

A) Représentation des fonctions Booléennes : ROBDD

On a présenté en cours les différentes fonctions permettant de construire le graphe ROBDD représentant une fonction Booléenne. La structure de données utilisée pour représenter un noeud BDD comporte un champs supplémentaire par rapport à la structure présentée en cours et en TME : Le champs **FLAG** est un indicateur binaire qui sera utilisé à la question Q4 : Dans certains parcours du graphe ROBDD, on souhaite marquer le fait qu'un noeud a déjà été rencontré : Si sa valeur est nulle, le noeud n'a encore été rencontré, sinon le noeud a déjà été rencontré. Ce champ FLAG n'est pas initialisé lors de la construction du graphe.

```
typedef struct bdd {  
    long      INDEX;  
    struct bdd *HIGH;  
    struct bdd *LOW;  
    char      FLAG;  
} bdd_t
```

Q1) (2 points) Représenter graphiquement les deux graphes ROBDD correspondant à la fonction $F = ABC + A'D$, pour les deux ordonnancements : $A > B > C > D$, et $D > C > B > A$. Pour faciliter la lecture, on attachera (en commentaire) à chaque noeud du graphe une expression Booléenne définissant la fonction Booléenne représentée par ce noeud.

On appelle "profondeur" du graphe ROBDD associé à la fonction F le nombre de noeuds BDD qui se trouvent sur le chemin le plus long entre le noeud racine représentant F , et l'un des deux noeuds terminaux représentant les constantes 0 et 1. On notera cette profondeur $\text{Prof}(F)$. On cherche à écrire la fonction $\text{ProfBdd}()$ qui prend pour unique argument un pointeur sur le noeud BDD représentant la fonction F , et renvoie la profondeur du ROBDD associé à F .

```
int ProfBdd(bdd_t *p)
```

Q2) (2 points) Ecrire en français la relation de récurrence permettant de calculer $\text{Prof}(F)$ en fonction de $\text{Prof}(FH)$ et $\text{Prof}(FL)$, si FH et FL sont les deux cofacteurs résultant de la décomposition de shannon de F . On n'oubliera pas de définir les cas terminaux.

Q3) (2 points) Ecrire en langage C la fonction $\text{ProfBdd}()$.

Le nombre total de noeuds BDD nécessaires pour représenter une fonction Booléenne F dépend à la fois de la table de vérité de la fonction F , et de l'ordre choisi pour les variables Booléennes appartenant au support de F . Pour pouvoir comparer différents ordonnancements de variables, on souhaite écrire en langage C une fonction qui prend pour unique argument un pointeur sur le noeud BDD représentant la fonction F , et renvoie le nombre total de noeuds utilisés pour représenter la fonction F . Attention : un même noeud BDD peut faire partie du sous graphe représentant FH et aussi du sous graphe représentant FL .

Q4) (3 points) Proposer - en Français - un algorithme permettant de calculer le nombre de noeuds BDD nécessaires pour représenter une fonction F, en utilisant le champs FLAG de la structure bdd_t. Définir précisément la relation de récurrence entre F, FH, et FL, permettant de calculer le nombre de noeud BDD associés à la fonction F.

Q5) (1 point) Ecrire en langage C la fonction ResetBdd() qui prend pour unique argument un pointeur sur le noeud BDD représentant la fonction F, et initialise à 0 le champs FLAG de tous les noeuds BDD appartenant au sous-graphe ROBDD représentant F.

```
void ResetBdd(bdd_t *p)
```

Q6) (2 points) Ecrire en Langage C la fonction CountBdd() qui prend pour unique argument un pointeur sur le noeud BDD représentant une fonction F, et qui renvoie le nombre de noeuds non marqués (tels que FLAG = 0) du graphe ROBDD représentant F.

```
int CountBdd(bdd_t *p)
```

B) Placement de cellules et partitionnement de graphe

L'algorithme de min-cut présenté en cours peut être utilisé pour optimiser le placement de cellules pré-caractérisées dans un plan. Cet algorithme prend en entrée un graphe non orienté, dont les noeuds ont été répartis en deux ensembles disjoints L et R, et cherche à optimiser cette partition en minimisant le nombre d'arcs coupants la frontière entre L et R.

