Algorithmique et Structures de Données 2

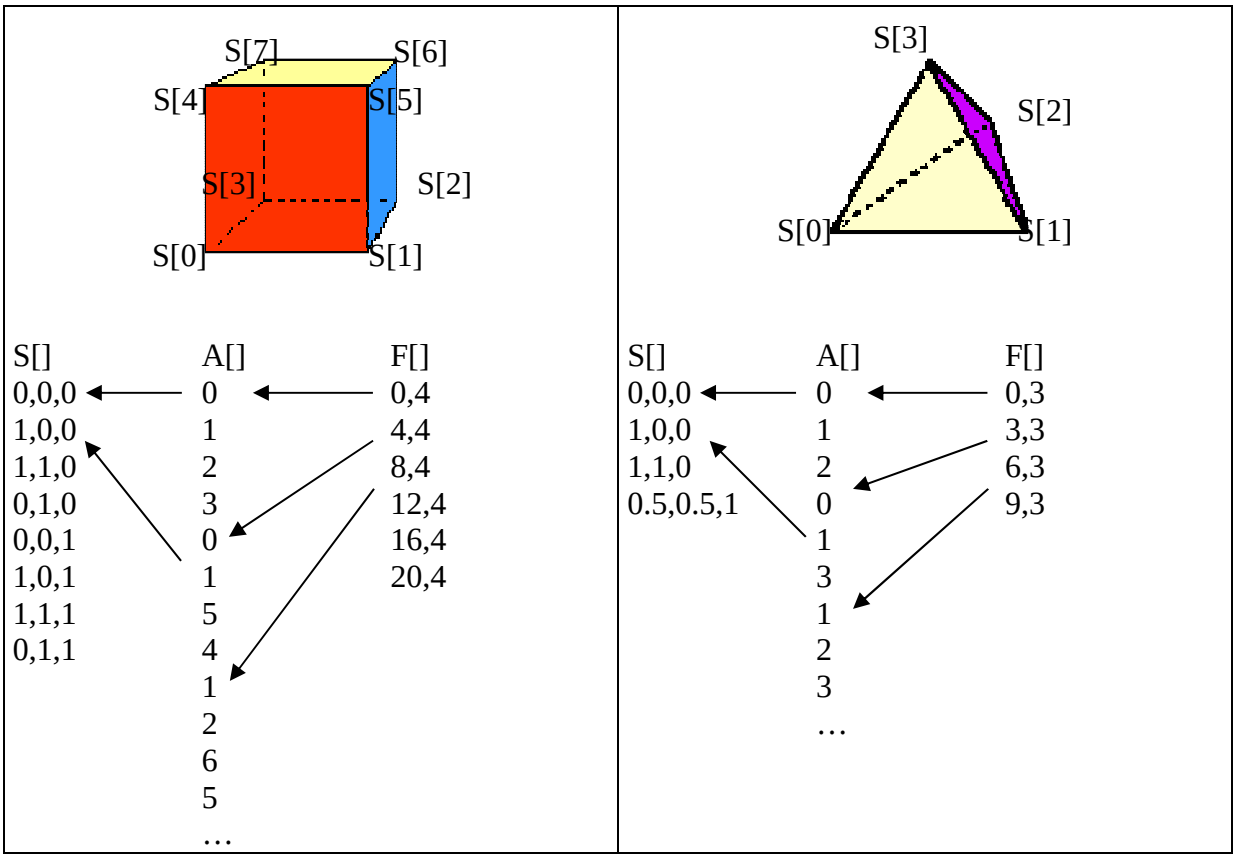
Devoir de TP libre
Année 2014 - 2015

Jean-Michel Dischler

Cadre du problème

On se propose de travailler avec des objets de type **polyèdres**. Un polyèdre est défini par un ensemble de **sommets** 3D (trois coordonnées de l'espace) et par un ensemble de **faces** polygonales. Ces faces peuvent être des triangles, des quadrilatères, etc.

Nous utilisons une structure *Polyedre* composée de trois listes: la première, *S* est la liste des sommets (un sommet étant un triplet de coordonnées réelles), la suivante, *A*, les arêtes, une arête étant un couple de sommets (*A* est une liste d'entiers où chaque entier représente un indice de sommet – 'est donc une table d'adressage indirect) et enfin la troisième, *F*, les faces. Une face est une liste d'arêtes. On la représente à l'aide d'un couple d'entiers, pour lequel le premier entier représente un indice dans la liste *A* des arêtes et le second le nombre d'arêtes du polygone. Par exemple, si F[k] = (p,n), alors la face numéro *k* est bordée par les arêtes (S[A[p]], S[A[p+1]]), (S[A[p+1]], S[A[p+2]]), ..., (S[A[p+n-2]], S[A[p+n-1]]), (S[A[p+n-1]] , S[A[p]]).
Les deux figures ci-dessous illustrent deux cas simples: un cube et un tétraèdre (la liste des arêtes n'est pas complète pour des raisons de place).



