IUT - Institut universitaire de technologie de Lille | Université de Lille

BUT Informatique, R2.07 Graphes, 2021–2022

TP Modélisation par graphe de configurations

Contact: iovka.boneva@univ-lille.fr

Objectifs: À l'issue du ce TP vous devriez être capable de:
- définir un graphe implicite correspondant à un graphe de configurations,
- modéliser et résoudre un problème en utilisant un graphe de configurations et un parcours.

1 Modélisation par un parcours dans un graphe de configurations

Prise en main des nouvelles fonctions de Grapp Dans la feuille d'exercices dédiée à Grapp, faire
- l'exercice « Définition d'un graphe implicite » de la section « Graphe implicite : `load -implicit` ». (Ne pas faire l'exercice « Graphe implicite valué ».)
- l'exercice de la section « Recherche de chemin : `find-path` ».

** Exercice 1.1 : Résolution du casse-tête loup chèvre chou

Ce casse-tête a été étudié en cours. Rappelons les éléments du graphe de configurations qui permet de le résoudre :
- Une configuration indique lesquels des personnages sont sur la rive gauche et lesquels sont sur la rive droite. La configuration est représentée par une chaine de longueur quatre, où chaque position correspond à un personnage :
  - berger : position 0
  - loup : position 1
  - chèvre : position 2
  - chou : position 3

  Par exemple, `"GDGD"` correspond à la configuration où le berger et la chèvre sont sur la rive gauche et le loup et le chou sont sur la rive droite.
- Les successeurs d'une configuration sont obtenus en faisant le berger traverser la rivière, seul ou avec un autre personnage. Par exemple, les successeurs de `"GDGD"` sont :
  - `"DDGD"` le berger traverse seul,
  - `"DDDD"` le berger traverse avec la chèvre.
- La configuration initiale est `"GGGG"`, càd tous les personnages sur la rive gauche.

Q1: Récupérez le fichier `LoupChevreChou.java` qui contient une implémentation incomplète d'un graphe implicite.

Q2: Complétez la fonction `initialNodes` en ajoutant la configuration initiale aux sommets initiaux.

Q3: Que fait la fonction `sommet` ?

Q4: Que fait la fonction `configurationPerdante` ?

Q5: Que fait la fonction `lautreRive` ?

Q6: Compléter la fonction `successors` pour inclure les cas de figure manquant.

Q7: Tester votre implémentation en utilisant la fonction `main` avec des tests qui vous semblent pertinents. Corriger si nécessaire.

Q8: Charger le graphe implicite ainsi défini.

Q9: Effectuer un parcours en largeur d'abord depuis la configuration initiale.

Q10: Quelle est la configuration finale qui correspond à la solution du casse-tête ?

Q11: Utiliser la commande `find-path` sur l'arbre de parcours pour trouver le chemin dans le graphe des configurations qui correspond à une solution.

Q12: En déduire la suite d'actions à effectuer par le berger pour traverser la rivière.

** Exercice 1.2 : Résolution du casse-tête des seaux

Ce casse-tête et sa modélisation par graphe de configurations a été étudié en TD. Vous devez maintenant trouver la solution du casse-tête en implémentant le graphe implicite des configurations et en effectuant un parcours dans ce graphe.

Q13: Créer le fichier `Seaux.java` qui contient une implémentation de l'interface `ImplicitGraph`.

Q14: Implémenter les méthodes `type` et `name`.

Rappelons qu'une configuration pour le problème des seaux est donnée par deux entiers, alors que les sommets d'un graphe implicite sont des chaines de caractères. Comme les entiers sont compris entre 0 et 5, ils peuvent être

représentés par un seul caractère chacun. Donc, les sommets du graphe pourront être des chaines de longueur deux contenant deux chiffres, par ex. `"00"`, `"35"`.

Q15: Implémenter la méthode `initialNodes` qui retourne la liste de configurations initiales pour le casse-tête.

Q16: Écrire une méthode `sommet` qui prend en paramètre une configuration, càd deux entiers représentant les contenus des deux seaux, et retourne le sommet correspondant, càd une chaine de longueur deux.

Il vous reste à faire la méthode `successors`.

