Théorie des Graphes

Graphes arboricoles et forestiers

TPS 1A INOC

theoleyre@unistra.fr

Objectif : qu'est ce qu'un arbre et quels sont les différents algorithmes permettant de les calculer et exploiter ?

Durée : 2 heures

Exercice 1: Arbres et forêts couvrants

Soit $G = (S, A)$ un graphe non orienté, avec S l'ensemble de ses sommets et A celui de ses arêtes.

1. Rappeler une condition nécessaire portant sur le nombre de ses arêtes pour que G soit un arbre.
2. Montrer que si G est connexe il admet au moins un arbre couvrant, et que cet arbre couvrant est unique si et seulement si G est un arbre.
3. Montrer que si G est un arbre, il admet 2^{n-1} forêts couvrantes, où n est le nombre de sommets de G.

Exercice 2: Implémentation

On propose les structures de données suivantes pour représenter un arbre par la liste de ses sommets, chaque sommet contenant un pointeur vers son père qui définit l'arborescence. La racine de l'arbre pointe vers `NULL`.

```
struct sommet {
          int id;
          struct sommet * pere;
};

typedef struct sommet sommet;

struct maillon {
        sommet * elem;
        struct maillon * suivant;
};

struct liste {
        struct maillon * tete;
};
```

```
int est_vide_liste (struct liste l);

void init_liste (struct liste *l);

void ajout_liste(struct liste *l, sommet * elem);

///Attention: retourne NULL si liste vide
///           premier element sinon
sommet * tete_liste (struct liste l);

///Attention: retourne NULL si liste vide
sommet * supp_tete_liste(struct liste *l);

struct arbre {
        sommet * racine;
        struct liste arborescence;
};
```

Écrire le corps de la fonction de prototype :

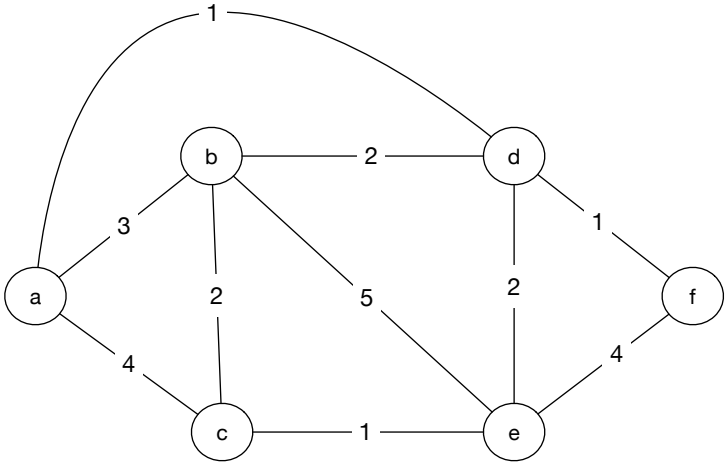
```
void traitement_Profondeur (struct arbre * ar, sommet * depuis, void (*traitement) (sommet *) );
```

qui applique à chaque sommet de l'arbre enraciné en `depuis` (par exemple la racine) la fonction `traitement`, avec une stratégie de parcours en profondeur dans un premier temps. On ne demande pas d'implémenter la fonction `traitement` qui pourrait être, par exemple, la fonction d'affichage suivante. Redéfinissez certaines de vos structures et fonctions (en particulier pour la gestion de vos listes) et implantez un algorithme de parcours en largeur selon un modèle analogue.

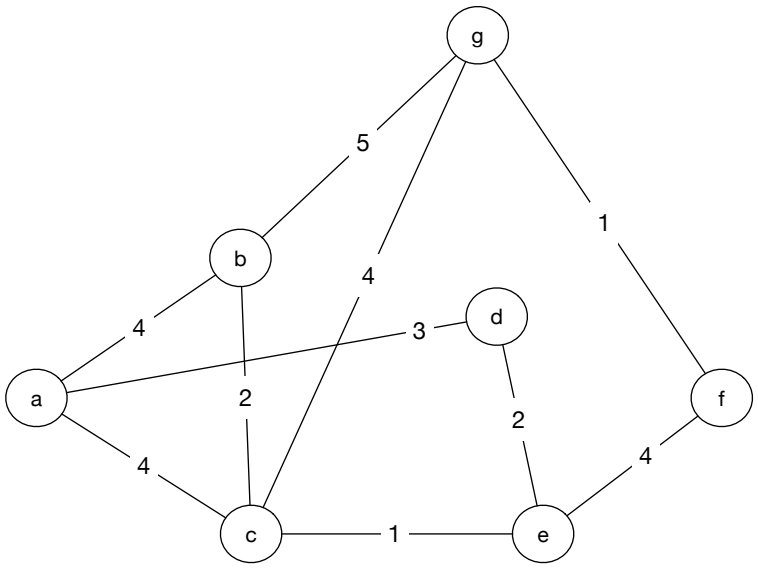
```
void affiche_sommet (sommet * s) {
        printf("sommet %i, ",s->id);
        if (s->pere==NULL) printf("racine\n");
        else printf("pere %i \n", s->pere->id);
}
```

Exercice 3: Arbres couvrants de poids minimum

On considère le graphe donné par la figure ci-après. Déterminer un arbre couvrant de poids maximal en utilisant successivement les algorithmes de Kruskal, puis de Prim.



Même question avec le graphe ci-après.



Exercice 4: Algorithme de Prim

Montrer que l'algorithme de Prim termine et est correct. Évaluer son coût dans le pire des cas en justifiant.

Exercice 5: Graphe Biparti

Pouvez vous démontrer qu'un graphe est biparti si et seulement s'il ne contient aucun cycle de longueur impaire ?

Exercice 6: Largeur arborescente

1. Montrer qu'un arbre T admet une décomposition arborescente de largeur 1.
2. Donner une décomposition arborescente du graphe C_5 [1] de largeur 3.
3. Cette décomposition est-elle la plus petite pour C_5 ? Quelle est la largeur de décomposition d'un graphe C_n ?

1. anneau comportant 5 sommets.