Structure de données et algorithmes

Projet 2: Structures de données

19 mars 2021

L'objectif pédagogique de ce projet est de vous familiariser avec la conception, l'implémentation, et l'analyse de structures de données. L'objectif concret sera de mettre au point un programme permettant d'étudier le phénomène de percolation, qui trouve des applications dans de nombreux domaines, allant de la science des matériaux à l'économie. Il s'agira d'écrire un programme permettant de déterminer le seuil critique de percolation d'un matériau, c'est-à-dire la densité à partir de laquelle le matériau devient poreux. La solution à ce problème mettra en œuvre une structure de données de type union-find, que vous devrez également implémenter dans une version potentiellement améliorée par rapport à la version vue au cours.

1 Modèle de percolation

Soit une grille de taille $N \times N$ composée d'un ensemble de $N \times N$ cellules carrées disposées en N lignes et N colonnes. Chaque cellule est soit ouverte, soit fermée. Deux exemples de grille 20×20 sont représentées à la figure 1. Un cellule ouverte sera dite remplie si elle peut être connectée à une cellule ouverte sur la rangée supérieure de la grille via une chaîne de cellules ouvertes adjacentes les unes aux autres (c'est-à-dire se touchant à droite, à gauche, en dessous ou au dessus). On dira que le système constitué par cette grille *percole* s'il y a au moins une cellule remplie dans la rangée inférieure de la grille. Autrement dit, le système percole dès qu'il y a une chemin passant uniquement par des cellules ouvertes entre une cellule ouverte de la rangée supérieure et une cellule ouverte de la rangée inférieure. La figure 2 montre les grilles de la figure 2 dans lesquelles les cellules remplies ont été marquées en bleu. La grille de gauche ne percole pas alors que celle de droite percole. Ce système simple permet de modéliser différents phénomènes, tels que l'écoulement d'un liquide dans un matériau poreux (les cellules ouvertes représentant le vide dans le matériau par lesquelles le liquide peut s'écouler), la propagation d'un feu de forêt (les cellules vides représentant les arbres de la forêt par lesquels le feu peut se propager) ou encore la conductivé d'un matériau composite fait d'isolant et de métal (les cellules ouvertes représentant le matériau métallique par lequel le courant peut passer).

Une question qui intéresse les chercheurs est l'influence de la densité des cellules ouvertes sur la possibilité que le système percole, lorsqu'on suppose que la position des cellules ouvertes pour une densité donnée est purement aléatoire. Malgré la simplicité du système, il n'est pas possible de dériver une équation mettant en relation ces deux grandeurs et l'étude du phénomène doit passer obligatoirement par des simulations numériques. Les scientifiques qui ont étudié la question ont observé qu'il existait un seuil critique en terme de densité (correspondant à une transition de phase) en dessous duquel la probabilité que le système percole est proche de 0 et au dessus duquel la probabilité que le système percole est proche de 1. On se propose dans le cadre de ce projet d'estimer ce seuil critique de percolation.

Estimation du seuil critique. Pour estimer le seuil critique, on réalisera un grand nombre T de répétitions de l'expérience suivante :

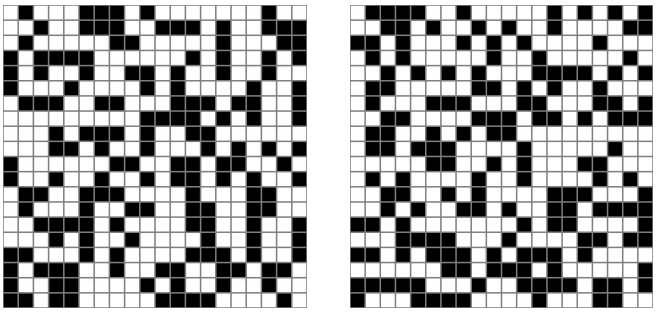

FIGURE 1 – Deux grilles 20×20 de densité $d = 0,6$. En blanc, les cellules ouverts, en noir, les cellules fermées.