Q17: Dans la méthode `successors`, déclarer deux variables `int petitOld` pour le contenu du petit seau et `int grandOld` pour le contenu du grand seau. Calculer les valeurs de ces variables pour la configuration donnée en paramètre.

Il faut maintenant énumérer toutes les configurations successeurs de la configuration donnée en paramètre. Pour cela, vous devez considérer les six actions possibles (remplir, vider, transvaser), chacune de ces actions donnant lieu à une configuration à ajouter à la liste des successeurs.

Q18: Écrire la fonction `successors`. N'oubliez pas de la tester à l'aide d'une fonction `main`.

Q19: Charger le graphe implicite correspondant au casse-tête des seaux.

Q20: Effectuer un parcours en largeur d'abord depuis la configuration initiale.

Q21: Visualisez le résultat du parcours.

Vous constatez que le graphe est suffisamment grand pour qu'il soit difficile de trouver le chemin visuellement.

Q22: Trouver un chemin depuis la configuration initiale jusque la configuration finale à l'aide de la commande `find-path`.

Q23: Traduire le chemin trouvé en une suite d'actions qui permettent de résoudre le casse-tête.

* Exercice 1.3 : Résolution du casse-tête des cavaliers

Ce casse-tête a été étudié en TD, ainsi que sa modélisation par un graphe de configurations. Vous devez maintenant trouver la solution à l'aide d'un parcours dans un graphe implicite, en suivant les mêmes étapes que pour les exercices précédant.

Q24: Créer la classe `Cavaliers` qui implémente `ImplicitGraph` et implémenter les méthodes `name` et `type`.

Rappelons qu'une configuration dans ce casse-tête est donnée par quatre entiers compris entre 0 et 9.

Q25: Comment allez-vous représenter une configuration par une chaine de caractères (càd un sommet) ? Écrire la méthode `String sommet(int posB1, posB2, posN1, posN2)` qui retourne le sommet correspondant à la configuration donnée par les quatre positions des cavaliers.

Q26: Implémenter la méthode `initialNodes`.

Pour calculer les successeurs d'une configuration, nous allons définir deux méthodes d'aide :

- la méthode `List<Integer> deplacements(int positionCavalier)` qui pour chaque position possible d'un cavalier retourne la liste des positions vers lesquelles ce cavalier peut se déplacer ;
- la méthode `boolean sommetCorrect(int posB1, int posB2, int posN1, int posN2)` qui retourne vrai si et seulement si les positions des cavaliers données en paramètre correspondent à une configuration correcte. Rappelons qu'une configuration est correcte si tous les cavaliers se trouvent à des positions différentes.

Q27: Écrire la méthode `successors`, que vous n'oublierez pas de tester.

Q28: Charger le graphe implicite ainsi défini et faire un parcours depuis la configuration initiale.

Rappelons que ce problème a quatre configurations finales possibles.

Q29: Pour chacune des configurations finales, calculer le chemin depuis la configuration initiale. Garder la solution la plus courte (qui correspond au plus petit nombre de déplacements de cavaliers).

Q30: La solution étant trop longue, on ne va pas la construire entièrement. Il vous est demandé de construire uniquement les 5 premiers déplacements de la solution.

Il est possible de résoudre le casse tête par n'importe quel parcours. Essayons avec un parcours en profondeur d'abord.

Q31: Effectuer un parcours en profondeur d'abord dans le graphe de configurations, puis trouver le chemin depuis la configuration initiale jusque la configuration finale donnée par le parcours en profondeur d'abord. Quelle est la longueur de la solution trouvée ?

Q32: Pourquoi un parcours en largeur est-il plus intéressant qu'un parcours en profondeur pour résoudre tous les problèmes considérés jusque présent dans ce TP ?

2 Optimisation par programmation dynamique

Prise en main des nouvelles fonctions de Grapp Faire l'exercice « Graphe implicite valué » de la section « Graphe implicite : `load -implicit` ».

*** Exercice 2.1 : Résolution du problème du sac à dos**

Nous retournons sur le problème du sac à dos vu en TD, pour calculer sa solution. Terminer d'abord l'exercice de TD correspondant, si ce n'est pas déjà fait.

