Les Trains

Directives

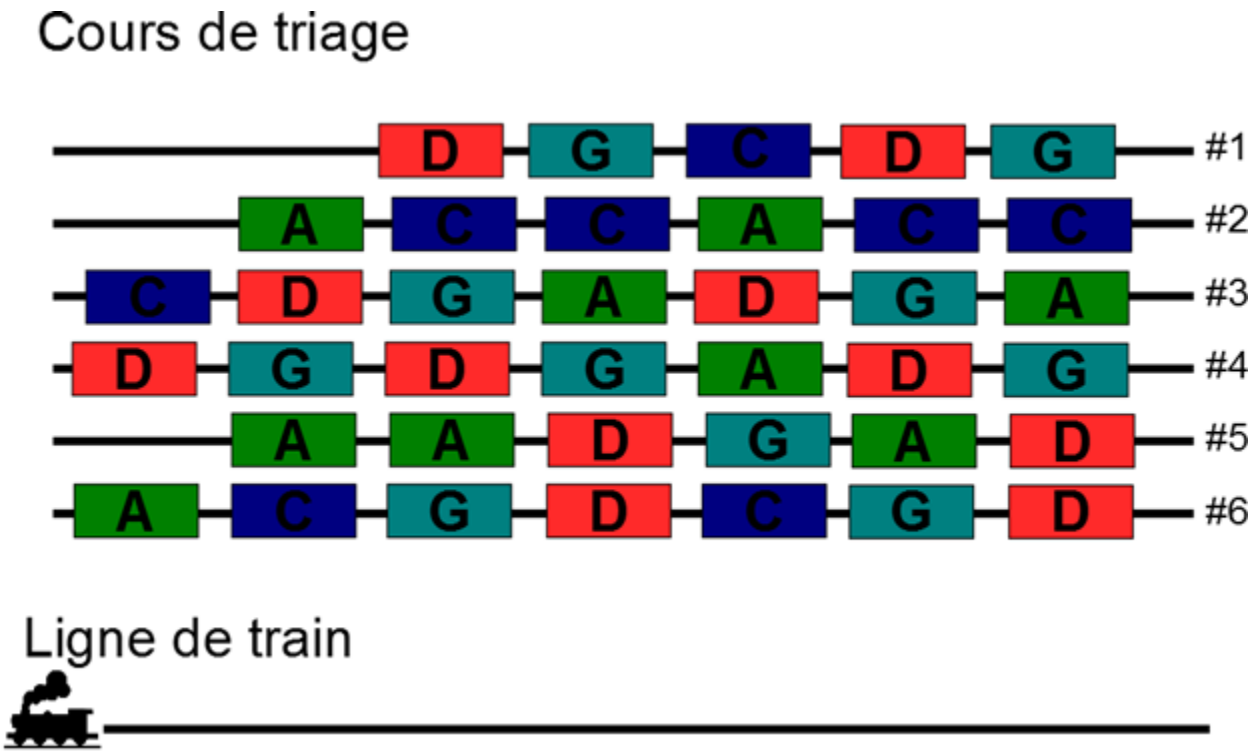
Vous aurez 48 heures pour fournir une solution C# fonctionnelle à la problématique qui suit. La solution doit avoir une représentation visuelle (voir la section Requis) et doit fonctionner avec l'interface **ITrainsStarter.cs** (que vous ne pouvez modifier).

Une fois les 48 heures écoulées, vous serez invités à venir travailler dans les bureaux de **GSoft** avec des membres de l'équipe de développement et un environnement reflétant notre ambiance journalière. Nous travaillerons tous ensemble dans votre code, mais avec de nouvelles contraintes ajoutées à la problématique. Il faut donc essayer d'avoir une solution bien architecturée, compréhensible et qui permettra de facilement diviser les tâches.

Problématique

Éric est chef de triage dans la gare de triage au Canadian National. Son travail quotidien consiste à organiser, planifier et déplacer des wagons pour la mise en place de trains.

Une gare de triage consiste en plusieurs « lignes » de wagons et une ligne de train. Prenez ce modèle :



Cette gare de triage a six lignes de wagons. Chaque wagon est représenté par un rectangle et chaque lettre représente une destination différente.

