Maps et tables de hachage

Algorithmique II
EISTI – ING1

2011–2012

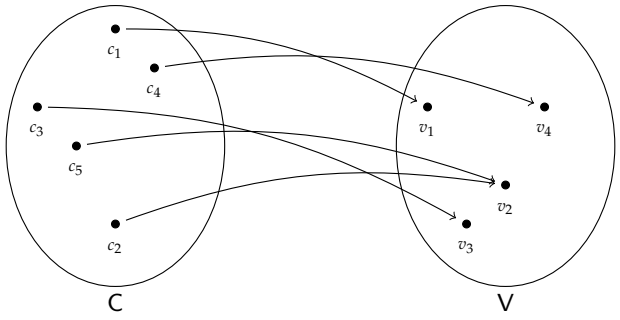
Rémi Vernay
remi.vernay@eisti.eu

I – Maps
1 – Définition

Map

- Association « clef, valeur »
- Une clef donne une seule valeur
- Une valeur peut être obtenue par plusieurs clefs
- Aucune valeur n'existe sans clef
- Aucune clef n'existe sans valeur

I – Maps
1 – Définition



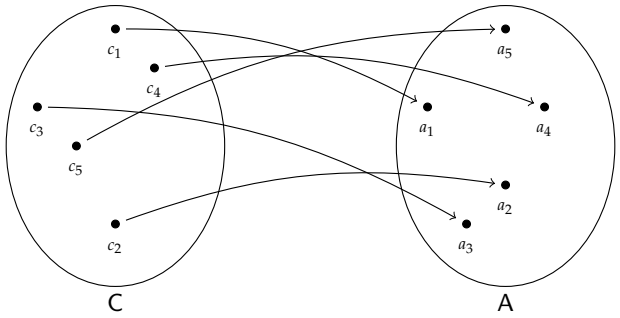
Soient C l'ensemble des clefs et V l'ensemble des valeurs

$$f : \begin{array}{ccc} C & \longrightarrow & V \\ c_i & \longmapsto & f(c_i) \end{array}$$

Il s'agit donc d'une application surjective.

I – Maps
1 – Définition

- Application surjective
  - problème
    - ⋆ $f(c_j) = f(c_k) \Rightarrow$ affectation de $f(c_j)$ écrase $f(c_k)$
  - solution
    - ⋆ utilisation d'alvéoles



Soient C l'ensemble des clefs et A l'ensemble des alvéoles

$$\begin{array}{rccl} g: & C & \longrightarrow & A \\ & c_i & \longmapsto & g(c_i) = a_i \end{array}$$

Il s'agit donc d'une application bijective.

- Utilisation d'alvéoles
  - les clefs sont liées aux alvéoles
  - la donnée est stockée dans l'alvéole
  - modifier le contenu d'une alvéole n'affecte aucune autre
  - concept théorique identique

- Propriétés
  - connaissance de la méthode de stockage des clefs inutile
  - connaissance de la méthode de stockage des valeurs inutile
  - accès direct à chaque valeur en fonction de sa clef

- Déclaration du conteneur Map
  - un type pour les clefs
  - un type pour les valeurs

`map: Map<type_clef> de type_valeur`

Exemple :

`map: Map<chaîne> de réel`

Méthodes disponibles

- procédure créerMap(map: Map<Élément> de Élément (S))
- procédure stocker(map: Map<Élément> de Élément (E/S) ; clef: Élément ; valeur: Élément)
- procédure supprimer(map: Map<Élément> de Élément (E/S) ; clef: Élément)
- fonction estVide(map: Map<Élément> de Élément): booléen
- fonction cardinalite(map: Map<Élément> de Élément): entier
- fonction clefExiste(map: Map<Élément> de Élément ; clef: Élément): booléen
- fonction valeurExiste(map: Map<Élément> de Élément ; valeur: Élément): booléen
- fonction valeurDe(map: Map<Élément> de Élément ; clef: Élément): Élément
- procédure listeClefs(map: Map<Élément> de Élément ; clefs: Liste de Élément (S))
- procédure copier(map1: Map<Élément> de Élément ; map2: Map<Élément> de Élément (S))

Préconditions

- créerMap(map) ⇔ aucune
- stocker(map,clef,valeur) ⇔ map initialisée
- supprimer(map,clef) ⇔ clef existante
- estVide(map) ⇔ map initialisée
- cardinalite(map) ⇔ map initialisée
- clefExiste(map,clef) ⇔ map initialisée
- valeurExiste(map,valeur) ⇔ map initialisée
- valeurDe(map,clef) ⇔ clef existante
- listeClefs(map,clefs) ⇔ map initialisée
- copier(map1,map2) ⇔ map1 initialisée

Postconditions

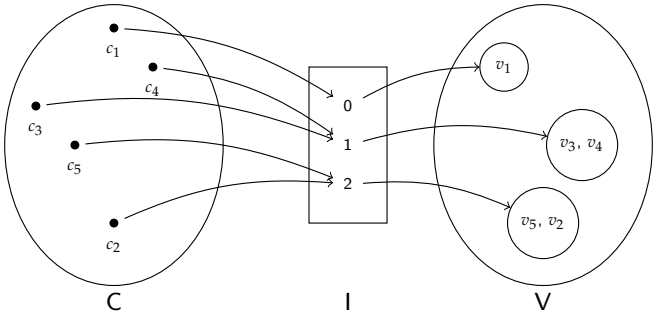
- créerMap(map) ⇒ map initialisée
- stocker(map,clef,valeur) ⇒ cardinalite(map)>0
- supprimer(map,clef) ⇒ map initialisée
- estVide(map) ⇒ aucune
- cardinalite(map) ⇒ aucune
- clefExiste(map,clef) ⇒ aucune
- valeurExiste(map,valeur) ⇒ aucune
- valeurDe(map,clef) ⇒ aucune
- listeClefs(map,clefs) ⇒ clefs initialisée avec des clefs existantes
- copier(map1,map2) ⇒ map2 initialisée