Nous allons opérer quelques changements dans la modélisation, pour la rendre plus simple à programmer (mais aussi un peu plus plus compliquée à comprendre :

— Une configuration (càd un sommet du graphe) sera donnée par deux nombres : (x, y) est la configuration du niveau x dans laquelle le sac contient y litres. La configuration initiale correspondant au sac vide est alors $(0, 0)$. Seul le sommet correspondant au sac plein sera représenté différemment, par la chaine "plein".

— Le niveau d'une configuration indique quel article on doit choisir depuis cette configuration (et donc tous les articles précédents ont déjà été choisis). On choisit l'article A au niveau 0, l'article B au niveau 1, et l'article C au niveau 2. Au niveau 3 on a déjà choisi tous les articles, il suffit alors d'ajouter des arêtes de poids 0 vers le sommet "plein".

Q33: Récupérer sur Moodle la fichier `SacADos.java` qui contient la solution du problème du sac à dos.

Q34: Charger et visualiser le graphe.

Vous pourriez remarquer que les sommets du niveau 3 ne sont pas les même que dans la solution faite en TD, pour deux raisons :

— D'une part, en TD nous avions fait une optimisation « à la main » en évitant de prendre trop peu d'articles de type C. Cette optimisation n'est pas faite par le programme, qui adopte une approche générique. Donc, il y a des sommets dans le graphe implicite qui ne sont pas présents dans le graphe fait en TD.

— D'autre part, la modélisation adoptée pour le graphe implicite est plus succincte, car un sommet ne porte plus l'information des articles déjà choisis, mais uniquement le poids total du sac. En effet, l'information des articles choisis n'est pas utile, car elle peut être reconstruite à partir des chemins. Ainsi, dans le graphe implicite, tous les sommets ayant le même volume sont fusionnés. Il y a donc des sommets qui sont présents dans le graphe fait en TD, mais pas dans le graphe implicite.

Q35: Résoudre le problème du sac à dos en calculant un chemin maximal depuis le sommet "_0_0" jusqu'au sommet "plein".

Q36: Utiliser la commande `find-path` pour trouver le chemin optimal correspondant.

Q37: En déduire la solution du problème du sac à dos, c'est à dire combien d'objets il faut prendre de quel type.

Considérons maintenant une autre situation et un autre sac. Supposons que nous disposons d'un sac de 25 litres, et des articles suivants :

Article	volume unitaire	valeur nutritive
A	7	15
B	4	11
C	2	4
D	3	10

Q38: Modifier le fichier `SacADos.java` pour calculer la solution de ce nouveau problème. Astuce : vous devriez modifier uniquement les constantes au début du fichier, qui décrivent les données du problème.

Q39: À partir du chemin trouvé, reconstruire la solution : quelle quantité faut-il prendre de quel article ?

*** Exercice 2.2 : Programmation dynamique : problème d'investissement**

On dispose de six millions à investir dans trois régions. Le Tableau 1 donne les bénéfices résultant des sommes investies (on supposera que les investissements se font en nombre entier de millions, et qu'il n'est pas possible d'investir plus que 4 millions dans la même région).

	Région A	Région B	Région C
1 million	0,2	0,1	0,4
2 millions	0,5	0,2	0,5
3 millions	0,9	0,8	0,6
4 millions	1	1	0,7

TABLE 1 – Bénéfices escomptés par région en fonction du montant investi.

On cherche à déterminer la politique d'investissement de manière à maximiser les gains. Nous allons utiliser un graphe de configurations à niveaux, similaire à celui utilisé pour le problème du sac à dos.

Une configuration est définie par une paire de nombres (x, y) :

— x est le numéro du niveau, 0, 1, 2 ou 3 correspondant au nombre de régions pour lesquelles on a déjà choisi la somme à investir,

— y est le nombre de millions déjà investis.

Par exemple, la configuration $(2, 4)$ correspond à l'état où on a déjà investi 4 millions en globalité dans les régions A et B (il reste donc 2 millions à investir dans la région C).

Les arêtes correspondent à un choix de somme à investir dans la prochaine région, pour passer d'un niveau au niveau suivant. Le poids d'une arête est le gain attendu par la somme investie.

Q40: Définir un graphe implicite qui correspond à la modélisation de ce problème.

Q41: Résoudre le problème en cherchant un plus long chemin dans le graphe de configurations.

Q42: En déduire quelle somme investir dans quelle région pour optimiser les gains.