ENSTA Paris
2023-2024

Résolution de problèmes algorithmiques – IN103

Alexandre Chapoutot

Feuille d'exercices 5

Objectif(s)

- Manipuler des graphes ;
- Écrire un parcours de graphes pour résoudre des problèmes comme trouver la sortie d'un labyrinthe ou calculer l'existence d'un chemin ;
- Introduction au problème de la coloration de graphes.

PRÉPARATION

Cette première partie permet de préparer votre environnement (**si vous ne l'avez pas fait au dernier TD**) de travail afin de pouvoir utiliser facilement la bibliothèque logicielle **libin103** spécialement développée pour cet enseignement.

1. A la racine de votre compte, créez un répertoire nommé `Library` s'il n'a pas déjà été créé, puis placez vous dans ce répertoire.

   ```
   mkdir ~/Library; cd ~/Library
   ```

2. Téléchargez l'archive `libin103-1.4.tar.gz` sur le site du cours

   ```
   wget https://perso.ensta-paris.fr/~chapoutot/teaching/in103/practical-work/libin103-1.4.tar.gz
   ```

3. Désarchivez l'archive

   ```
   tar -xvzf libin103-1.4.tar.gz
   ```

4. Allez dans le répertoire `libin103-1.4` et compilez la bibliothèque. Quelle commande faut-il utiliser ?
   — Il faut utiliser la commande `make`. A la fin de la compilation vérifier la présence du fichier `libin103.a` dans le répertoire `source`.
   — Également, vous pouvez exécuter la commande `make check` pour compiler et exécuter les programmes de tests.

Matériel pour le TP

Récupérez l'archive associé à cette séance de TP à l'adresse :

```
https://perso.ensta-paris.fr/~chapoutot/teaching/in103/practical-work/in103-td5.tar.gz
```

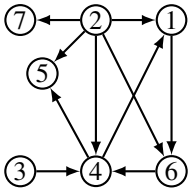
EXERCICES

Exercice 1 – Représentations mathématico-informatiques des graphes

Fonctions utiles pour cet exercice

- `integer_graph_init`
- `integer_graph_destroy`
- `integer_graph_ins_vertex`
- `integer_graph_ins_edge`

Nous allons considérer le graphe suivant



Question 1
Donnez à la main la description du graphe sous la forme d'une matrice d'adjacence.

Question 2
Donnez à la main la description du graphe sous la forme d'une liste d'adjacence.

Question 3
À l'aide du programme donné en cours et du squelette de programme donné dans le répertoire `exo1`, construisez informatiquement le graphe à l'aide de la bibliothèque `libin103` et affichez le pour vérifier que vous obtenez une liste d'adjacence équivalente à celle construite à la main.

Question 4
Mettez en œuvre le parcours en largeur d'un graphe. Le prototype de la fonction sera

```
void bfs(integer_graph_t* graph, int root);
```

Exercice 2 – Modélisation et parcours

Le matériel de cet exercice se trouve dans le répertoire `exo3`. qL'ennoncé de l'exercice est une traduction de l'exercice donné

`https://www.hackerearth.com/practice/algorithms/graphs/topological-sort/practice-problems/algorithm/oliver-and-the-game-3/`

Oliver et Bob sont les meilleurs amis du monde. Ils ont passé toute leur enfance dans la belle ville de Byteland. Les habitants de Byteland vivent heureux avec le roi.

La ville a une architecture unique avec un total de N maisons. Le manoir du roi est un très grand et magnifique bungalow dont l'adresse est 1. Les autres maisons de Byteland ont une adresse unique (par exemple, A), sont reliées par des routes et il y a toujours un chemin unique entre deux maisons de la ville. Notez que le manoir du roi fait également partie de ces maisons.

Oliver et Bob ont décidé de jouer à cache-cache en prenant la ville entière comme arène. Dans le scénario du jeu, c'est à Oliver de se cacher et à Bob de le trouver.

