CPE Lyon - 3IRC - Année 2018/19

Techniques et Langages d'Internet

TP 7 et 8 - Mini projet

Ce projet couvre les deux dernières séances de TP de ce cours de TLI, qui auront lieu le mercredi 20 mars et le mardi 2 avril. Il est à rendre pour le vendredi 5 avril à 18h.

Le sujet fait appel à des compétences variées et transversales : algorithmique, POO, web, un peu de maths... Pour tenir compte de la différence d'expérience, le sujet est légèrement plus long pour le groupe 1 (IUT Info). Par conséquent, les deux membres de chaque binôme doivent appartenir au **même** groupe.

Prenez le temps de lire le sujet dans son intégralité. Il y a de nombeuses spécifications et fonctionnalités à respecter, mais il paraît plus long qu'il ne l'est réellement.

Vous pouvez utiliser jQuery, ainsi que les librairies mentionnées dans la partie **Lecture et écriture des fichiers**. Les autres bibliothèques sont interdites par défaut, sauf avis contraire des intervenants.

Procédez avec **méthode** et de manière **incrémentale** : essayez d'abord d'implémenter les fonctionnalités simples et de base, puis ajoutez les fonctionnalités plus évoluées. Réfléchissez aussi à la répartition des tâches dans votre binôme.

Sujet du projet : Editeur de graphes

Préambule : un peu de théorie...

Un **graphe** est un objet mathématique très simple, qu'on peut représenter comme un ensemble de points, appelés **sommets** ou **noeuds** dont certains, appelés **voisins**, sont reliés par des **arêtes** :

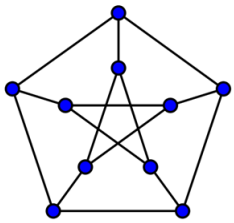


FIGURE 1 – Le graphe de Petersen.

La puissance des graphes réside dans la simplicité de ce formalisme : ils permettent en effet de modéliser toute situation où des objets, des personnes, sont en **relation** : réseaux de télécommunication, réseaux sociaux, réseaux routiers.... Ils permettent également de modéliser un nombre considérable de problèmes importants dans l'industrie : problèmes d'affectation optimale, problèmes d'emplois du temps, problèmes de transports, problèmes de tournées de véhicules.... Toute l'histoire de Google a démarré avec un algorithme de graphe simple mais original, appelé **PageRank**, qui représentait le web sous forme de graphe et appliquait un score à chaque page en fonction de ses voisins (l'algorithme a été considérablement enrichi depuis !).

Plus formellement, un graphe est donc un couple $G = (V, E)$ où V est l'ensemble des sommets (*vertices* en anglais) et E est l'ensemble des arêtes (*edges* en anglais), càd un ensemble de paires de sommets. Rien n'interdit de donner des noms, des numéros, ou plus généralement des **étiquettes** aux sommets (cf. Fig. 2).

Le graphe ci-dessus est **non orienté** : on ne distingue pas une « extrémité initiale » ou une « extrémité finale » d'une arête. Parfois, cette distinction est nécessaire ; par exemple, en gestion de projet, on peut

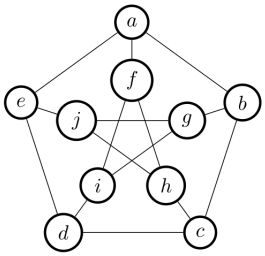


FIGURE 2 – Le graphe de Petersen étiqueté.

représenter chaque tâche par un sommet d'un graphe, et indiquer par une flèche de A vers B que la tâche A doit être terminée avant la tâche B. On parle alors de **graphe orienté** :

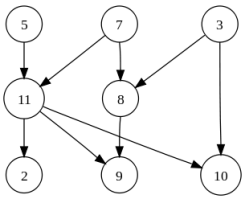


FIGURE 3 – Un exemple de graphe orienté.

Dans un graphe orienté, on ne parle plus d'arête, mais d'**arc** entre deux sommets.

L'algorithme PageRank original de Google, évoqué ci-dessus, fonctionne en fait sur un graphe orienté, et attribue un score à une page en fonction du nombre et du score des pages qui pointent sur elle :

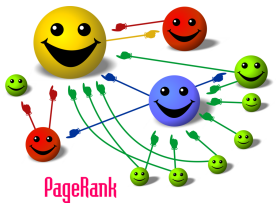


FIGURE 4 – Illustration de PageRank

Passons à la pratique...

