Programmation élémentaire

Lem in

Responsable Astek astek_resp@epitech.eu

Abstract:

Table des matières

.1 Détails administratifs

- Rendu : `~/rendu/prog_elem/lem_in`
- Votre binaire devra compiler avec un Makefile.
- Votre binaire se nommera `lem_in`

`Attention aux droits de vos fichiers et de vos répertoires`

.2 Sujet

Des étudiants en magie ont fabriqué Hex, une machine à penser. À base de fourmis pour les calculs, de ruches et d'abeilles pour la mémoire, d'une souris pour... euh oui pourquoi au fait ?, de fromage (pour nourrir la souris), d'une plume pour écrire.

```
Pour plus d'info référez-vous aux livres** de Terry Pratchett,
histoire de changer du 42. (**) Ook !
```

Et qu'est ce qu'on en fait ?

On va s'intéresser plus particulièrement à sa partie calculs. Son fonctionnement ? Simple ! On monte une fourmilière avec tout son lot de tunnels et de salles, on met des fourmis d'un côté et on regarde comment elles trouvent la sortie.
Comment on monte une fourmilière ? On a besoin de tubes et de boîtes.

On relie les boîtes entre elles par plus ou moins de tubes. Un tube relie deux boîtes entre elles et pas plus.
Une boîte peut être reliée à une infinité d'autres boîtes par autant de tubes qu'il en faudra.

Ensuite on enterre le tout (où vous voudrez) pour que les fourmis ne puissent pas tricher en regardant avant de commencer.

Comme ici, on est pas trop bricolage à coup de boîtes, scotch et bouts de ficelle, on va en faire une version hightech.
Le but du projet sera donc de faire un simulateur de "Hex".

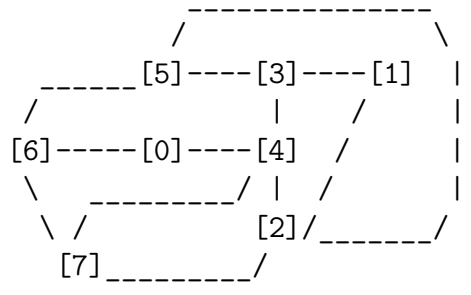
Votre programme va recevoir sur l'entrée standard la description de la fourmilière pour la forme suivante :

```
nombre_de_fourmis\n
les_salles\n
les_tubes\n
```

La fourmilière est donnée par ses liaisons :

```
##start
1 23 3
2 16 7
#commentaire
3 16 3
4 16 5
5 3 9
6 1 0
7 4 8
##end
0 9 5
0-4
0-6
1-3
4-3
5-2
3-5
#autre commentaire
4-2
2-1
7-6
7-2
7-4
6-5
```

Ce qui représente :



On a donc deux parties :

- La définition des salles sous la forme suivante : nom coord_x coord_y
- Puis la définition des tubes : A-B

Attention, les noms des salles ne seront pas forcément des chiffres dans l'ordre et encore moins continus (on pourra trouver des salles aux noms zdfg, 256, qwerty, etc ...).

Les coordonnées des salles seront toujours entières. À noter qu'il est possible d'insérer des commentaires commençant par des `#`.
Les lignes commençant par un `##` sont des commandes modifiant les propriétés de la ligne qui vient juste après.

Par exemple : `##start` indique l'entrée de la fourmilière et `##end` la sortie. Toute commande inconnue sera ignorée.

Le but du projet est de trouver le moyen le plus rapide de faire traverser la fourmilière par `n` fourmis.

Évidemment, il y a quelques contraintes. Pour arriver le premier, il faut prendre le chemin le plus court (et pas forcément pour autant le plus simple), ne pas marcher sur ses congénères, tout en évitant les embouteillages.

Au début du jeu, toutes les fourmis sont sur la salle indiquée par la commande `##start`. Le but est de les amener sur la salle indiquée par la commande `##end` en prenant le moins de tours possible. Chaque salle peut contenir une seule fourmi à la fois (sauf `##start` et `##end` qui peuvent en contenir autant qu'il en faut). À chaque tour vous pouvez déplacer chaque fourmi une seule fois et ce suivant un tube (la salle réceptrice doit être libre).

Vous devez sortir le résultat sur la sortie standard sous la forme suivante :

```
1  nombre_de_fourmis\n
2  les_salles\n
3  les_tubes\n
4  Px-y Pz-w Pr-o ...\n
```

Où `x, z, r` sont des numéros de pions et `y, w, o` des noms de salles.

La première ligne non conforme ou vide entraine la fin de l'acquisition de la fourmilière et son traitement normal avec les données déjà aquises.

Exemples :

- Si on a en entrée :

```
1  3\n
2  ##start
3  0 1 0\n
4  ##end
5  1 5 0\n
6  2 9 0\n
7  3 13 0\n
8  0-2\n
9  2-3\n
10 3-1\n
```

Cela représente le graphe suivant :

`[0]-[2]-[3]-[1]`

Vous devez sortir :

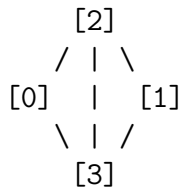
```
3\n
##start
0 1 0\n
##end
1 5 0\n
2 9 0\n
3 13 0\n
0-2\n
2-3\n
3-1\n
P1-2\n
P1-3 P2-2\n
P1-1 P2-3 P3-2\n
P2-1 P3-3\n
P3-1\n
```

Voilà, c'est fini donc c'est fait en 5 tours (ce message ne fait pas partie de la sortie).

- Si on a en entrée :

```
3\n
2 5 0\n
##start
0 1 2\n
##end
1 9 2\n
3 5 4\n
0-2\n
0-3\n
2-1\n
3-1\n
2-3\n
```

Cela représente le graphe suivant :



Vous devez sortir :

```
3\n
2 5 0\n
##start
0 1 2\n
##end
```

```
6     1 9 2\n
7     3 5 4\n
8     0-2\n
9     0-3\n
10    2-1\n
11    3-1\n
12    2-3\n
13    P1-2 P2-3\n
14    P1-1 P2-1 P3-2\n
15    P3-1\n
```

Voilà, c'est fini donc c'est fait en 3 tours.

Ce n'est pas aussi simple que ça en a l'air. Quant à savoir quel type d'opération les étudiants en magie pouvaient bien faire avec un tel ordinateur, tout ce qu'on en sait aujourd'hui, c'est que l'électricité c'est plus fiable :)

.3 Bonus

En bonus, pourquoi ne pas coder un visualisateur de fourmilière ?

Soit en deux dimensions, vue de "dessus". Voir pourquoi pas du point de vue d'une fourmi dans un environnement de couloirs en 3D comme dans un wolf.

Pour l'utiliser il suffirait d'un `./lem_in < map_fourmilliere.txt | ./visu-hex`

À noter que puisque les commandes et commentaires sont répétés sur la sortie, il est donc possible de passer des commandes spécifiques au visualisateur (pourquoi pas des couleurs, des niveaux ?)

Vous l'aurez noté, les coordonnées des salles seront utiles seulement ici.