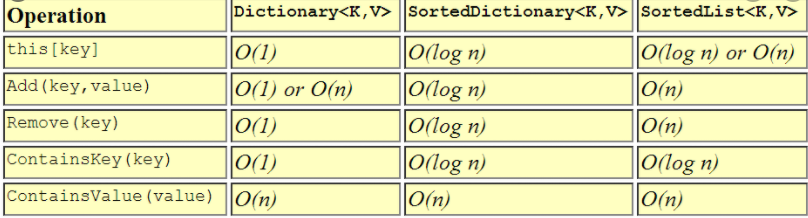
Datové struktury s klíčem, indexem a hashované

**S klíčem**

**C#**

* Dictionary
* SortedDictionary
* ConcurrentDictionary
* ListDictionary
* SortedList

implementují IDictionary<TKey,TValue>

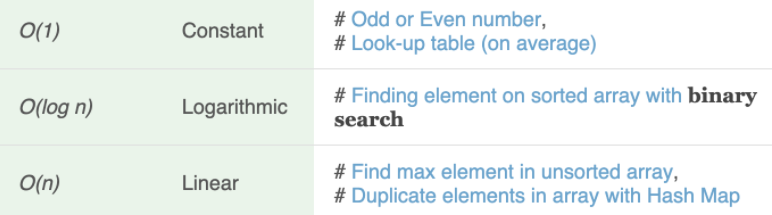


Nejrychlejší

O(1)

O(log n) je druhý nejrychlejší

O(n) až třetí



**Java**

**Map – Java**

Map<Key,Value> mapa = new HashMap<Key,Value>(); (Map je interface, HashMap je objekt)

**Metody**

|  |  |
| --- | --- |
| get(key) | Vrátí value k danému klíči |
| put(key,value) | Vložení |
| containsKey(key) | Bool, ptá se, jestli mapa obsahuje ke |
| containsValue(value) | Bool, ptá se, jestli mapa obsahuje value |
| Set<typeOfKey>keySet() | Vrátí množinu obsahující klíče |

Další: remove(key), size(), isEmpty(), clear(), Collection<V>values()

V Javě jsou dvě univerzální implementace mapy – HashMap a TreeMap a další WeakHashMap, LinkedHashMap a starej ale stále použitelný HashTable

**HashMap vs TreeMap**

HashMap je rychlejší, TreeMap automaticky třídí klíče.

HashMap má implementované metody hashCode() a equals(), které se využívají pro zjištění jedinečnosti klíče

HashMap má implementovanou metodu compareTo(), pro porovnání a zatřídění klíčů. Instance vkládané jako klíč musí implementovat rozhraní Comparable

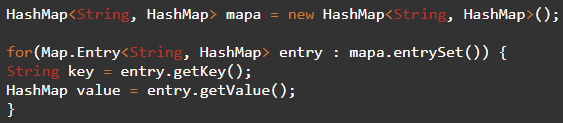
Hledání přes klíče je rchlejší než přes hodnoty (většinou).

