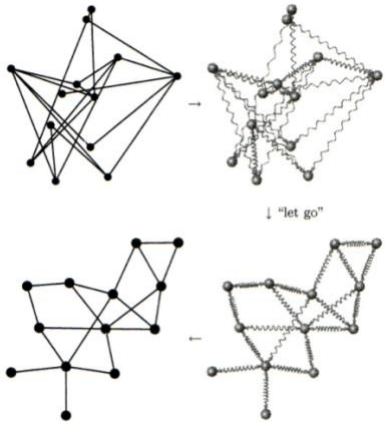
Visualisation d'informations

D3 (4)

Les algorithmes par modèle de force permettent de placer les sommets des réseaux/graphes dans le plan. Le principe est de simuler un modèle physique masse-ressort. Dans cette simulation, les sommets du graphe sont représentés par des objets physiques et deux objets physiques sont reliés par un ressort si les sommets du graphe correspondant sont reliés par une arête.



Ces algorithmes suivent le schéma général suivant :

```
Répéter n fois :
   Pour chaque sommet v :
      Calculer les forces d'attraction appliquées sur v par ses voisins
      Calculer les forces de répulsion appliquées sur v par tous les
             autres sommets
   Pour chaque sommet v :
      Déplacer le sommet v en fonction de ces forces
```

Suivant l'algorithme utilisé, le calcul des forces n'est pas le même (utilisation de différents modèles physiques). Voici un exemple :

- Calcul de la force d'attraction exercée sur un sommet v :

$$f_{attr}(v) = \sum_{u \in Neigh(v)} \frac{dist_R(u,v)^2}{length^2} * (p(u) - p(v))$$

- Calcul de la force de répulsion exercée sur un sommet v :

$$f_{rep}(v) = \sum_{u \in V, u \neq v} \frac{length}{dist_R(u,v)^2} * (p(v) - p(u))$$

- Calcul de la force totale exercée sur un sommet v :

$$f_{totale}(v) = f_{attr}(v) + f_{rep}(v)$$

`Neigh(v)` correspond à la liste des voisins de v.
`dist(u,v)` est la distance euclidienne entre u et v.
`p(v)` est la position de v.
`V` est l'ensemble des sommets du graphe.
`length` est la longueur constante des ressorts au repos.

Dans ce TP, nous allons afficher un graphe avec D3 en utilisant un modèle de force. Les exercices sont librement adaptés d'un code proposé par Mike Bostok[1]. Ceci nous permettra aussi de faire de brefs rappels sur les principales fonctions D3 déjà étudiées précédemment.

Exercice 1 : Afficher les sommets d'un graphe

Commencez par créer une page HTML.

Rappel : Une page HTML correctement formatée doit débuter par une balise doctype, contenir une balise `html`, une balise `head`, une balise `body` et une balise `title`. Vous pouvez aussi utiliser une balise `meta` pour définir le type d'encodage (e.g. UTF 8).

Importez la librairie D3.js.

Dans le body, insérez un script JavaScript (balise `script`) et définissez deux variables (`w` et `h`) pour la largeur et la hauteur de votre visualisation.

Ajoutez maintenant un objet SVG à votre body en utilisant les fonctions D3 appropriées :

```
var b = d3.select("body");
var svg = b.append("svg");
svg.attr("width", w);
svg.attr("height", h);
```

Ou de manière plus concise :

```
var svg = d3.select("body").append("svg")
     .attr("width", w)
     .attr("height", h);
```

Ces deux codes sont équivalents.

Avant d'aller plus loin, nous allons définir une fonction retournant un chiffre aléatoire. Elle nous permettra d'attribuer des valeurs aux positions des sommets de notre graphe (en attendant d'ajouter l'algorithme de forces). Si vous ne la connaissez pas déjà, trouvez sur Internet la fonction JavaScript permettant de renvoyer une valeur aléatoire comprise entre 0 et 1. Utilisez-là pour créer une fonction `rand(mi,ma)` permettant de renvoyer une valeur aléatoire comprise entre `mi` et `ma`.

Ouvrez le fichier miserables.json et étudiez sa structure. Assurez-vous que vous comprenez comment les sommets et les arêtes sont encodés. Chargez le graphe dans votre page HTML :

```
d3.json("miserables.json").then(function(graph) {
});
```

[1] http://bl.ocks.org/mbostock/4062045

Il vous reste maintenant à afficher les sommets et les arêtes du graphe. Commencez par ajouter un cercle pour chaque élément de « nodes » (voir le fichier miserables.json). Voici une solution possible d'après les TP précédents :

```
for(i=0;i<graph.nodes.length;i++){
        var node = svg.append("circle");    // ajout d'un cercle
        node.attr("stroke", "#aaaaaa");     // couleur de la bordure
        node.attr("r", 5);                  // largeur du rayon
        node.attr("id", "node"+graph.nodes[i].id); // identifiant
        node.attr("cx", rand(1, w));        // position X
        node.attr("cy", rand(1, h));        // position y
        node.attr("class", ".node");        // ajout d'une classe
}
```

