INF3105 – Dictionnaire associatif basé sur un ABR

Jaël Champagne Gareau

Université du Québec à Montréal (UQAM)

Été 2024

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

UQÀM

Dictionnaire (*map*)

- Structure associative pour stocker un ensemble de paires (**clé**, **valeur**).
- Accès optimisé à partir de la clé.

Exemple :

Clé (Sigle)	Valeur (Titre)
INF1120	Programmation I
INF1132	Mathématiques pour l'informatique
INF2120	Programmation II
INF2171	Organisation des ordinateurs et assembleur
INF3105	Structures de données et algorithmes
INF4230	Intelligence artificielle

Utilisation (1)

```
1  ArbreMap<string, string> dictionnaire; // [sigle] --> titre
2  dictionnaire["INF3105"] = "Structures de donnees et algorithmes";
3  dictionnaire["INF1120"] = "Programmation I";
4  dictionnaire["INF4230"] = "Intelligence artificielle";
5
6  cout << "INF1120" << " : " << dictionnaire["INF1120"] << endl;
7  cout << "INF3105" << " : " << dictionnaire["INF3105"] << endl;
8  cout << "INF4230" << " : " << dictionnaire["INF4230"] << endl;
```

Utilisation (2)

```
1  ArbreMap<string, string> dictionnaire;
2  dictionnaire["INF3105"] = "Structures de donnees et algorithmes";
3  dictionnaire["INF1120"] = "Programmation I";
4  dictionnaire["INF4230"] = "Intelligence artificielle";
5
6  Liste<string> cours;
7  cours.inserer("INF1120");
8  cours.inserer("INF3105");
9  cours.inserer("INF4230");
10
11 for(Liste<string>::Iterateur iter = cours.debut(); iter; ++iter)
12    cout << cours[iter] << " : " << dictionnaire[cours[iter]] << endl;
```

Utilisation (3)

```
1  ArbreMap<string, string> dictionnaire;
2  dictionnaire["INF3105"] = "Structures de donnees et algorithmes";
3  dictionnaire["INF1120"] = "Programmation I";
4  dictionnaire["INF4230"] = "Intelligence artificielle";
5
6  for(ArbreMap<string, string>::Iterateur iter = dictionnaire.debut(); iter; ++iter)
7      cout << iter.cle() << " : " iter.valeur() << endl;
```

Plusieurs choix d'implémentation

1. Implémenter les classes ArbreAVL et ArbreMap indépendamment.
   - Duplication de code.
   - Simple ?
2. Implémenter ArbreAVL à l'aide d'un ArbreMap.
   - C'est le choix qui a été fait en Java : un `TreeSet` (ArbreAVL) est un `TreeMap` où seule la clé est utilisée (la valeur est un `nullptr`).
3. **Implémenter ArbreMap à l'aide d'un ArbreAVL**.
   - Simple.
   - C'est le choix fait en INF3105.

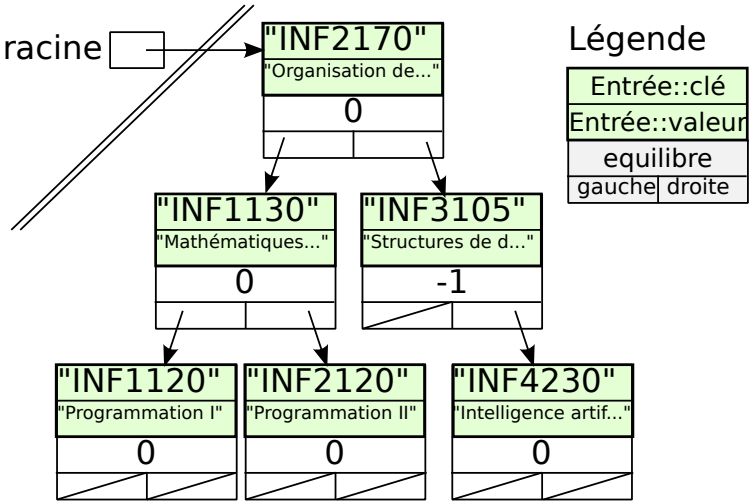
Classe Entrée

```
1  template <class K, class V>
2  struct Entree {
3      Entree(const K& c) : cle(c), valeur() {}
4      K cle;
5      V valeur;
6
7      // La classe Entree doit implémenter l'opérateur <
8      bool operator < (const Entree& e) const {
9          return cle < e.cle;
10     }
11 };
```