Votre objectif est de réaliser d'une part une bibliothèque JavaScript offrant un certain nombre de fonctions de manipulation de graphes sous la forme d'une API (c'est-à-dire des fonctions appelables par un utilisateur), et d'autre par un éditeur de graphes permettant les mêmes opérations mais sous forme "visuelle" (et s'appuyant évidemment sur votre bibliothèque).

En particulier, cette bibliothèque et l'éditeur doivent permettre *a minima* de :

- **créer** et **modifier** un graphe, orienté ou non, en ajoutant des sommets et des arêtes / arcs ;
- **enregistrer** / **lire** un graphe dans / depuis un fichier ;
- appliquer certains **algorithmes** sur des graphes

Cette bibliothèque doit être utilisable directement dans le navigateur d'un utilisateur, sans avoir besoin de recourir à un serveur.

Spécifications fonctionnelles de l'interface graphique

Votre travail doit respecter les spécifications suivantes ; néanmoins, rien ne vous empêche d'ajouter des fonctionnalités supplémentaires que vous jugeriez intéressantes

- la manipulation du graphe se fait sur un *canvas*, qu'on appellera *zone de travail* dans la suite
- on **ajoute un sommet** en cliquant sur la zone de travail
- on **ajoute une arête / un arc** en sélectionnant successivement deux sommets
- on peut **sélectionner** un sommet en cliquant dessus ; la sélection est mise en évidence graphiquement (changement de couleur, d'épaisseur de bordure, etc.). On **déselectionne** un objet sélectionné en cliquant à nouveau dessus
- on **supprime** un sommet en cliquant dessus puis en appuyant sur la touche "suppr" du clavier ; la suppression d'un sommet entraîne la suppression de tous les arcs / arêtes qui y sont connectés. On supprime une arête / un arc de la même façon qu'on le crée
- on peut **modifier l'étiquette** d'un sommet en double-cliquant sur celui-ci
- les **arêtes** sont représentés par des segments
- les **arcs** sont représentés par des segments fléchés, la pointe de la flèche étant du côté du sommet destination
- on peut **déplacer un sommet** par "glisser-déposer" (drag and drop) ; évidemment, les arêtes / arcs connectés à ce sommet doivent être redessinés
- un composant graphique permet à l'utilisateur de **basculer** entre le mode "graphe non orienté" et le mode "graphe orienté" (le passage de l'un à l'autre doit faire apparaître / disparaître les pointes des flèches)
- un bouton **`Réinitialiser`** permet d'effacer tout le contenu de la zone de travail
- l'utilisateur doit pouvoir exporter son graphe dans un format lui permettant de reprendre son travail plus tard (cf. ci-dessous pour les spécifications de ce format) ; bien sûr il doit également être en mesure d'importer un fichier de ce format
- l'import d'un fichier ne correspondant pas au format attendu provoque l'affichage d'un message d'erreur
- avant d'exporter un graphe, l'utilisateur doit spécifier un nom ; ce nom sera utilisé comme nom de base pour les fichiers exportés
- une liste déroulante contient les noms des algorithmes que l'utilisateur peut exécuter sur le graphe. On lance l'exécution par un clic sur un bouton **`Exécuter`**. La sortie de l'algorithme doit être visible à la fois sur l'interface, et exportée dans un fichier JSON (voir plus loin le format attendu pour chaque algorithme). Un clic dans la zone de travail efface le résultat de l'algorithme.
  ⚠ Attention : certains algorithmes ne fonctionnent que sur un graphe orienté ou non orienté ; la liste déroulante doit adapter son contenu (les algorithmes proposés) en fonction du type de graphe choisi.

Spécifications du format de fichier

Les fichiers exportés devront impérativement respecter la syntaxe JSON suivante :

```
{
    "graph": {
        "name": "nom_du_graphe",
        "directed": "false",
        "vertices": [
            { "id": "0", "label": "v1", "pos": {"x":"100", "y":"100"}},
            { "id": "1", "label": "v7", "pos": {"x":"150", "y":"50"}},
            { "id": "2", "label": "v19", "pos": {"x":"83", "y":"27"}},
            ...
        ],
        "edges": [
            {"id1":"0", "id2":"1"},
            {"id1":"0", "id2":"2"},
            ...
        ]
    }
}
```

⚠ Attention, ces fichiers seront testés ensuite par une application de validation. Il est donc important de respecter la syntaxe ci-dessus, faute de quoi vos fichiers pourraient être rejetés !. De même, votre fonction d'import de fichiers sera testée sur différents fichiers, valides ou non, et il faut donc qu'elle réagisse correctement dans chacun des cas.

