Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

MINISTÈRE DE L'ENVIRONNEMENT,
DE L'ENERGIE ET DE LA MER

MINISTÈRE DU LOGEMENT
ET DE L'HABITAT DURABLE

Examen professionnel de vérification d'aptitude aux fonctions de programmeur

- Session 2016 -

Épreuve écrite
INFORMATIQUE

Analyse critique d'un programme rédigé dans un langage évolué choisi par le candidat.

Langage : **C++**

Durée : 5 heures

Coefficient : 4
Notation : sur 20

Nombre de pages du sujet : 8 (y compris cette page)

Matériel :
Aucun matériel autorisé.

Documents :
Aucun document autorisé.

Observations :
La lisibilité et la qualité de l'expression écrite seront prises en compte dans l'appréciation de la copie.

Remarque générale :

Il est vivement conseillé aux candidats de travailler sur un brouillon et de prévoir un temps de rédaction au propre sur la copie définitive.

1 – le sujet comporte deux parties :
un sujet général commun aux différents langages noté sur 6 (temps estimé 1h30)
un sujet spécifique au langage choisi noté sur 14 (temps estimé 3h30)

2 – si éventuellement il vous manquait des informations particulières pour développer votre sujet, il vous revient de retenir les hypothèses adaptées à votre solution, en les explicitant clairement.

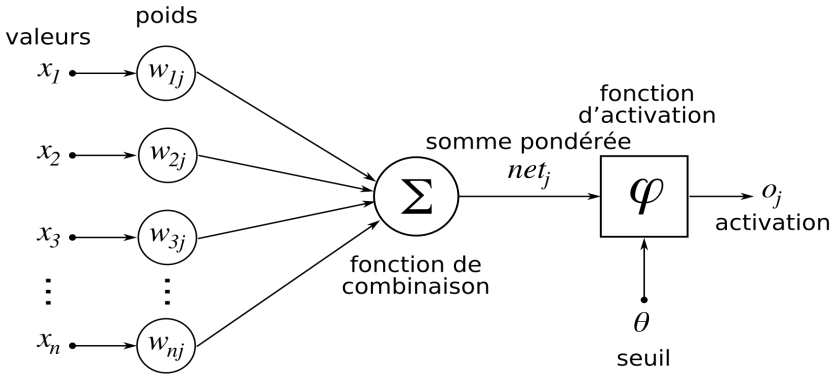
3 – aucun document ou matériel électronique (calculette, ordinateur …) n'est autorisé.

1 Sujet commun – Réseaux de neurones

1.1 Description

Le fonctionnement du cerveau humain a inspiré une classe d'algorithme appelé réseaux de neurones. L'exemple le plus simple est le « Perceptron ». Il s'agit d'un ensemble de neurones organisés sous forme de couches. La première est appelée couche d'entrée et la dernière couche de sortie.

Fonctionnement d'un neurone

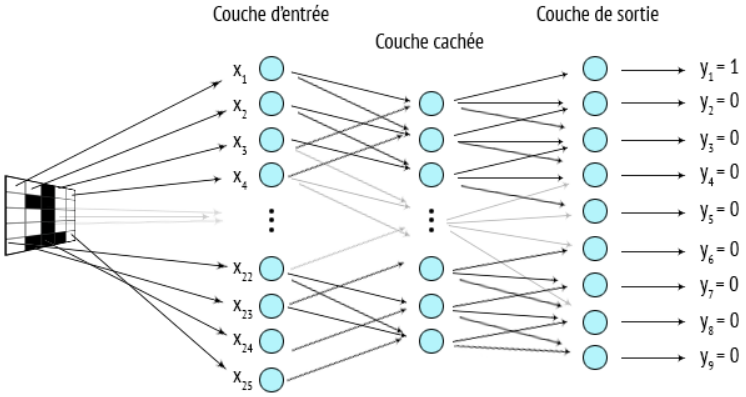


Chaque neurone est relié à tous ceux de la couche précédente et dispose d'un poids sur chacune de ses connexions. On effectue alors la somme pondérée de l'ensemble de ses entrées. Ce qui donne pour le neurone j : $net_j = \sum_{i=1}^{n} w_{ij} \cdot x_i$

