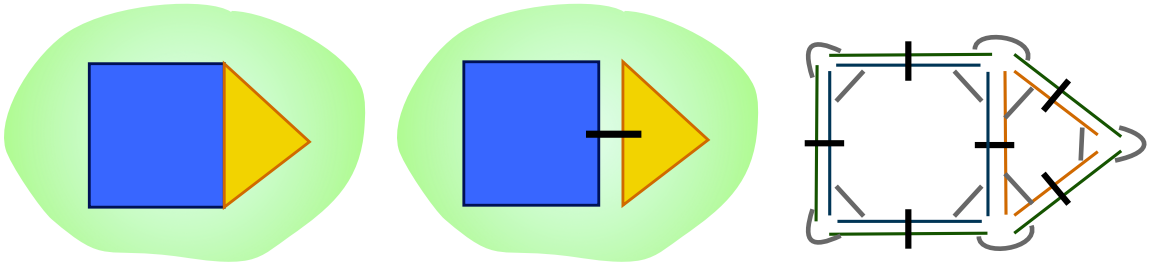
Examen d'Algorithmique IG3

Christophe Fiorio, Jérôme Fortin, Eleonora Guerrini

janvier 2015 – durée 2h00

1 Problématique : cartes combinatoires $2d$

Les cartes combinatoires permettent de représenter un objet en le subdivisant en éléments topologiques de dimension 0 à n. Ainsi on peut définir une 2-carte en décomposant un objet $2d$ selon ses faces, ses arêtes et ses sommets et en notant les relations d'incidence entre ces éléments : sommets d'une arête et d'une face, arête d'une face, et face. La Figure 1 présente une telle décomposition. L'objet de la figure 1.a est composé d'un triangle adjacent à un carré. On commence par



a. Un objet 2d. b. Faces disjointes. c. Arêtes disjointes.

Figure 1 – Processus de décomposition d'un objet pour obtenir la carte correspondante

décomposer l'objet selon ses faces (voir figure 1.b), l'adjacence entre les deux faces est marquée par un trait noir. On peut ensuite décomposer l'objet selon ses arêtes [1] (figure 1.c) et noter l'adjacence entre les éléments par un trait gris. Dans la théorie des cartes, ces éléments sont appelés *brins* et sont abstraits. Ils permettent de définir une carte. Il suffit en effet de définir les relations d'adjacence entre les éléments topologiques (sommets, arêtes et faces en $2d$) par des relations entre les brins. On notera β_i la relation entre les brins définissant l'adjacence entre des éléments de dimension i. Ainsi β_2 définit l'adjacence entre faces, et β_1 l'adjacence entre arêtes.

Rappelons dans un premier temps quelques notions mathématiques de base : une *permutation* d'un ensemble fini E est une séquence ordonnée de tous les éléments de E, chaque élément apparaissant exactement une fois. Par exemple si $E = \{a, b, c\}$, il existe 6 permutations de E :

$$abc, acb, bac, bca, cab, cba$$

Une *involution* σ dans un ensemble E est une permutation qui satisfait $\sigma(\sigma(x)) = x$. Un point fixe x d'une fonction σ est tel que $\sigma(x) = x$.

Nous pouvons maintenant donner la définition d'une 2-carte :

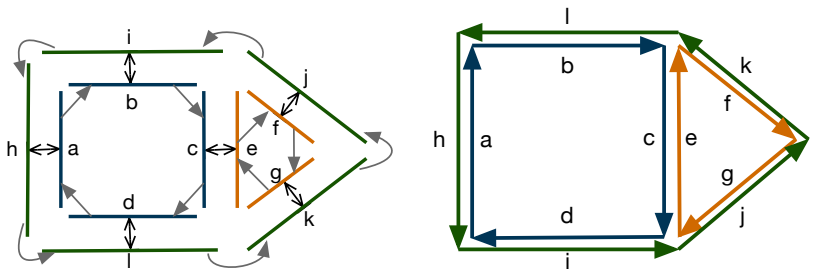
définition 1 *Une 2-carte combinatoire, (ou 2-carte) est un triplet $M = (D, \beta_1, \beta_2)$ où :*

1. *D est un ensemble fini de brins ;*
2. *β_1 est une* permutation *sur D ;*

1. Notez les arêtes vertes qui délimitent la face extérieure (l'espace extérieure).

3. β_2 *est une* involution *sur* D.

