Université d'Ottawa
Faculté de génie

École de science informatique
et de génie électrique

uOttawa
L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Electrical Engineering
and Computer Science

Devoir de programmation 1 (7.5%) CSI2110/CSI2510

Date limite : 11 octobre, 23h59 2025

Politique de soumission tardive : ***1 à 24 heures de retard sont acceptées avec 30 % de réduction ; aucun devoir n'est accepté après 24 heures de retard.***

Description du problème

Dans ce devoir, vous allez découvrir le **type abstrait de données de partition** (également appelé union-find) et l'utiliser dans une application. La première tâche consiste à implémenter la **partition TAD** à l'aide de l'implémentation de séquence. La deuxième tâche consistera à résoudre un problème qui implique de garder une trace des îlots de terre dans une zone inondée.

1 Implémentation par séquence du TAD de partition

La **partition TAD** comporte les opérations suivantes:

- makeCluster(E *x*) : Crée un cluster singleton contenant le nouvel élément *x* et renvoie sa position.
- union(*p*, *q*) : fusionne les clusters contenant les positions *p* et *q*.
- find(*p*): Renvoie la position du leader du cluster contenant la position *p*.

Il existe plusieurs façons d'implémenter cette structure de données. Dans ce devoir, vous devez utiliser l'**implémentation par séquence** de la **partition TAD** décrite dans le manuel aux pages 672-673 (voir ces pages en annexe), qui est mise en œuvre à l'aide de plusieurs listes chaînées.

En utilisant l'**implémentation** par **séquence,** le temps d'exécution de makeCluster(*x*), find(*p*) et union(*p*, *q*) doit être O(1), O(1) et O(min(n1,n2)), respectivement, où n1 est la taille du cluster de p et n2 est la taille du cluster de q.

En plus des méthodes habituelles ci-dessus, le **TAD de partition** doit fournir les méthodes suivantes:

- element(p) : renvoie l'élément qui a été stocké à la position p
- numberOfClusters() : renvoie le nombre actuel de clusters différents
- clusterSize(p) : renvoie la taille du cluster contenant p
- clusterPositions(p) : renvoie une liste de toutes les positions qui appartiennent au même cluster que la position p
- clusterSizes() renvoie une liste de toutes les tailles de cluster par ordre décroissant de taille

Le temps d'exécution des 3 premières méthodes doit être O(1), pour clusterPositions(p) et clusterSizes(), le temps doit être O(s), où s est la taille de la liste renvoyée (sauf que clusterSizes() peut passer plus de temps à trier sa liste renvoyée).

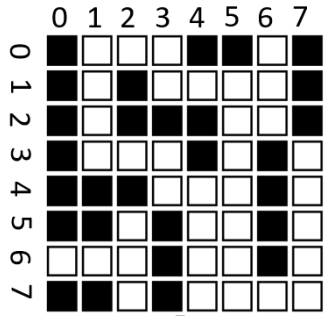
La partition doit être implémentée en tant que classe à l'aide d'un élément générique E :
`class Partition<E>`
Une « position » est une référence à un nœud qui stocke l'élément et toute autre information nécessaire, la classe de nœud doit être une classe utilisant un élément générique E : `classe Node<E>`

2 îles dans une zone inondée

Dans ce devoir, vous devriez écrire un programme pour garder une trace d'une région qui a été soumise à une inondation, de sorte que la surface terrestre a été divisée en « îles » à la suite de l'inondation.

PARTIE 2A Arpentage d'un paysage inondé en évolution

Vous lirez une carte initiale S par T qui contient des points/pixels de 0 (blanc, représentant l'eau) ou 1 (noir, représentant la terre). L'image suivante montre un exemple d'une région de 8 par 8 avec 7 îles :



(crédits : photo du professeur Jeff Ericson[1])

Notez que deux carrés font partie d'une même île s'ils partagent l'un de leurs quatre côtés avec un autre carré. Les cases qui se touchent dans un coin ne sont pas considérées comme une connexion terrestre.

Une fois que la carte initiale a été lue et que les îles ont été identifiées, vous devez fournir un **arpentage (enquête) du terrain** qui comprend les informations suivantes:

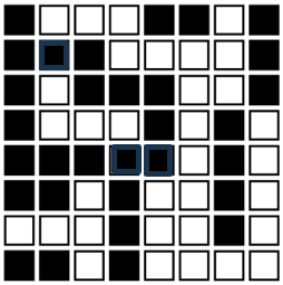
- Nombre d'îles
- Liste des tailles des îles (par ordre décroissant de taille comptée en nombre de carrés noirs); S'il n'y a pas d'îles, la liste vide est représentée par -1
- Superficie totale des îles (somme des tailles des îles)

Cependant, le niveau d'inondation peut diminuer, ce qui fait réapparaître certaines terres (c'est-à-dire que certains carrés blancs deviennent noirs) et modifier effectivement le paysage de la carte.