La sortie du neurone sera alors calculée en utilisant une fonction d'activation paramétré par un seuil Θ : $o_j = \varphi_\Theta(net_j)$ Cette fonction vaut presque 0 avant le seuil et monte très vite à 1 après le seuil. Un neurone laisse donc passer l'information seulement s'il est suffisamment excité sur ses entrées. Ainsi, l'information se propage de couche en couche jusqu'à atteindre la couche de sortie.

On peut utiliser un Perceptron pour catégoriser des images. On branche chaque pixel de l'image sur un neurone de la couche d'entrée. Il est alors possible d'entraîner le réseau en ajustant les poids de chaque neurone pour que la couche de sortie indique dans quelle catégorie l'image se trouve. On utilise alors autant de neurones en sortie que de catégories.
L'exemple classique consiste à reconnaître l'écriture manuscrite des chiffres. Lorsque l'on présente l'image du chiffre 1 en entrée : un tableau de pixel noté X, le réseau donnera en sortie un vecteur noté Y : {1,0,0,0,0,0,0,0,0,0} c'est-à-dire que seul le premier neurone sera activé. Lorsque l'on présente l'image du chiffre 2 on obtiendra {0,1,0,0,0,0,0,0,0,0} et ainsi de suite. Un réseau de neurone correctement entraîné sera capable d'identifier un chiffre dont l'image serait légèrement modifiée.

Perceptron



Pour des questions de lisibilité, l'ensemble des flèches n'ont pas été représentées

L'une des méthodes d'entraînement les plus utilisées s'appelle la rétro-propagation du gradient, basée sur l'idée de détecter les poids qui ont le plus d'influence sur les erreurs produites par le réseau et de les modifier en conséquence, en partant de la sortie et en propageant les erreurs vers l'entrée. On utilise le gradient afin de savoir dans quel sens modifier le poids.

On note en minuscule les éléments d'un tableau. Par exemple : x_i indique le $i^{ème}$ élément du tableau X.

Les étapes de l'algorithme sont les suivantes :

Pour chaque exemple (X, Y) :

On calcule la sortie du Perceptron noté S

Les valeurs de chaque neurone en sortie sont notées s_k

Pour chaque neurone de sortie k, on calcule un écart e_k basé sur l'erreur et le gradient :

$$e_k = s_k(1-s_k)(y_k - s_k)$$

Les valeurs de chaque neurone de la couche cachée sont notées o_j

Pour chaque neurone caché j, on propage l'erreur en calculant l'écart e_j :

$$e_j = o_j(1-o_j) \sum_{neurone(k)en\ sortie} e_k w_{jk}$$

On peut alors mettre à jour les poids w_{ij} en utilisant le coefficient d'apprentissage η qui est une variable donnée.

Pour les poids entre la couche cachée (neurones j) et la couche de sortie (neurones k) :

$$w_{jk} \leftarrow w_{jk} + \eta e_k o_j$$

Pour les poids entre la couche d'entrée (neurones i) et la couche cachée (neurones j) :

$$w_{ij} \leftarrow w_{ij} + \eta e_j x_i$$

On répète ce processus jusqu'à ce que la somme des erreurs soit inférieure à un seuil ε ou que le nombre d'itération dépasse le seuil N.

1.2 Travail demandé

On suppose que la fonction qui calcule $\varphi_\Theta(x)$ est donnée par phi(x).

1. Écrire en pseudo code la méthode qui calcule la sortie d'un Perceptron en fonction d'un ensemble x_i de valeurs en entrée.
2. Écrire en pseudo code la méthode d'entraînement par rétro-propagation du gradient. On fournit à cette méthode un ensemble d'exemples : une série de tableaux X contenant les entrées à présenter au Perceptron et une série de tableau Y contenant les sorties attendues.
   On suppose que les constantes suivantes sont données : N, ε et η

   Note : on utilisera la fonction de la question 1 pour obtenir la sortie du Perceptron et on suppose qu'elle stocke également les sorties intermédiaires des neurones de la couche cachée dans le tableau O

2ème partie

2 Questions de syntaxe

Langage C++

Q1 : Expliquez le mot clef **void**. Donner un exemple de son utilisation.