Le travail du chef de triage est de placer sur la ligne de train l'ensemble des wagons pour une destination sélectionnée, et ce de la manière la plus efficace possible. Pour ce faire, il utilise une petite locomotive de triage, qui sert à tirer ou pousser des trains entre les différentes lignes.

Voici quelques règles qu'il doit suivre :

- La locomotive de triage peut tirer de 1 à 3 wagons continus.
    - Exemple : Tirer de la ligne #5 les deux wagons A sur la ligne 1
- Chaque ligne de triage peut accommoder un maximum de 10 wagons. Elle peut en avoir moins (espaces libres).
- Le chef de triage ne peut tirer les wagons que de la gauche vers la ligne de train.
- Les wagons mis sur la ligne de train doivent être exclusivement ceux de la destination sélectionnée, et une fois mis sur la ligne de train, ils ne peuvent être déplacés.
- Idéalement, les wagons des lignes les plus proches sont déplacés. Il est plus coûteux de déplacer des wagons de la ligne 1 à la ligne 6 que de la ligne 1 à la ligne 2.
- Idéalement, les wagons sont déplacés en groupes (maximum de 3 par groupe). Il est moins coûteux de déplacer d'un coup 3 wagons

continus de la ligne 1 à la ligne 2 que de faire trois voyages de 1 wagon entre ces deux lignes.

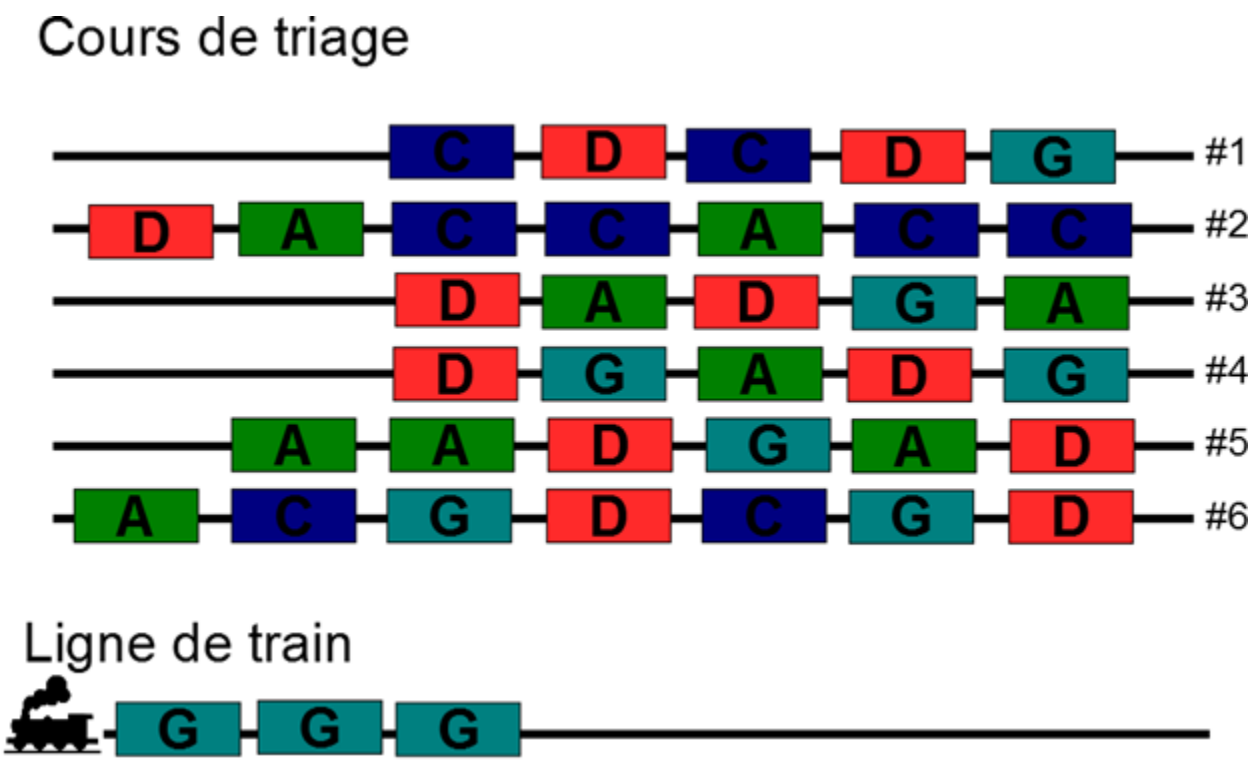
Éric a besoin d'un logiciel qui lui fournit la liste des mouvements à faire pour préparer sa ligne de train pour une destination particulière. Idéalement, ce serait la liste de mouvements la plus optimisée, celle qui requiert le moins de mouvements possible.