À chaque *phase* de rétablissement des inondations, nous apprenons les coordonnées de plusieurs nouvelles positions terrestres (carrés qui étaient autrefois blancs et sont devenus noirs). La carte et les structures de données connexes devraient être mises à jour, et un nouvel arpentage des terres devrait être imprimé.

[1] https://jeffe.cs.illinois.edu/teaching/algorithms/notes/11-unionfind.pdf

L'exemple suivant montre la carte après la phase 1 où les positions (1,1), (4,3), (4,4) deviennent des terres. Maintenant, il n'y a plus que 5 îles de tailles 20, 4, 3, 2, 2.



Le résultat de l'enquête pour l'enquête initiale et l'enquête après la phase 1 doit être produit comme suit :
7
9 5 4 3 3 2 2
28

5
20 4 3 2 2
31

PARTIE 2B Arpentage des îles avec identification du lac

Le niveau des inondations a été lent et les gens s'adaptent à leur nouvelle vie de résidents des îles, dont certaines contiennent maintenant plusieurs lacs. Dans l'exemple jouet « toy » de la page précédente, vous pouvez voir sur la carte qu'un lac est présent dans la plus grande île.

Certains experts ont remarqué que les données que vous fournissez dans votre programme dans la partie 1 n'étaient pas exactement ce qu'ils voulaient, parce que vous n'avez pas tenu compte de la formation des lacs comme faisant partie des îles. Eh bien, ils s'attendent à ce que la superficie (aire) des îles avec des lacs inclue la superficie des lacs qu'elles contiennent. Votre nouveau programme pour la partie B devrait effectuer des tâches similaires à celles de la partie A, mais votre arpentage devrait être mis à jour pour tenir compte de la présence de lacs et de cette nouvelle interprétation de la superficie d'une île.

L **'arpentage des terres pour la** PARTIE 2B doit inclure dans cet ordre:

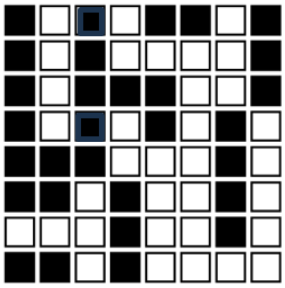
- Nombre d'îles
- Liste des tailles des îles (par ordre décroissant de taille) – notez que les lacs à l'intérieur d'une île doivent contribuer à la superficie de l'île; Une liste vide est représentée par -1
- Superficie totale des îles (somme des tailles des îles)
- Nombre total de lacs (en comptant tous les lacs qui existent dans toutes les îles)
- Superficie totale des lacs

Pour l'exemple de la page précédente, la sortie serait:
7
9 5 4 3 3 2 2
28
0
0

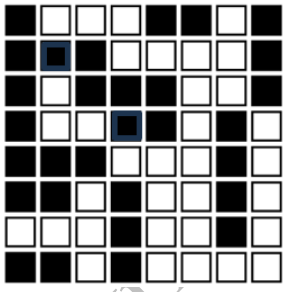
5
24 4 3 2 2
35
1
4

Notez qu'un lac doit être un ensemble de carrés blancs entourés de carrés noirs non seulement sur les bords mais aussi sur les « coins ». Une île peut avoir plusieurs lacs. On peut avoir une île avec de l'eau à l'intérieur qui ne forme pas un lac. Nous montrons les 3 exemples suivants, chacune avec 6 îles, et fournissons des informations sur la **superficie de la plus grande île** et si elle possède un lac :

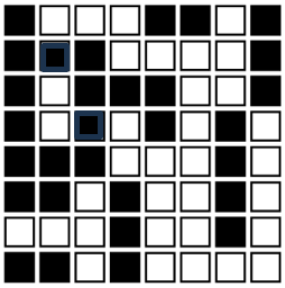
Aire = 16, pas de lac



Aire = 16, pas de lac



Aire = 18 avec 1 lac avec aire 2



3 Spécification concernant l'entrée et la sortie

Format d'entrée:
Les parties A et B utiliseront le même format d'entrée donné dans un fichier texte que celui spécifié ici. La première ligne contient deux nombres, S et T, spécifiant que la carte est une grille de lignes S et de colonnes T. Pour les lignes suivantes, chacune se compose d'une chaîne de 0 et de 1 spécifiant chaque ligne de la carte dans l'ordre. Après cela, il y aura une séquence de zéro ou plusieurs phases. La première ligne contiendra le nombre de phases F. Chaque phase est représentée par des lignes au format suivant : la première ligne d'une phase contient le numéro L des nouvelles cases terrestres, et la deuxième ligne contient L paires de coordonnées pour les cases L (au total 2L nombres). Les coordonnées sont de l'ordre i j, où 0<=i< S et 0<=j< T, représentant la position sur la carte, map[i,j], qui devrait devenir un 1.

