INF3105 – Tables de hachage (*Hash Tables*)

Jaël Champagne Gareau

Université du Québec à Montréal (UQAM)

Été 2024

http://cria2.uqam.ca/INF3105/

UQÀM

Nous avons vu ...

Structure	Accès aléatoire	Insertion	Recherche
Tableau (non trié)	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Tableau (trié)	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$
Liste chainée	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$
Arbre recherche équilibré	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

- **accès aléatoire** (ou **accès direct**) : accéder directement au i-ème élément d'une structure sans avoir à visiter d'autres positions précédentes/suivantes.

Comment faire mieux ?

Structure	Accès aléatoire	Insertion	Recherche
Tableau (non trié)	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Tableau (trié)	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$
Liste chainée	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$
Arbre recherche équilibré	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
Table de hachage	–	$\mathcal{O}(1)$	$\mathcal{O}(1)$

prog0.cpp

```cpp
#include <iostream>
using namespace std;
string description(char note) {
  // Retourne "Excellent" si note=='A', "Tres bien" si note=='B', etc.




  return "?";
}
int main() {
    while(cin && !cin.eof()) {
        char note; // ex.: A, B, C, D, E, X
        cout << "Entrez une note : " << endl;
        cin >> note;
        cout << "Description : " << description(note) << endl;
    }
  return 0;
}
```

prog1.cpp

```cpp
#include <iostream>
using namespace std;
string description(char note) {
  if(note == 'A') return "Excellent";
  if(note == 'B') return "Tres bien";
  if(note == 'C') return "Bien";
  if(note == 'D') return "Passable";
  if(note == 'E') return "Echec";
  if(note == 'X') return "Abandon";
  return "Erreur";
}
int main() {
    while(cin && !cin.eof()) {
        char note; // ex.: A, B, C, D, E, X
        cout << "Entrez une note : " << endl;
        cin >> note;
        cout << "Description : " << description(note) << endl;
    }
  return 0;
}
```

prog2.cpp

```cpp
#include <iostream>
#include <map>
#include <string>
using namespace std;
int main() {
    map<char, string> table; // Arbre rouge–noire de la STL
    table['A'] = "Excellent";
    table['B'] = "Tres bien";
    table['C'] = "Bien";
    table['D'] = "Passable";
    table['E'] = "Echec";
    table['X'] = "Abandon";
    while(cin && !cin.eof()) {
        char note;
        cout << "Entrez une note : " << endl;
        cin >> note;
        cout << "Description : " << table[note] << endl;
    }
}
```

prog3.cpp

```cpp
#include <iostream>
#include <map>
#include <string>
using namespace std;
int main() {
    string table[256]; // Mémoire: sizeof(string) * 256 peu importe n
    table['A'] = "Excellent"; // 'A' = 65
    table['B'] = "Tres bien"; // 'B' = 66
    table['C'] = "Bien";      // 'C' = 67
    table['D'] = "Passable";  // 'D' = 68
    table['E'] = "Echec";     // 'E' = 69
    table['X'] = "Abandon";   // 'X' = 88
    while(cin && !cin.eof()) {
        char note;
        cout << "Entrez une note : " << endl;
        cin >> note;
        cout << "Description : " << table[note] << endl;
    }
}
```

prog4.cpp

```cpp
#include <iostream>
#include <map>
#include <string>
using namespace std;
int main() {
    string table[26]; // Mémoire: sizeof(string) * 26 peu importe n
    table['A' - 'A'] = "Excellent";
    table['B' - 'A'] = "Tres bien";
    table['C' - 'A'] = "Bien";
    table['D' - 'A'] = "Passable";
    table['E' - 'A'] = "Echec";
    table['X' - 'A'] = "Abandon";
    while(cin && !cin.eof()) {
        char note;
        cout << "Entrez une note : " << endl;
        cin >> note;
        cout << "Description : " << table[note - 'A'] << endl;
    }
}
```