Implantation d'algorithmes sur les polyèdres en C

On se propose de définir les structures suivantes: **Point** (un triplet de flottants qui représente un point dans l'espace Euclidien 3D) et **Face** (un couple d'entiers). On définit par ailleurs des structures **ListPoints et ListeFaces,** modélisées par des tableaux contigües, ainsi qu'une structure **ListInt** qui permet de ranger des entiers. Ces trois structures sont des Listes et doivent donc proposer les opérations usuelles sur les listes : recherche (de l'élément en position k), insérer (en position k), supprimer (l'élément en position k), longueur, etc. On utilisera un **tableau borné** pour représenter ces listes.

Le type **Polyedre** est donnée par trois listes : une liste de Point, ***ListPoint***, une liste d'arêtes, de type ***ListInt*** et la liste de Face, de type ***ListeFace***.

1. Proposer une opération du type Polyedre permettant de charger un polyèdre à partir d'un fichier texte (en codage ascii).

Le fichier texte est structuré comme suit (exemple du tétraèdre):

```
OFF
4 4 12
0 0 0
1 0 0
1 1 0
0.5 0.5 1
3 0 1 2
3 0 1 3
3 1 2 3
3 0 2 3
```

La première ligne indique le format du fichier: OFF pour *Object File Format*. Les trois chiffres de la seconde ligne correspondent respectivement au nombre de sommets (longueur de la liste S[] des sommets), nombre de faces (longueur de la liste F[]) et nombre d'arêtes (longueur de la liste A[]). Suivent les coordonnées des points (trois coordonnées par ligne pour chaque sommet). Les chiffres suivants définissent les faces: chaque ligne correspond à une face. Le premier chiffre indique le nombre d'arêtes de la face (par exemple 3 pour un triangle), les chiffres qui suivent, sont les indices des sommets.
Note: les indices commencent à 0.

Ecrire également une fonction qui permet de faire une sauvegarde de polyèdre selon le même format.

2. Observateurs.

Implémenter les opérations suivantes pour le type **Polyedre**:

- NbSommet(p) -> renvoie le nombre de sommets du polyèdre p
- NbFace(p) -> renvoie le nombre de faces du polyèdre p
- GetSommet(p,i) -> renvoie le Point correspondant au sommet numéro i.
- NbSommetFace(p,i) -> renvoie le nombre de sommets de la face numéro i.
- SommetFace(p,i,j) -> renvoie l'indice du $j^{ème}$ sommet de la face numéro i.

3. Proposer une opération qui permet, pour un sommet* i *donné, de retourner la liste des faces contenant ce sommet.

L'opération prend en entrée l'indice du sommet *s* et renvoie une ListInt contenant les indices des faces concernées:

```
ListInt FacesIncidentes(Polyedre p, int s)
```

Lorsqu'une face contient un sommet donné, on dit que la face est **incidente** en ce sommet (d'où le nom de cette opération).

4. Proposer une opération qui permet, pour une face d'indice* i *donnée, de retourner la liste des faces adjacentes.

Deux faces sont dites **adjacentes** si elles partagent une même arête. Dans l'exemple du cube les faces numéro 0 et numéro 1 sont adjacentes, car elles partagent la même arête formée par les sommets d'indices 0 et 1. Note: deux faces ayant un même sommet ne sont pas forcément adjacentes.

La fonction prend en entrée l'indice de la face *f* et renvoie une liste *ListeInt* des indices des faces concernées:

```
ListInt FacesAdjacentes(Polyedre p, int f)
```

5. Proposer une fonction qui permet, pour une face d'indice* i *donnée, de retourner la liste des faces connectées.

Deux faces sont dites **connectées** si elles partagent un même sommet. Note: deux faces adjacentes sont forcément connectées.

```
ListInt FacesConnectees(Polyedre p, int f)
```

6. Calcul des composantes connexes.

Définition: une composante connexe d'un polyèdre est un ensemble **maximal** de faces connectées deux à deux. C'est à dire que pour toute face F[i] de cet ensemble, il existe au moins une face F[j] de cet ensemble telle que F[i] et F[j] soient connectées. Attention cet ensemble est *maximal*: cela signifie que toute face du polyèdre qui n'appartient pas à la composante connexe n'est connectée à aucune face de cette composante connexe. Un polyèdre est composé d'un ensemble de composantes connexes, où chacune d'elles peut être considérée comme un "sous-polyèdre" déconnecté des autres « sous-polyèdres ».

