Activité intégrative 2021

Itération 3

Durée : 10 heures

Cette itération a pour but d'implémenter plusieurs algorithmes parcourant les paragraphes du livre-jeu afin de produire des relevés (*Statement*). Ces relevés renseignent sur la structure du livre-jeu.

Planification de l'itération

1 : découverte et présentation de l'énoncé (30 min)

En grand groupe, le responsable présente les fonctionnalités attendues.

2 : ajout de la classe CheckSuperviserTests et implémentation d'interface (15 min)

Ajoutez le fichier `gamebook.supervisers.CheckSuperviserTests`, disponible sur HELMo-Learn. Son premier test vérifie que votre classe `CheckSuperviser` implémente l'interface `BookEditedEventHandler` définie lors de l'itération 2.

Implémentez cette interface !

3 : conception de l'application (1 heure 10)

Vous trouverez en fin de document la description des relevés attendus pour chaque groupe de laboratoire. Ces relevés prendront la forme de différentes classes implémentant la même interface **`gamebook.domains.GameBookStatement`** que vous devez concevoir :

- [10 min] Individuellement, parcourez les descriptions des relevés et concevez l'interface [1]`GameBookStatement`. À cette fin, consultez la description de la US unique de l'itération 3, ainsi que les méthodes de l'interface `gamebook.supervisers.CheckView`.
- [15 min] En petits groupes, mettez vos définitions en commun, imaginez les classes qui implémenteront votre interface, indiquez la classe qui créera les objets correspondants et comment ces objets seront communiqués au superviseur.
- [15 min] Pour chaque relevé à implémenter, imaginez au moins 3 exemples. Le premier partira du livre fourni dans l'énoncé général, les autres partiront d'autres livres à imaginer. Essayez de trouver des exemples intéressants (livres comportant des cycles, livres avec paragraphes inaccessibles, etc.).
- [30 min] En grand groupe, un ou plusieurs groupes présentent leurs solutions afin d'identifier d'éventuels problèmes de conception.

4 : implémenter et tester l'application (8 heures)

Chaque groupe implémentera les relevés repris dans la table ci-dessous. Les lettres correspondent aux lettres associées aux relevés dans la section Relevés.

Groupe	Relevé de base	Relevé exploitant la recherche en largeur
1	E	X
2	F	Y
3	A	Z
4	B	X
5	C	Y

[1] Pour rappel, une interface est une liste de méthodes publiques et abstraites qui décrivent comment collaborer avec un objet : une interface ne possède pas d'attributs d'objet.

6	D	Z

AI-3.1 : consulter des relevés

En tant que vérificateur, je souhaite consulter des relevés pour valider le livre avant sa publication.

Détails

- Le programme peut afficher un nombre quelconque de relevés
- Les relevés sont créés au lancement de l'application et ne changent plus pendant l'exécution… ce qui ne veut pas dire que leurs états sont figés 😊
- Tout relevé possède un nom générique, c'est-à-dire qu'il est indépendant de l'état du livre-jeu ou d'éléments aléatoires
- Tout relevé possède une description qui peut varier selon l'état du livre-jeu ou d'autres éléments aléatoires
- Les résultats du relevé dépendent de l'état du livre-jeu et, éventuellement, d'éléments aléatoires spécifiques (paragraphe pris au hasard par exemple)

Tests d'acceptation

Contrairement aux itérations précédentes, vous devrez compléter les tests d'acceptation fournis. À cette fin, n'hésitez pas à créer des relevés factices qui retourneront des résultats prévisibles.

- ✓ La vue affiche les résultats des relevés dès le démarrage du programme.
- ✓ Quand le livre-jeu est édité (ajout de nouveaux §, modification d'un choix, etc.), le programme met à jour les résultats des relevés.
- ✓ Quand un relevé n'a aucun résultat à présenter, le programme affiche le résultat fictif « Aucun résultat ».
- ✓ Quand l'utilisateur appuie sur le bouton « Analyse », le programme met à jour les résultats des relevés (ce bouton sert principalement pour les relevés aléatoires)

Relevés

Relevés de base

Ces relevés sont calculés en parcourant le livre-jeu sans ordre précis.

A. Relevé des § ne figurant dans la destination d'aucun choix

La description doit mentionner le titre du livre jeu et le nombre de § trouvés.

Chaque élément du résultat doit mentionner la position du § dans le livre-jeu suivie de son contenu.

