

ALGORITHMIQUE AVANCÉE (420-W31-SF)

TRAVAIL PRATIQUE 2 - REMISE PRINCIPALE

PONDÉRATION - 14%

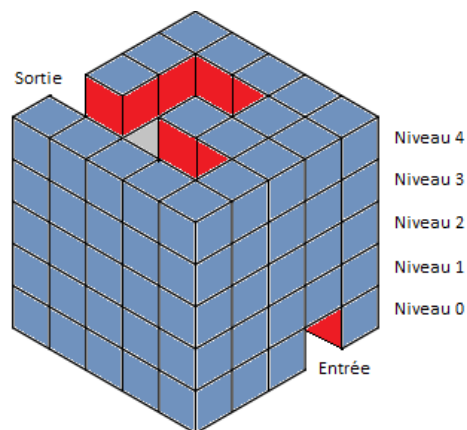
R.O.B

OBJECTIFS DU TRAVAIL

Ce travail pratique en équipe de 2 vise à maîtriser les 2 structures de données vues au cours jusqu'à maintenant, soient la **Pile** et la **File** dans un cas concret de résolution de problème. Il vise aussi à renforcer un concept algorithmique plus avancé : la **récurtivité**.

DISPOSITION LOGIQUE DES DONNEES

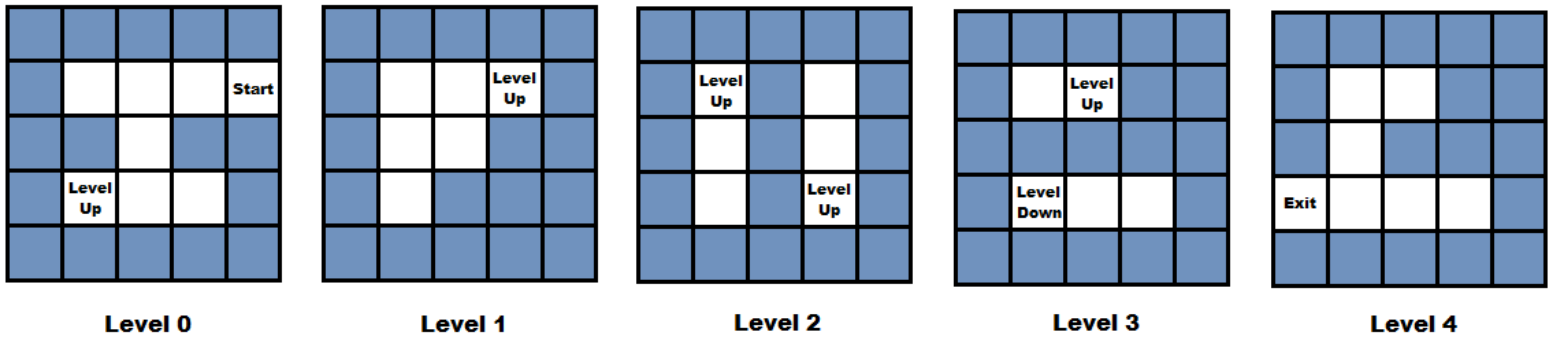
L'objet qui sera utilisé dans ce travail sera une structure en 3 dimensions (en l'occurrence, un cube) qui sera en fait un labyrinthe. Visuellement, le cube labyrinthe ressemblera à ceci :



Il aura les spécifications suivantes :

