L3/M1, 2005–2006

# Algorithmique effective – TD 3

## 1 Hors-d'œuvre

**Exercice 1**
Parcours de graphe.

1. Écrire un programme effectuant un parcours en largeur d'un graphe non-orienté. Le graphe est donné en entrée par le nombre de sommets, suivi de sa matrice d'adjacence. Le nombre de sommets sera inférieur à 1000. On écrira en sortie chaque sommet rencontré, ainsi que sa distance au premier sommet.
2. Pour des raisons obscures, dans nos graphes il n'y a pas plus de 50 sommets à même distance du sommet initial. On veut minimiser la taille de la file de traitement des sommets. Modifier le programme précédent en conséquence : il s'agit de réutiliser le début de la file lorsque l'on a atteint la fin.

**Exercice 2**
Mise en page.
Pour aligner à droite l'affichage dans une colonne de taille fixée, ou pour modifier le nombre de chiffres après la virgule d'un nombre flottant, il y a deux méthodes (C ou C++). Les expérimenter. Code utile :

```
printf("%10d\n", 19);          printf("%18.2f\n", 3.14159);
printf("%80s\n", "Alignement à droite.");

#include <iomanip.h>
std::cout << setw(10) << 19 << std::endl;
std::cout << setprecision(3) << setw(18) << 3.14159 << std::endl;
std::cout << setw(80) << "Alignement à droite." << std::endl;
```

## 2 Couplage dans un biparti, suite

**Exercice 3**
Couplage maximum.
Coder l'algorithme polynomial de recherche d'un couplage de poids maximum dans un graphe biparti pondéré. On programmera d'abord la recherche d'un couplage maximal dans un biparti non pondéré, en utilisant des listes d'adjacence. Ensuite, on se servira de ce programme pour coder le cas du biparti pondéré. Ce dernier sera en revanche donné par sa matrice d'adjacence (puisque le graphe biparti est complet). Comparer avec le Branch & Bound du DM 2.

## 3 Dessert

En DM pour le prochain TD, on s'intéresse aux flots dans un graphe. Soit un graphe orienté $G = (V, A)$ avec une capacité par arc (un entier positif, disons). Sur cette arête il peut circuler au maximum $c$ unité de flot (« trucs » par unité de temps). Si l'on choisit une source $s$ et une destination $t$, on peut alors faire circuler des trucs entre $s$ et $t$, en respectant le sens et la capacité de chaque arc. On cherche à maximiser le nombre de trucs que l'on peut faire circuler par unité de temps (en régime permanent). C'est le problème du flot maximum.

On cherche à calculer le flot maximum lorsqu'on nous donne un graphe orienté et pondéré en entrée. On codera l'algorithme polynomial de Ford-Fulkerson. On trouvera une introduction aux flots, ainsi qu'une description détaillée de l'algorithme dans le Cormen (chapitre 26 de la deuxième édition), par exemple. Il s'agit d'améliorer successivement le flot courant, jusqu'à obtenir un flot maximum, en trouvant à chaque étape un chemin entre $s$ et $t$ qui peut être amélioré.

Comme application, on donne le problème suivant, que l'on résoudra en utilisant des flots. Il peut y avoir d'autres méthodes, que vous pourrez aussi coder si le cœur vous en dit (facultatif).

**Exercice 4**

The Grand Dinner. PC : 111007 ; UVa : 10249

Each team participating in this year's ACM World Finals is expected to attend the grand banquet arranged for after the award ceremony. To maximize the amount of interaction between members of different teams, no two members of the same team will be allowed to sit at the same table.

Given the number of members on each team (including contestants, coaches, reserves, and guests) and the seating capacity of each table, determine whether it is possible for the teams to sit as described. If such an arrangement is possible, output one such seating assignment. If there are multiple possible arrangements, any one is acceptable.

*Input*

The input file may contain multiple test cases. The first line of each test case contains two integers, $1 \leq M \leq 70$ and $1 \leq N \leq 50$, denoting the number of teams and tables, respectively. The second line of each test case contains $M$ integers, where the $i$-th integer $m_i$ indicates the number of members of team $i$. There are at most 100 members of any team. The third line contains $N$ integers, where the $j$-th integer $n_j$, $2 \leq n_j \leq 100$, indicates the seating capacity of table $j$.

A test case containing two zeros for $M$ and $N$ terminates the input.

*Output*

For each test case, print a line containing either 1 or 0, denoting whether there exists a valid seating arrangement of the team members. In case of a successful arrangement, print $M$ additional lines where the $i$-th line contains a table number (from 1 to $N$) for each of the members of team $i$.

*Sample Input*

```
4 5
4 5 3 5
3 5 2 6 4
4 5
4 5 3 5
3 5 2 6 3
0 0
```

*Sample Output*

```
1
1 2 4 5
1 2 3 4 5
2 4 5
1 2 3 4 5
0
```