8INF803 Bases de données réparties Automne 2020
Devoir 2
50 % de la note finale.

Date de remise :

17 Décembre 2020

Remise : edmond.lachance@gmail.com OU
edmond_lachance@uqac.ca

Lien vers le devoir 1

Remise :

1. **Rapport en PDF, HTML ou Repo Github avec Markdown**
2. **Screenshots de l'exécution dans le document, ou alors fichiers images**
3. **(Optionel) Screenshots ou vidéo de l'exécution sur Cluster**

Vous pouvez utiliser Github ou Google Drive/Skydrive pour vos fichiers.

Équipes :

1 à 4 personnes

Les bonus possibles

Vous pouvez obtenir plusieurs bonus pour ce travail, car c'est le genre de travail qu'on peut sans cesse améliorer en mettant beaucoup de temps. Les bonus peuvent être très massifs, et bien entendu l'excédent gagné avec les bonus augmente directement votre note finale.

Bonus 1. Visualisation d'un exercice du devoir

1. Vous pouvez montrer une visualisation du graphe des créatures et leurs sorts. Le graphe devrait avoir des milliers de noeuds!
2. Vous pouvez montrer une visualisation du combat tour par tour. La technologie de visualisation est votre choix. Quelques projets précédents ont fait de la visualisation de combat de la façon suivante :
    a. Dessin des créatures et leurs actions sur une fenêtre GUI. La fenêtre est refresh à chaque tour de combat.
    b. Visualisation du combat sur le browser avec Websockets et CSS (Ils n'utilisaient pas canvas si je me souviens correctement)
    c. Visualisation 3D
    d. Vous écrivez des fichiers .svg ou des fichiers image sur le disque à chaque itération (jamais fait, peut être intéressant aussi).

Quelques technologies de visualisation : yedEditor, d3.js, Gephi, GraphStream ou GraphViz (pour donner quelques exemples).

Voici 2 exemples de visualisation avec Yed :

Exemple 1 (Itérations de coloriage). Généré depuis Spark GraphX (On écrit dans un XML à chaque itération)
Exemple 2 (36mb PNG)
Exemple 2 (version SVG)

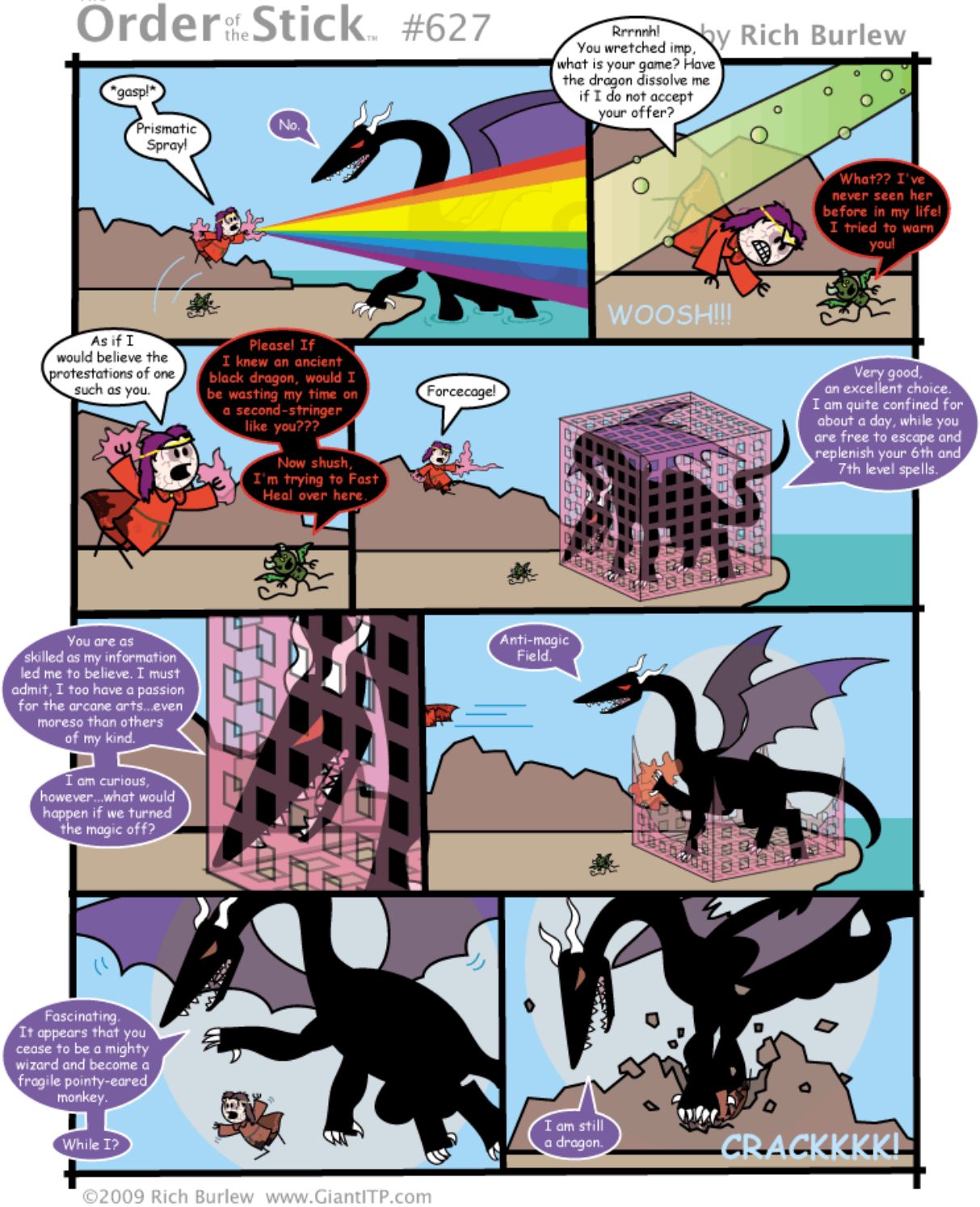
Bonus 2. Exécution sur un Cluster d'au moins 2 machines en réseau.

