allez produire !

Cahier des charges

Étape 1 - Créer un graphe, vérifier que c'est un graphe de Markov, dessiner le graphe

Fichiers de données

Pour créer des graphes, vous disposerez de **fichiers de données**, pour ne pas avoir à faire de saisies répétitives.

Ces fichiers auront le format suivant :

Ligne 1 : **nombre de sommets**

Lignes suivantes : **sommet_de_départ sommet_d_arrivée probabilité**

Exemple : toujours pour le même graphe à 4 sommets, le fichier 'exemple1.txt' est donc le suivant :

```
4
1 1 0.95
1 2 0.04
1 3 0.01
2 2 0.9
2 3 0.05
2 4 0.05
3 3 0.8
3 4 0.2
4 1 1
```

On y retrouve toutes les valeurs non nulles de la matrice M.

Structures de données à utiliser pour stocker le graphe

Pour stocker un graphe, on pourrait utiliser un tableau 2D (une matrice) comme pour la représentation matricielle – cependant ces matrices pour les graphes de Markov contiennent une majorité de 0, qui ne nous sont pas utiles pour le moment.

- La structure de données à utiliser est donc **une liste d'adjacence** :

Pour chaque sommet, on stocke, dans une liste chaînée (simple), les arêtes issues de ce sommet, avec : le sommet d'arrivée (un int), et la probabilité associée (un float)

Toutes ces listes (1 par sommet), sont stockées dans un **tableau**.

Exemple de la liste d'adjacence pour le graphe exemple

```
Liste pour le sommet 1:[head @] -> (3, 0.01) @-> (2, 0.04) @-> (1, 0.95)
Liste pour le sommet 2:[head @] -> (4, 0.05) @-> (3, 0.05) @-> (2, 0.90)
Liste pour le sommet 3:[head @] -> (4, 0.20) @-> (3, 0.80)
Liste pour le sommet 4:[head @] -> (1, 1.00)
```

structures à créer

- une structure de type « cellule » avec : sommet d'arrivée, probabilité, et pointeur vers la « cellule » suivante. Chaque « cellule » représente une arête.
- une structure de type « liste » avec un champ « head ». Chaque liste stocke toutes les arêtes 'sortant' d'un sommet
- une structure « liste d'adjacence » : un tableau dynamique de « liste » avec sa taille (sa taille étant le nombre de sommets)

fonctions à créer

- une fonction pour créer une « cellule » ;
- une fonction pour créer une « liste » vide ;
- une fonction pour ajouter une cellule à une liste ;
- une fonction pour afficher une liste ;
- une fonction pour créer une liste d'adjacence 'vide' à partir d'une taille donnée – on crée le tableau de listes, chacune étant initialisée avec la liste vide ;
- Une fonction pour afficher une liste d'adjacence.

Créer un graphe

Pour créer un graphe, utilisez un des fichiers texte fournis en exemple.

voici une base de code pour vous aider à effectuer la création d'une telle liste à partir d'un fichier texte :

```
liste_adjacence readGraph(const char *filename) {
    FILE *file = fopen(filename, "rt"); // read-only, text
    int nbvert, depart, arrivee;
    float proba;
    declarer la variable pour la liste d'adjacence
    if (file==NULL)
    {
        perror("Could not open file for reading");
        exit(EXIT_FAILURE);
    }

    // first line contains number of vertices
    if (fscanf(file, "%d", &nbvert) != 1)
    {
        perror("Could not read number of vertices");
        exit(EXIT_FAILURE);
    }

    Initialiser une liste d'adjacence vide à partir du nombre de
sommets

    while (fscanf(file, "%d %d %f", &depart, &arrivee, &proba) == 3)
    {
        // on obtient, pour chaque ligne du fichier les valeurs
        // depart, arrivee, et proba
        Ajouter l'arête qui va de 'depart' à 'arrivée' avec la
        probabilité 'proba' dans la liste d'adjacence
```

```
    }

    fclose(file);
    return la liste d'adjacence remplie;
}
```

Validation de l'étape 1

Vous êtes capable de charger un fichier texte ;

Vous êtes capable d'afficher la liste d'adjacence.

Étape 2 – Vérifier le graphe

Vous devez vérifier que la somme de toutes les probabilités 'sortantes' d'un sommet est bien égale à 1.

Puisqu'il s'agit de valeurs de type **`float`**, il y aura parfois des arrondis : il faudra vérifier, en fait, que la somme des probabilités est comprise entre 0,99 et 1.

Ecrivez une fonction qui vérifie si un graphe est bien un graphe de Markov, et qui affiche les messages :
`Le graphe est un graphe de Markov`
ou

`Le graphe n'est pas un graphe de Markov`
avec les sommets en cause, par exemple

`la somme des probabilités du sommet 3 est 0.97`

Validation de l'étape 2

Votre programme affiche correctement les messages – testez sur les différents fichiers de données (certains ont été produits par ChatGPT et contiennent des erreurs).

Étape 3 – Dessiner le graphe

Pour dessiner le graphe, nous allons utiliser l'outil en ligne et gratuit Mermaid : https://www.mermaidchart.com/

Cet outil nous permettra d'obtenir un dessin, à partir d'un fichier texte avec un format spécifique, assez proche de celui des fichiers de données que nous utilisons.

Voici le fichier à créer, toujours pour l'exemple du graphe à 4 sommets.

```
---
config:
   layout: elk
   theme: neo
   look: neo
---

flowchart LR
A((1))
B((2))
C((3))
D((4))

A -->|0.01|C
A -->|0.04|B
A -->|0.95|A
B -->|0.05|D
B -->|0.05|C
B -->|0.90|B
C -->|0.20|D
C -->|0.80|C
D -->|1.00|A
```

On y retrouve :

- des directives de configuration (à conserver)
- Des noms de sommets (A,B,C,D),suivi des numéros à afficher
- Les arêtes, avec leurs probabilités.

Vous disposez d'une fonction **`char *getId(int num) ;`** qui vous fournira une chaîne de caractères associé à un entier : « A » pour 1, « B » pour 2, …, « Z » pour 26, « AA » pour 27, « AB » pour 28,….

- ➢ Ecrivez une fonction qui prend en paramètre une liste d'adjacence (représentant un graphe), et qui produit le fichier texte au format attendu.

Ensuite, copiez/collez le contenu de votre fichier dans l'éditeur de code Mermaid.

Validation de l'étape 3

A partir du fichier **`exemple_valid_step3.txt`**, vous devriez obtenir le visuel suivant (après quelques manipulations très simples sur mermaid) :