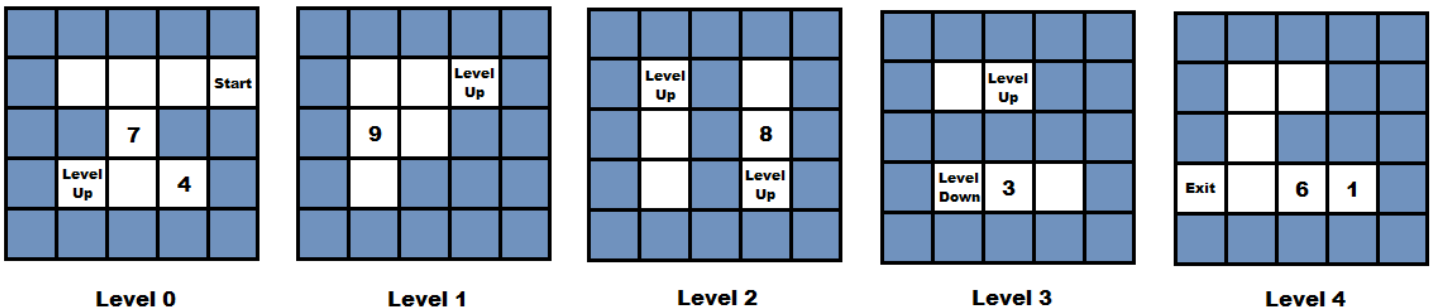
- Il a un seul point d'entrée et une seule sortie
- Il existe assurément un (ou plusieurs) chemins menant à la sortie
- Il est composé de niveaux superposés et tous délimités par un mur extérieur fermé (c'est pour cette raison que si on regarde le cube de côté, on ne voit pas l'intérieur)
- Des murs internes délimitent les chemins
- Les niveaux ont un « plancher » et un « plafond »
- Des « trous » permettent de monter ou de descendre de niveau, ce sont uniquement par ces « trous » que l'on pourra monter ou descendre dans le labyrinthe.
- Une unité du cube sera appelée un « bloc »

Suivant ces spécifications, le cube illustré ci-haut pourrait avoir l'anatomie interne suivante :



N'oubliez pas que les niveaux ont des planchers et des plafonds. Autrement dit, les cases « Level Up » et « Level Down » sont les seules façons de monter ou de descendre. Même si naturellement on pourrait penser qu'un bloc de libre « au-dessus » ou « en-dessous » de notre position courante serait accessible dans le niveau supérieur ou inférieur, ce n'est pas le cas. Cette remarque sera cruciale lors de la construction de la structure.

Finalement, certains blocs du cube contiendront des points (valeurs numériques comprises entre 1 et 9 inclusivement). Ces points serviront dans une des problématiques à résoudre. Nous aurons donc au final des cubes qui auront par exemple cette constitution :



En résumé, les blocs possibles dans le cube seront les suivantes :

- Bloc d'entrée
- Bloc de sortie
- Un mur
- Un bloc vide (possible d'y circuler)
- Un bloc numérique (possible d'y circuler)
- Un bloc d'accès au niveau supérieur(*)
- Un bloc d'accès au niveau inférieur(*)

(*) Note : Ces blocs sont en fait des espaces vides circulables, la seule différence est que le prochain pas fera en sorte qu'on se retrouvera à un niveau différent.

FORMAT DES DONNEES

La composition des cubes sera détaillée dans un fichier texte. Les caractères correspondant aux blocs possibles sont les suivants :

Type de bloc	Caractère dans le fichier texte
Bloc d'entrée (Start)	S
Bloc de sortie (Exit)	E
Mur	*
Bloc circulaire	Espace vide
Bloc numérique (circulaire)	Valeur du nombre
Bloc d'accès au niveau supérieur (Up)	U
Bloc d'accès au niveau supérieur (Down)	D

Par exemple, la représentation textuelle du niveau 0 serait la suivante :

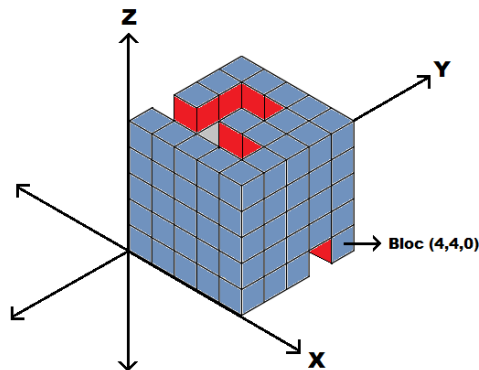
	<pre>* * * * * * S * * 7 * * * U 4 * * * * * *</pre>
--	--

Le fichier texte qui représentera un cube contiendra tous les niveaux séparés par le symbole « + ». Ce délimiteur facilitera le stockage en mémoire de la structure.

Exemple : Voir fichiers fournis avec l'énoncé

STOCKAGE DES DONNEES


Le stockage des blocs en mémoire devra être fait dans un tableau **statique** 3 dimensions **dont les indices respectent absolument cette représentation du cube dans l'espace**. La coordonnée d'un bloc est précisée par l'emplacement de son coin inférieur gauche dans l'espace. **Aucun bloc ne doit être NULL**. Tous les blocs adjacents devront être chaînés. Si le bloc est en périphérie, ses chaînages « extérieurs » devront être mis à NULL.