Tutoriel Cluster ici

Si vous faites exécuter sur un cluster, me montrer des screenshots qui montrent que tout calcul sur le cluster (ou alors une vidéo).

The Order of the Stick #627

by Rich Burlew



Exercice 1 : La guérison

Dans le dernier TP, vous avez trouvé un sort (verbal seulement) pour sortir Pito de son pétrin. Pito a utilisé le sort ***Dimension Door*** *pour s'enfuir du feu. Cependant, Pito a été brûlé sévèrement par l'incendie et il va mourir de ses blessures s'il n'y a personne pour l'aider. Et Pito ne connaît aucun sort qui puisse l'aider dans la situation présente...*
C'est à ce moment que le pouvoir de la bloodline de Pito se met en action. Ce pouvoir (que vous incarnez encore une fois de plus) va permettre à Pito de se concentrer et d'accéder à un grand pouvoir.
Ce grand pouvoir va lui permettre d'accéder à un savoir incroyable. Pito va pouvoir visualiser tous les sorts de soins ainsi que les noms des créatures qui possèdent ces derniers.
Une fois qu'il a vu tout ça, il conjure une créature qui peut le soigner. Cette créature, un Ange Solar (Pito n'a jamais entendu parler d'eux) soigne les brûlures de Pito. Pito s'endort, épuisé. L'Ange, qui est toujours là, ne repars pas tout de suite.

Ce que vous devez faire :

1. Vous devez crawler le bestiaire de Pathfinder pour aller chercher toutes les créatures. On s'intéresse ici au nom de la créature, et à la liste de ses sorts. Le caster level de la créature peut également être intéressant, mais c'est tout à fait optionnel. Plusieurs choix s'offrent à vous. Si vous crawlez avec le langage Scala, vous pouvez directement mettre vos créatures dans un tableau comme un ArrayBuffer et ensuite, vous n'avez qu'à transformer le tableau en RDD avec le SparkContext et le tour est joué.
   Sinon, si vous crawlez les créatures dans un fichier JSON, vous pouvez charger le contenu du fichier JSON avec l'API DataFrame JSON.
2. Maintenant que vous avez un RDD, vous pouvez faire des actions/transformations sur celui-ci. Vous allez utiliser l'API des RDD de Spark (entre autre) pour créer une batch view comme celle-ci (voir table en bas). Cette Batch View sera très utile pour Pito, car il va pouvoir visualiser rapidement les créatures qui peuvent le tirer d'affaire et faire son choix. Au niveau de l'architecture Lambda, cette Batch View irait dans une BD de Serving Layer (probablement une BD distribuée key-value).