Lecture et écriture de fichiers

Pour lire un fichier, vous pourrez utiliser l'objet **FileReader** (https://developer.mozilla.org/fr/docs/Web/API/FileReader). Par exemple, le code ci-dessous permet d'ouvrir et d'afficher un fichier image :

```
<input type='file' accept='image/*' onchange='openFile(event)'>
...
<script>
  var openFile = function(event) {
    var input = event.target;

    var reader = new FileReader();
    reader.onload = function(){
      var dataURL = reader.result;
      var output = document.getElementById('output');
      output.src = dataURL;
    };
    reader.readAsText(input.files[0]);
  };
</script>
```

Pour simplifier, l'écriture se fera aussi en local, c'est-à-dire sans avoir recours à un serveur. Vous pourrez utiliser par exemple la bibliothèque **FileSaver.js** (https://github.com/eligrey/FileSaver.js), comme suit :

```
var blob = new Blob(["Hello, world!"], {type: "text/plain;charset=utf-8"});
FileSaver.saveAs(blob, "hello world.txt");
```

Calcul du PageRank

Lorsque vous disposerez d'un éditeur de graphes fonctionnel, vous pourrez simuler un (relativement petit) sous-ensemble du web : chaque sommet représente une page, et un arc du sommet A vers le sommet B signifie que la page A contient au moins un lien vers la page B.

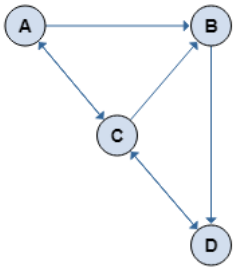
On vous demande à présent d'implémenter l'algorithme **PageRank** qui est (entre autres) à l'origine du succès de Google (il tire d'ailleurs son nom de Larry Page, le cofondateur de l'entreprise). L'idée derrière PageRank est que plus il y a de pages légitimes qui pointent vers une page P, meilleur sera le PageRank de P. La version présentée est très simplifiée, mais donne néanmoins l'idée du fonctionnement de l'algorithme.

Cet algorithme est itératif. A chaque itération, on applique la formule suivante sur chacune des pages :

$$PR_i(p_k) = \sum_{j \in L(k)} \frac{PR_{i-1}(p_j)}{d(p_j)}$$

où $L(k)$ est l'ensemble des pages qui pointent sur k, et $d(p_j)$ est le nombre de pages pointées par la page j. Autrement dit, chaque page donne une même fraction de son PageRank aux pages sur lesquelles elle pointe.

Prenons un exemple : considérons le graphe suivant :



- **initialisation :** on donne aux 4 pages une valeur de PageRank égale, soit 1/4 chacune
- **itération 1 :** calculons le nouveau PageRank de A : seule la page C, de PageRank égal à 1/4, pointe sur A ; comme C pointe sur 3 pages, elle donne $\frac{1/4}{3}$ à chacune de ces pages, dont A. Ainsi, $PR_1(A) = \frac{1}{12}$. On procède de même pour les autres pages. A la fin de la première itération, on obtient les PageRanks suivants (ramenés au même dénominateur) :

Page	Itération 0	Itération 1
A	1/4	2/24
B	1/4	5/24
C	1/4	9/24
D	1/4	8/24

- **itération k :** on recommence comme précédemment, mais à partir des résultats de l'itération k-1.

Arrêt de l'algorithme : on peut voir les résultats de chaque itération comme un **vecteur**. On arrête l'algorithme lorsque deux itérations successives donnent des résultats suffisamment proches, i.e. lorsque les vecteurs correspondants sont suffisamment proches. Un moyen de comparer deux vecteurs est de calculer l'angle qu'ils forment, grâce à leur produit scalaire :

$$cos(\theta) = \frac{u \cdot v}{||u|| \times ||v||} = \frac{\sum u_i v_i}{||u|| \times ||v||}$$

Ainsi, plus $cos(\theta)$ est proche de 0, plus les deux vecteurs sont orthogonaux, i.e. "différents". A l'inverse, plus $cos(\theta)$ se rapproche de 1, plus les deux vecteurs deviennent similaires. On peut par exemple choisir d'arrêter l'algorithme lorsque $cos(\theta) \geq 0,99999$.