Q2 : À quoi sert le mot clef ***unsigned*** ?

Q3 : Dans le code suivant :

```
int n;
int* p = &n;
int& r = *p;
```

Que représentent p et r par rapport à n ?

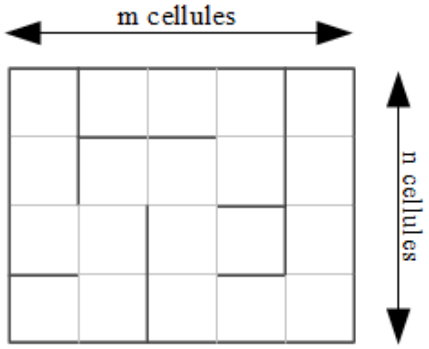
Q4 : Dans la définition d'une classe, que signifie cette syntaxe :**virtual void test() = 0;**

Q5 : A quoi sert le mot clef **inline** ?

3 Problème

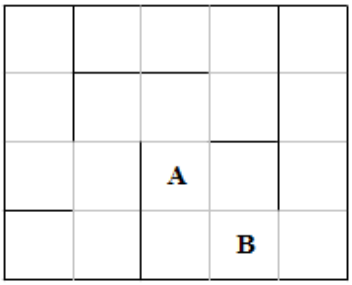
3.1 Description

Vous avez en charge l'implémentation d'un système générant automatiquement des labyrinthes de taille arbitraire. Un labyrinthe est inscrit dans un tableau de m x n cellules. Un mur est matérialisé par un segment horizontal ou vertical entre deux cellules :

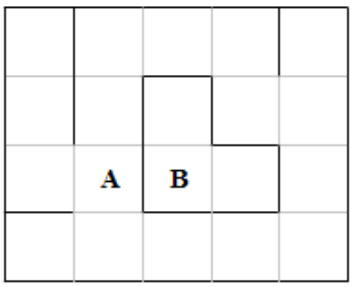


a) Labyrinthe parfait de taille 5 x 4

Il est ici sujet des labyrinthes parfaits, c'est-à-dire qu'entre deux cellules quelconques du labyrinthe, il existe toujours un et un seul chemin les reliant.



b) Labyrinthe non parfait; deux chemins possibles entre A et B



c) Labyrinthe non parfait; aucun chemin ne permet de relier A et B (îlot central)

On suppose que le labyrinthe est entièrement entouré de murs extérieurs.

3.2 Travail demandé

Pour plus de clarté, vous expliquerez et/ou illustrerez en début de chaque question le fonctionnement des méthodes/fonctions avant l'écriture du code.

1. Algorithme 1 : Exploration exhaustive

On part d'un labyrinthe où tous les murs sont fermés (quadrillage). On choisit aléatoirement une cellule, on la marque comme visitée. Parmi les cellules voisines non visitées, on en choisit une au hasard, on ouvre le mur et on recommence avec la nouvelle cellule. S'il n'y a pas de cellule voisine, on revient à la case précédente et on recommence. Lorsque l'on est revenu à la case de départ et qu'il n'y a plus de possibilités, le labyrinthe est terminé.

1a. Écrivez la classe Labyrinthe représentant seulement la structure de donnée adaptée à cet algorithme, ainsi que le constructeur de cette classe initialisant la structure conformément à l'algorithme à partir des nombres entiers m et n.