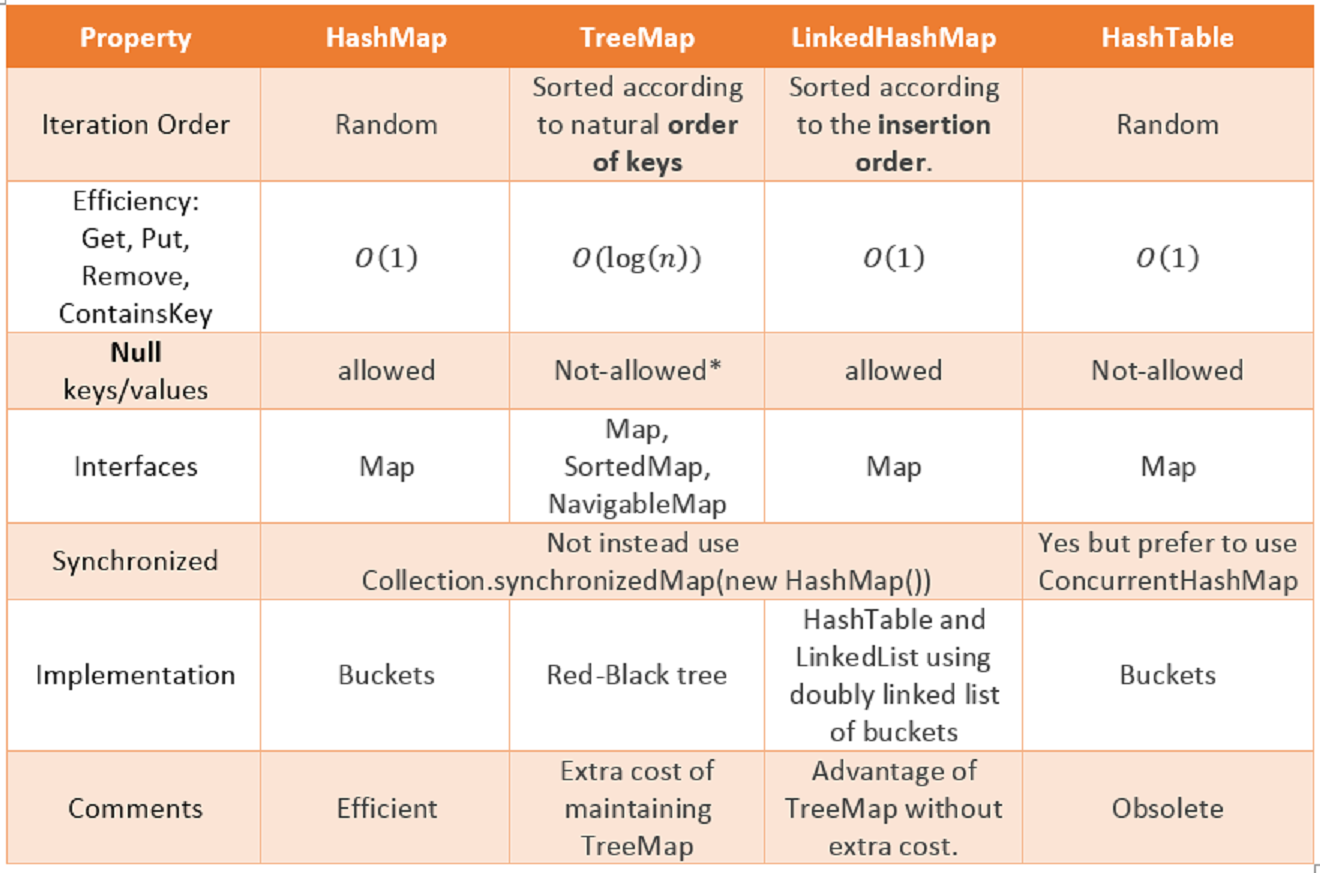
Mapa.put(1,“a“);

Mapa.put(1,“b“);

Value pro key 1 je “b“;

**Procházení mapy**





**Dictionary**

Generický typ

Using System.Collection.Generic

Implementuje IDictionary IEnumerable IDeserializationCallback ISerializable

Na haldě je uložen rozloženě.

Do slovníku se ukládá dvojice objektů: klíč a k němu přiřazená hodnota.

Key nemůže být null, ale value může.

Slovník je abstraktní datová struktura, skládající se z množiny unikátních klíčů a množiny hodnot, kde každý klíč je asociován (mapován) s jednou hodnotou nebo s množinou hodnot.

Nemá index, místo něj mám unikátní klíč (různého datového typu).

Použití slovníků má jedno logické omezení – ve slovníku nemohou být duplicitní klíče.

Často se používá i v situaci, kdy chceme zajistit rychlý přístup k prvkům seznamu dle klíče.

Při deklaraci slovníku a při volání konstruktoru je možné (a velmi vhodné) určit typy klíčů a typy hodnot.

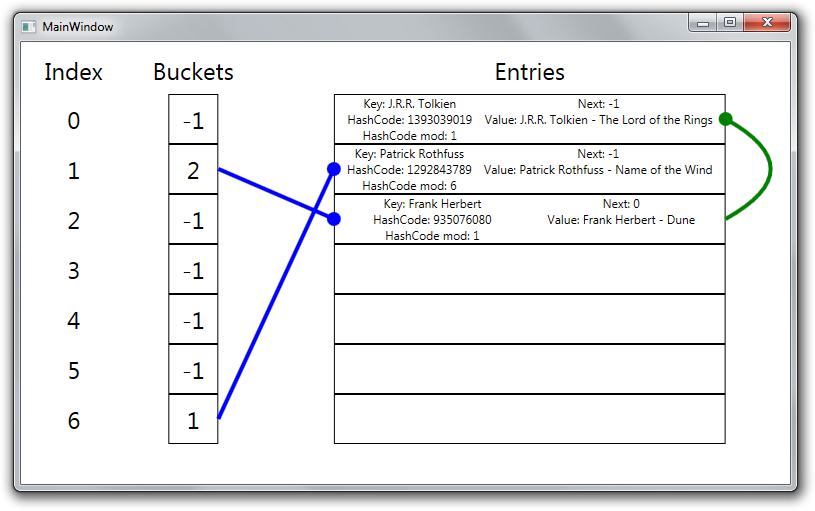
Velikost slovníku roste podle potřeby.