Les coordonnées du niveau 0 étant par exemple :

(0,4,0)	(1,4,0)	(2,4,0)	(3,4,0)	(4,4,0)
(0,3,0)	(1,3,0)	(2,3,0)	(3,3,0)	(4,3,0)
(0,2,0)	(1,2,0)	(2,2,0)	(3,2,0)	(4,2,0)
(0,1,0)	(1,1,0)	(2,1,0)	(3,1,0)	(4,1,0)
(0,0,0)	(1,0,0)	(2,0,0)	(3,0,0)	(4,0,0)



 Les chaînages sont cruciaux lors du stockage du cube. Référez-vous à la struct Block pour les pointeurs à utiliser. Gardez en tête que même si un bloc a un bloc supérieur ou inférieur de chaîné, la seule façon d’y mettre le pied est si ce bloc est de type Up (U) ou Down (D).

Exemple de chaînage pour le bloc (2,1,3) :

Bloc	Coordonnée
Au-dessus	(2,1,4) (circulable si le bloc courant est de type « U »)
En-dessous	(2,1,2) (circulable si le bloc courant est de type « D »)
À gauche	(1,1,3) (circulable si pas un mur (*))
En face	(2,2,3) (circulable si pas un mur (*))
À droite	(3,1,3) (circulable si pas un mur (*))
En arrière	(2,0,3) (circulable si pas un mur (*))

PROBLEMES A RESOUDRE

Le programme à produire est un programme en C++ de type Console. On aura un visuel très simple qui contiendra les auteurs du programme (crédits), un menu permettant la résolution de 2 problèmes ainsi que la possibilité de quitter définitivement le programme.

Problème 1 : Le chemin de sortie

À partir du bloc d'entrée, vous devrez demander au robot ROB de trouver le chemin de sortie en explorant le cube. La solution sera **obligatoirement stockée dans une pile de blocs**. Tant que ROB peut progresser dans le cube, il empilera les blocs sur lesquels il marche. S'il arrive dans un cul-de-sac, ROB rebrousse chemin en dépilant les blocs du cul-de-sac pour repartir dans une autre direction dès que possible.

Lorsque la sortie sera atteinte, on affichera à la console le chemin à suivre **dans le bon ordre** (**attention** : la pile de calcul doit être inversée car le premier pas effectué depuis l'entrée se retrouve au fond de la pile). Par exemple :

```
Le chemin de sortie est :  
[2,0,0]  
[2,1,0]  
[2,2,0]  
[2,2,1]  
[1,2,1]  
[1,2,2]  
[1,3,2]  
[1,3,3]  
[1,3,4]  
[1,2,4]  
[1,1,4]  
[0,1,4]  
Appuyez sur une touche pour continuer...
```



Les points distribués dans le labyrinthe ne serviront pas dans cet algorithme

Afin d'avoir une constance dans la résolution de cet algorithme, nous allons avoir un **ordre d'inspection prédéterminé**, autrement dit ROB va **systematiquement** regarder s'il y a un chemin possible dans **l'ordre suivant** :

- 1) Au plafond (un trou pour monter de niveau)
- 2) Au plancher (un trou pour descendre de niveau)
- 3) À gauche
- 4) À droite
- 5) En face (sur la prochaine valeur Y)
- 6) En arrière (sur la valeur Y qui nous précède)

Indice : L'inspection se fait sur le contenu du bloc (son caractère dans le fichier texte puisque tous les blocs sont non NULL)



Il est impératif de suivre cet ordre dans votre algorithme

Gardez en tête que si le robot veut faire ce déplacement :



Il doit se rendre sur le bloc d'en arrière et non sur celui d'en avant à cause de votre chaînage de pointeurs (Rappel : voir exemple de chaînage à la page 4).

Problème 2 : Le total des points

À partir du bloc d'entrée, vous devrez encore une fois faire appel à ROB pour explorer le cube au complet afin de collecter tous les points disponibles dans ce dernier. On utilisera cette fois-ci une approche **récursive**. L'idée est de **diviser** le problème (effectuer un appel récursif) lorsque ROB arrive à une intersection où plusieurs avenues sont possibles. Effectuer un appel récursif systématiquement sur le prochain bloc ne constitue pas une division du problème.