EXEMPLE DE FICHIER D'ENTRÉE 1 (correspond à la carte de la première image)

```
8 8
10001101
10100001
10111001
10001010
11100010
11010010
00010010
11010000
0
```

EXEMPLE DE FICHIER D'ENTRÉE 2

```
8 8
10001101
10100001
10111001
10001010
11100010
11010010
00010010
11010000
1
3
1 1 4 3 4 4
```

EXEMPLE DE FICHIER D'ENTRÉE 3

```
3 4
0000
0000
0000
2
9
0 0 0 1 0 2 0 3 1 0 2 0 2 1 2 2 1 3
1
2 3
```

Format de SORTIE:

Pour les parties A et B, les résultats fourniront les résultats des diverses enquêtes séparées par une ligne vide. Les valeurs doivent être données dans l'ordre exact indiqué dans les descriptions de l'arpentage des terres pour chaque partie. Il est à noter que les sondages ont des formats différents dans les parties A et B.

À la fin de la partie 2A et de la partie 2B, nous avons déjà donné des exemples de sorties pour une entrée qui correspond à **SAMPLE INPUT FILE 2.**

Ici, nous montrons la sortie pour **SAMPLE INPUT FILE 3**

SAMPLE OUPUT FILE 3 (POUR LA PARTIE 2A)
0
-1
0

1
9
9

1
10
10

SAMPLE OUPUT FILE 3 (POUR LA PARTIE 2B)
0
-1
0

1
9
9
0 ← typo corrigé ici! de 1 à 0
0

1
12
12
1
2

4 Algorithmes

Algorithmes pour la partie 2A

Vous devez garder une trace des îles en utilisant le Partition TAD.

Une esquisse des principales étapes de création de la partition initiale en îlots (avant l'impression du premier levé) est donnée ci-dessous:

```
Créez un cluster de tableaux auxiliaires S par T pour effectuer
le suivi des positions du cluster avec toutes les entrées
initialisées sur null
Créer BP un nouvel objet de la classe Partition
for each black grid point i,j:
     p = BP.makeCluster(info(i,j));
     cluster[i,j]=p

for each black grid point i,j:
     for each black grid point k,l adjacent to i,j:
          if  BP.find(cluster[i,j])!= BP.find(cluster[k,l]) then
                        BP.union(cluster[i,j],cluster[k,l])
```

Toute phase ultérieure ajoutera une liste de nouvelles positions noires en utilisant une méthode similaire. La nouvelle liste de clusters peut être utilisée pour mettre à jour les îles comme suit :

```
for each point i,j in the new list
     p = BP.makeCluster(info(i,j));
     cluster[i,j]=p
     change grid point i,j to black

for each point i,j in the new list
     for each black grid point k,l adjacent to i,j:
          if  BP.find(cluster[i,j])!= BP.find(cluster[k,l]) then
              BP.union(cluster[i,j],cluster[k,l]))
```

Les éléments d'information qui doivent être imprimés pour l'arpentage du terrain peuvent être obtenus en invoquant des méthodes spécifiques de la partition de classe, à condition que l'élément info(i,j) stocke des informations pertinentes comme i et j.

Algorithmes pour la partie 2B

Les mêmes algorithmes que dans la partie 2A doivent être utilisés pour garder une trace de la partition actuelle BP des points noirs en îlots.
Vous devez concevoir votre propre algorithme qui, une fois BP créé, peut identifier les composants/grappes (clusters) blancs et décider si chacun d'entre eux forme un lac et pour quelle île, en tenant compte du changement de taille qui doit être pris en compte pour l'île correspondante.

Astuce : Une option est de créer un nouvel objet WP de la classe Partition pour garder une trace des composants/clusters blancs (white clusters), mais de changer le concept de cluster **pour que les points blancs** mettent dans le même cluster des points blancs qui ne touchent qu'un coin. Ensuite, pour chaque

groupe de points blancs, vérifiez que leurs côtés ne touchent pas plus d'une île ni un bord extérieur de la carte ; Si c'est vrai, alors c'est un lac de l'île unique qui est touché par certains de ses points.

