Entraînement au concours ACM-ICPC

Conteneurs des bibliothèques standard C++ et Java

Mars 2016

Pierre Senellart

Plan

Généralités

Collections de base

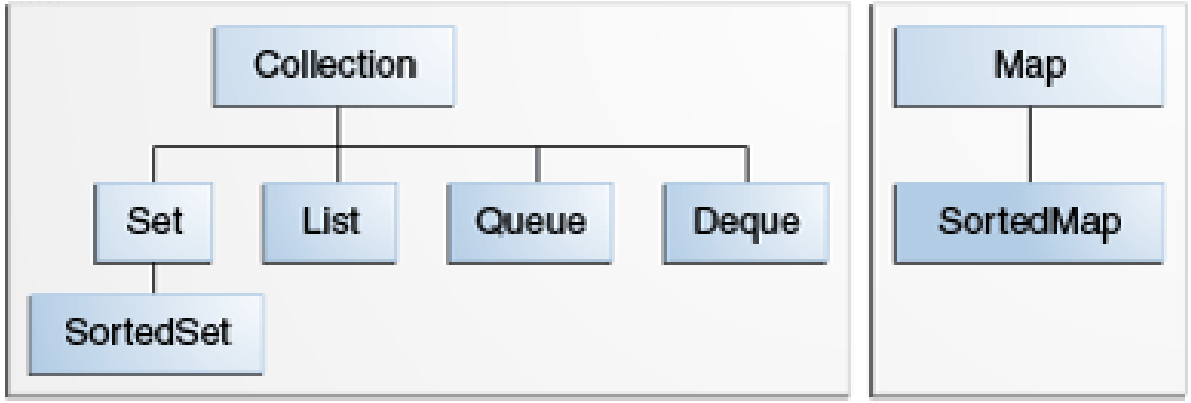
Collections à usage plus rare

Collections

- Les collections regroupent un nombre arbitraire d'éléments de même type
- Suivant les collections, la complexité des opérations suivantes varie :
  - accéder au n-ième élément
  - rechercher un élément
  - ajouter un élément à une position arbitraire
  - supprimer un élément
  - parcourir les éléments dans l'ordre
- Cas particulier, tableaux associatifs : peuvent être vus comme collections de paires (clef, valeur)
- Important de choisir la collection la mieux adaptée aux besoins d'un problème

Collections en Java

- Hiérarchie d'interfaces



- Chaque classe concrète implémente une ou plusieurs de ces interfaces
- Les collections standards sont dans le paquetage `java.util`
- `Vector`, `Hashtable` : collections thread-safe. Inutiles dans un contexte single-thread

Collections en C++

- Pas d'héritage, C++ utilise le mécanisme de la programmation générique (templates) pour permettre à des fonctions de manipuler des collections diverses
- Toutes les collections sont dans l'espace de nom `std` (ou `std::tr1` pour `unordered_set`, `unordered_map`, etc., sur les compilateurs pré-C++ 2011)
- Certaines collections (`stack`, `queue`, `priority_queue`) sont des adapteurs qui peuvent utiliser plusieurs classes (`list`, `vector`...) comme implémentation

Itérer sur une collection (1/2)

- En C++ 1998 :

```
for(collection_type::const_iterator
      it=collection.begin(),itend=collection.end();
      it!=itend;
      ++it) {
  value_type v=*it;
  /* Faire quelque chose avec v */
}
```

Remplacer `const_iterator` par `iterator` quand on modifie la collection.

Itérer sur une collection (2/2)

- En C++ 2011 :

```
for(auto v : collection) {
  /* Faire quelque chose avec v */
}
```

Utiliser `auto &` pour modifier la collection.

- En Java :

```
for(value_type v : collection) {
  /* Faire quelque chose avec v */
}
```

Algorithmes de la bibliothèque standard

- Grand nombre d'algorithmes polymorphes dans la bibliothèque standard s'appliquant à des collections quel que soit leur type
- En C++, `#include <algorithm>` ; les algorithmes prennent en argument des itérateurs vers le début et la fin
- En Java, `import java.util.Collections ;` ; les algorithmes prennent en argument les collections
- Pour un tri :

```
std::sort(collection.begin(),collection.end());
```

```
Collections.sort(collection);
```

- Grand nombre d'algorithmes disponibles : tri, recherche, recherche dichotomique, copie, inversion, remplissage, énumération des permutations d'un ensemble...

Plan

Généralités

Collections de base

Collections à usage plus rare

Tableaux de taille fixe

- Tableaux de base du langage `value_type[size]` en C++ et Java
- En C++ (mais pas en C 1999 !) la taille a besoin d'être connue à la compilation
- Il existe aussi une classe `array<value_type,size>` en C++ 2011, plus homogène avec les autres collections, et rendant certaines interactions avec la bibliothèque standard plus pratiques
- Occupation mémoire minimale (on ne peut pas faire mieux)
- Accès à un élément arbitraire : $O(1)$
- Pas de suppression, pas d'ajout d'élément
- Initialisation avec memset : `memset(t, 0, sizeof(value_type) * size)` (plus efficace qu'une boucle)

Tableaux de taille variable

- `vector` en C++, `ArrayList` en Java
- Peut typiquement occuper 2 fois plus d'espace qu'un tableau de taille fixe
- Accès à un élément arbitraire : $O(1)$
- Ajout/suppression à la fin du tableau : $O(1)$ amorti
- Ajout/suppression à un autre endroit : $O(n)$
- Pratique pour indexer des éléments par des entiers quand il n'y a pas (ou peu) de « trous » dans les entiers indexant
- Il existe aussi `deque` en C++, `ArrayDeque` en Java pour ajout/suppression en temps constant à la fois au début et à la fin du tableau