On peut voir un exemple de 2-carte à la figure 2.*a*. Sur cette figure les brins sont représentés par des traits, β_1 par des flèches grises et β_2 par des doubles flèches noires. On peut également représenter implicitement les relations β_1 et β_2 en dessinant les brins avec des flèches pointant vers l'image par β_1 du brin et en plaçant en vis-à-vis les brins reliés par β_2 (voir figure 2.*b*).



a. Représentation complète. b. Représentation implicite.

FIGURE 2 – Deux représentations de la même carte combinatoire.

En nommant chaque brin par une lettre, on peut représenter les relations β_i par le tableau suivant :

D	a	b	c	d	e	f	g	h	i	j	k	l
β_1	b	c	d	a	f	g	e	i	j	k	l	h
β_2	h	l	e	i	c	k	j	a	d	g	f	b

Intuitivement, une orbite $<f_1, \ldots, f_k>$ est l'ensemble de tous les brins que l'on peut atteindre par un parcours en appliquant dans n'importe quel ordre, n'importe quelle combinaison des permutations $f_1, \ldots, f_k$ [2] à partir d'un brin donné. Ces orbites permettent de définir les éléments de l'objet : sommet, arête, faces, objets, etc... Ainsi, en deux dimensions, l'orbite $<\beta_1 \circ \beta_2>$, notée $<\beta_{21}>$, permet de représenter un sommet. Par exemple, $<\beta_{21}>(c) = \{c, f, l\}$ permet de représenter le sommet dont est issu c, tandis que $\{c, e\}$ définit l'arête entre la face carrée et la face triangulaire. Un élément de l'objet est donc définit par un ensemble de brins. Par exemple, $\{e, f, g\}$ définit la face triangulaire.

Au final, dans une carte, les brins sont des éléments abstraits qui ne portent pas réellement d'information propre mais les éléments de la carte (sommets, arêtes, faces, objets) sont définis par des ensembles de brins. Par exemple la figure 3.a présente deux sommets isolés et la figure 3.b présente une arête construite à partir de ces deux sommets.

2 Questions

D'une manière générale, vous pouvez considérer que vous disposez de tous les types abstraits définis en cours : `Pile`, `File`, `Liste`, `Arbre`, `ABR`, `ARN` et `TableHachage` et les utiliser sans avoir à les redéfinir pour tout type d'objets dont vous avez besoin. Par exemple, considérez que le type `PileInt` est défini si vous en avez besoin.

* **Question 1: Compréhension du sujet :**

a) Quel ensemble d'applications (au sens mathématique du terme, c'est à dire combinaisons de fonctions β_i et leurs inverses) permet de définir l'orbite permettant d'obtenir une face ?

2. Ici, les fonctions f seront bien sûr les permutations β_i mais aussi leurs inverses β_1^{-1} et β_2^{-1}, ainsi que toute composition de ces permutations comme par exemple $\beta_1 \circ \beta_2$.

b) Une carte peut éventuellement représenter plusieurs objets ; la carte de l'exemple n'en représente qu'un seul. Quel est selon vous l'ensemble de fonctions permettant de définir l'orbite qui donnera l'ensemble des brins d'un objet.

c) Donnez l'orbite de la face issue du brin a, et celle issue du brin l.

Réponse 1:

a) $<\beta_1>$ définit une face. En effet, il suffit de suivre les brins en appliquant autant de fois qu'il le faut β_1 pour « faire le tour » de la face ;

> Il n'est pas nécessaire de rajouter β_1^{-1} puisque β_1 est une permutation ; de même, il n'est pas nécessaire, d'un point de vue applications mathématiques, de limiter le nombre de compositions de β_1 puisqu'à partir d'un certain moment, les brins que l'on voudrait rajouter sont déjà dans l'ensemble, ce qui n'augmente pas l'ensemble puisqu'un ensemble possède chaque élément en un et un seul exemplaire.