5 Spécifications du devoir et répartition des notes (60 points)

L'entrée DOIT être lue à partir de l' **entrée standard** et la sortie doit être envoyée à la **sortie standard**. Pour la lecture à partir d'un fichier d'entrée et l'écriture dans un fichier de sortie, nous allons rediriger l'entrée et la sortie via la ligne de commande comme indiqué ci-dessous. Si vous ne savez pas comment cela fonctionne, demandez à un assistant d'enseignement ou au professeur.

Votre code sera testé via la ligne de commande Partie 2A comme suit

```
javac Partition.java Node.java IslandSurvey.java
java IslandSurvey < map1.txt > map1Output.txt
```

Et pour la partie 2B, comme suit:

```
javac Partition.java Node.java IslandLakeSurvey.java
java IslandLakeSurvey < map1.txt > map1Output.txt
```

Vous devez remettre les fichiers suivants sans aucun sous-répertoire

1 toutes les classes nécessaires à l'exécution de votre code (*.java), notamment:
→ **`Partition.java`** (comme décrit dans la section 1)
`Node.java` (comme décrit dans la section 1)
`IslandSurvey.java` (classe qui implémente **part2A** ; Il contient une méthode main)
`IslandLakeSurvey.java` (classe qui implémente **part2B** ; Il contient une méthode main)

2 Deux exemples d'entrées que vous concevez pour vérifier l'exactitude de votre code

3 Un court rapport au format pdf avec le nom : `report.pdf` (1-3 pages)
un court rapport contenant les sections suivantes :
1) Statut du code : expliquer l'état du code soumis (fonctionnel, partiellement fonctionnel, bogues connus, et tout ce qui peut aider au marquage)
2) Tests : décrivez votre stratégie de test avec les sorties et les résultats que vous avez choisis.
3) Programme
Incluez toute information pertinente sur l'implémentation de votre classe Partition, l'implémentation dans la partie 2A et l'implémentation dans la partie 2B
4) Ressources
Donnez référence à toutes les ressources utilisées pour mener à bien votre devoir.

4 Incluez tous les fichiers d'entrée fournis par le professeur et la sortie produite par votre programme, en ajoutant le suffixe « Output » au nom du fichier. Si un fichier d'entrée est appelé `map1.txt` le fichier de sortie doit être enregistré dans `map1Output.txt`
De plus, incluez les sorties pour vos deux exemples d'entrées.

Système de notation (sur 60 points) - **l'efficacité compte!**

- **PARTIE 1 Mise en œuvre par séquence du TAD de partition (20 points)** - **l'efficacité compte!**
  makeCluster - 1 point
  union - 5 points
  find - 4 points
  element(p) - 1 point
  numberOfClusters() - 1point
  clusterSize(p) - 1 point
  clusterPositions(p) - 2 point
  clusterSizes() - 4 points
  Généralités (autres parties) - 1 point
  Si la complexité des opérations a un big_Oh plus grand que celui spécifié, une déduction peut être appliquée jusqu'à 10 points.
  Si vous utilisez une implémentation différente du TAD de partition qui n'est pas l'implémentation de la séquence, aucune note ne sera donnée, mais vous serez noté normalement pour les parties 2A et 2B.
- **PARTIE 2A Arpentage d'un paysage inondé en évolution (15 points)** - **l'efficacité compte!**
  Résultat de test correct : 5 points (les erreurs détectées peuvent entraîner d'autres déductions)
  10 points seront distribués pour l'exactitude de pièces spécifiques, ainsi que pour la clarté et l'efficacité
- **PART 2B Surveying islands with lake identification (15 points)** **efficiency counts!**
  Correct test output: 5 points (errors detected can lead to more discounts)
  10 points seront attribués pour l'exactitude de certaines parties, ainsi que pour la clarté et l'efficacité
- **Général pour toutes les parties : documentation (10 points)**
  rapport
  qualité du code et commentaires clairs ajoutés tout au long du code pour expliquer ses parties et ses commandes spécifiques
- **Déductions en cas de non-respect des spécifications:** (jusqu'à -15 points si les spécifications ne sont pas respectées même si le programme est correct)
  - L'entrée et la sortie se font à l'aide des E/S standard ; la ligne de commande avec redirection fonctionne

- Le format de la sortie est exactement celui spécifié (cela signifie qu'un programme automatique peut tester l'exactitude de votre sortie sans ajustements)
- Les classes Java sont 4 classes java avec des noms exacts tels que spécifiés
- Deux exemples d'entrées ont été inclus
- Les fichiers d'entrée fournis et leur sortie ont été inclus, ou une justification dans le rapport selon laquelle certains fichiers d'entrée produisent une erreur ou une sortie incorrecte.
- Le rapport a été inclus au format PDF avec le nom spécifié.

Appendice:

`Voir les pages ci-jointes du manuel.`