Algorithmie Avancée
TP/TD Connexion et chemins

A. Document Algo_TDParE_Birmele.pdf
Vous traiterez les exercices **1.3** à **1.5**

B. Plus court chemin dans un graphe non pondéré avec File d'attente (lien avec BFS)
Dans un premier temps, vous utilisez votre code manipulant des listes doublement chaînées comme file d'attente. La terminologie : enfiler correspond à insérer (à vous de choisir la place) et défiler à supprimer de la liste tout en récupérant la valeur de la cellule (le numéro x du sommet).

```
// Procédure qui recherche le plus court chemin depuis un sommet de référence
//        Paramètres :
//        adjacence : matrice d'adjacence du graphe
//        ordre : nombre de sommets
//        s : numéro de sommet de référence
//        l : tableau dynamique alloué des longueurs minimales des sommets depuis s
//        pred : tableau dynamique alloué des prédécesseurs des sommets

void  plusCourtChemin (int**adjacence, int ordre, int s, int *l, int *pred) {
        // Variables locales
int *marques ; // tableau dynamique indiquant si les sommets sont marqués ou non
int x, y ; // numéros de sommets intermédiaires
t_file *f ; // file d'attente de sommets à créer en s'inspirant des listes doublement chaînée avec un .h
et un .c dédié

        // Allouer le tableau marques de taille « ordre »
        …
        // Initialiser les marquages et les longueurs minimales à 0
        for (x=0 ; x<ordre ; x++) {
        marques[x] = 0 ;
        l[x] = 0 ;
        }

        // Marquer le sommet s à 1
        marques[s] = 1 ;

        // Créer (allouer) la file f et enfiler s dans f
…

        while (…) { // Tant que la file f n'est pas vide
        …    // Défiler le premier sommet x de la file f

        // Pour tous les sommets y adjacents à x et non marqués
                for (y=0 ; y<ordre ; y++)
                        if (adjacence[x][y] && !marques[y]) {
                        marques[y] = 1 ;  // marquer le sommet y

                        … // enfiler le sommet y dans f

                        pred[y] = x ; // x est le prédécesseur de y
                        l[y] = l[x]+1 ; // // incrémenter la longueur de y
                }
        }
}
```

Analyser ce code en lien avec les TP précédents, les cours également, et de sa complexité mémoire et calcul.

Activités supplémentaires :

Obtient-on un arbre couvrant avec l'algorithme précédent ? Pourquoi ?

Quel est le diamètre d'un graphe ? d'un arbre ?

Obtient-on le diamètre de ce graphe avec l'algorithme précédent ?

Que faudrait-il faire pour l'obtenir si on imagine le *BFS(G)* comme un flot de propagation d'arêtes marquées partant d'un point source *s* ?

Générez en C des graphes aléatoires simples via matrice d'adjacence (*drand48() ou autre au choix*) de plus en plus grand et vérifiez la complexité de l'algorithme.

Bonus : Eventuellement en R ou Python afficher une courbe de complexité avec en abscisse la taille du problème et en ordonnée le temps de calcul.