**Struktura:** HashTable s linky na pole kvůli kolizím

**Zachované pořadí:** ano

**Označení místa uložení:** key

**Dvě stejné hodnoty:** klíč musí být unikát, value může



Dictionary<Key,Value> dic = new Dictionary<Key,Value>();

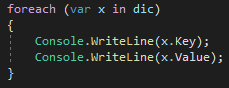
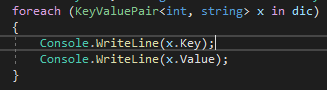
dic.Add(1,“a“);

dic.Add (1,“b“); // dic.TryAdd (1,“b“); se vyhne vyjímce, ale hodnotu to nezmění

Vyvolá se ArgumentException, ale když se chytí, tak hodnota pro 1 je “a“.

**Procházení**

Var x je typu KeyValuePair (v tomhle přípradě <int,string>)

**Metody**

Jsou podobné jako v Javě

|  |  |
| --- | --- |
| TryGetValue(key, out value) | Vrátí bool k danému klíči, na out i hodnotu value |
| Add(key,value) | Vložení |
| ContainsKey(key) | Bool, ptá se, jestli mapa obsahuje key |
| ContainsValue(value) | Bool, ptá se, jestli mapa obsahuje value |

Další metody: Clear(), Equals(), Remove(key), GetType(),GetValueOrDefault(key)

Add() – O(1), s kolizí v poli buckets O(n)

Remove() - O(1), s kolizí v poli buckets O(n)

Dictionary[key] - O(1), s kolizí v poli buckets O(n)

**Vlastnosti:** Values (kolekce hodnot), Keys (kolekce klíčů), dic[key] (get v Java), Count

**SortedDictionary**

System.Collections.Generic

Implementuje IDictionary IEnumerable

Hodně podobný Dictionary.

Je automaticky seřazený podle klíče

Sorted taky znamená, že key musí implementovat IComparer proměny ho mají, ale vlastní třídy ne, pokud se použije IComparable tak bude upřednostněn

SortedDictionary má rychlejší vkládání a odebírání pro neseřazená data než SortedList<Key,Value> a to O(log n) vs O(n)

**Struktura:** binární strom (red-black)

Red-Black tree - je binární strom s jedním dvouhodnotovým příznakem v uzlu navíc. Tento příznak představuje barvu uzlu, která může být červená nebo černá. Zajišťuje, že žádná cesta z kořene do libovolného listu stromu nebude dvakrát delší než kterákoli jiná, to znamená, že strom je přibližně vyvážený.

Musí splňovat

* Každý uzel je bud’ černý nebo červený.
* Každý list (NULL) je černý.
* Jestliže je daný uzel červený, pak jeho potomci jsou černí.
* Každá cesta z libovolného uzlu do listu obsahuje stejný počet černých uzlů.

**Zachované pořadí:** ne

**Označení místa uložení:** key

**Dvě stejné hodnoty:** klíč musí být unikát, value může

Procházení stejné jako Dictionary, metody taky.

Používá na pozadí SortedSet

Add() – O(log n)

Remove() - O(log n)

Dictionary[key] - O(log n)

**ListDictionary**

Není generický

System.Collections.Specialized

Implementuje ICollection IDictionary IEnumerable

Je implementován s jednosměrným linkedlistem

Je doporučena k použití pouze při menším počtu prvků, než je 10

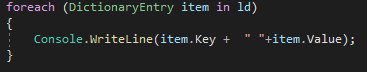
Neměl by být použit, pokud je výkon důležitý pro velký počet prvků.