Assurez-vous de bien comprendre toutes les lignes du code, et si ce n'est pas le cas, revoyez les TP précédents. Ouvrez votre fichier HTML dans votre navigateur. Si le code est correct, vous verrez les points s'afficher à des positions aléatoires :

D3 est basé sur une approche déclarative, permettant d'appliquer des opérations sur n'importe quel ensemble d'objets, par exemple les objets d'une certaine classe. La fonction `selectAll(".nomClass")` permet de récupérer cet ensemble. Remplacez le code de création des cercles précédent par celui-ci (vous devriez obtenir le même résultat) :

```
var classNodes = svg.selectAll(".node"); // Sélection d'une classe
var dataNodes = classNodes.data(graph.nodes); // Ajout des données
var nodes = dataNodes.enter().append("circle"); // Ajout des cercles
nodes.attr("stroke", "#aaaaaa");
nodes.attr("r", 5);
nodes.attr("id", function(d) { return "node"+d.id; } );
nodes.attr("cx", function(d) { return rand(1, w); } );
nodes.attr("cy", function(d) { return rand(1, h); } );
```

Ou de manière plus concise :

```
var nodes = svg.selectAll(".node")
        .data(graph.nodes)
        .enter().append("circle");
nodes.attr("stroke", "#aaaaaa")
        .attr("r", 5)
        .attr("id", function(d) { return "node"+d.id; } )
        .attr("cx", function(d) { return rand(1, w); } )
        .attr("cy", function(d) { return rand(1, h); } );
```

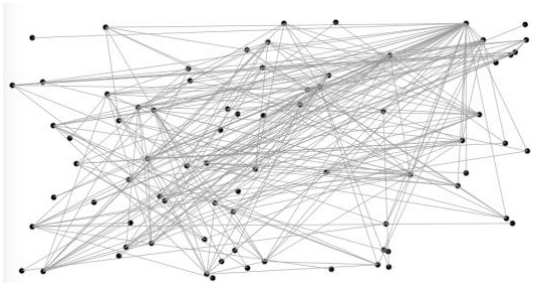
Exercice 2 : Afficher les arrêtes d'un graphe

Utilisez la même technique pour ajouter les arêtes du graphe. Dans ce cas, vous devez ajouter des objets graphiques de type `"line"` au lieu de `"circle"`. Lorsqu'une ligne est droite, on peut directement définir les attributs x1, y1, x2, y2, qui correspondent aux coordonnées des

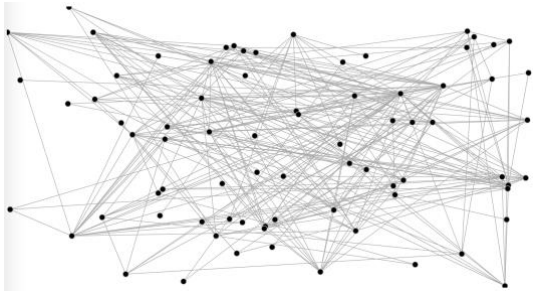
deux points aux extrémités de la ligne, dans notre cas les deux sommets reliés par le lien. L'attribut x1 sera par exemple :

```
links.attr(
        "x1",
        function(d){
              return d3.select("#node"+d.source).attr("cx");
        }
);
```

Voici le diagramme que vous devez obtenir :



Comme vous pouvez l'observer, les liens apparaissent « au-dessus » des sommets. Pour qu'ils apparaissent en-dessous, créez-les avant les sommets (attention, les attributs des positions des liens doivent rester après le positionnement des sommets, car ils utilisent ces positions).



Exercice 3 : Ajouter des couleurs aux sommets et des épaisseurs aux arêtes

Vous allez maintenant ajouter des couleurs à vos sommets en fonction des groupes dans lesquels ils apparaissent (voir le fichier miserables.json). Pour cela, vous allez utiliser une fonction d'échelonnage de D3 qui calcule des couleurs[2] (vous pouvez placer la ligne suivante avant le chargement des données) :

```
var color = d3.scaleOrdinal(d3.schemeCategory10);
```

Vous pouvez ensuite ajouter un attribut « fill » à vos sommets :

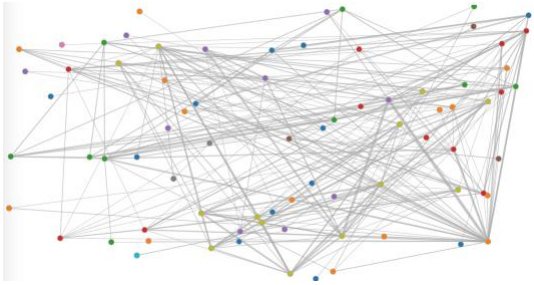
```
.attr("fill", function(d) { return color(d.group); });
```

Vérifier que les couleurs des sommets s'affichent bien.