Exemple

Exemples de mouvements de wagons avec le modèle ci-dessus :
Destination sélectionnée : G

- Déplacer 1 wagon (D) de la ligne #1 sur la ligne #2
- Déplacer 1 wagon (G) de ligne #1 sur la ligne de train
- Déplacer 2 wagons (C et D) de la ligne #3 sur la ligne 1
- Déplacer 1 wagon (G) de la ligne #3 vers la ligne de train
- Déplacer 1 wagon (D) de la ligne #4 sur la ligne 3
- Déplacer 1 wagon (G) de la ligne #4 vers la ligne de train

Après ces déplacements, nous aurions une gare de triage dans cet état :



Évidemment, le travail n'est pas encore terminé (il reste encore des wagons G à mettre sur la ligne de train).
Mais ça vous donne une idée de ce qu'Éric attend comme liste de mouvements de wagons.

Requis

- Le logiciel doit pouvoir lire un fichier qui indiquera l'état des lignes de triage (voir plus bas pour une explication du format du fichier).
- Le logiciel recevra l'information de 3 façon possible : Texte, WebAPI (JSON), XML
- Le logiciel devra demander à l'utilisateur la destination du train (représentée par une lettre)
- Le logiciel calculera la *meilleure* liste de mouvements à effectuer pour préparer le train et écrira à l'écran la série de mouvements que le chef de triage devra faire dans l'ordre pour préparer son train sur la ligne de train.
- Le logiciel devra aussi répondre à l'interface **ITrainsStarter.cs** et sa méthode **Start()** (afin de rouler des tests unitaires).

Format du fichier de lecture

Texte

Le fichier de lecture représentant l'état initial de la gare de triage consiste en six lignes de 10 caractères. Chaque caractère correspond à un wagon pour une destination particulière. Les zéros représentent un espace vide.

Un exemple de contenu de fichier de lecture (données similaires à notre modèle initial plus haut, mais avec 10 wagons par ligne) :

```
00DGCDGEFA
0ACCACCGDE
CDGADGADEE
DGDGADGCAA
0AADGADCGD
ACGDCGDEGD
```

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<yard>
  <lines>
<line>00DGCDGEFA</line>
<line>0ACCACCGDE</line>
<line>CDGADGADEE</line>
<line>DGDGADGCAA</line>
<line>0AADGADCGD</line>
<line>ACGDCGDEGD</line>
</lines>
</yard>
```

WEB API

`https://sgd-dev-test-technique-trains.azurewebsites.net/api/trainyard`

`format de retour :`

```
{"configuration":"00DGCDGEFA\r\n0ACCACCGDE\r\nCDGADGADEE\r\nDGDGADGCAA\r\n0AADGADCGD\r\nACGDCGDEGD"}
```

Exemples d'entrées et de sorties de l'interface **ITrainsStarter.cs**

Ce fichier a pour but de permettre une interface entre votre code et nos tests unitaires. Certains tests de base sont fournis avec la solution pour mieux comprendre l'intéraction. Voici un exemple simple d'entrées et de sorties pour cette classe. Nous avons utilisés seulement deux lignes de trains pour simplifier le résultat. La ligne 0 représente ici la "Ligne de train". Dans les deux exemples ci-dessous, la destination C a été choisie.

En entré (string array) :

```
00000ACDGC
00000000DG
```

En sortie (string) :

```
A,1,2;C,1,0;DG,1,2;C,1,0
```

Voici un exemple plus complexe.

En entré (string array) :

```
00000AGCAG
000DCACGDG
```

En sortie (string) :

```
AG,1,2;C,1,0;AGD,2,1;C,2,0;A,2,1;C,2,0
```

Si la solution est impossible, une string vide devrait être retourné.