IFT339
Structures de données

Thème 13 : Adressage dispersé

Aïda Ouangraoua
Département d'informatique

UNIVERSITÉ DE
SHERBROOKE

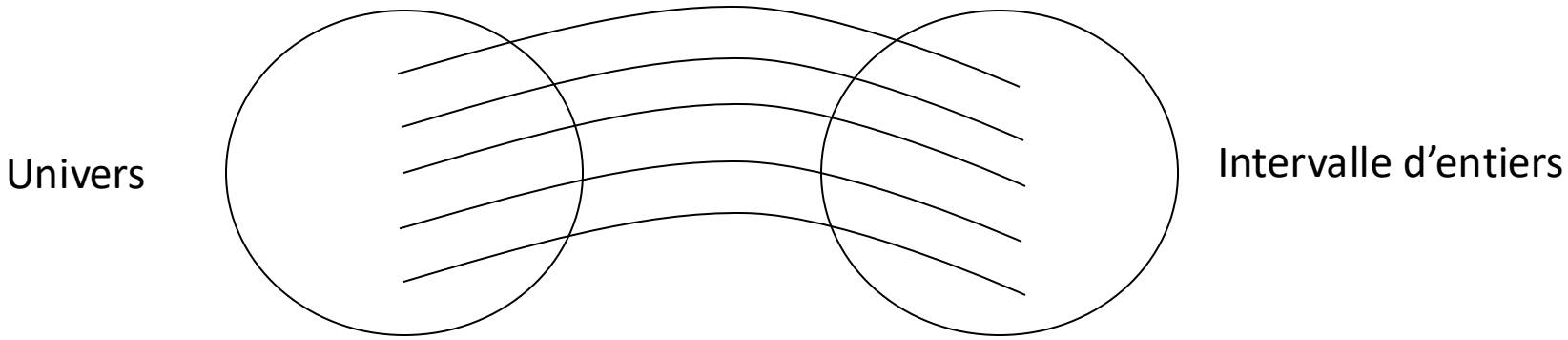
Adressage dispersé

- Arbres AVL: opérations en O(log(n)).
- Pour n très grand, log(n) est encore trop grand.
- Peut-on avoir un conteneur non linéaire et non-contigu avec des opérations en O(1) en moyenne ?

Adressage dispersé

- Idée:
  - Combinaison de tableaux et de listes chainées
  - Table de hachage: ensemble de couples (clé, valeur)
  - Pas d'ordre sur les éléments
  - Accès à un élément en temps constant en utilisant sa clé

- Piste de solution:
  - Chaque élément (cle, val) de l'ensemble E est stocké dans une case d'index hash(cle) telle que hash est définie de l'Univers (Ensemble des clés possibles) vers un intervalle d'entiers, et f est bijective.
    Ex: E = {(1, val1),(2,val2), (10^6,val3)}, Univers = {i | 0 <= i <= 10^6}, hash(x) = x



Adressage dispersé

- Idée:
  - Combinaison de tableaux et de listes chainées
  - Table de hachage: ensemble de couples (clé, valeur)
  - Pas d'ordre sur les éléments
  - Accès à un élément en temps constant en utilisant sa clé

- Piste de solution:
  - Chaque élément (cle, val) de l'ensemble E est stocké dans une case d'index hash(cle) telle que hash est définie de l'Univers (Ensemble des clés possibles) vers un intervalle d'entiers, et f est bijective.
    Ex: E = {(1, val1),(2,val2), (10^6,val3)}, Univers = {i | 0 <= i <= 10^6}, hash(x) = x

  - Problème:
    - Dans cet exemple, l'Univers, l'ensemble des clés possibles, est très grand par rapport à la taille de E.
    - Le stockage nécessite un tableau de la taille de l'univers : $1+10^6$

Adressage dispersé

- Solution:
  - Restreindre les index possibles en regroupant les clés

- Fonction de hachage h:
  - définie de l'ensemble des clés vers un intervalle [0, … k-1] tel que k << | Univers |
  - associe à chaque clé x de l'Univers un index adresse h(x) telle que 0 <= h(x) < |