b) $<\beta_1, \beta_2>$: il faut utiliser les deux fonctions qui permettent de parcourir les brins d'une arête (β_1) puis de changer de face (β_2) pour avoir toutes les faces de l'objet.

c) $<\beta_1>(a) = \{a, b, c, d\}$ et $<\beta_1>(l) = \{h, i, j, k, l\}$

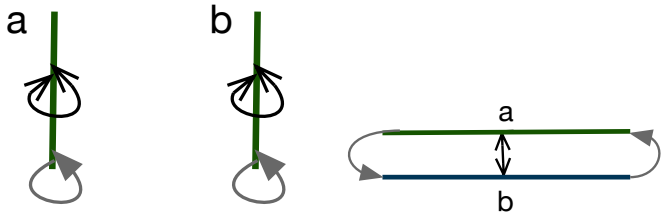
Question 2: Spécification fonctionnelle ***

On veut disposer d'un type `Carte` permettant de manipuler une carte combinatoire $2d$, c'est à dire créer la carte, lui rajouter des brins, obtenir l'image d'un brin par β_1 et β_2, obtenir à partir d'un brin l'ensemble des brins formant un sommet, une arête, une face ou un objet, et enfin coudre[3] deux brins selon β_1 ou β_2 tout en conservant l'intégrité de la carte.

Vous supposerez que vous disposez d'un type `ListeBrins` permettant de représenter un ensemble de brins, pour une orbite par exemple.

Réponse 2:

3. Coudre est le terme employé pour indiquer qu'on rajoute une relation β_i entre deux brins



Brins	a	b
β_1	a	b
β_2	a	b

Brins	a	b
β_1	b	a
β_2	b	a

a. 2 sommets. b. 1 arête.

Figure 3 – deux sommets et une arête

Il est mainte fois répété dans le sujet qu'un brin est abstrait, il n'est donc pas nécessaire que le brin soit géré par l'utilisateur ; en revanche, avoir un type brin est utile puisque ce sont les éléments constitutifs d'une carte. On a donc besoin d'un type, que l'utilisateur va devoir manipuler mais qu'il ne devrait pas gérer, au sens de « création et suppression d'instances », et qui est totalement dépendant de la carte. On applique donc la technique de la fabrique [a] qu'on a vu pour les itérateurs : la carte gère la création et destruction des instances du type abstrait Brin. D'autre part, le brin n'a pas à avoir de fonctionnalité particulière puisqu'il est abstrait. On pourra éventuellement lui octroyer un identifiant afin que l'utilisateur puisse différencier un brin d'un autre, surtout si il a besoin de faire des affichages.