B. Relevé du nombre de fois que chaque § figure dans une destination

La description doit mentionner le titre du livre jeu.

Chaque élément du résultat doit mentionner la position du § dans le livre-jeu suivie du nombre de fois correspondants et du contenu du § (exemple : 1 (0). Vous êtes en…).

C. Relevé des § contenant un choix vers un § pioché aléatoirement

La description doit mentionner le titre du livre jeu, la position du § pioché aléatoirement et le nombre de § trouvés.

Chaque élément du résultat doit mentionner la position du § dans le livre-jeu suivie du contenu du §.

D. Relevé des § contenant pas de choix vers un § pioché aléatoirement

La description doit mentionner le titre du livre jeu, la position du § choisi et le nombre de § trouvés.

E. Chaque élément du résultat doit mentionner la position du § dans le livre-jeu suivie du contenu du §.Relevé des § ne contenant pas de choix sur eux-mêmes

La description doit mentionner le titre du livre jeu et le nombre de § trouvés.

Chaque élément du résultat doit mentionner la position du § dans le livre-jeu suivie du contenu du §.

F. Relevé des § résultant d'une lecture du livre jeu en effectuant des choix aléatoires, jusqu'à arriver à un § terminal.

La description doit mentionner le titre du livre jeu, le nombre de § lus.

Chaque élément du résultat doit mentionner la position du § dans le livre-jeu suivie du contenu du §.

On veillera à ce que le relevé respecte l'ordre de la lecture aléatoire.

Relevés exploitant la recherche en largeur

Les relevés qui suivent exploitent une table permettant de trouver les plus courts chemins depuis le § de départ.

X. Trouver le plus petit des plus courts chemins vers un § terminal

La description doit mentionner le titre du livre, le nombre de § traversés et le numéro du paragraphe ciblé.

Chaque élément du résultat doit mentionner la position du § dans le livre-jeu suivie du contenu du §. ==> le tableau chelou

On veillera à ce que le résultat respecte l'ordre imposé par le chemin.

Y. Trouver le plus grand des plus courts chemins vers un § terminal

La description doit mentionner le titre du livre, le nombre de paragraphes traversés et le numéro du paragraphe ciblé.

Chaque élément du résultat doit mentionner la position du § dans le livre-jeu suivie du contenu du §.

On veillera à ce que le résultat respecte l'ordre imposé par le chemin.

Z. Trouver le plus court chemin vers un § pioché aléatoirement

La description doit mentionner le titre du livre, le nombre de paragraphes traversés et le numéro du paragraphe ciblé.

Chaque élément du résultat doit mentionner la position du § dans le livre-jeu suivie du contenu du §.

On veillera à ce que le résultat respecte l'ordre imposé par le chemin.

Calcul du plus court chemin dans un graphe orienté

Un livre-jeu peut être vu comme un graphe orienté où les sommets correspondent aux § et les arcs représentent les choix (on peut libeller chaque arc avec le libellé du choix).

Dans cette représentation, le parcours d'un livre-jeu en largeur (*Breadth First Search – BFS*) consiste à parcourir tous les § du livre-jeu en partant du premier, puis en explorant ses § voisins, ensuite les voisins des voisins, etc. jusqu'à ce que tous les § aient été parcourus. Pendant ce parcours, nous construisons une table qui sert ensuite à construire les chemins les plus courts du § de départ vers tous les autres § du livre.

Pour illustrer son fonctionnement, nous partons du livre-jeu donné en exemple.

Etape 1 – Parcours du livre-jeu

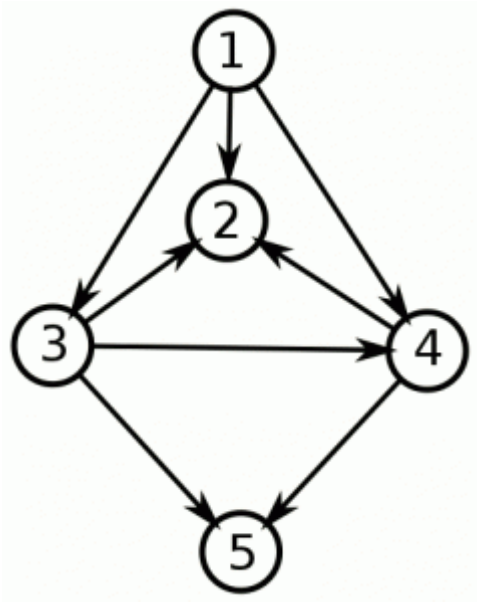
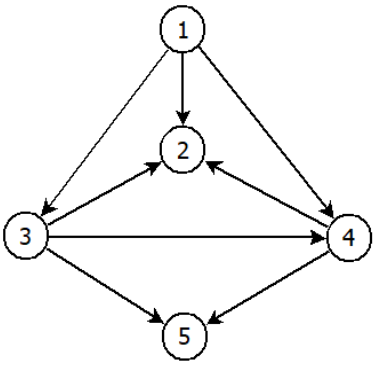