Classe ArbreMap

```
1  template <class K, class V>
2  class ArbreMap {
3    public:
4      // ...
5      bool contient(const K&) const;
6      void enlever(const K&);
7      void vider();
8      const V& operator[] (const K&) const;
9      V& operator[] (const K&);
10   private:
11     struct Entree {
12       Entree(const K& c) : cle(c), valeur() {}
13       K cle;
14       V valeur;
15       bool operator < (const Entree& e) const { return cle < e.cle; }
16     };
17     ArbreAVL<Entree> entrees;
18 };
```

Représentation



Test d'existence d'une clé

```
1  template <class K, class V>
2  bool ArbreMap<K,V>::contient(const K& cle) const {
3     // Construire un objet Entrée avec la clé recherchée
4     // La valeur n'est pas importante, car l'operateur < ne la considère pas
5     const Entree entree(cle);
6     return entrees.contient(entree);
7  }
```

```
1  template <class K, class V>
2  bool ArbreMap<K,V>::contient(const K& cle) const {
3     // Le compilateur appelle le constructeur de Entrée avec clé automatiquement
4     return entrees.contient(cle); // ou entrees.contient(Entree(cle));
5  }
```

Opérateurs []

```
1  template <class K, class V>
2  const V& ArbreMap<K,V>::operator[] (const K& cle) const {
3      typename ArbreAVL<Entree>::Iterateur iter = entrees.rechercher(cle);
4      // L'élément doit exister!
5      return entrees[iter].valeur;
6  }
7
8  template <class K, class V>
9  V& ArbreMap<K,V>::operator[] (const K& cle) {
10     typename ArbreAVL<Entree>::Iterateur iter = entrees.rechercher(cle);
11     if(!iter) {
12         // Choix d'implémentation : si l'élément n'existe pas, on le crée
13         entrees.inserer(Entree(cle));
14         iter = entrees.rechercher(cle);
15         // Choix alternatif : création explicite (Ex: TreeMap.put() en Java).
16     }
17     return entrees[iter].valeur;
18 }
```

Itérateur d'ArbreMap

- Permet d'itérer sur les entrées, c'est-à-dire les paires (clé, valeur).
- Un itérateur d'ArbreMap<K,V> est tout simplement un itérateur d'ArbreAVL<Entree>.
- Encapsulation d'un itérateur d'ArbreAVL dans un itérateur d'ArbreMap (la classe Entree demeure privée à ArbreMap).

```
1  template <class K, class V>
2  class ArbreMap {
3     class Entree { /*...*/ };
4     ArbreAVL<Entree> entrees; // représentation
5   public:
6     class Iterateur {
7       public:
8         Iterateur(ArbreMap& a) : iter(a.entrees.debut()) {}
9         Iterateur(typename ArbreAVL<Entree>::Iterateur i) : iter(i) {}
10        operator bool() const { return iter.operator bool(); };
11        Iterateur& operator++() { ++iter; return *this; }
12        const K& cle() const { return (*iter).cle; }
13        V& valeur() { return (V&) (*iter).valeur; }
14      private:
15        typename ArbreAVL<Entree>::Iterateur iter;
16    };
17    Iterateur debut() { return Iterateur(*this); }
18    Iterateur fin() { return Iterateur(entrees.fin()); }
19    Iterateur rechercher(const K& cle) { return Iterateur(entrees.rechercher(cle)); }
20    Iterateur rechercherEgalOuSuivant(const K& cle) { return Iterateur(entrees.rechercherEgalOuSuivant(cle)); }
21    Iterateur rechercherEgalOuPrecedent(const K& cle) { return Iterateur(entrees.rechercherEgalOuPrecedent(cle)); }
22 };
```