   Vous allez faire un index inversé avec les créatures. Au lieu de présenter l'information sous la forme suivante :

   Nom créature => sort1, sort2, sort3

   On montre plutôt :

Nom sort => créature1, créature2, créature3

Pour faire la chose suivante, on peut utiliser les opérations flatMap et reduceByKey ou aggregateByKey. Attention aux doublons!
Vous pouvez également utiliser les fonctions explode et groupByKey si vous travaillez avec les DataFrame directement.

Voici quelques fonctions de l'API RDD qui seront sûrement utiles :
Map, flatmap (voir lien), reduceByKey.
reduceByKey et flatMap ressemblent beaucoup au map/reduce comme vu sur MongoDB. On peut utiliser reduceByKey lorsqu'on a un PairRDD. Pour avoir un pairRDD, il faut que chaque élément du RDD soit un Tuple2.

Exemple de Batch View (le RDD final après les traitements)
Voici votre information présentée sous forme d'index inversé.

Avant :

Créature	**Liste de sorts (tronquée)**
Solar	*etherealness*, *mass heal*, *miracle*, *storm of vengeance* *fire storm* ,*holy aura* ,*mass cure critical wounds* *destruction* , *dictum* , *ethereal jaunt*, *holy word* , *regenerate* ……..
Bralani	*blur*, *charm person*, *gust of wind*, *mirror image*, *wind wall*, *lightning bolt*, *cure serious wounds*

Après:

Sort	**Créatures qui ont le sort**
Cure light wounds	Solar, Planetar , Lillend
Cure moderate wounds	Solar, Planetar
Cure serious wounds	Movanic Deva, Monadic Deva, Bralani
Cure critical wounds	Solar, Planetar
Heal	Solar, Planetar Ghaele
Mass Heal	Solar
Banishment	Solar, Planetar

Voici un exemple fait par une ancienne équipe (Pommeret, Da Costa, Duminy).

Ensuite, une fois que votre RDD est complet. Vous pouvez le sauver sur votre disque avec **saveAsTextFile**(*path*) ou d'une autre façon. Faites un screenshot pour la remise.

Vous pouvez également “collect” le RDD sur votre driver program et ensuite le sauver sur le disque, l’envoyer dans une BD etc. Le RDD devient un Array du type de votre RDD lorsqu’il est ramené sur le driver program.

Vous devez sauver les infos suivantes sur chaque créature :

- Nom de la créature
- Liste des sorts préparés et spontanés. Vous pouvez les mélanger ensemble, dans le même array.
- Le caster level et le DC des sorts car c’est une information qui détermine la force des sorts, et la difficulté pour résister au sort. Mais c’est tout à fait optionnel. Le jeu de D&D est plutôt complexe pour les DCs des sorts car le DC peut être différent pour chaque sorts à cause de nombreux facteurs (école spécialisée, feats, niveau du sort 1,2...9) etc.

Voici un exemple en JSON.

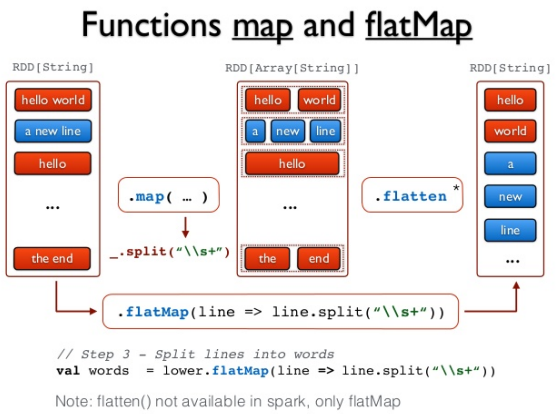
```
{
   "name":"Solar",
   "spells":[
      "cure light wounds",
      "cure moderate wounds",
      "cure serious wounds",
      "cure critical wounds",
      "heal",
      "mass heal"
   ]
}
```

Et voici un exemple avec une classe Scala. Le ArrayBuffer est une classe très utile en Scala.

```
class creature(val name : String) extends Serializable {
  var spells =  ArrayBuffer[String]()
 def addspell(spell : String) : Unit = {
    spells += spell
  }
}

var solar = new creature("Solar")
solar.addspell("cure light wounds")
solar.addspell("heal")
```

Petit rappel sur le fonctionnement de flatmap :



Exemple de stratégie possible pour le crawling :

Chacun des livres du Beastiary (Bestiaire) a une page. Sur la page, il y a des liens vers la fiche de chaque créature.
Donc vous pouvez aller sur chacune des pages de Bestiaire (il y a plusieurs Bestiaires) et vous crawler tous les liens que vous trouvez.
Pour chaque créature vous prenez son nom et vous allez prendre également la liste de tous ses sorts.
Ensuite avec Spark et les transformations/actions sur les RDD, vous faites la Batch View spell-> liste de créatures.

