Devoir maison

Recherche opérationnelle S3.

Afizullah RAHMANY
Numéro d'étudiant : 20170171

Ce devoir maison a été généré automatiquement et aléatoirement pour Afizullah RAHMANY. Il contient 3 exercices : un exercice d'application du cours sur 1 point, un exercice intermédiaire sur 1 point et un algorithme à coder sur 2 points. Pour répondre à ce devoir, vous devez rendre, sur `exam.ensiie.fr`*, un fichier texte nommé* `rahmanyafizullah_20170171.txt`*. Ce fichier devra contenir* ***dans l'ordre*** *les solutions des 3 exercices tels que demandé dans chaque exercice. Toute ligne vide dans le fichier sera ignorée (vous pouvez donc sauter des lignes entre 2 exercices, ou au sein d'un exercice). ATTENTION : un non respect du format pour un exercice entraînera la note de 0 pour l'exercice. Il vous est possible de tester votre format ainsi que votre dernier exercice avec le script* `check` *qui vous a été fourni avec ce sujet. Vous pouvez par exemple le copier avec votre fichier solution sur une machine de l'école et exécuter*

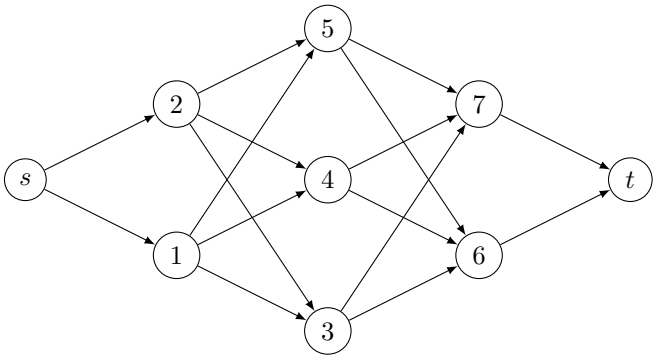
```
./check rahmanyafizullah_20170171.txt
```

Si votre fichier est au bon format, le script vous le dira explicitement. Le code de votre dernier exercice peut être dans ce cas testé sur 100 tests unitaires. Si un test ne passe pas, le script vous l'affichera pour que vous puissiez corriger votre code. Vous pouvez demander de l'aide aux professeur/chargés de TD, ou aux autres élèves, mais sachez que le sujet est différent pour tout le monde.

Exercice — ***#1-3-4 Algorithme de marquage***

Utilisez l'algorithme de marquage dans le réseau suivant où s et t sont respectivement la source et le puits. On a déjà calculé un flot admissible. Ce flot et les capacités des arcs sont indiquées dans le tableau.

s	1 : [3/3], 2 : [5/37],
1	3 : [3/6], 4 : [0/7], 5 : [0/7],
2	3 : [4/4], 4 : [1/3], 5 : [0/1],
3	6 : [7/7], 7 : [0/3],
4	6 : [1/1], 7 : [0/9],
5	6 : [0/6], 7 : [0/6],
6	t : [8/30],
7	t : [0/25],



Afin de garantir l'unicité du marquage, on suppose que l'algorithme trie tous les arcs dans l'ordre alphanumérique (l'ordre du tableau : $(s, 1)$, $(s, 2)$, ...). Puis, à chaque itération, il choisit le premier arc dont une des deux extrémités peut être marqué et marque ce noeud. Il réénumère les arcs en partant du début à chaque itération.

Vous donnerez cette réponse au format suivant : vous noterez en premier le numéro de l'exercice précédé par un `#`,
puis 1 ligne contenant 8 entiers et/ou lettres 'E' séparés par un espace. Pour chaque noeud, dans l'ordre, en commençant par 1 et en terminant par t, indiquez le marquage de ce noeud : le numéro du noeud multiplié par 1 ou -1 selon le signe du marquage. Si le noeud est non marqué, indiquez 'E'.

Par exemple :

```
# 1-3-4
0 4 8 12 16 20 24 E E
```

Exercice — *#2-2-2 Algorithme de Johnson : 3 machines*

4 tâches doivent être effectuées sur une machine M_1 puis sur une machine M_2 puis, enfin, sur une machine M_3. À la fin de son exécution sur M_1, chaque tâche peut commencer son exécution sur M_2 si la machine est disponible ou sinon elle doit attendre qu'elle le soit. De même, elle doit ensuite être exécutée sur une machine M_3. Les durées d'exécution des tâches sur les trois machines sont données dans le tableau suivant :

Durée $t(j, M)$	j_1	j_2	j_3	j_4
M_1	60	160	120	70
M_2	20	150	10	50
M_3	80	180	20	60

Appliquer l'algorithme de Johnson adapté au cas de 3 machines pour trouver la durée minimum d'exécution minimum sur ce problème.

Vous donnerez cette réponse au format suivant : vous noterez en premier le numéro de l'exercice précédé par un `#`,
puis 1 ligne contenant un entierégal à la durée minimum d'exécution.

Par exemple :

```
# 2-2-2
500
```

Exercice — *#3-1-7 Algorithme de Floyd Warshall*

Coder l'algorithme de Floyd Warshall pour calculer pour chaque paire de noeud d'un graphe i et j, le prédecesseur de j dans un plus court chemin de i à j. Utilisez le langage C.

Pour cela, vous coderez une fonction avec la signature suivante :

```
void floyd_warshall_pred_algorithm(int n, int** weights, int** pred);
```

- `n` est le nombre de noeuds du graphe, numérotés de 1 à n.
- `weights` est un tableau de `n` × `n` entiers où `weights[i-1][j-1]` est le poids de l'arc reliant les noeuds i et j ou -1 si cet arc n'existe pas.
- `pred` est un tableau de `n` × `n` entiers. En entrée, `pred[i-1][j-1]` = -1 pour tout $1 \leq i, j \leq n$. Pour tout $1 \leq i, j \leq n$, votre fonction doit affecter dans `pred[i-1][j-1]` le prédécesseur du noeud j dans un plus court chemin de i à j si un tel chemin existe et -1 sinon.

On garantit, que, quelque soit l'entrée $2 \leq n \leq 300$, les poids sont positifs. S'il existe plusieurs solutions possibles choisissez en une arbitrairement.

Votre fonction doit compiler avec `gcc -Wall -Wextra -ansi` et se terminer en moins de 1 seconde. Vous pouvez déclarer d'autres fonctions ou des structures mais ne pouvez ni écrire de `#define` ou de `#include`. Vous pouvez utiliser tout ce qui est défini dans `stdlib.h` à l'exception de `system`. Cette bibliothèque sera inclue automatiquement. Ne mettez pas de commentaire. Elle sera testée sur 100 cas. Votre note est de 2 si tous les tests sont réussis, 1 si plus de 10 pourcents mais pas tous, et 0 sinon.

Vous donnerez cette réponse au format suivant : vous noterez en premier le numéro de l'exercice précédé par un `#`, puis écrivez votre code.

Par exemple :

```
#3-1-7
void floyd_warshall_pred_algorithm(int n, int** weights, int** pred){
}
```