Oliver peut se cacher dans n'importe quelle maison de la ville, y compris le manoir du roi. Comme Bob est une personne très paresseuse, pour trouver Oliver, soit il se dirige vers le manoir du roi (il s'arrête quand il l'atteint), ou il s'éloigne du manoir par n'importe quel chemin possible jusqu'à la dernière maison sur ce chemin.

Oliver court se cacher dans une maison (par exemple, X) et Bob commence le jeu à partir de sa maison (par exemple, Y). Si Bob atteint la maison X, il trouvera certainement Oliver.

Étant donné Q requêtes, vous devez dire à Bob s'il est possible pour lui de trouver Oliver ou non.

Les requêtes peuvent être des deux types suivants :

— $0\ X\ Y$: Bob se déplace vers le manoir du roi.

— $1\ X\ Y$: Bob s'éloigne du manoir du roi.

ENTRÉE
- La première ligne de l'entrée contient un seul entier N, le nombre total de maisons dans la ville.
- Les $N-1$ lignes suivantes contiennent deux entiers A et B séparés par des espaces, indiquant une route entre les maisons situées aux adresses A et B.
- La ligne suivante contient un seul entier Q représentant le nombre de requêtes.
- Les Q lignes suivantes contiennent trois nombres entiers séparés par un espace, représentant chaque requête comme expliqué ci-dessus.

SORTIE Affichage de "YES" ou "NO", une réponse par ligne, pour chaque requête en fonction de la réponse à la requête.

CONSTRAINTES
- $1 \leqslant N \leqslant 10^5$
- $1 \leqslant A, B \leqslant N$
- $1 \leqslant Q \leqslant 5 \cdot 10^5$
- $1 \leqslant X, Y \leqslant N$

Exemple de fichier d'entrée	Sortie associée
9 1 2 1 3 2 6 2 7 6 9 7 8 3 4 3 5 5 0 2 8 1 2 8 1 6 5 0 6 5 1 9 1	YES NO NO NO YES

Explications
- Requête 1 Bob va de 8 vers 1 et rencontre 2 sur son chemin.
- Requête 2 Bob part de 8 en s'éloignant de 1 et ne rencontre jamais 2.
- Requête 3 Bob part de 5 en s'éloignant de 1 et ne rencontre jamais 6.
- Requête 4 Bob va de 5 vers 1 et ne rencontre jamais 6.
- Requête 5 Bob part de 1 en s'éloignant de 1 et rencontre Oliver à 9. Il peut prendre les deux chemins suivants $1 \to 2 \to 6 \to 9$ ou $1 \to 2 \to 7 \to 8$, 9 apparaît dans au moins l'un des deux.

Pensez à dessiner le graphe pour faciliter la compréhension.

Les fichiers exemples sont donnés dans le répertoire `exo3` sous le nom de `digraph-dfs.in` et `digraph-dfs.out`. Pour exécuter le test, utilisez la commande suivante

```
./digraph-dfs.x < digraph-dfs.in
```

et si vous voulez tester la sortie par la même occasion, utilisez la commande

```
./digraph-dfs.x < digraph-dfs.in | diff - digraph-dfs.out
```

Cette dernière commande ne doit rien afficher si la sortie de votre programme et le contenu du fichier `digraph-dfs.out` sont identiques.

Question 1
Comment modéliser la ville et ses parcours depuis ou vers le bungalow du roi ?

Question 2
Proposez une solution algorithmique à ce problème. On passera par une fonction auxiliaire nommée `check` dont le prototype sera

```
bool check (integer_graph_t* graph, int start, int place)
```

Cette fonction vérifie s'il est possible d'atteindre la cachette de Oliver représentée par l'argument `place` depuis la maison de Bob représentée par l'argument `start`.

POUR S'ENTRAÎNER À LA MAISON

Exercice 1 – Calculer le chemin de sortie d'un labyrinthe

Fonctions utiles pour cet exercice

- `integer_graph_init`
- `integer_graph_destroy`
- `integer_graph_ins_vertex`
- `integer_graph_ins_edge`

Dans l'exercice 2 de la feuille 4 de TP, nous avons développé un algorithme de génération de labyrinthes à l'aide de la structure de données union-find. Dans cet exercice, nous allons mettre en place un parcours de graphe pour trouver le chemin vers la sortie du labyrinthe généré.