http://paizo.com/pathfinderRPG/prd/bestiary/monsterIndex.html
http://paizo.com/pathfinderRPG/prd/bestiary2/additionalMonsterIndex.html

Spell-Like Abilities (CL 20th)

Constant—*detect evil*, *detect snares and pits*, *discern lies* (DC 21), *true seeing*

At Will—*aid*, *animate objects*, *commune*, *continual flame*, *dimensional anchor*, *greater dispel magic*, *holy smite* (DC 21), *imprisonment* (DC 26), *invisibility* (self only), *lesser restoration*, *remove curse*, *remove disease*, *remove fear*, *resist energy*, *summon monster VII*, *speak with dead* (DC 20), *waves of fatigue*

3/day—*blade barrier* (DC 23), *earthquake* (DC 25), *heal*, *mass charm monster* (DC 25), *permanency*, *resurrection*, *waves of exhaustion*

1/day—*greater restoration*, *power word blind*, *power word kill*, *power word stun*, *prismatic spray* (DC 24), *wish*

Spells Prepared (CL 20th)

9th—*etherealness*, *mass heal*, *miracle*, *storm of vengeance* (DC 27)

8th—*fire storm* (DC 26), *holy aura* (2) (DC 26), *mass cure critical wounds* (2)

7th—*destruction* (DC 25), *dictum* (DC 25), *ethereal jaunt*, *holy word* (DC 25), *regenerate*

Ensuite au niveau du code source, vous avez la situation suivante :

```
href="/pathfinderRPG/prd/coreRulebook/spells/consecrate.html#consecrate" >consecrate</a></i>, <i><a
href="/pathfinderRPG/prd/coreRulebook/spells/cureModerateWounds.html#cure-moderate-wounds" >cure moderate wounds</a></i> (2), <i><a
```

Donc, chaque fois que dans un href on voit /spells/ on peut être sûr à 100 % que c'est un lien vers un spell. Et donc on récupère "Cure Moderate Wounds" comme un spell pour cette créature.

Suite de l'exercice 1

Introduction

Historiquement dans ce cours, l'exercice 1 est très bien réussi et l'exercice 2 est plus difficile. Et puisque les équipes de 4 ont l'habitude de séparer le travail en deux, l'équipe 1 se retrouve avec moins de travail. Cette année, j'ai décidé d'améliorer la difficulté de l'exercice 1, tout en étant équitable par rapport à la quantité de travail supplémentaire que cela représente. Sachez qu'au niveau de la correction, je vais distribuer de nombreux points bonus aux équipes qui implémentent ces éléments, ou une partie de ces éléments. Il devient donc possible pour une équipe qui a un super bon exercice 1 mais un mauvais exercice 2 d'avoir une très bonne note.

Dans cette partie supplémentaire, vous allez construire un vrai outil de recherche pour naviguer les données des sorts de Pathfinder.
Vous avez normalement les sorts, et les créatures de Pathfinder. Pourquoi ne pas avoir un outil qui permet de chercher des choses sur les deux? Je vous demande donc de réaliser un tel outil. L'outil existant Advanced Spell Search est déjà très bon, mais il a de nombreux problèmes.

1. La recherche dans le texte est très mauvaise et ne marche presque pas du tout.
2. On ne peut pas voir les créatures qui possèdent les sorts que l'on recherche. Très embêtant si on joue une classe comme Summoner!
3. L'onglet Advanced Output fonctionne mal car les sommaires sont limités aux résultats qui contiennent au maximum 50 sorts. Cette limite est très décevante également.

Voici l'outil que vous suggère d'implémenter. Vous pouvez faire ça différemment, mais c'est ma vision à moi :

1. Interface graphique pour utiliser l'application.
2. On peut trier les sorts avec les choses suivantes :
    a. Recherche texte dans le nom du sort
    b. Composantes (V,S,M)
    c. Classe du sort
3. On peut chercher dans le texte de la description du sort.
4. Une fois qu'on a notre liste de résultats, la liste des sorts qui match nos critères, on peut cliquer sur un sort pour ouvrir directement sa page web.
5. On affiche également une liste de créatures qui possèdent les sorts sur la liste, ou alors qui possèdent au moins un sort sur cette liste (votre choix ici). On peut également cliquer sur la créature pour ouvrir sa page.
6. Afficher une image par rapport au sort, ou par rapport à la créature.