a. Design Pattern Factory

Fonctionnalités de l'interface

- `créerCarte` : $\rightarrow$ **Carte** -- *crée une carte vide de brin*
- β_1 : **Carte** $\times$ **Brin** $\rightarrow$ **Brin** -- *renvoie l'image par β_1 du brin passé en paramètre ; ERREUR si le brin passé en paramètre n'existe pas dans la carte.*
- β_0 : **Carte** $\times$ **Brin** $\rightarrow$ **Brin** -- *renvoie l'image par β_1^{-1} du brin passé en paramètre ; ERREUR si le brin passé en paramètre n'existe pas dans la carte.*
- β_2 : **Carte** $\times$ **Brin** $\rightarrow$ **Brin** -- *renvoie l'image par β_2 du brin passé en paramètre ; ERREUR si le brin passé en paramètre n'existe pas dans la carte.*
- `est`β_1`Cousu` : **Carte** $\times$ **Brin** $\rightarrow$ **Boolean** -- *renvoie* **true** *si le brin est cousu par β_1,* **false** *sinon ; ERREUR si le brin passé en paramètre n'existe pas dans la carte.*
- `est`β_0`Cousu` : **Carte** $\times$ **Brin** $\rightarrow$ **Boolean** -- *renvoie* **true** *si le brin est cousu par β_0,* **false** *sinon ; ERREUR si le brin passé en paramètre n'existe pas dans la carte.*
- `est`β_2`Cousu` : **Carte** $\times$ **Brin** $\rightarrow$ **Boolean** -- *renvoie* **true** *si le brin est cousu par β_2,* **false** *sinon ; ERREUR si le brin passé en paramètre n'existe pas dans la carte.*
- `Sommet` : **Carte** $\times$ **Brin** $\rightarrow$ **ListeBrins** -- *renvoie la liste (ensemble) des brins composant le sommet dont est issu le brin passé en paramètre ; ERREUR si le brin n'existe pas dans la carte*
- `Arête` : **Carte** $\times$ **Brin** $\rightarrow$ **ListeBrins** -- *renvoie la liste (ensemble) des brins composant l'arête dont est issu le brin passé en paramètre ; ERREUR si le brin n'existe pas dans la carte*
- `Face` : **Carte** $\times$ **Brin** $\rightarrow$ **ListeBrins** -- *renvoie la liste (ensemble) des brins composant la face dont est issu le brin passé en paramètre ; ERREUR si le brin n'existe pas dans la carte*
- `Coudre`$_{\beta_1}$: **Carte** $\times$ **Brin** $\times$ **Brin** $\rightarrow$ **Carte** -- *coud les deux brins passés en paramètre par β_1 ; si le premier brin est déjà cousu par β_1 à un autre brin que lui-même, le second brin sera cousu à ce brin : insertion dans la permutation ; si le second brin est déjà cousu β_0 à un autre brin que lui-même, alors ERREUR ; si l'un des deux brins n'existent pas dans la carte, alors ERREUR ; si la couture β_1 est réalisée, celle par β_0 est automatiquement mise à jour, y compris les décousures qu'il y aurait du avoir.*
- `Coudre`$_{\beta_2}$: **Carte** $\times$ **Brin** $\times$ **Brin** $\rightarrow$ **Carte** -- *coud les deux brins passés en paramètre par β_2 ; si le premier brin est déjà cousu par β_2 à un autre brin que lui-même, alors ERREUR ; si l'un des deux brins n'existent pas dans la carte, alors ERREUR ; si la couture par β_2 est réalisée, alors le second brin est alors cousu par β_2 au premier : respect de l'involution.*
- `créerSommet` : **Carte** $\rightarrow$ **Carte** $\times$ **Brin** -- *crée un nouveau sommet isolé dans la carte en rajoutant le brin nécessaire et renvoie un brin de ce sommet. Note : le brin est cousu à lui-même par β_1 et β_2.*
- `créerArête` : **Carte** $\rightarrow$ **Carte** $\times$ **Brin** -- *crée une nouvelle arête isolée dans la carte en rajoutant le 2 brins nécessaires, en les cousant entre eux par β_2 et β_1 et renvoie un brin de cette arête*
- `coudreArête` : **Carte** $\times$ **Brin** $\times$ **Brin** $\rightarrow$ **Carte** -- *coud entre elles deux arêtes consituées chacune de deux brins au moins cousus par β_2. Le premier brin doit appartenir à la première arête, le second à la second arête. Le premier brin sera cousu par β_1 au second. Si le premier brin était déjà cousu par β_1 alors un « croisement » sera créé : précisemment le β_1 du premier brin deviendra le β_1 du β_2 du second brin. Si le second brin passé en paramètre*

était déjà cousu par β_0 à un brin, alors un croisement sera également créé, et c'est le β_{02} du second brin qui deviendra le β_1 du premier brin. Si l'un des deux brins n'existe pas dans la carte où qu'il n'appartiennent pas à une arête correctement formée, alors ERREUR.

- `ajouteBrin` : **Carte** → **Carte** × **Brin** *-- crée un nouveau brin et le rajoute dans la carte.*
- `idBrin` : **Brin** → **int** *-- renvoie l'identifiant du brin sous forme d'entier. Un identifiant est unique dans la carte à laquelle il appartient : deux brins d'une même carte ne peuvent pas avoir le même identifiant.*

Fonctionnalités internes

- `créerBrin` : Carte → Brin *-- crée un brin appartenant à la carte : rajoute un nouveau brin à la carte ; ce brin est cousu à lui-même par β_1 et β_2.*
- `estDansCarte` : **Carte** × **Brin** → **Boolean** *-- **true** si le brin appartient bien à la carte*

Question 3: Description logique : structure de données **

On veut désormais pouvoir disposer d'une structure de données permettant de représenter un tel type `Carte` défini à la question précédente ; sachant qu'on n'aura jamais besoin de plus N brins, N étant considérée comme une constante connue à l'avance, proposez une structure de données basée sur un ou des tableaux.