Résultat : A la fin de l'algorithme, on classe les pages par PageRank décroissant. Si on s'arrêtait après l'itération 1, le classement des pages serait donc le suivant :

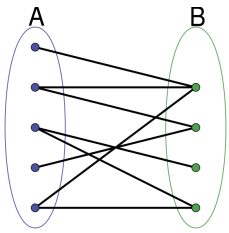
Page	PageRank	Classement
A	2/24	4
B	5/24	3
C	9/24	1
D	8/24	2

Rendu attendu : le résultat de l'algorithme doit s'afficher sur la page web et doit faire apparaître pour chaque sommet la valeur calculée de son PageRank ainsi que son classement. Ces résultats doivent également être exportés en JSON, dans un fichier se nommant `XXX_PageRank.json`, où `XXX` est le nom du graphe.

***** Fin de la partie obligatoire pour le groupe Réseaux / GEII *****

Bipartition

Un graphe **non orienté** est **biparti** si on peut classer ses sommets en deux ensembles A et B de sorte que les arêtes du graphe ont toutes une extrémité dans A et l'autre extrémité dans B :



De manière équivalente, un graphe est biparti si et seulement si on peut colorier ses sommets avec deux couleurs (par exemple rouge et bleu), de sorte que tous les voisins d'un sommet rouge sont bleus et inversement.

Travail demandé : implémentez un algorithme qui teste si un graphe est biparti. Si le graphe n'est pas biparti, un message s'affiche à l'écran ; si le graphe est biparti, ses sommets seront coloriés en rouge ou bleu, pour mettre en évidence la bipartition, et un fichier JSON est généré avec le format suivant :

```
{
    "algorithm": {
        "name": "biparti",
        "classe_A": [
{"id": "0"}, {"id": "1"}, {"id": "4"}, {"id": "7"}, {"id": "8"}
],
        "classe_B": [
{"id": "2"}, {"id": "3"}, {"id": "5"}, {"id": "6"}
        ]
    }
}
```

Algorithme de Dijkstra

L'algorithme de Dijkstra permet de calculer le plus court chemin entre des sommets du graphe. Il est utilisé dans les GPS, dans Google Maps, en intelligence artificielle, en robotique, dans les jeux vidéos... pour calculer des itinéraires optimaux.

Il fonctionne sur des graphes orientés ou non, mais pour simplifier, on ne considérera que des graphes non orientés. Cependant, les graphes sont **pondérés** : chaque arête a un **poids** ; typiquement, si les sommets du graphe représentent des villes, le poids d'une arête correspond à la distance entre les deux villes.

L'algorithme est expliqué dans cette courte vidéo : https://www.youtube.com/watch?v=MybdP4kice4

Travail demandé : rajoutez la gestion des poids sur les arêtes du graphe (dans l'éditeur et au niveau des structures de données), puis implémentez l'algorithme de Dijkstra. L'exécution de l'algorithme doit mettre en évidence le plus court chemin entre deux sommets sélectionnés ; la sortie en JSON doit avoir le format suivant :

```
{
    "algorithm": {
        "name": "dijkstra",
        "length": "78",
        "path": [{"id": "0"}, {"id": "1"}, {"id": "4"}, {"id": "7"}, {"id": "8"}]
    }
}
```

***** Fin de la partie obligatoire pour le groupe Info *****

Pour ceux qui auraient tout terminé, quelques idées pour poursuivre :

- implémenter la sélection des arêtes
- pour les graphes orientés, lorsqu'on crée deux arcs en sens contraires (un arc de u vers v et un arc de v vers u), dessiner des arcs courbés afin de mieux les distinguer
- améliorer l'interface graphique en rajoutant par exemple des menus
- ajouter des boutons `annuler / refaire`
- ajouter des contrôles pour gérer des styles (taille des sommets, couleur des sommets, épaisseur des arêtes, style des arêtes...)
- implémenter d'autres algorithmes de graphes (coloration, calcul des composantes connexes, recherche de chemin eulérien, recherche de cycle hamiltonien...)