Il existe cependant une différence entre la structure de données **graph_t** représentant le graphe à partitionner, et la structure de données **lofig** représentant la *net-list* de cellules. En particulier, un signal de la structure lofig peut connecter un nombre quelconque de connecteurs d'instances, alors qu'un arc de la structure graph_t définit une relation entre deux noeuds du graphe.

On rappelle ci-dessous les différentes structures de données utilisées pour représenter le graphe et la net-list

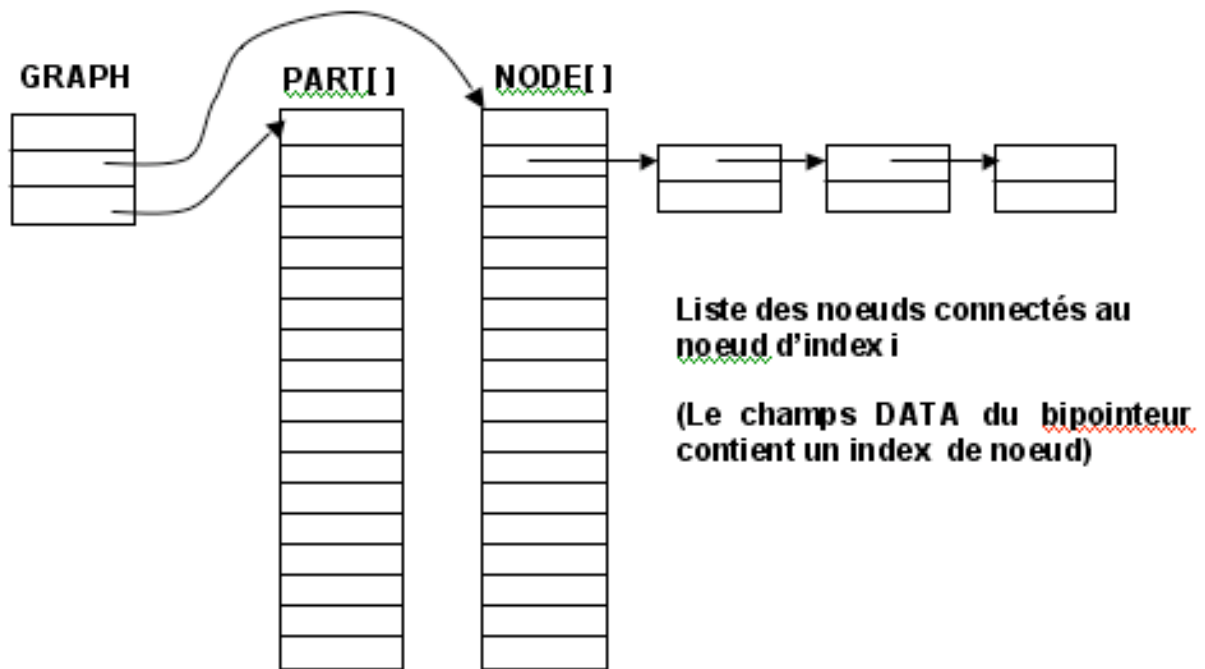
Représentation du graphe

```
typedef struct graph {
    unsigned    NBNODE;    // nombre de noeuds du graphe
    chain_t     * NODE;    // pointeur sur le tableau NODE[NBNODE]
    unsigned    * PART;    // pointeur sur le tableau PART[NBNODE]
} graph_t
```

```
typedef struct chain {
    struct chain *NEXT ;
    int          DATA ;
} chain_t ;
```

La fonction cons_chain() alloue un objet de type chain_t, initialise les deux champs et renvoie un pointeur sur l'objet créé en mémoire.

```
chain_t * cons_chain(chain_t * next , int data)
```