1b. Votre collègue a codé les méthodes suivantes de la classe Labyrinthe :

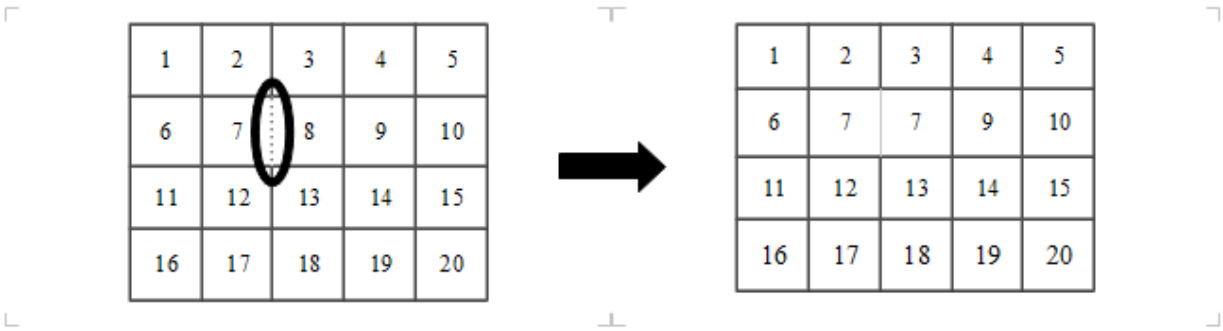
- `ouvrirMur(...)` : Enlève le mur demandé du labyrinthe.
- `listerVoisins(...)` : Donne la liste/ensemble des cellules voisines, visitées ou non.

En adoptant une stratégie récursive, implémentez la solution en écrivant le code de la méthode récursive `parcours(...)` ayant pour paramètre la position en cours. Le labyrinthe modifié doit être une solution. Veillez à appeler les méthodes codées par votre collègue, sans écrire leur implémentation.

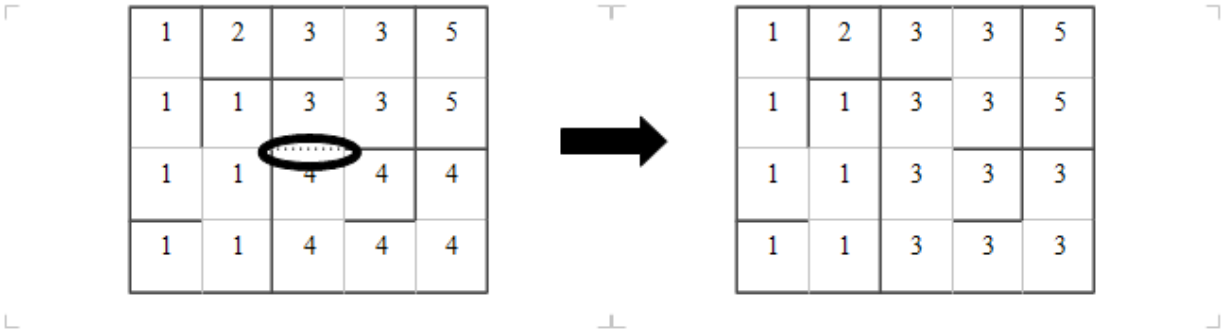
Méthode 2 : Fusion aléatoire de chemins

On part aussi d'un labyrinthe où tous les murs sont fermés, et l'on associe une valeur numérique différente à chaque cellule. À chaque itération, on choisit un mur à ouvrir et si les deux cellules possèdent des valeurs différentes, on ouvre le mur et les deux chemins contenant les deux cellules prennent la même valeur (peu importe laquelle). Lorsque toutes les cellules ont la même valeur, le labyrinthe est terminé.

Exemple à l'initialisation



Exemple à une itération quelconque



2a. En réutilisant la structure de données Labyrinthe, écrivez la méthode `miseAJourZones(...)` qui permet de mettre à jour le labyrinthe lors d'une itération, avec comme paramètre le mur ou les cellules adjacentes au mur.

Vous pouvez, si besoin, enrichir votre structure de données d'informations supplémentaires. Dans ce cas, veillez à préciser clairement les attributs ajoutés.

2b. Proposez une structure de données différente pour que la mise à jour des valeurs soit plus performante.