Réponse 3:

La structure de données était donnée dans le sujet : il suffit de reprendre les tableaux de l'exemple. Si vous aviez choisi un identifiant de brin sous forme de caractère ou chaîne de caractères, alors il fallait utiliser des tables de hachage au lieu de tableaux

$$\textbf{Carte} = \left\{ \begin{array}{lcl} \beta_1 & : & \texttt{array}[\text{n}] \text{ of } \textbf{int} \\ \beta_2 & : & \texttt{array}[\text{n}] \text{ of } \textbf{int} \\ \beta_0 & : & \texttt{array}[\text{n}] \text{ of } \textbf{int} \\ n & : & \textbf{int} \end{array} \right\}$$

Les β_i sont les tableaux codant les applications β_i, l'indice correspond à l'identifiant du brin concerné. n est le nombre de brins dans la carte, il est initialisé à 0 à la création de la carte.

Brin =int

Un brin est simplement un entier : on n'a pas besoin d'autres informations a priori sur le brin.

Question 4: Description logique : algorithmes *

a) Donnez l'algorithme de la fonction permettant d'obtenir l'ensemble des brins d'une arête de la carte à partir d'un brin.[4].

b) Donnez l'algorithme de la fonction permettant d'obtenir l'ensemble des brins d'un sommet de la carte à partir d'un brin.[4].

Réponse 4:

fonction `Arête`(*c* : **Carte** ; *b* : **Brin**)→**ListeBrins**
  if *not* `estDansCarte`(*c*,*b*) **then**
    ERREUR
  end if
  d : **Brin** = β_2(*b*)
  l : **ListeBrins** = creerListeBrins()
  Ajouter(b,l)
  if *d*≠*b* **then**
    Ajouter(d,l)
  end if
  return *l*
endf

4. Si il vous semble nécessaire de rajouter une ou plusieurs fonctions, donnez ici leur spécification fonctionnelle. Vous n'avez pas à définir un nouveau type.

```
fonction Sommet(c : Carte ; b : Brin)→ListeBrins
    if not estDansCarte(c,b) then
        ERREUR
    end if
    d : Brin = b
    l : ListeBrins = creerListeBrins()
    Ajouter(b,l)
    while (β2(β1(b)) ≠ d) and (β2(β1(b)) ≠ b) do
        b=β2(β1(b))
        Ajouter(b,l)
    end while
    return l
endf
```

** **Question 5: Dessin**

On supposera que toutes nos arêtes sont des segments de droite et qu'on dispose d'un type `Segment`. On veut pouvoir éventuellement dessiner notre objet et donc pouvoir associer à chaque arête un `Segment`.

a) Modifier la spécification fonctionnelle du type `Carte` afin d'être capable de récupérer le `Segment` d'une arête.

b) Si nécessaire, proposez une modification de la structure de données.

c) Proposez le(s) algorithme(s) de(s) éventuelle(s) fonction(s) supplémentaire(s) que vous avez rajoutées au type `Carte`.

Réponse 5:

Spécification fonctionelle

Non seulement on rajoute une fonctionnalité mais par là-même on change le sens de la carte puisque désormais à chaque arête est associée un **Segment**. Il faut donc faire en sorte que lorsqu'une arête est créée, un Segment lui soit associé. On va donc modifier la fonction `créerArête` mais aussi la fonction $\texttt{Coudre}_{\beta_2}$ qui doit désormais s'assurer qu'on dispose d'un **Segment** rattaché à au moins l'un des deux brins car coudre par β_2 crée de fait une arête. Il nous faut aussi la possibilité de ratacher un **Segment** à l'arête d'un brin.