Saisie de noms d'étudiants avec leur moyenne.

```
programme Moyenne
    nb, i: entier
    valeur: réel
    clef: chaîne
    mapMoyennes: Map<chaîne> de réel
    créerMap(mapMoyennes)
    répéter
        écrire("Combien d'étudiants ?")
        lire(nb)
    jusqu'à nb⩾0
    pour i←1 à nb faire
        répéter
            écrire("Nom de l'étudiant ?")
            lire(clef)
        jusqu'à non(clefExiste(mapMoyennes,clef))
        répéter
            écrire("Sa moyenne ?")
            lire(valeur)
        jusqu'à 0⩽valeur et valeur⩽20
        stocker(mapMoyennes,clef,valeur)
    fin pour
    ...
```

- Tables de hachage
  - diminue la complexité des méthodes courantes
    - recherche
    - insertion
    - suppression
  - une des implémentations possibles des maps
    - adressage direct
    - méthodes de hachage
    - adressage ouvert

- Problématique
  - méthode par adressage direct trop coûteux
    - index contenant autant de clefs que de valeurs
    - type des clefs trop volumineux (chaînes, etc.)
  - tenter de
    - ne stocker que des entiers comme index
    - limiter la taille de l'index



Soient C l'ensemble des clefs, I celui des index et V celui des valeurs.

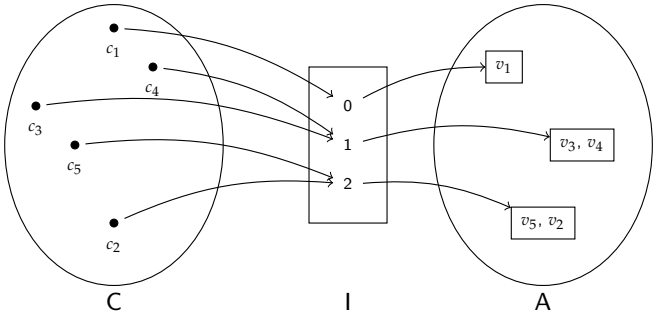
$$\begin{array}{rccc} v \circ h : & C & \longrightarrow & V \\ & c_i & \longmapsto & v \circ h(c_i) \end{array}$$

$\Rightarrow$ Problème de collisions.

- Fonctionnement
  - une fonction de hachage $h(c)$
    - renvoie une valeur de hachage $i \in \{0, 1, 2, \ldots, n-1\}$
    - les valeurs de hachage de $h(c)$ doivent être équiprobables
  - un stockage en fonction de $h(c) \Rightarrow$ collisions

- Plusieurs valeurs à stocker par valeur de hachage $h(c)$
  - utilisation d'alvéoles contenant des listes
  - utilisation de structures pour les éléments de la liste

```
type StructValeur = structure
    clef: Élément
    valeur: Élément
fin type
```

Utilisation de listes.

- On stocke la structure comprenant la clef et la valeur
- Le nombre de valeurs stockées par liste reste faible
- La recherche est rapide
- Les valeurs de hachage doivent être bien réparties

Exemple de fonction de hachage : clef du numéro de sécurité sociale

$$h(c) = 97 - (c - \left\lfloor \frac{c}{97} \right\rfloor \times 97)$$
$$h(c) \in \{0, 1, 2, \ldots, 96\}$$

$$1\ 95\ 74\ 10\ 023\ 068 \Rightarrow c = 1957410023068$$
$$h(1957410023068) = 10$$

- Fonctions de hachage
  - différentes méthodes
    - créées sur mesure
    - méthode dite de la division
    - méthode dite de la multiplication
    - hachage universel
- On doit avoir la même probabilité d'obtenir toutes les valeurs de hachage
  - il faut tendre au maximum vers cet absolu

préconditions : `numero` est un numéro de sécurité sociale valide
postconditions : $0 \leqslant$ h(`numero`) < 97
fonction h(numero: entier): entier
    retourner 97-(numero mod 97)
fin fonction

type Assure = **structure**
    numero: entier
    nom: chaîne
    prenom: chaîne
fin type


```
programme Exemple
    alveoles: Tableau de Liste de Assure
    i, numero: entier
    nom, prenom: chaîne
    créerTableau(alveoles,97)
    pour i←0 à 96 faire
        créerListe(alveoles[i])
    fin pour
    ...
    lire(nom)
    lire(prenom)
    lire(numero)
    enregistrer(alveoles,nom,prenom,numero)
    ...
fin programme
```



```
procedure enregistrer(alveoles: Tableau de Liste de Assure (E/S) ; nom, prenom: chaîne ;
numero: entier)
    assure: Assure
    nouveau: booléen
    l: Liste de Assure
    nouveau←VRAI
    l←alveoles[h(numero)]
    tant que non(estVide(l) et nouveau faire
        si tête(l).numero=numero alors
            nouveau←FAUX
            tête(l).nom←nom
            tête(l).prenom←prenom
        fin si
        l←reste(l)
    fin tant que
    si nouveau alors
        assure.numero←numero
        assure.nom←nom
        assure.prenom←prenom
        ajouter(alveoles[h(numero)],assure)
    fin si
fin programme
```