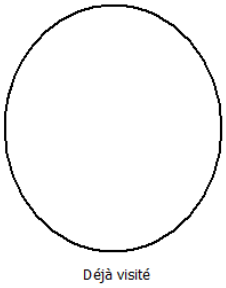
Figure 1 livre-jeu d'exemple modélisé comme graphe

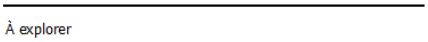
Situation de départ

Nous devons créer deux collections et initialiser une table pour effectuer le parcours et calculer les plus courts chemins.

La première collection contient les § déjà visités. Elle évitera notamment d'explorer indéfiniment des §. La seconde collection mémorise les § suivants à explorer. **Cette deuxième collection demande de respecter l'ordre des ajouts.**







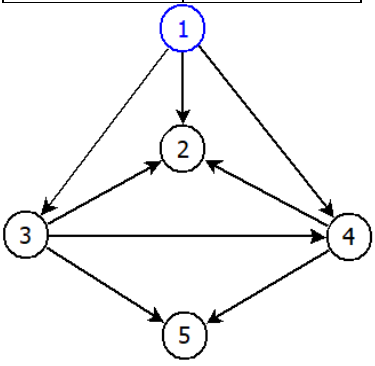
La table mémorise, pour chaque §i, son § précédent dans le plus court chemin allant du 1er paragraphe jusqu'au §i.

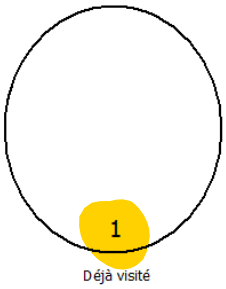
Au départ, tous les éléments de la table sont initialisés à null.

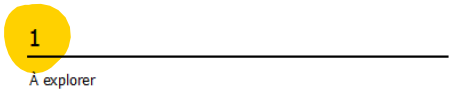
Paragraphe	Précédents
1	Null
2	Null
3	Null
4	Null
5	Null

Nous débutons le parcours par le 1er §. Pour le 1er§, le § précédent est évident : il n'y en a pas. Nous gardons la valeur null. Nous plaçons le 1er§ dans la collection des nœuds à explorer et l'ajoutons à la collection des nœuds visités.

Paragraphe	Précédents
1	Null
2	Null
3	Null
4	Null
5	Null







Une fois l'état initial défini, nous entrons dans une boucle qui se répète jusqu'à ce qu'il n'y ait plus de paragraphes à explorer.

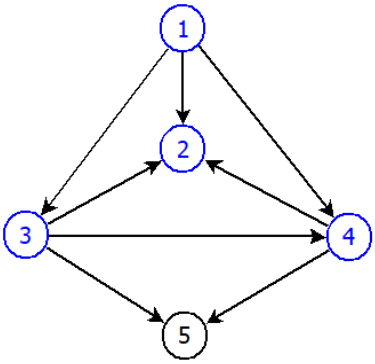
Explorations des voisins

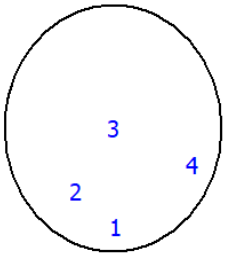
Nous devons explorer le premier § disponible dans les § à explorer. Nous le retirons de cette collection et y ajoutons tous ses voisins qui n'ont pas été déjà visités.

Pour chaque voisin non visité, nous mettons à jour les entrées correspondantes de la table. Le précédent d'un voisin est le dernier § retiré des § à explorer. Dans notre exemple, il s'agit du § 1.

Paragraphe	Précédents
1	Null
2	**1**
3	**1**
4	**1**
5	Null

Nous ajoutons les nouveaux § visités à nos collections. Dans la figure suivante, remarquez que §1 a été retiré des § à explorer, mais qu'il reste dans les § déjà visités. Dans notre exemple, le §2 est le prochain § à explorer. Cependant, l'ordre d'ajout parmi les voisins n'est pas important.