prog5.cpp

```cpp
class CodePerm {
    string code;
  public:
    CodePerm(const string& sigle) : code(sigle) {
        // Validation du code permanent
    }
    operator int() const {
        int i = 0;
        i += lesigle[0] – 'A'; i *= 26;
        i += lesigle[1] – 'A'; i *= 26;
        i += lesigle[2] – 'A'; i *= 26;
        i += lesigle[3] – 'A'; i *= 10;
        i += lesigle[4] – '0'; i *= 10;
        i += lesigle[5] – '0'; i *= 10;
        i += lesigle[6] – '0'; i *= 10;
        i += lesigle[7] – '0'; i *= 10;
        i += lesigle[8] – '0'; i *= 10;
        i += lesigle[7] – '0'; i *= 10;
        i += lesigle[8] – '0'; i *= 10;
        i += lesigle[9] – '0';
        return i;
    }
};
```

```cpp
int main() {
    // Taille : 26^4 * 10^8 = 45 697 600 000 000 = 45.697 Ti cases
    string* table = new string[26*26*26*26*10*10*10*10*10*10*10*10];
    // Besoin en mémoire : 45.697 Ti * sizeof(string)

    table[CodePerm("CODE00000001")] = "Etudiant 1";
    table[CodePerm("CODE00000002")] = "Etudiant 2";
    table[CodePerm("CODE00000003")] = "Etudiant 3";
    table[CodePerm("CODE00000004")] = "Etudiant 4";
    table[CodePerm("CODE00000005")] = "Etudiant 5";

    while(cin && !cin.eof()) {
        string codeperm;
        cout << "Entrez un code permanent : " << endl;
        cin >> codeperm;
        cout << "Description : " << table[CodePerm(codeperm)] << endl;
    }
    delete[] table;
}
```

Domaine des valeurs possibles très grand

- Un objet a un domaine (D) de valeurs possibles.
- Un objet peut toujours être traduit en un nombre entier.
- La taille du domaine des valeurs possibles (nombre d'objets différents) peut être très grand.
- Nombre de chaînes de longueurs 8 avec [A-Z] :
  $26^8 = 208\,827\,064\,576 \approx 2.09 \cdot 10^{11}$.
- Nombre de chaînes de longueurs 8 avec [a-zA-Z0-9] :
  $(2 \times 26 + 10)^8 \approx 2.18 \cdot 10^{14}$.
- Nombre de chaînes de caractères de longueurs 10 : $(256)^{10} \approx 1,2 \cdot 10^{24}$.

Observation

- Généralement, le nombre d'objets (n) qu'on veut réellement stocker est de plusieurs ordres de grandeurs inférieur à la taille du domaine (D).
- Exemple avec les codes permanents :
  - Nombre d'étudiants dans une université : 40 000
  - Nombre d'étudiants dans un cours : 60

Réduction de la taille d'adressage

Fonctions :

- division : /
- modulo : %

Exemple 1

```
1  unsigned int f(const string& s) {
2      unsigned int h = 0;
3      for(int i = 0; i < s.size(); i++) {
4          h *= 256;  h += s[i];
5      }
6      return h;
7  }
8  int main() {
9      bool table[10] = {false, ...};
10     while(cin) {
11         string s;
12         cin >> s >> ws;
13         cout << f(s) << " ";
14         int ar = f(s) % 10; // réduction adresse
15         cout << table[ar] ? "existe" : "nouveau";
16         table[ar] = true;
17         cout << endl;
18     }
19 }
```

```
1  a // 97
2  b // 98
3  c // 99
4  d // 100
5  e // 101
6  f // 102
7  g // 103
```

Exemple 2

```
1  unsigned int f(const string& s) {
2      unsigned int h = 0;
3      for(int i = 0; i < s.size(); i++) {
4          h *= 256;  h += s[i];
5      }
6      return h;
7  }
8  int main() {
9      bool table[10] = {false, ...};
10     while(cin) {
11         string s;
12         cin >> s >> ws;
13         cout << f(s) << " ";
14         int ar = f(s) % 10; // réduction adresse
15         cout << table[ar] ? "existe" : "nouveau";
16         table[ar] = true;
17         cout << endl;
18     }
19 }
```

```
1  alpha //1819306081 ==>1
2  bravo //1918989935 ==>5
3  charlie //1919707493 ==>3
4  delta //1701606497 ==>7
5  echo //1701013615 ==>5
6  foxtrot //1953656692 ==>2
7  golf //1735355494 ==>4
8  hotel //1869899116 ==>6
9  india //1852074337 ==>7
10 juliett //1768256628 ==>8
11 kilo //1802071151 ==>1
12 lima //1818848609 ==>9
13 mike //1835625317 ==>7
14 november //1835165042 ==>2
15 oscar //1935892850 ==>0
16 papa //1885433953 ==>3
17 quebec //1700947299 ==>9
18 romeo //1869440367 ==>7
19 sierra //1701999201 ==>1
20 tango //1634625391 ==>1
21 uniform //1718579821 ==>1
22 victor //1668575090 ==>0
23 whiskey //1936418169 ==>9
24 x−ray //762470777 ==>7
25 yankee //1852532069 ==>9
26 zulu //2054515829 ==>9
```