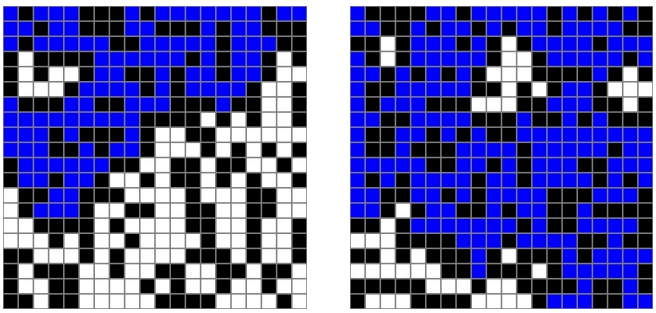

FIGURE 2 – Deux grilles 20×20 de densité $d = 0,6$. Les cellules bleus représentent les cases remplies. Seule la grille de gauche percole de part l'existence d'un passage de haut en bas.

1. On initialise une grille $N \times N$ avec toutes les cellules fermées.
2. Tant que la grille ne percole pas :
   (a) On choisit une cellule au hasard parmi toutes les cellules fermées.
   (b) On ouvre cette cellule.

A chaque expérience, la proportion de cellules ouverte à la sortie de la boucle donne une estimation du seuil critique de percolation qui nous intéresse. On estimera la moyenne de ce seuil sur les T expériences et pour mesurer la stabilité de ce seuil, on rapportera également les 5ème et 95ème centiles sur les T expériences.

La structure union-find permet d'implémenter facilement ces expériences. L'idée est d'utiliser la structure pour maintenir les sous-ensembles de cellules accessibles les unes des autres. Initialement, la structure contiendra un ensemble de singletons correspondant chacun à une cellule et l'ensemble sera mis à jour à chaque ouverture d'une cellule en réalisant les opérations d'union appropriées. Il y aura percolation dès qu'une cellule de la première ligne se retrouvera dans le même sous-ensemble qu'une cellule de la dernière ligne.

2 Implémentation

Cette section décrit les différentes structures et fonctions qu'on vous demande d'implémenter.

Structure Union-find (fichiers `UnionFindList.c`, `UnionFindTree.c` et `UnionFind.h`)

La structure union-find maintient une partition d'un ensemble d'objets en sous-ensembles disjoints. Pour simplifier l'implémentation, on supposera que les objets sont représentés par les entiers

```
|o##ooo##oo|
|oooo## #oo|
|ooooo# ##o|
|#o##ooo##o|
|ooooo## ##|
|##o##  ## |
|#oo# #    |
|#oooo# #  |
|#oooo# # #|
|oo#ooo#  #|
```

FIGURE 3 – Une représentation en texte d'une grille 10×10 (qui percole)

de 0 à $Q-1$, où Q sera supposé connu à la création de la structure. Vous devez implémenter les fonctions suivantes de l'interface :

- ufCreate : qui prend comme argument le nombre d'objets Q et crée une partition constituée des Q singletons contenant les objets 0 à $Q-1$.
- ufUnion : qui réunit les ensembles contenant les deux objets donnés en argument.
- ufFind : qui renvoie un représentant de l'ensemble d'objets qui contient l'objet donné en argument.
- ufComponentsCount : qui renvoie le nombre de sous-ensembles dans la structure.
- ufFree : qui libère l'espace mémoire alloué pour la structure.

On vous demande de fournir (et de comparer ci-dessous) deux implémentations de cette structure : l'implémentation présentée dans les transparents du cours (union-find pondéré par liste-chaînées) et une implémentation à base d'arbres. Plusieurs variantes de cette structure–plus ou moins raffinées–existent. Vous êtes libres d'implémenter celle de votre choix.

Pour vous renseigner sur cette dernière implémentation, vous pouvez consulter toutes les sources que vous souhaitez.

Structure Percolation (fichiers `Percolation.c` et `Percolation.h`)

Un type de données Percolation sera défini permettant de modéliser une grille et de réaliser l'expérience décrite ci-dessus. Ce type de données devra fournir l'interface suivante :

- percCreate : prenant comme argument une taille de grille N et créant une grille complètement fermée de $N \times N$ cellules.
- percFree : libérant la mémoire allouée par la structure de percolation
- percSize : renvoyant la taille N de la grille
- percOpenCell : prenant comme argument une cellule (i, j) et l'ouvrant dans la grille courante.
- percIsCellOpen : prenant comme argument une cellule et renvoyant `true` si elle est ouverte, `false` sinon.
- percIsCellFull : prenant comme argument une cellule (i, j) et renvoyant `true` si elle est remplie, `false` sinon.
- percPercolates : renvoyant `true` si la grille percole, `false` sinon.
- percPrint : affichant une représentation textuelle de la grille courante. Une cellule fermée sera représentée par le caractère `#`, un cellule ouverte non remplie par une espace et une cellule ouverte remplie par le caractère `o`. On représentera des "murs" sur les bords gauche et droit par le caractère `|`. Une exemple d'affichage attendu est montré à la figure 3.
- percFree : qui libère l'espace mémoire alloué pour la grille.