Besoin d'itérateur constant ?

```
1 void afficher(const ArbreMap<string, string>& dictionnaire) {
2     ArbreMap<string, string>::Iterateur iter = dictionnaire.debut();
3     for(; iter; ++iter) {
4         cout << iter.cle() << " : " << iter.valeur() << endl;
5         iter.valeur() = "Nouveau titre";
6     }
7 }
```

Solution #1 / Déclaration

Deux (2) classes (`IterateurConst` et `Iterateur`) :

```
1  template <class K, class V>
2  class ArbreMap {
3     class Entree { /*...*/ };
4     ArbreAVL<Entree> entrees; // représentation
5   public:
6     class Iterateur {
7        typename ArbreAVL<Entree>::Iterateur iter;
8      public:
9        V& valeur() const {return (V&) (*iter).valeur;}
10    };
11    class IterateurConst {
12       typename ArbreAVL<Entree>::Iterateur iter;
13     public:
14       const V& valeur() const {return (*iter).valeur;}
15    };
16 }
```

Solution #1 / Utilisation

Deux (2) classes (`IterateurConst` et `Iterateur`) :

```
1  void afficher(const ArbreMap<string, string>& dict) {
2    ArbreMap<string, string>::IterateurConst iter = dict.debut(); // debut() const;
3    for(; iter; ++iter)
4      cout << iter.cle() << " : " << iter.valeur() << endl;
5  }
6
7  void modifier(ArbreMap<string, string>& dict) {
8    ArbreMap<string, string>::Iterateur iter = dict.debut(); // debut() non const;
9    for(; iter; ++iter)
10     iter.valeur() = "Nouveau titre";
11 }
```

Solution #1 / Cas libstdc++

C'est la solution retenue dans la bibliothèque standard de C++ (STL).

```
1  void afficher(const map<string, string>& dict) {
2      map<string, string>::const_iterator iter = dict.begin();
3      for(; iter != dict.end(); ++iter)
4          cout << iter->first << " : " << iter->second << endl;
5  }
6
7  void modifier(map<string, string>& dict) {
8      map<string, string>::iterator iter = dict.begin();
9      for(; iter != dict.end(); ++iter)
10         iter->second = "Nouveau titre";
11 }
```

Solution #2

- Un itérateur ne permet pas de retourner la valeur non const.
- Pour modifier la valeur, on force l'usage de l'opérateur [].
- Plus simple à implémenter.

```
1  template <class K, class V> class ArbreMap {
2      ...
3      class Iterateur {
4          ...
5          const V& valeur();
6      };
7  };
8
9  void afficherEtModifier(ArbreMap<string, string>& dict) {
10     ArbreMap<string, string>::Iterateur iter = dict.debut(); // debut() non const;
11     for(; iter; ++iter) {
12         cout << iter.cle() << " : " << iter.valeur() << endl;
13         dict[iter.cle()] = "Nouveau titre"; // complexité ?
14     }
15 }
```

Solution #2 améliorée

```
1  template <class K, class V> class ArbreMap {
2      ...
3      class Iterateur {
4          ...
5          const V& valeur();
6      };
7      V& operator[](const Iterateur& iter) { return (V&) (*iter).valeur; }
8      const V& operator[](const Iterateur& iter) const { return (*iter).valeur; }
9  };
10 void afficherEtModifier(ArbreMap<string, string>& dict) {
11     ArbreMap<string, string>::Iterateur iter = dict.debut(); // debut() non const;
12     for(; iter; ++iter) {
13         cout << iter.cle() << " : " << iter.valeur() << endl;
14         dict[iter] = "Nouveau titre"; // complexité ?
15     }
16 }
```

Problème : compter la fréquence des mots

Écrivez un programme qui lit : «a b c d a b c a b a»
Et retourne :

- a : 4
- b : 3
- c : 2
- d : 1

Le Programme

```
1  int main() {
2      ArbreMap<char, int> compteurs;
3      while(cin >> ws) {
4          char lettre;
5          cin >> lettre;
6          compteurs[lettre]++;
7      }
8
9      for(ArbreMap<char, int>::Iterateur iter = compteurs.debut(); iter; ++iter)
10         cout << iter.cle() << " : " << iter.valeur() << endl;
11 }
```

Problème : Historique de température

Question

Écrivez un programme qui permet (1) d'estimer la température à une date donnée et (2) de calculer la température moyenne entre deux dates. On suppose que la température varie de façon linéaire entres deux dates pour lesquelles la température est connue. Par exemple, la température au temps 0.5 est de 9.9. Votre solution doit être construite autour d'une classe `Historique` qui permet de garder l'historique des températures en mémoire. Vous devez choisir une représentation de la classe `Historique` et coder les fonctions `estimeTemperature` et `calculeMoyenne`.