Exemple 2

```
1  unsigned int f(const string& s) {
2      unsigned int h = 0;
3      for(int i = 0; i < s.size(); i++) {
4          h *= 256;  h += s[i];
5      }
6      return h;
7  }
8  int main() {
9      bool table[10] = {false, ...};
10     while(cin) {
11         string s;
12         cin >> s >> ws;
13         cout << f(s) << " ";
14         int ar = f(s) % 10; // réduction adresse
15         cout << table[ar] ? "existe" : "nouveau";
16         table[ar] = true;
17         cout << endl;
18     }
19 }
```

```
1  www.alpha.com //778268525 ==>5
2  www.bravo.com //778268525 ==>5
3  www.charlie.com //778268525 ==>5
4  www.delta.com //778268525 ==>5
5  www.echo.com  //778268525 ==>5
6  www.foxtrot.com //778268525 ==>5
7  www.golf.com  //778268525 ==>5
8  www.hotel.com //778268525 ==>5
9  www.india.com //778268525 ==>5
10 www.juliett.com //778268525 ==>5
11 www.kilo.com  //778268525 ==>5
12 www.lima.com  //778268525 ==>5
13 www.mike.com  //778268525 ==>5
14 www.november.com  //778268525 ==>5
15 www.oscar.com //778268525 ==>5
16 www.papa.com  //778268525 ==>5
17 www.quebec.com  //778268525 ==>5
18 www.romeo.com //778268525 ==>5
19 www.sierra.com  //778268525 ==>5
20 www.tango.com //778268525 ==>5
21 www.uniform.com //778268525 ==>5
22 www.victor.com  //778268525 ==>5
23 www.whiskey.com //778268525 ==>5
24 www.x-ray.com //778268525 ==>5
25 www.yankee.com  //778268525 ==>5
26 www.zulu.com  //778268525 ==>5
```

Objectif

- Éviter que des objets ayant des valeurs proches obtiennent la même adresse réduite.
- Idée : construire une fonction le plus chaotique possible.
- Fonction chaotique : une faible variation de l'entrée entraîne une grande variation en sortie.

Exemple de fonction de hachage

```
1  unsigned int hash(const string& s) {
2      unsigned int h = 0;
3      for(int i = 0; i < s.size(); i++) {
4          h *= 31;  h += s[i];
5      }
6      return h;
7  }
8  int main() {
9      bool table[10] = {false, ...};
10     while(cin) {
11         string s;
12         cin >> s >> ws;
13         cout << hash(s) << " ";
14         int ar = hash(s) % 10; // réduction adresse
15         cout << table[ar] ? "existe" : "nouveau";
16         table[ar] = true;
17         cout << endl;
18     }
19 }
```

```
1  www.alpha.com //2706318682==>2
2  www.bravo.com //1546345990==>0
3  www.charlie.com //1374347758==>8
4  www.delta.com //2692525300==>0
5  www.echo.com //692053361==>1
6  www.foxtrot.com //3677518702==>2
7  www.golf.com //1787073812==>2
8  www.hotel.com //2713091888==>8
9  www.india.com //2453665907==>7
10 www.juliett.com //841337625==>5
11 www.kilo.com //124416821==>1
12 www.lima.com //2383303109==>9
13 www.mike.com //3285493312==>2
14 www.november.com //665463428==>8
15 www.oscar.com //4290280812==>2
16 www.papa.com //525507732==>2
17 www.quebec.com //3111054519==>9
18 www.romeo.com //3972339830==>0
19 www.sierra.com //3255468804==>4
20 www.tango.com //3651535813==>3
21 www.uniform.com //542194032==>2
22 www.victor.com //2755519183==>3
23 www.whiskey.com //2659943040==>0
24 www.x-ray.com //2954925905==>5
25 www.yankee.com //55855025==>5
26 www.zulu.com //2348746866==>6
```

Collisions

- Se produit quand deux objets ayant des valeurs différentes produisent la même adresse réduite.
- Fonction de hachage parfaite (sans collisions) rarement possible en pratique.
- Besoin d'un mécanisme de gestion de collision.

Stratégies

- Ouvert.
  - Linéaire.
  - Quadratique.
- Structure externe.
  - Liste chaînée.
  - Arbre binaire de recherche.

Implémentation

L'implémentation sera vue au prochain labo.