- `dessin` : **Carte** × **Brin** → **Segment** -- *Retourne le segment associé à l'arête du brin passé en paramètre. ERREUR si le brin n'est pas dans la carte.*
- `rattacheDessin` : **Carte** × **Brin** × **Segment** → **Carte** -- *Ratache un* **Segment** *au brin. Si celui-ci est cousu par β_2 à un brin, ratache également ce segment à ce brin. Si le brin avait déjà un* **Segment**, *le remplace. ERREUR si le brin n'est pas dans la carte.*
- `créerArête` : **Carte** × **Segment** → **Carte** × **Brin** -- *crée une nouvelle arête isolée dans la carte en rajoutant le 2 brins nécessaires, en les cousant entre eux par β_2, en rattachant à chacun le* **Segment** *donné en paramètre et renvoie un brin de cette arête*

Structure de données

Il faut donc pouvoir associer un **Segment** à un brin. Il suffit de rajouter un tableau de **Segment**.

$$\textbf{Carte} = \left\{ \begin{array}{lcl} \beta_1 & : & \texttt{array[n] of } \textbf{int} \\ \beta_2 & : & \texttt{array[n] of } \textbf{int} \\ \beta_0 & : & \texttt{array[n] of } \textbf{int} \\ \textbf{dessins} & : & \texttt{array[n] of } \textbf{Segment} \\ n & : & \textbf{int} \end{array} \right\}$$

Algorithmes des fonctions supplémentaires ou modifiées

fonction `dessin`(*c :* **Carte** *; b :* **Brin**)→**Segment**
  if *not* `estDansCarte`(*c,b*) **then**
    ERREUR
  end if
  return *c.dessins[b]*
endf

fonction `rattacheDessin`(*c :* **Carte** *; b :* **Brin** *; s :* **Segment**)→**Carte**
  if *not* `estDansCarte`(*c,b*) **then**
    ERREUR
  end if
  c.dessins[b]=s
  c.dessins[β_2`(b)`]=s
  return *c*
endf

fonction `créerArête`(*c :* **Carte** *; s :* **Segment**)→**Carte**
  b1=`créerBrin`(*c*)
  b2=`créerBrin`(*c*)
  `Coudre`$_{\beta_1}$`(c,b1,b2)`
  `Coudre`$_{\beta_1}$`(c,b2,b1)`
  `Coudre`$_{\beta_2}$`(c,b2,b1)`
  `rattacheDessin(c,b1,s)`
  return *c*
endf

Question 6: Cartes dynamique *

Malheureusement, il s'avère impossible de connaître le nombre maximum de brins ; d'autre part on souhaite optimiser l'espace mémoire et donc utiliser un espace mémoire proportionnel au nombre de brins réellement dans la carte.

Compte-tenu de ces nouvelles contraintes, donnez une spécification fonctionnelle pour le type `Carte`.

Réponse 6:

Une modification de la structure de données n'entraîne pas d'ajout de fonctionnalités et donc pas de modification de la spécification fonctionnelle. Elle reste donc inchangée.

Question 7: Cartes dynamique : révision de la structure de données *

On abandonne donc l'idée d'utiliser des tableaux, ou même des tables de hachage. Proposez une nouvelle structure de données pour le(s) type(s) de votre spécification fonctionnelle.

Réponse 7:

$$\textbf{Carte} = \left\{ \begin{array}{lcl} n & : & \textbf{int} \\ brins & : & \textbf{ListeBrins} \end{array} \right\}$$

$$\textbf{Brin} = \left\{ \begin{array}{lcl} id & : & \textbf{int} \\ \beta_1 & : & \textbf{Brin} \\ \beta_2 & : & \textbf{Brin} \\ \beta_0 & : & \textbf{Brin} \\ dessin & : & \textbf{Segment} \end{array} \right\}$$

Question 8: Cartes dynamique : révision des algorithmes *

Compte-tenu de votre nouvelle structure de données, proposez un algorithme pour la fonction renvoyant l'ensemble des brins d'un sommet.

Réponse 8:

L'algorithme ne change pas!!!

Seules changent les fonctions `créerCarte`, β_1, β_2, β_0, `est`β_1`Cousu`, `est`β_2`Cousu`, `est`β_0`Cousu`, `Coudre`$_{\beta_1}$, `Coudre`$_{\beta_2}$, `ajouteBrin`, `créerBrin` et `estDansCarte`.