Entrée

```
0.00 9.7
1.00 10.1
2.00 10.2
3.00 10.5
4.00 10.3
5.00 10.7
6.01 11.7
7.00 12.4
8.01 14.7
```

L'entrée pourrait être désordonnée.

Représentation

```
1  class Historique {
2    private:
3      // Représentation ?
4
5    public:
6      Historique();
7      ~Historique();
8      void ajouter(float date, float temperature);
9      float estimeTemperature(float date) const;
10     float calculeMoyenne(float datedebut, float datefin) const;
11 };
```

Représentation

```
1  class Historique {
2    private:
3      // Association d'un temps (float) avec une température (float)
4      ArbreMap<float, float> donnees;
5
6    public:
7      Historique();
8      ~Historique();
9      void ajouter(float date, float temp) { donnees[date] = temp };
10     float estimeTemperature(float date) const;
11     float calculeMoyenne(float datedebut, float datefin) const;
12 };
```

Estimer : Ébauche 1

```
1 float Historique::estimerTemperature(float date) const {
2     return donnees[date];
3 }
```

Estimer : Ébauche 2

```
1  float Historique::estimerTemperature(float date) const {
2      ArbreMap<float,float>::IterateurConst iter1 = donnees.rechercherEgalOuPrecedent(date);
3      ArbreMap<float,float>::IterateurConst iter2 = donnees.rechercherEgalOuSuivant(date);
4      return (iter1->valeur + iter2->valeur) / 2.0;
5  }
```

Estimer : Ébauche 3

```
1  float Historique::estimerTemperature(float date) const {
2      ArbreMap<float,float>::IterateurConst iter = donnees.rechercherEgalOuPrecedent(date);
3
4      float d1 = iter.cle();
5      float t1 = iter.valeur();
6      ++iter;
7      float d2 = iter.cle();
8      float t2 = iter.valeur();
9
10     return t1 + (t2 – t1) * (date – d1) / (d2 – d1);
11 }
```

Estimer : Version finale

```
1  float Historique::estimerTemperature(float date) const {
2      ArbreMap<float,float>::IterateurConst iter = donnees.rechercherEgalOuPrecedent(date);
3
4      // Si aucune donnée avant date, le mieux qu'on puisse retourner, c'est la première donnée
5      if(!iter) return donnees.debut().valeur();
6      if(iter.cle() == date) return iter.valeur(); // Si on trouve la date, aucune interpolation
7
8      // Sinon, on doit faire une interpolation lineaire avec la donnee suivante
9      float d1=iter.cle();
10     float t1=iter.valeur();
11     ++iter;
12     float d2=iter.cle();
13     float t2=iter.valeur();
14
15     return t1 + (t2 – t1) * (date – d1) / (d2 – d1);
16 }
```

```
1  float Historique::calculerMoyenne(float debut, float fin) const {
2      ArbreMap<float,float>::IterateurConst iter = donnees.rechercherEgalOuPrecedent(debut);
3      if(!iter) iter = donnees.debut();
4
5      float sommeT = 0; // accumulateur de température * durée
6      while(iter && iter.cle() < fin) {
7          float d1, d2, t1, t2;
8          if(iter.cle() < debut) { d1 = debut; t1 = estimerTemperature(debut); }
9          else { d1 = iter.cle(); t1 = iter.valeur(); }
10         ++iter;
11         if(!iter) break;
12         if(iter.cle() > fin) { d2 = fin; t2 = estimerTemperature(fin); }
13         else { d2=iter.cle(); t2=iter.valeur(); }
14         sommeT += ((t2 + t1) / 2.0) * (d2 – d1);
15     }
16     return sommeT / (fin – debut); // retourner la moyenne
17 }
```