Exercice 1

1) Expliquer comment la structure de tas peut être utilisée dans l'algorithme de Dijkstra.
2) Réécrire l'algorithme de principe en utilisant le tas.

Rép. :

1)

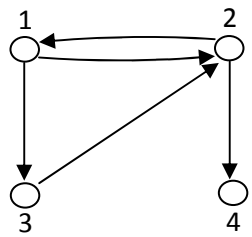
1) On construit un tas dont chaque élément représente le sommet avec le poids associé
2) La recherche du min va consister à supprimer la racine du tas
3) La mise à jour des distances va consister à changer la valeur dans le tas et donc réorganiser le tas.
4) Algo de principe :

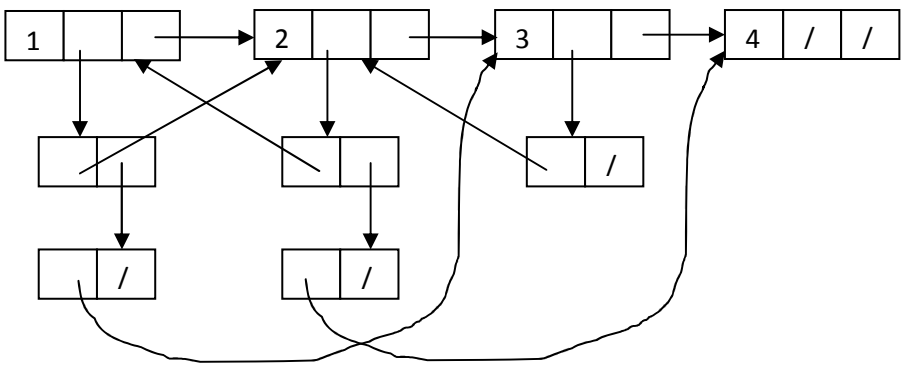
2)

```
Algorithme Dijkstra rapide
Entrée : un graphe orienté valué de n sommets m arcs
Sortie : le plus court chemin entre un sommet d'origine et les
autres sommets

InitialiserTas (Tas) ;
InsérerDansTas (Tas, sommet S, 0); //initialiser distance(S,S) à 0
pour i = 1 à n-1
      insérerDansTas (Tas, sommet i, +∞) ; //initialiser distance(S, i) à 0

tant qu'il reste des sommets non marqués faire
   pour i = 1 à n faire
      //trouver le sommet T qui donne la distance minimale
      suppMinDuTas(Tas, T, distance(S,T));
      Marquer (sommet T) ;
      pour tout successeur U du sommet T faire
         si distance(S,T)+ poids(T,U) < distance(S,U)
         alors
            min = distance(S,T)+ poids(T,U) ;
            //mettre à jour la valeur associée au noeud U, et donc réorganiser le tas
            MAJduTas (Tas, sommet U, min)
```

Exercice 2

Une variante de la structure de données « liste d'adjacence » permet de représenter l'ensemble des sommets par une liste chaînée, où on associe à chaque sommet un pointeur vers son successeur qui appartient à la liste chaînée des sommets, comme dans l'exemple suivant :





1) Déclarer cette structure
2) Ecrire les fonctions : ajoutArc(u,v) ; suppArc(u,v)

Rép :

1) Déclaration

```
typedef struct bloc1 {
    int sommet ;
    bloc2 *succ ;
    bloc1 *suiv ;
}
typedef struct bloc2 {
    bloc2 *succ ;
    bloc1 *suiv ;
}
```

2) Ajouter un arc entre deux sommets u,v (on suppose que l'arc n'existe pas)

```
void ajoutArc (bloc1 *tete, int u, int v) {
    bloc1 *courant, *pteur_u ;
    bloc2 *pteur_v, nouv ;

    //chercher  u
    Courant = tete ;
    while (courant != NULL && courant->sommet != u) courant = courant->suiv ;
    If (courant != NULL) pteur_u = courant ;

    //chercher  v
    Courant = tete ;
    while (courant != NULL && courant->sommet != v) courant = courant->suiv ;
    If (courant != NULL) pteur_v = courant ;

    //le plus simple est d'insérer en tete de la liste verticale
    Nouv = new bloc2 ;
    Nouv->succ = pteur_u->succ ;
    Nouv->suiv = pteur_v ;
    Pteur_u->suiv =nouv ;
}
```

3) Supprimer un arc uv

```
void suppArc (bloc1 *tete, int u, int v) {
        bloc1 *courant1, *courant2, *precedent ;
        //chercher  u
        courant1 = tete ;
        while (courant != NULL && courant->sommet != u) courant = courant->suiv ;
        If (courant != NULL) pteur_u = courant ;
        //chercher le pteur vers v dans la liste verticale
        courant2 = courant1->succ ; precedent = courant2 ;
        while (courant2 != NULL && courant2->suiv->sommet !=v) {
                precedent = courant2 ;
                courant2 = courant2->succ ;
        }
        If (courant2->suiv->sommet ==v) { //supprimer courrant2
                q = courant2 ;
                precedent->succ = courant2->succ ;
                delete q ;

        }
}
```