//Elementy nejsou v žádném zaručeném pořadí (náhodné)

ListDictionary dic = new ListDictionary();

**Procházení**

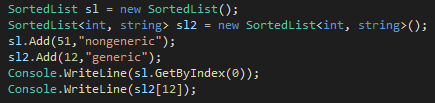
Procházení pomocí DictionaryEntry, var tady nejde použít (KeyValuePair nejde)



Schází mu metody jako TryGetValue() a TryAdd()….

**SortedList a SortedList<Key,Value> negenerický a generický typ**

Hlavní rozdíl: Negenerický umožňuje získání prvků pomocí indexů



**Struktura:** 2 pole, jedno pro klíče, druhé pro hodnoty

**Zachované pořadí:** ne

**Označení místa uložení:** key

**Dvě stejné hodnoty:** klíč musí být unikát, value může

Add() – O(n), O(log n) přidán na konec listu

Remove() - O(n)

Dictionary[key] - O(log n)

**SorterList**

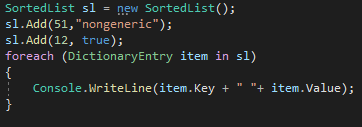
System.Collections

Implementuje ICollection IDictionary IEnumerable ICloneable

Výhoda je následující: key musí zůstat stejného datového typu, ale value nemusí

Byl vytvořen ve verzi 1.1 a je tu pořád kvůli zpětné kompatibilitě

Procházení pomocí DictionaryEntry, var tady nejde použít (KeyValuePair nejde)



Metody: Add(key,value), Clear(), Clone(), ContainsKey(), GetByIndex(int), IndexOfValue(int), IndexOfKey(int), Remove(key), RemoveAt(int), SetByIndex(int, value)

Podtržený jsou metody pracující s indexem

Vlastnosti: Capacity - Získá nebo nastaví kapacitu SortedList objektu.

**SortedList<Key,Value>**

System.Collections.Generic

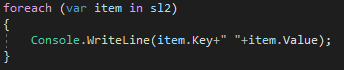
Implementuje ICollection IDictionary IEnumerable

SortedList<Key,Value> používá méně paměti než SortedDictionary<Key,Value> a je rychlejší při procházení všech prvků

Chová se jako Dictionary, ale je seřazený

**Procházení**

zase pomocí KeyValuePair<key,value> , takže var je zase ok



**Metody:** Add(key,value), Clear(), ContainsKey(), IndexOfKey(key), IndexOfValue(value), RemoveAt(int)

**ConcurrentDictionary**

System.Collections.Concurrent

Implementuje IDictionary IEnumerable

Představuje kolekci párů klíč/hodnota, které jsou bezpečné pro přístup z více vláken, ke kterým lze přistupovat více vlákny současně.

Je thread-safe (až na metody AddOrUpdate() nebo GetOrAdd())

Interně spravuje zamykání a poskytuje snadné rozhraní pro přidávání / aktualizaci položek

Procházení pomocí KeyValuePair<Key,Value>

**Indexy**

* Pole
* List
* ArrayList
* Collection
* ObservableCollection

**Pole**

Pole může být jednorozměrné, multidimenzionální nebo vícenásobné.

Pole, je kolekce, do které lze snadno vkládat větší množství hodnot stejného typu. U pole se musí definovat jeho velikost. Velikost nejde později změnit (v Java, C#. V PHP lze velikost změnit). První index má hodnotu 0.

**Plusy:**

pokud potřebujeme uložit předem daný počet prvků, je pole ze všech datových struktur nejefektivnější.

pole umožňuje vkládat přímo primitivní datové typy, u ostatních datových struktur se musí převést na objekty.

je možné vytvářet jednorozměrná i vícerozměrná pole.

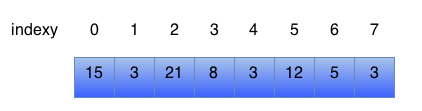
Všechny hodnoty v poli zabírají stejně místa

**Mínusy:**

Pole má pouze k dispozici metody třídy Object !

je třeba předem znát počet prvků, které do něj budete ukládat

špatně se v něm vyjadřuje neexistence prvku – pokud máme pole pro 20 čísel a zatím máme vloženo pouze 10 čísel, potřebujeme pomocnou proměnnou pro vyjádření, která políčka po jsou neobsazena



**deklarace**

**C#**

typ [] pole = new typ[počet prvků];