Question 1

Quel type de parcours de graphes allons-nous utiliser pour trouver le plus cours chemin ?

Le matériel pour cet exercice se situe dans le répertoire `exo2`.

Question 2

Pour trouver le chemin dans un labyrinthe avec un parcours en largeur, il faut mémoriser en plus de l'ordre des nœuds visités, son parent afin d'exhiber le chemin de l'entrée vers la sortie.

On écrira une fonction dont le prototype est

```
void vers_la_sortie(integer_graph_t* graph, int entree, int sortie);
```

qui affichera la succession de sommets parcourus pour aller de l'entrée du labyrinthe vers la sortie.

Question 3

La fonction `generate_maze` est une modification de l'algorithme de génération du labyrinthe demandé dans la feuille 4. La principale différence est qu'elle renvoie (par effet de bord) la liste des cases adjacentes dans le labyrinthe. En d'autres termes, au lieu de renvoyer la liste des murs du labyrinthe, la fonction renvoie la liste des cases qui forment les couloirs du labyrinthe.

1. Expliquer comment sont calculées les cases des couloirs.
2. Utiliser cette fonction `generate_maze` pour construire le graphe associé aux couloirs du labyrinthe généré et appliquer la fonction `vers_la_sortie` pour trouver le chemin entre la case de départ 0 et la case d'arrivée numéro dimension$^2 - 1$.

APPROFONDISSEMENT

Exercice 1 – Coloration de graphe

Le problème de colorier un graphe est de positionner des couleurs à chaque sommet du graphes tels que les voisins ont une couleur différente.

Le matériel pour cet exercice se situe dans le répertoire `exo4`.

Question 1

Supposons un graphe $G = (V,E)$ tel que $\mid V \mid = n$. On suppose que les nœuds sont représentés par des entiers positifs, c'est-à-dire, $V = \{1,2,\ldots,n\}$. On impose donc un ordre sur le nœuds du graphe. Les couleurs sont également représentées par des entiers positifs. Le nombre minimum de couleurs nécessaire pour colorier un graphe est nommé *nombre chromatique* du graphe. C'est un problème NP-difficile pour lequel plusieurs algorithmes ont été proposés dans la littérature.

Notons $c(x)$ la couleur associée au nœud x. Imaginez un algorithme glouton pour colorier un graphe.

Question 2

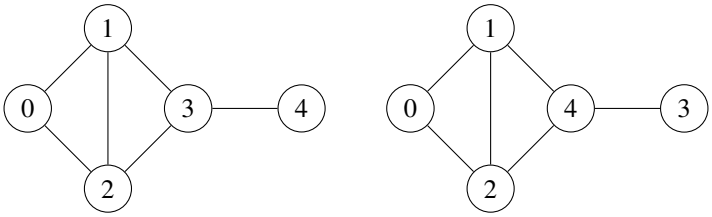
Développez une fonction nommée `greedyColoring` qui permet de colorier un graphe et dont le prototype est :

```
int greedyColoring (integer_graph_t* graph, int** colors);
```

On supposera que le premier sommet du graphe est 0, le tableau `colors` sera alloué dans la fonction `greedyColoring`.

Question 3

Testez votre fonction sur les graphes suivants :



Que constatez-vous ?

Question 4 – Application

Nous voulons résoudre le problème donné

`https://open.kattis.com/problems/vivoparc`

Les fichiers exemples sont donnés dans le répertoire `exo4` sous le nom de `vivoparc.in` et `vivoparc.out`. Pour exécuter le test, utilisez la commande suivante

```
./vivoparc.x < vivoparc.in
```

et si vous voulez tester la sortie par la même occasion, utilisez la commande

```
./vivoparc.x < vivoparc.in | diff - vivoparc.out
```

Cette dernière commande ne doit rien afficher si la sortie de votre programme et le contenu du fichier `vivoparc.out` sont identiques.