On utilisera **obligatoirement une file de blocs** pour stocker les blocs qui possèdent des points (seulement ceux-ci).

Lorsque la récursivité aura terminé, on affichera les coordonnées des blocs trouvés avec le nombre de points qu'ils contiennent (le contenu de la file). On affichera aussi le total des points :

```
6 points dans le bloc [1,2,1]
3 points dans le bloc [1,3,2]
2 points dans le bloc [1,1,2]
5 points dans le bloc [2,3,3]
8 points dans le bloc [3,2,3]
Pour un total de 24 points
Appuyez sur une touche pour continuer...
```

Aperçu du menu

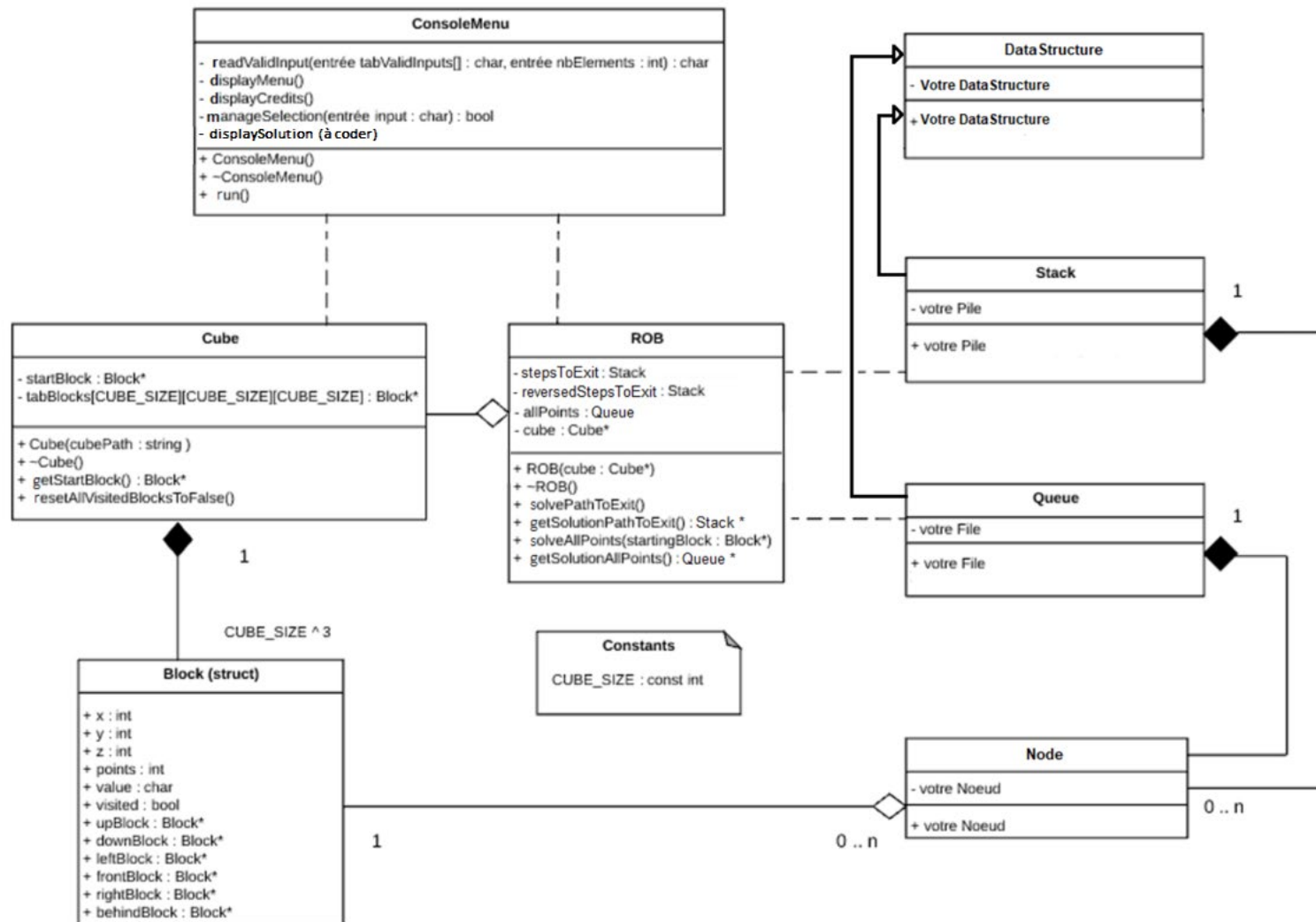
Le menu permettra la résolution des 2 problèmes et affichera le résultat demandé. On pourra faire ces opérations tant qu'on ne demande pas de quitter :

```
-----
                          TRAVAIL PRATIQUE 2
                          R.O.B
                          AUTEUR  :
                          Daniel Huot
-----
Que voulez-vous faire ?
Appuyez sur 1 pour solutionner l'algorithme du chemin de sortie
Appuyez sur 2 pour solutionner l'algorithme de tous les points
Appuyez sur q pour quitter le programme.
```