Liste doublement chaînée

- `list` en C++, `LinkedList` en Java
- Accès au premier ou dernier élement : $O(1)$
- Accès à l'élement suivant ou précédent : $O(1)$
- Accès à un élément arbitraire : $O(n)$
- Ajout/suppression à un endroit arbitraire : $O(1)$

Arbre de recherche équilibré (ensemble)

- `set` en C++, `TreeSet` en Java
- Les éléments peuvent être parcourus dans l'ordre (ordre naturel, ou ordre défini par un comparateur)
- Accès à un élément : $O(\log(n))$
- Ajout/suppression d'un élément arbitraire : $O(\log(n))$

Table de hachage (ensemble)

- `std::tr1::unordered_set` en C++ pré-2011, `std::unordered_set` en C++ 2011, `HashSet` en Java
- Les éléments sont parcourus dans un ordre arbitraire (celui de la fonction de hachage)
- Accès à un élément : $O(1)$ amorti
- Ajout/suppression d'un élément arbitraire : $O(1)$ amorti
- En Java, il existe aussi `LinkedHashSet` qui combine table de hachage et liste chaînée pour accès rapide et parcours dans l'ordre où les éléments ont été insérés

Arbre de recherche équilibré (tableau associatif)

- Pour stocker des paires (clef, valeur)
- `map` en C++, `TreeMap` en Java
- Les éléments peuvent être parcourus dans l'ordre de leur clef (ordre naturel, ou ordre défini par un comparateur)
- Accès à un élément par sa clef : $O(\log(n))$
- Ajout/suppression d'un élément arbitraire par sa clef : $O(\log(n))$

Table de hachage (tableau associatif)

- `std::tr1::unordered_map` en C++ pré-2011, `std::unordered_map` en C++ 2011, `HashMap` en Java
- Les éléments sont parcourus dans un ordre arbitraire (celui de la fonction de hachage)
- Accès à un élément par sa clef : $O(1)$ amorti
- Ajout/suppression d'un élément arbitraire par sa clef : $O(1)$ amorti
- En Java, il existe aussi `LinkedHashMap` qui combine table de hachage et liste chaînée pour accès rapide et parcours dans l'ordre où les éléments ont été insérés

Plan

Généralités

Collections de base

Collections à usage plus rare

Liste simplement chaînée

- `forward_list` en C++ 2011, n'existe pas en Java ; facile à implémenter à la main
- Accès au premier élement : $O(1)$
- Accès à l'élement suivant : $O(1)$
- Accès à un élément arbitraire : $O(n)$
- Ajout/suppression à un endroit arbitraire : $O(1)$

Pile (Last In First Out)

- `stack` en C++, `Stack` en Java
- Accès au premier élement : $O(1)$
- Ajout/suppression au début : $O(1)$
- Accès à un autre élément ou ajout/suppression à un autre endroit impossibles

File (First In First Out)

- `queue` en C++, n'importe quelle classe implémentant l'interface `Queue` en Java
- Accès au premier élémént : $O(1)$
- Suppression du premier élément : $O(1)$
- Ajout à la fin : $O(1)$
- Accès à un autre élément ou ajout/suppression à un autre endroit impossibles

File de priorité

- `priority_queue` en C++, `PriorityQueue` en java
- Les éléments sont insérés et gardés triés dans la file de priorité, par leur ordre naturel ou par un ordre de priorité défini par un comparateur ; deux éléments peuvent avoir la même priorité
- Accès à l'élément de plus haute priorité : $O(1)$
- Suppression de l'élément de plus haute priorité : $O(\log(n))$
- Ajout d'un élément : $O(\log(n))$
- Accès à un autre élément ou suppression à un autre endroit impossibles
- Modification dynamique des priorités impossible
- En C++, implémenter un `bool operator<(T a)` dans la classe T pour fournir la fonction de comparaison des objets de type T

Multi-ensembles

- `multiset` (arbre) ou `unordered_multiset` (table de hachage) en C++ ; n'existe pas en Java
- Pareil qu'un `set` / `unordered_set` mais plusieurs éléments identiques (ou, en tous cas, se comparant identiquement) peuvent être présents dans l'ensemble

Tableaux associatifs à valeurs multiples

- `multimap` (arbre) ou `unordered_multimap` (table de hachage) en C++ ; n'existe pas en Java
- Pareil qu'un `map` / `unordered_map` mais plusieurs éléments identiques (ou, en tous cas, se comparant identiquement) peuvent être présents dans l'ensemble

Licence de droits d'usage

Contexte public } avec modifications

Par le téléchargement ou la consultation de ce document, l'utilisateur accepte la licence d'utilisation qui y est attachée, telle que détaillée dans les dispositions suivantes, et s'engage à la respecter intégralement.

La licence confère à l'utilisateur un droit d'usage sur le document consulté ou téléchargé, totalement ou en partie, dans les conditions définies ci-après et à l'exclusion expresse de toute utilisation commerciale.
Le droit d'usage défini par la licence autorise un usage à destination de tout public qui comprend :
– le droit de reproduire tout ou partie du document sur support informatique ou papier,
– le droit de diffuser tout ou partie du document au public sur support papier ou informatique, y compris par la mise à la disposition du public sur un réseau numérique,
– le droit de modifier la forme ou la présentation du document,
– le droit d'intégrer tout ou partie du document dans un document composite et de le diffuser dans ce nouveau document, à condition que :
  – L'auteur soit informé.

Les mentions relatives à la source du document et/ou à son auteur doivent être conservées dans leur intégralité.

Le droit d'usage défini par la licence est personnel et non exclusif.
Tout autre usage que ceux prévus par la licence est soumis à autorisation préalable et expresse de l'auteur : `sitepedago@telecom-paristech.fr`