Pour les deux exemples précédents, il n'y a qu'une seule composante connexe.

En utilisant les fonctions précédentes, proposer un algorithme qui construit l'ensemble des composantes connexes d'un polyèdre donné. Pour un polyèdre *P* donné, la fonction retourne le nombre de composantes connexes, tout en remplissant un tableau *cc[]* de polyèdres, où chacun de ces polyèdres est une composante connexe. Note: s'il n'y a qu'une seule composante connexe, la fonction renvoie 1 et place une copie du polyèdre dans *cc[0]*. S'il y a plusieurs composantes connexes, chacun des polyèdres résultant de cet algorithme ne doit contenir qu'un nombre minimal de sommets (donc pas tous les sommets de *P*).

```
int Connexe(Polyedre p, Polyedre cc[])
```

Où cc est un tableau contenant des Polyedre.

Optimisation

Un polyèdre peut être considéré comme un graphe où les nœuds sont les sommets. On se propose à présent d'utiliser une nouvelle structure de données pour gérer des polyèdres que nous appelons **PolOptim**. Cette structure va utiliser, en plus des listes précédentes (ListFaces, ListePoints et ListInt), des listes chaînées d'entiers, ***Listc*** ainsi que des tables pour modéliser les arêtes du graphes.

1. Implémenter une liste chaînée d'entiers : Listc

On écrira les opérations usuelles: adjtete(), tete(), suivant(), estvide() et longueur().

1. Table de listes chaînée d'entiers : TabListc

Cette structure doit permettre de construire une « table de listes chaînées d'entiers ». Donc, à chaque élément de la table d'indice i correspond une liste chaînée de type Listc. On utilisera simplement un tableau pour représenter cette table. On supposera que sa taille reste constante. Proposer des opérations permettant de créer une table TabListec de taille N fixée et de mettre à jour un élément d'indice i dans cette table. Ecrire également une opération permettant de récupéré la valeur de l'élément d'indice i (donc la liste Listc correspondante).

2. Construction dePolOptim .

On définit un type « polyèdre optimisé » ***PolOptim*** comme étant un Polyedre, auquel on ajoute deux tables TabListc.

- La première ***sa*** est de la même taille que S. **sa** donne directement pour chaque sommet d'indice *i* la liste des faces incidentes (en fait, les indices des faces incidentes au sommet i).
- La seconde ***fi*** est de la même taille que F. **fi** donne directement pour chaque face d'indice *i* la liste des faces connectées.

Proposer un algorithme en complexité linéaire qui permette de construire un *PolOptim* à partir d'un *Polyedre*. La fonction prend en entrée un polyèdre *P* et créer un polyèdre de type *PolOptim*.

```
PolOptim Convertir(Polyedre p)
```

3. Implémentation des opérations.

Ré-implémenter l'ensemble des opérations précédentes, en les adaptant à la nouvelle structure PolOptim:

```
ListInt FacesIncidentes(PolOptim p, int s)
ListInt FacesAdjacentes(PolOptim p, int f)
ListInt FacesConnectees(PolOptim p, int f)
int Connexe(PolOptim p, PolOptim cc[])
```

Comparer la vitesse d'exécution de ces fonctions avec la structure initiale *Polyedre*.

On définira dans chacun des cas un *"main"* (programme de test) pour réaliser les jeux d'essais. Deux fichiers sont fournis à titre d'exemple : TOUR.OFF et CLAVIER.OFF (voir ci-dessous)

Remarques

Ce devoir pratique est à rendre au plus tard pour le **vendredi 8 mai 2015**. Il sera envoyé sous forme électronique. Le devoir électronique contiendra :

- les codes sources de votre programme C (uniquement les fichiers .c et .h, pas d'exe, ni de .o).
- un document word/rtf/pdf illustrant et documentant la réalisation (max. 10 pages)
- un document texte montrant le résultat de l'exécution des deux programmes (jeu d'essai sur les fichiers CLAVIER et TOUR)

Vous aurez à disposition sur le web, deux exemples de polyèdres sur lesquels vous pourrez exécuter des jeux d'essai et que vous pouvez télécharger sous:

https://dpt-info.u-strasbg.fr/~dischler/poly/

Vous avez également des images qui vous montrent ces objets en 3D. Notez que le format OFF est un format standard. Des logiciels gratuits comme par exemple *Blender* vous permettent de les visualiser sur votre PC.

Le devoir est à réaliser en binômes. Le nom du fichier que vous enverrez sera celui de vos noms de famille respectifs. Par exemple pour Fuchs et Meyer, le fichier portera le nom : *fuchs_meyer.tgz*. Les données doivent être compressées avec tar ou zip et doivent se décompresser dans un répertoire portant ce même couple de noms.