Attention, la résolution d'un algorithme viendra changer certaines valeurs dans les blocs, pour passer d'un algorithme à l'autre, vous devrez réinitialiser certaines valeurs (voir méthode de la classe Cube à ce sujet)

DIAGRAMME DE CLASSES





1) Vous devez implémenter toutes les classes, variables et méthodes publiques du diagramme de classes.

2) Aucune classe, méthode ou variable publique (+) ne peut être ajoutée mais vous pouvez ajouter autant de méthodes privées (-) que vous voulez (principe d'encapsulation). De plus, vous ne pouvez pas ajouter, modifier ni enlever aucune variable membre.

COMMENTAIRES DANS LE CODE

Afin de guider le professeur dans la correction des algorithmes, les instructions contenues dans les méthodes **solvePathToExit** et **solveAllPoints** devront être commentées de façon rigoureuse et pertinente.

FICHIERS DE TRAVAIL

Une solution de départ ainsi que 2 fichiers textes de cubes seront donnés avec ce travail, ils sont tous 2 valides et possèdent des solutions aux 2 problèmes :

- cube5.txt (cube à 5 niveaux, 27 points dans le cube)
- cube10.txt (cube à 10 niveaux, 170 points dans le cube)

Le professeur possède également des fichiers de dimensions correspondantes pour la correction, tous 2 valides et avec des solutions aux 2 problèmes.

Conseil : Développez les algorithmes sur le plus petit cube pour faciliter le débogage !

TEMPS DE TRAVAIL EN CLASSE

5,5 heures de travail en classe seront accordées pour effectuer ce travail. Le professeur a déterminé la date de remise en prenant pour acquis que ces heures seraient assidûment travaillées par les 2 membres de l'équipe, tout comme les heures de travail personnel.

Il a aussi pris pour acquis que les structures de données étaient prêtes, testées et robustes.

ÉQUITÉ DU TRAVAIL D'ÉQUIPE

L'activité dans le dépôt Git sera inspectée afin de s'assurer que les 2 coéquipiers ont contribué de façon équitable au travail. Il va de soi qu'une bonne séparation du travail et que des revues de code entre pairs sont encouragées afin de ne pas être pénalisé pour une erreur de codage ou de conception de votre coéquipier.

Si le professeur juge que la contribution est inéquitable (exemple 20%-80%), l'étudiant qui n'a pas assez contribué verra sa note pondérée à la baisse au prorata de sa participation. L'autre étudiant ne sera pas en situation de bonus en telle situation.

MODALITES DE REMISE

La date de remise de ce travail est le : **dimanche 25 août, fin de journée (23h59)**

Le travail devra être remis à 2 endroits :

1) Remise git :

- Assurez-vous que le dépôt contient le code final avant l'heure limite. Les dépôts d'équipe seront clonés immédiatement après l'heure de tombée pour correction.

2) Remise LEA (remise de sécurité **obligatoire**)

- Remise par 1 membre de l'équipe seulement (désignez un responsable)
- Solution nettoyée (clean/nettoyer solution + .vs + extern supprimés)
- Le travail est archivé dans un fichier nommé XXXXXXXX_XXXXXXX_TP2.zip où XXXXXXXX représente vos matricules
- Le programme compile et s'exécute sur Visual Studio 2022 (logiciel supporté pour le cours, voir plan de cours)



Les programmes qui ne compilent pas ou ne s'exécutent pas ne seront pas corrigés.



Bon travail !