TP2- Plus courts chemins, Arbres Couvrants, Problèmes de transport.

Objectifs: Utiliser les algorithmes présentés en cours (recherche de plus courts chemins, arbres couvrants minimaux, et calcul de flots maximaux).

Nous utilisons la librairie de fonctions Matlab BGL (*Boost Graph Library*) contenant quelques algorithmes définis sur des graphes. Les fonctions disponibles de la librairie sont visibles dans help contents.

La matrice de distances doit être stockée sous forme creuse (sparse). La fonction *sparse(D)* permet de réaliser cette opération.

```
%%%% TP2_exo_1.m %%%%%%%
addpath('matlab_bgl');      %load graph libraries
addpath('matlab_tpgraphe'); %load tp ressources
load TPgraphe.mat;          %load data
%% INIT %%
G=sparse(D);
```

Exercice 1

On cherche à trouver l'arbre des plus courts chemins.

Q1) Choisir une source *src* et utiliser la fonction *shortest_paths*(*G*, *src*) pour obtenir les plus courts chemins partant du sommet choisi et afficher l'arbre obtenu à l'aide de *viz_spt*. Commenter la longueur de l'arbre observé.

```
%% Choose arbitrary src node for SPT
src=14;
%% Compute SPT rooted in src node
[wp spt_]=shortest_paths(G,src); %wp=weight path, spt_=shortsetpathtree structure
%% Vizualize
viz_spt(G,spt_,pos,cities);
```

Exercice 2

On s'intéresse maintenant à l'arbre couvrant de poids minimal.

Q1) Utiliser les fonctions *Prim_mst* et *Kruskal_mst* pour calculer les arbres couvrants minimaux. Comparer les arbres trouvés.

Q2) Modifier les valeurs de quelques arêtes pour trouver un arbre obtenu par l'algorithme de Prim différent de celui obtenu avec Kruskal. On pourra ajouter le paramètre *struct('root',sommet)* pour choisir un *sommet* initial pour l'algorithme de Prim.

```
%% Compute MST by PRIM
%% Changer les valeurs initiales pour obtenir deux arbres différents entre PRIM et KRUSKAL
mst_=prim_mst(G,struct('root',XXX à faire));
%% Vizualize
viz_mst(G,mst_,pos,cities);
%% Compute MST by KRUSKAL
mst_=kruskal_mst(G);
%% Vizualize
viz_mst(G,mst_,pos,cities);
```

Exercice 3

Un opérateur de réseau privé souhaite interconnecter les capitales européennes pour acheminer un trafic d'entreprise. La capacité des liens est inversement proportionnelle aux distances selon la formule suivante : *Capacité = [10000/Distance]*.
Les liens avec une capacité inférieure à 11 ne sont pas exploitables.

On ajoute une source et un puit pour modéliser ce problème sous forme de problème de transport. Le trafic généré par la source et qui devrait être reçu par le puit est de 10.

Q1) Compléter le code suivant pour mettre à jour la matrice de capacités *bw*.

Q2) La fonction *vis_cut* affiche la coupure minimale. Les sommets reliées à la source sont marquées avec le symbole 'o' et ceux reliés au puit sont marqués avec '+'. Commenter le résultat affiché.

Q3) L'opérateur souhaite monter en charge avec un trafic généré de l'ordre de 100 pour chaque site. Quelle est la valeur du flot maximal obtenu pour les capacités considérées. Commenter les résultats affichés.

Q4) Quelle propriété du graphe valué considéré fait que la coupure minimale contient le plus souvent des liens incidents aux sites source ou puit.

Q5) Trouver deux sites (site source et site puit) pour lesquels la coupure ne passe pas par les arêtes incidentes à ces sites.

```
%   TP2_exo_1.m
%% Choose arbitrary srcs, dsts and virtual capacities for these nodes
srcs=[2, 21]; %for Europe
dsts=[10, 16];  %for Europe
virtual_capacity=100;%for Europe
%% Create bandwidth graph from distance graph
n=size(D,1);
%% add virtual src & dst (so create graph with two extra nodes)
bw=zeros(n+2,n+2);
%% n+1th node is for virtual src;virtual src connected to srcs cities
bw(n+1,srcs)=virtual_capacity;
bw(srcs,n+1)=virtual_capacity;
%% n+2th node is for virtual dst;virtual dst connected to dsts cities
bw(n+2,dsts)=virtual_capacity;
bw(dsts,n+2)=virtual_capacity;
```

```
%% bandwidth is inversely proportional to distance
bw(1:n,1:n)=XXX à faire;
%% Inf is on the diagonal, so change it to 0
bw(bw==Inf)=0;
%% links with too less bw are not interesting for operators
XXX Filtrage des liens non expoitables à faire;%for Europe
%% Compute Max Flow bw virtual src & dst nodes
Gbw=sparse(bw);
[max_val cut_ R F]=max_flow(Gbw,n+1,n+2);
%% Vizulize
viz_cut(Gbw,cut_,pos,cities,srcs,dsts)
```

Exercice 4

Q1) Reprendre les questions *Q1*, *Q2* et *Q3* de l'*exercice 3* avec des villes allemandes. La capacité exploitable dans ce cas est de 120.
Dans ce cas d'étude, le trafic généré par chacun des deux sites 2 et 14 est de 800. Ce trafic est reçu par les sites 11 et 21.
On utilisera la fonction *dataGermany* pour remplacer les données européennes par les données allemandes.

Q2) L'opérateur prévoit un budget pour augmenter les capacités de trois liens afin d'améliorer la valeur du flot. Quels liens proposez vous en priorité et avec quelles capacités ?

Exercice 5

On s'intéresse à la répartition de la taille des composantes connexe d'un graphe.

Le fichier *TP2_exo_2.m* utilise la fonction *components(sparse(D))* à ce titre.

Q1) Déterminer le nombre et l'ordre des composantes connexes dans le graphe des distances entre villes allemandes.

Q2) On généré un graphe aléatoirement et on étudie la répartition des composantes en fonction de la probabilité d'existence d'une arête entre deux sommets. A partir de quelle probabilité obtient t-on un graphe connexe ?