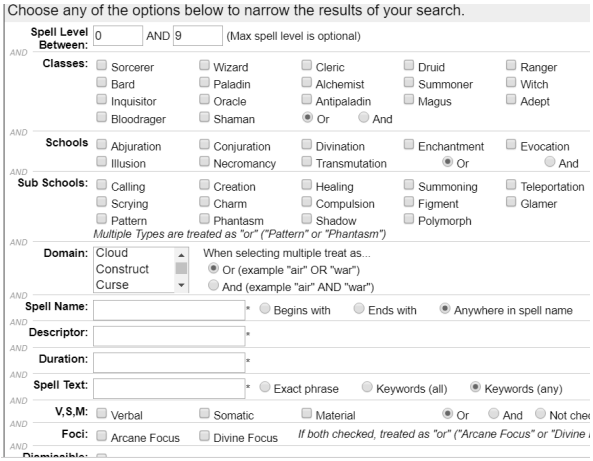
Voici des images pour les sorts (reddit, fichier RAR de 1 go+)
Ici aussi (Google drive)

1. *Interface graphique web ou desktop pour utiliser votre base de données de sorts/monstres.*

Vous pouvez utiliser la technologie de votre choix. Moi par exemple, je connais Websockets et formulaires HTML + Events DOM. Vous pouvez parler avec votre processus Scala avec des events Websockets.
Vous pouvez également utiliser une autre base de données (MongoDB, MySQL, PostgresSQL) pour faire le full-text search et les autres requêtes.

2. *Vous devez pouvoir chercher des sorts en utilisant plusieurs critères pour affiner la recherche*

Voir Advanced Spell Search pour un exemple.
École du sort : (Transmutation, Conjuration, Illusion, Nécromancie etc)
Les classes qui peuvent posséder le sort : Ranger, Wizard, Inquisitor, Cleric etc
Verbal / Somatic / Material.



3. *Implémenter une forme de full-text search.*

Les descriptions de certains sorts peuvent être très longues. Et c'est parfois dans la description du sort qu'on trouve la perle rare.
Lorsque vous irez chercher les informations sur les sorts, enregistrez également le contenu de la description.

Un exemple d'utilisation. On veut chercher les sorts qui donnent des "morale bonus".
On écrit donc morale bonus dans le champs texte et on clique sur "Rechercher les sorts".
On lance maintenant une recherche distribuée sur notre RDD de sorts. La tâche étant distribuée, de nombreux processeurs vont se mettre à rechercher dans les descriptions de texte pour les mots "morale" et "bonus". Lorsque ces deux termes sont présents dans une description, nous avons un match. On crée un RDD avec tous les match trouvés.

On devrait trouver les sorts suivants parmi les résultats. Chacun de ces sorts donne un "morale bonus".

1. Heroism
2. Heroism Greater
3. Rage

Ensuite, sur ce RDD, on applique une autre transformation. On avait coché une option pour n'avoir que des sorts disponibles pour les alchimistes. On doit donc parcourir notre RDD et

filtrer pour enlever les sorts qui ne sont pas pour les alchimistes. On enlèverait donc *Heroism Greater* du RDD.

Lorsqu'on lance la requête, vos résultats doivent montrer la liste des sorts qui sont satisfaisants pour les critères. Lorsqu'on clique sur un sort dans la liste des résultats, on doit pouvoir voir une liste de Monstres qui possèdent ce sort. Ça serait même pas mal d'afficher un petit résumé du monstre lorsqu'on clique sur un monstre dans la liste affichée.

Comment implémenter tout ça?
Vous pouvez garder toutes vos données dans des RDDs Spark et pré-programmer des requêtes Spark.

SparkRDD ← Requête formulée avec flatmap, map, join, reduce, aggregate etc

Ou alors

SparkDataFrame ← Requête avec SparkSQL sur le DataFrame/Dataset.

4. *Afficher les créatures du bestiaire qui possèdent les sorts.*

Par exemple, une recherche dont le résultat donne le sort heal nous affiche également les créatures suivantes :

1. Angel, Solar
2. Azata, Ghaele
3. Trumpet Archon
4. Angel, Monadic Deva

Ces outils existants seront peut-être une source d'inspiration pour vous :
Advanced Spell Search, outil de Paizo
Beastiary Search, outil de Paizo
Voir cette présentation
Ce pdf aussi