Une cellule de la grille sera représentée par une paire de coordonnées (i, j) ou (row, col), avec $i \in \{0, \ldots, N-1\}$ représentant la ligne de la cellule (de haut en bas) et $j \in \{0, \ldots, N-1\}$ représentant la colonne de la cellule (de gauche à droite).

Pour implémenter ces fonctions, vous devez utiliser une structure de type union-find telle que définie ci-dessus. Vous devez vous arranger pour que toutes les fonctions de l'interface, sauf percCreate, et percPrint, prennent un temps constant plus éventuellement un nombre *constant* d'appels aux fonctions de l'interface de la structure union-find (ufFind, ufUnion, ou ufCount). La fonction percCreate pourra être $\Theta(N^2)$ et aucune contrainte n'est imposée sur percPrint.

Fichiers `Threshold.c` et `Threshold.h`

Le fichier `Threshold.c` contiendra l'implémentation d'une unique fonction :

thresholdEstimate : prenant comme argument une taille de grille N, un nombre T de répétitions et renvoyant un pointeur vers un tableau de trois valeurs réelles contenant la moyenne du seuil estimé sur les T expériences et les valeurs des 5ème et 95ème centiles

3 Rapport et analyse théorique

On vous demande de répondre aux questions suivantes dans le rapport :

1. **Analyse théorique** :
   (a) Décrivez la variante d'union-find à base d'arbres que vous avez implémentée.
   (b) Donnez, sans les justifier, les complexités en temps au pire et au meilleur cas des deux implémentations des fonctions ufUnion et ufFind en fonction du nombre d'objets Q présents dans la structure. Citez vos sources.
   (c) Expliquez et justifiez vos choix d'implémentation de la structure Percolation. Décrivez et justifiez toutes les méta-données associées à cette structure. Montrez que les contraintes en terme de complexité données plus haut sont bien satisfaites pour toutes les fonctions de l'interface.
   (d) Analysez les complexités en temps et en espace au pire et au meilleur cas de la fonction thresholdEstimate en fonction de N, la taille de la grille, et de T le nombre de répétitions pour l'implémentation à base de liste-chaînées.
   (e) Faites la même analyse pour l'implémentation à base d'arbres.
2. **Analyse empirique** :
   (a) Tracez un graphe de l'évolution de la valeur moyenne du seuil critique et des centiles pour N croissant et une valeur de T suffisamment élevées (≥ 100). Discutez brièvement cette courbe.
   (b) Tracez un graphe de l'évolution des temps de calcul de la fonction thresholdEstimate en fonction de la taille de la grille N pour les deux implémentations du union-find. Utilisez une valeur de T suffisamment élevée.
   (c) Discutez ces courbes en les mettant en parallèle avec l'analyse théorique.

4 Deadline et soumission

Le projet est à réaliser **par groupe de deux étudiant.e.s** pour le **15 avril 2021, 23h59** au plus tard. Il doit être soumis via la plateforme de soumission (`http://submit.montefiore.ulg.ac.be/`).

Le projet doit être rendu sous la forme d'une archive `tar.gz` contenant :

1. Votre rapport (5 pages maximum) au format PDF. Soyez bref mais précis et respectez bien la numération des (sous-)questions.
2. Les fichiers `UnionFindList.c` `UnionFindTree.c`, `Percolation.c` et `Threshold.c`.

Vos fichiers seront évalués avec les flags habituels (`--std=c99 --pedantic -Wall -Wextra -Wmissing-prototypes -DNDEBUG`) sur les machines ms8xx. Ceci implique que :

- Les noms des fichiers doivent être respectés.
- Le projet doit être réalisé dans le standard C99.
- La présence de *warnings* impactera négativement la cote finale.
- Un projet qui ne compile pas sur ces machines recevra une cote nulle (pour la partie code du projet).

Notez que le choix d'implémentation de l'union-find se fait à la compilation.

Un projet non rendu à temps recevra une cote globale nulle. En cas de plagiat avéré, l'étudiant se verra affecter une cote nulle à l'ensemble des projets.

Les critères de correction sont précisés sur la page web des projets.

Bon travail !