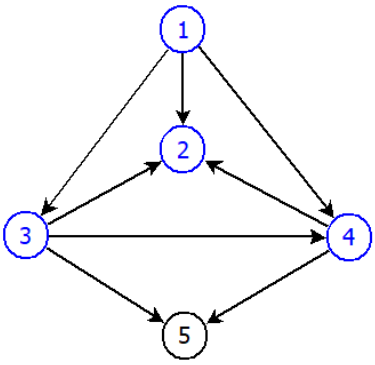


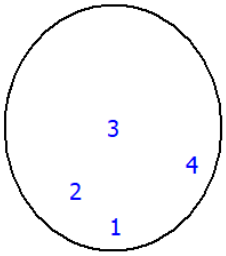
Déjà visité

2 4 3

À explorer

Nous retirons le prochain § à explorer : il s'agit du §2. Ce dernier est terminal : il n'y a pas de nouveaux nœuds ajoutés et la table n'est pas mise à jour.





Déjà visité

4 3

À explorer

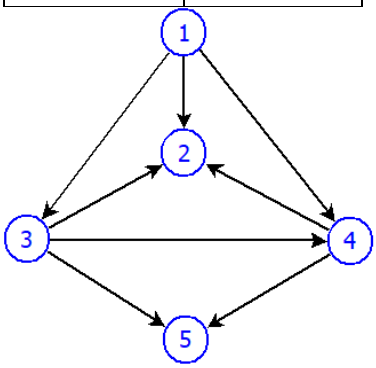
Le prochain § à explorer est le §4. Ce dernier a pour voisin le §2, déjà visité, et le §5. Ce dernier n'étant pas encore visité, nous l'ajoutons aux collections et mettons à jour nos tables en conséquence.

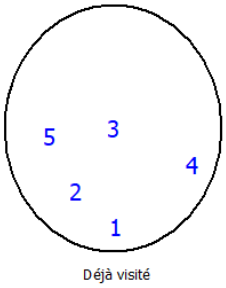
Paragraphe	Précédents
1	Null
2	1

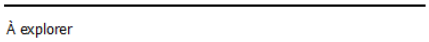
3	1
4	1
5	**4**

Il reste deux § à explorer. Cependant, ces § ont déjà été visités : nous ne modifierons plus nos tables, ni la collection des § déjà visités. Voici la situation en fin d'exploration.

Paragraphe	Précédents
1	Null
2	1
3	1
4	1
5	4







À ce stade, nous pouvons passer à la construction des plus courts chemins.

Etape 2 - Construction du plus court chemin

La table des précédents permet de construire le plus court chemin allant du §1 à un autre §. Pour ce faire, il faut ajouter le précédent du § de destination au début du chemin, puis d'ajouter le précédent du précédent au début du chemin et de poursuivre jusqu'à ne plus avoir de précédent.

Pour illustrer, construisons le plus court chemin du §5. Au départ, ce chemin est composé de §5, c'est-à-dire le § de destination.

5

[Repétition 1] Comme il n'est pas null, nous ajoutons le précédent de §5 au début du chemin.

Paragraphe	Précédents
1	Null
2	1
3	1
4	1
5	**4**

4	5

[Répétition 2] Comme il n'est pas null, nous ajoutons le précédent de § 4 au début du chemin.

Paragraphe	Précédents
1	Null

2	1
3	1
4	1
5	4

1	4	5

[Sortie] Comme son précédent est null, nous avons terminé la construction du chemin pour le §5.

Consignes et contraintes

Algorithmique

Algo 1 – Plans de test

Rédigez un plan de tests pour le relevé de base :

- Ecrivez les tests dans la classe de tests correspondant à votre relevé de base, sous le commentaire de titre « PLAN DE TEST »
- Les noms des tests doivent indiquer clairement ce qui est testé
- Des titres (commentaires) doivent structurer vos tests. Pour rappel, un plan de test doit absolument comporter :
  - Un échantillon des différents cas « normaux » distingués par l'algorithme,
  - Un échantillon des différents cas d'erreurs reconnues et traitées par le programme,
  - Les cas limites, en particulier, celui de l'ensemble vide ou de la valeur 0,
  - Un maximum de combinaisons de cas.