Vous allez maintenant utiliser la valeur associée aux liens dans le fichier miserables.json pour modifier l'épaisseur des arêtes de la visualisation. Pour rappel, l'attribut correspondant à l'épaisseur d'une ligne est stroke-width. Donnez-lui comme valeur la racine carrée (`Math.sqrt(laValeur)`) de la valeur contenue dans le fichier.

[2] https://github.com/d3/d3-scale/blob/master/README.md#scaleOrdinal

Enfin, donnez une opacité de 0.6 aux arêtes en modifiant l'attribut « stroke-opacity ».



Exercice 4 : Utiliser un modèle de forces pour placer les sommets

Vous allez d'abord créer une instance de la classe de D3 « forceSimulation », en précisant la liste des sommets et les différentes forces que vous voulez appliquer (la dernière permet de spécifier le centre de la visualisation):

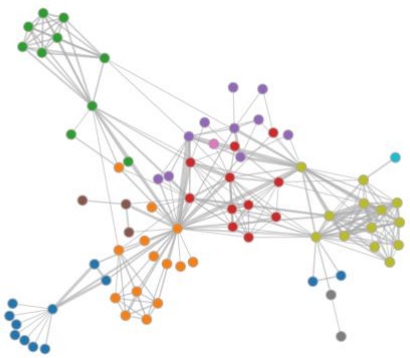
```
var forceDirectedLayout = d3.forceSimulation()
        .nodes(graph.nodes)
        .force("repulsion", d3.forceManyBody())
        .force("attraction", d3.forceLink(graph.links))
        .force("center", d3.forceCenter(w / 2, h / 2));
```

Supprimez maintenant les lignes permettant de calculer la position des sommets (cx et cy) et la position des arêtes (x1, x2, y1 et y2). C'est D3 qui va se charger de donner ces positions. A chaque itération de l'algorithme de placement, la fonction ci-dessous est exécutée (« tick »). Le code permet d'attribuer les valeurs calculées (d.x et d.y) aux positions des sommets et des arêtes.

```
forceDirectedLayout.on("tick", function() {
        links
              .attr("x1", function(d) { return d.source.x; })
              .attr("y1", function(d) { return d.source.y; })
              .attr("x2", function(d) { return d.target.x; })
              .attr("y2", function(d) { return d.target.y; });

        nodes
              .attr("cx", function(d) { return d.x; })
              .attr("cy", function(d) { return d.y; });
});
```

Ajoutez cette fonction et vérifiez que tout marche bien.



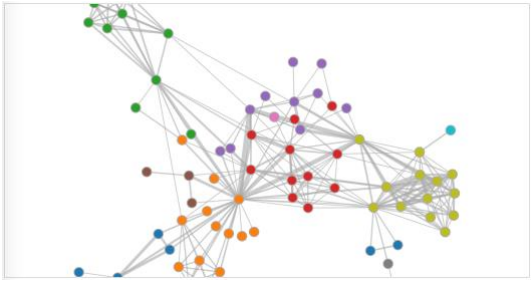
Exercice 5 : Interactions

Ajoutez sur les sommets une info-bulle permettant de visualiser les noms des personnages lorsque le curseur de la souris reste une seconde au-dessus :

```
.append("title").text(function(d) { return d.name; });
```

Ajoutez un zoom (cf. TP précédents, rappelez-vous, il faut ajouter une bordure au SVG et mettre tous les éléments graphiques dans un objet de type « g ») :

```
var z = d3.zoom();
z.on("zoom", function({transform}){
        gContenantTousLesElements.attr("transform", transform);
});
svg.call(z);
```



Pour finir, vous pouvez ajouter la possibilité de déplacer les sommets à l'aide de la souris. Pour cela, il faut utiliser la fonction « call » sur les cercles pour appeler la fonction « drag » de D3 :

```
.call(d3.drag()
        .on("start", dragstarted)
        .on("drag", dragged)
        .on("end", dragended));
```

Il faut ensuite définir les fonction `dragstarted`, `dragged` et `dragended` ainsi :

```
function dragstarted(e, d) {
      if (!e.active) forceDirectedLayout.alphaTarget(0.3).restart();
      d.fx = d.x;
      d.fy = d.y;
}

function dragged(e, d) {
      d.fx = e.x;
      d.fy = e.y;
}

function dragended(e, d) {
      if (!e.active) forceDirectedLayout.alphaTarget(0);
      d.fx = null;
      d.fy = null;
}
```