Représentation de la net-list

```
typedef struct lofig {
    struct lofig    *NEXT ;
    char            *NAME ;
    struct locon    *LOCON ;
    struct losig    *LOSIG ;
    struct loins    *LOINS ;
    void            *USER ;
} lofig_list ;

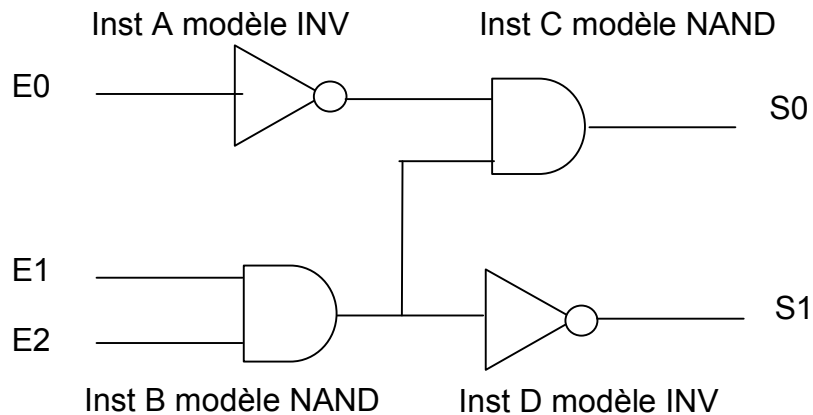
typedef struct locon {
    struct locon    *NEXT ;
    char            *NAME ;
    void            *ROOT ;
    struct losig    *SIG ;
    char            TYPE ;
    void            *USER ;
} locon_list ;

typedef struct losig {
    struct losig    *NEXT ;
    long            INDEX ;
    struct bip      *NAMECHAIN ;
    char            TYPE ;
    void            *USER ;
} losig_list ;

typedef struct loins {
    struct loins    *NEXT ;
    char            *INSNAME ;
    char            *FIGNAME ;
    struct locon    *LOCON ;
    void            *USER ;
} loins_list ;
```

Q7) (2 pts) Précisez ce que représentent les nœuds et les arcs du graphe non orienté représenté par la structure `graph_t` par rapport à la netlist.
 Quel est le type C des éléments des deux tableaux `PART[]` et `NODE[]`.
 Expliquer ce que représentent ces deux tableaux.

Q8) (1pts) Représenter le graphe (avec des ronds et des traits) correspondant à la netlist ci-dessous, vous choisirez arbitrairement les numéros de nœuds, mais vous indiquerez sur votre graphe à quelle instance ils correspondent. Représentez la structure `graph_t` correspondante.



On cherche donc à écrire la fonction `lofig2graph()`, qui construit en mémoire la structure `graph_t` à partir d'un pointeur sur la structure `lofig`:

```
graph_t *lofig2graph(lofig_list *ptfig).
```

L'idée générale de l'algorithme :

- On commence par "enrichir" la structure de donnée `lofig`, en utilisant les champs `USER` des différents objets pour y ajouter les informations dont on a besoin pour construire la structure `graph_t` :
 - Le nombre total de noeuds à créer dans le graphe est écrit dans le champ `USER` de la structure `lofig`,
 - L'index d'un nœud (choisi arbitrairement), est ajouté dans le champ `USER` de l'objet `loins` associé.
 - La liste des connecteurs branchés sur chaque signal est ajouté dans le champ `USER` de l'objet `lofig` par un appel à la fonction vue au partiel :


```
void lofigchain (struct lofig * lofig)
```
- Lorsque ce travail est effectué, on utilise la fonction `malloc()` pour créer les deux tableaux associés à la structure `graph_t`.
- Il reste alors à créer les listes chaînées représentant les arcs du graphe.

Notez

- que pour la construction du graphe, on ne prend pas en compte les connecteurs externes.
- que la fonction n'est pas chargée d'initialiser le contenu du tableau `PART[]`.

Q9) (3 points) Décrire précisément – en Français – les différentes étapes de l'algorithme réalisé par la fonction `lofig2graph()`.

Q10) (2 points) Ecrire en langage C la fonction `lofig2graph()`