Algo 2 – Réponses aux questions

Répondez aux questions ci-dessous en commentaires dans votre code source. Pour chaque question, nous indiquons l'endroit où la réponse doit être écrite.

Choix de collections

Le calcul du plus court chemin nécessite 3 collections principales pour

1. Mémoriser les § déjà visités (Etape 1)
2. Stocker les § suivants à explorer (Etape 1)
3. Représenter un chemin de § (Etape 2)

Pour chacune de ces 3 collections, répondez aux questions suivantes : **OK**

- Quelle interface de collection (`List, Queue, Deque, (Sorted)Set, (Sorted)Map, etc.`) allez-vous choisir pour mémoriser les paragraphes ? Justifiez votre choix en rappelant la caractéristique principale de l'interface choisie, et en identifiant les principales méthodes dont vous aurez besoin. Avez-vous notamment besoin d'accéder à un élément précis ? Si oui, sur base de quelle sorte de clé ?
- Quelle implémentation de collection (`ArrayList, LinkedList, ArrayDeque, HashSet, TreeMap, etc.`) allez-vous choisir pour mémoriser les paragraphes ? Justifiez votre choix en déterminant les CTT des méthodes mentionnées plus haut, et en comparant avec la/les autre(s) implémentation(s) possible(s).

Répondez à cette question, pour les 3 collections, dans la Javadoc de la méthode effectuant le calcul de chemin le plus court.

Post-conditions de la méthode de production du relevé de base

Dans la Javadoc de GameBookStatement, indiquez les post-conditions de l'opération de validation du livre pour le relevé de base. En d'autres termes, qu'est-ce qui fait que le nom, la description et les résultats obtenus sont bien corrects ? Quelles conditions seront respectées par le nom, la description et les résultats ? **OK**

CTT de la production du relevé de base

Pour répondre à cette question, identifiez les boucles, les imbrications, mais aussi les collections utilisées et leurs opérations. Quand vous répondez, n'oubliez pas d'indiquer à quoi correspondent vos libellés N,M, etc. et d'expliquer les étapes de l'algorithme. Si celui-ci contient des « if », indiquez quel est le pire des cas et donnez uniquement la CTT pour celui-ci. **OK**

Répondez à cette question dans la Javadoc de la classe du relevé de base.

CTT du calcul du plus court chemin

Voici la CTT à évaluer selon votre groupe de labo :

Groupe	CTT de
1	Etape 1 de l'algorithme
2	Etape 1 de l'algorithme
3	Etape 1 de l'algorithme
4	Etape 2 de l'algorithme
5	Etape 2 de l'algorithme
6	Etape 2 de l'algorithme

OK

Pour répondre à cette question, identifiez les boucles, les imbrications, mais aussi les collections utilisées et leurs opérations. Quand vous répondez, n'oubliez pas d'indiquer à quoi correspondent vos libellés N,M, etc. et d'expliquer les étapes de l'algorithme. Si celui-ci contient des « if », indiquez quel est le pire des cas et donnez uniquement la CTT pour celui-ci.

Répondez à cette question dans la Javadoc de la méthode effectuant le calcul de chemin le plus court.

Programmation Orientée Objet

Définissez l'interface **`gamebook.domains.GameBookStatement.`** Votre interface ne doit trahir aucun détail d'implémentation : réfléchissez aux types des résultats de vos méthodes en conséquence.

Définissez une classe par relevé dans un package **`gamebook.domains.statements`**. Vos classes doivent implémenter l'interface `GameBookStatement`. Vos classes doivent être correctement encapsulées : si vous retournez des collections mutables, veillez à retourner des copies de ces collectons, etc. Si vous remarquez que certaines instructions se répètent parmi vos relevés, n'hésitez pas à factoriser ce code en définissant une classe mère…

Vous devez adaptez le constructeur du `CheckSuperviser` afin que ce dernier reçoive des objets implémentant l'interface `GameBookStatement`. Le `CheckSuperviser` sera indépendant des classes implémentant `GameBookStatement`.

Au niveau des tests unitaires du CheckSuperviser, nous vous recommandons de créer une classe factice FakeStatement qui implémente votre interface GameBookStatement et de transmettre des objets de cette classe factice au CheckSuperviser. Les objets de cette classe pourront retourner des résultats prédéfinis, ce qui facilitera vos tests. Complétez les méthodes `CheckSuperviser.verifyViewRefreshedFor Statement(GameBookStatement stmt)` et `CheckSuperviser. verifyStatementRefreshed()` en conséquence.