Exemple : E = {(1, val1),(2,val2), (10^6,val3)}, Univers = {i | 0 <= i <= 10^6}, h(x) = x % 8;

(10^6,val3)	**(1, val1)**	**(2,val2)**					

alors k = 8, et h(1) = 1; h(2) = 2, h(10^6)= 0

Adressage dispersé

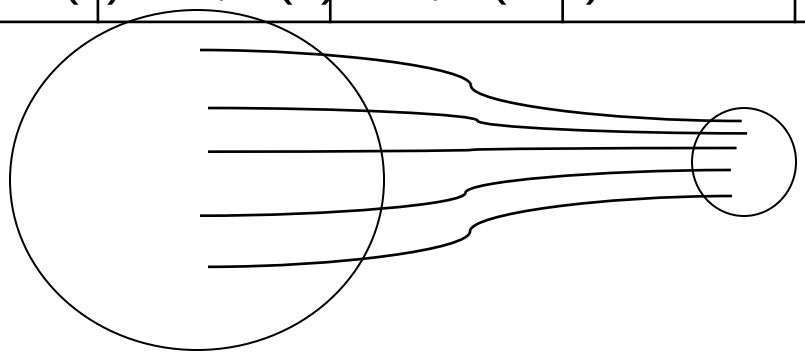
- Solution:
  - Restreindre les index possibles en regroupant les clés

- Fonction de hachage h:
  - définie de l'ensemble des clés vers un intervalle [0, … k-1] tel que k << | Univers |
  - associe à chaque clé x de l'Univers un index adresse h(x) telle que 0 <= h(x) < k

Exemple : E = {(1, val1),(2,val2), (10^6,val3)}, Univers = {i | 0 <= i <= 10^6}, h(x) = x % 8;

(10^6,val3)	**(1, val1)**	**(2,val2)**					

alors k = 8, et h(1) = 1; h(2) = 2, h(10^6)= 0



Adressage dispersé

- Problème : possibilité d'avoir des collisions
  - Plusieurs éléments peuvent avoir la même adresse

  Exemple: E = {(0, val1), (1, val2), (2, val3), (25, val4), (64, val5), (10^6, val6)}, h(x) = x%8

(0, val1) (10^6,val6)(64, val5)	**(1, val2) (25, val4)**	**(2,val3)**					

- Il faut une stratégie de gestion des collisions.

Propriétés d'une bonne fonction de hachage

- Distribuer les adresses de la façon la plus uniforme possible (pour minimiser les collisions)
- Compresser l'ensemble des index [0… k-1]. (k << taille de l'Univers)
- Facile à calculer en O(1)

Propriétés d'une bonne fonction de hachage

- Distribuer les adresses de la façon la plus uniforme possible (pour minimiser les collisions)
- Compresser l'ensemble des index [0… k-1]. (k << taille de l'Univers)
- Facile à calculer en O(1)
- Composée de 2 sous-fonctions
  - Code de hachage h1: associe à chaque clé un entier

    Exemple: h1: chaîne de caractères → entier
  - Fonction de compression h2: associe à chaque entier un index dans l'ensemble [0.. k-1] , k étant la taille de la table de hachage

    h(cle) = h2(h1(cle))

Choix de la fonction de hachage

- Exemple1: Stocker les dossiers d'étudiants de l'UdeS identifiés par leurs matricules:

  - Taille de l'Univers : 10^8 clés (matricules) possibles

  - Taille de l'ensemble : environ 40 000 étudiants

  - Fonction de hachage pour k = 10 000 ➔ clés sur [0 … 9999]
    - ne garder que les 4 chiffres de poids les plus faibles

    h1(x) = x ; //x est un matricule

    h2(x) = x % 10 000

    h(56312473) = 2473

Choix de la fonction de hachage

- Exemple2: Stocker les dossiers d'étudiants de l'UdeS identifiés par leurs noms:

  - Taille de l'Univers : 26^{10} clés (noms) possibles
    (en se limitant à 10 caractères pour les noms)

  - Taille de l'ensemble : environ 40 000 étudiants

  - Fonction de hachage pour k = 10 000 ➔ clés sur [0 … 9999]
    - ne garder que les 4 chiffres de poids les plus faibles

    h1(x) = int.from_bytes(str.encode(x),'little') // conversion en entiers
    h2(x) = x % 10 000
    h1('Berger') = 125779852813634
    h('Berger') = 3634

Pré-traitement sur les positions des chiffres de h1(x)

- Tous les chiffres ne contiennent pas la même quantité d'information

  - Exemple: Le premier chiffre du matricule étudiant est souvent 1,2. Il ne porte donc pas beaucoup d'information.

  - Idée:
    - Calculer la quantité d'information $q(y_i)$ pour chaque position y_i
    - Choisir les 4 positions ayant des quantités d'information maximum

  Si h1(x) produit un nombre de longueur m : $y_{m-1}y_{m-2}..y_1y_0$,

$$q(y_i) = -[prob(y_i=0) * log(prob(y_i=0)) + prob(y_i=1) * log(prob(y_i=1)) + \ldots + prob(y_i=9 * log(prob(y_i=9))]$$

Entropie

Pré-traitement sur les positions des chiffres de h1(x)

- Exemple: si h1(x) retourne des nombres binaires sur 4 bits: $b_3b_2b_1b_0$

$prob(b_i=0) = p_i$

$q(b_i) = - p_i * \log(p_i) – (1 - p_i) * \log(1 - p_i)$

$q(b_i) = 0$ si $p_i = 0$ ou $p_i = 1$; $q(b_i) = 1$ (maximum) si $p_i = 0,5$

Autre fonction pour le calcul de la quantité d'information:

$q2(b_i) = p_i * (1 - p_i)$

$q2(b_i) = 0$ si $p_i = 0$ ou $p_i = 1$; $q(b_i) = 0,25$ (maximum) si $p_i = 0,5$

- On a 2^4 nombres possibles. Dans l'exemple ci-dessous 2^2 nombres suffisent pour avoir une seule collision.

1 0 1 0
1 1 1 0
1 0 1 1
1 0 0 0
1 1 0 0

Gestion des collisions

- On ne peut pas garantir que deux clés n'auront jamais le même index
- Que faire en cas de collision ?

Gestion des collisions

- Considérer le facteur de charge : $\alpha = n / N$, n étant la taille de E (ensemble des éléments stockés), et N la taille de la table de hachage

  Exemple: $E = \{(1, val1),(2,val2), (10^6,val3)\}$; $h(x) = x \% 8$; $\alpha = 3 / 8$

- Si $\alpha > 1$, on aura au moins une collision. On doit donc toujours choisir $\alpha < 1$.

  Le tableau de hachage doit être plus grand que l'ensemble des éléments stockés.

Gestion des collisions

- Sondage linéaire: prendre le prochain index disponible
- Sondage quadratique: prendre l'index immédiatement après ($+1^2$) si disponible, sinon 4 plus loin ($+2^2$), sinon 9 plus loin ($+3^2$), etc.
- Double hachage: si h(x) génère une collision, alors appliquer une seconde fonction de hachage g(x)
- Adressage en chaîne : placer dans un conteneur tous les éléments ayant le même index (exemple: liste ou set)
- Exemple: E = {(0, val1), (1, val2), (2, val3), (25, val4), (64, val5), (10^6, val6)}, h(x) = x%8

Exemple: Écrire les fonctions insert et erase

```
#ifndef _umap_h
#define _umap_h


template <typename Tclef, typename
Tvaleur>
class umap{
 private:
  vector<pair<Tclef,Tvaleur>> tab;
  size_t hash(Tclef)
 public:
 void insert(pair<Tclef,Tvaleur>)
 void erase(pair<Tclef,Tvaleur>)
}
#endif
```

- Sondage linéaire
- Sondage quadratique