Devoir maison

Recherche opérationnelle S3.

Zeyu CHEN
Numéro d'étudiant : 20170019

Ce devoir maison a été généré automatiquement et aléatoirement pour Zeyu CHEN. 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* `chenzeyu_20170019.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 chenzeyu_20170019.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 — ***#1-1-1 Problème de partition d'entiers : méthode itérative***

On souhaite partager l'ensemble d'entiers $X = \{x_1, x_2, \ldots, x_8\}$ en deux parties de sommes égales. On utilise pour cela la version itérative l'algorithme de programmation dynamique vu en cours. Pour tout $0 \leq i \leq 8$ et tout $0 \leq j \leq (\sum_{x \in X} x)/2$, on calcule $f(i, j)$ qui vaut TRUE s'il existe, parmi $x_1, x_2, \ldots, x_i$, des entiers dont la somme fait j et FALSE sinon. (si $i = 0$, la somme fait 0)

$$X = \{6, 10, 2, 1, 9, 4, 6, 2\}$$

A l'itération 4, on obtient les résultats suivants :

	0	1	2	3	4	5	6	7	8	9	10
$f(4, j)$	True	True	True	True	False	False	True	True	True	True	True
		11	12	13	14	15	16	17	18	19	20
$f(4, j)$		True	True	True	False	False	True	True	True	True	False

Donnez la valeur de $f(5, j)$ pour tout $1 \leq j \leq 20$.

Vous donnerez cette réponse au format suivant : vous noterez en premier le numéro de l'exercice précédé par un `#`,
puis 1 ligne contenant 21 entiers binaires 0 ou 1 séparés par 1 espace où le j^{e} chiffre de la ligne est 0 si $f(5, j-1)$ est FALSE, et 1 sinon.

Par exemple :

```
# 1-1-1
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

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	200	160	10	110
M_2	70	90	10	40
M_3	90	130	140	200

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 — ***#3-3-5 Calcul d'une coupe minimum***

Coder en C un algorithme qui, connaissant un flot maximum d'un réseau, renvoie une coupe de capacité minimum.

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

```
void minimum_cut_algorithm(int n, int** flow, int** capa, int* cut);
```

- `n` est le nombre de noeuds, numérotés de 1 à n. La source s est 1, le puits t est n.
- `flow` est un tableau de `n` $\times$ `n` entiers où `flow[i-1,j-1]` vaut -1 si l'arc (i, j) n'existe pas ou le flux de cet arc sinon.
- `capa` est un tableau de `n` $\times$ `n` entiers où `capa[i-1,j-1]` vaut -1 si l'arc (i, j) n'existe pas ou la capacité de cet arc sinon.
- `cut` est un tableau de `n` entiers. En entrée de la fonction, `cut[i-1] = -1` pour tout noeud i du graphe. Votre fonction doit calculer une coupe de capacité minimum (S, T) séparant s et t. Pour chaque noeud i, votre fonction doit affecter dans `cut[i -1]` la valeur 0 si $i \in S$ ou la valeur 1 si $i \in T$.

On garantit, que, quelque soit l'entrée $2 \leq n \leq 1000$ et le flot est maximum. S'il existe plusieurs solutions, 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 :

```
#3-3-5
void minimum_cut_algorithm(int n, int** flow, int** capa, int* cut){
}
```