**Java**

typ [] pole = new typ[počet prvků]; nebo typ pole [] = new typ[počet prvků];

**deklarace s rovnou s prvky**

int[] pole = {1, 2, 3, 4, 5, 6};

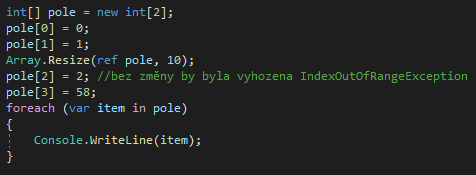
**procházení**

pole se dá dobře procházet pomocí for (díky metodě lenght) nebo foreach

Třída Array. (C#) Arrays (Java) Metody pro práci s polem

**C#**

Sort(), Copy(), Clear(), Reverse(), Find(), IndexOf(), Resize(ref pole, nová velikost)

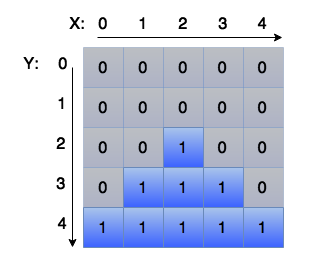


Na pole jdou také volat metody z Linq

Pokud se pokusíme vypsat index, který není v poli tak dostaneme IndexOutOfRangeException

**Metody**

Max(), Min(), Avarage(), First(), Last(), Contains(), Reverse()

**Dvourozměrné pole**

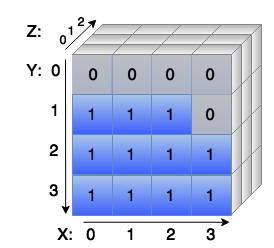
index je složen z X a Y

int[,] pole = new int[5, 5];

pole[0,0] = 1;

java

int[][] pole = new int[5][5];

**více rozměrná pole**

index je složen z X a Y a Z

**c#**

int[,,] pole = new int[5,5,5];

**java**

int[][][] pole = new int[5][5][5]

**Seznamy**

Seznamy (anglicky a často i česky list) jsou kolekce, které umožňují prvky za běhu programu přidávat a mazat.

**List**

Generická kolekce

System.Collections.Generic

Implementuje IEnumerable IList

Může měnit svoji velikost i zaběhu programu

velikost se dynamicky zvětšuje podle potřeby.

Do konstruktoru se může zadat počáteční kapacita.

**Struktura:** pole

**Zachované pořadí:** ano

**Označení místa uložení:** index

**Dvě stejné hodnoty:** ano

Add() – O(1)

Remove() - O(n)

List[key] - O(1)

Metody: Add(T), BinarySearch(T), Clear(), Constains(T), CopyTo(kolekce,start index), IndexOf(T), Insert(T,index), Remove(T, Sort(), ToString()

Procházení foreach, for má i metody ForEach(action T)

**ArrayList**

Negenerická kolekce

System.Collections

Implementuje ICollection IEnumerable IList ICloneable

C# není doporučeno používat ArrayList místo něho radši List<T>

ArrayList nemůže používat Linq na rozdíl od listu

Může měnit svoji velikost i zaběhu programu

Metody podobné Listu

**Collection**

Generická

System.Collections.ObjectModel

Implementuje IEnumerable IList

Oproti listu nemá pevné metody a ty se dají přepsat

**ObservableCollection**

generická

System.Collections.ObjectModel

Implementuje IEnumerable IList

Představuje dynamickou kolekci dat, která poskytuje oznámení o tom, že se položky přidají, odeberou, nebo když se obnoví celý seznam.

**Hash**

**Hashtable**

negenerická

System.Collections

Implementuje ICollection IDictionary IEnumerable ICloneable IDeserializationCallback ISerializable

Ukládá data jako key a value

Data jsou uspořádány podle hashCodu key

Objekty použity jako key musí mít metody GetHashCode() a Equals() nebo IComparer

Key nemůže být null, ale value může

Key a value můžou být jakéhokoliv datového typu

**HashSet**

generická

System.Collections.Generic

Třída poskytuje operace s vysokým výkonem.

Sada je kolekce, která neobsahuje žádné duplicitní prvky.

Prvky nejsou seřazené, ani je nelze seřadit pomocí nějaké metody

Dá se představit jako Dictionary bez Value (jen Key)

poskytuje mnoho matematických sad operací, jako je například sčítání v sadě (sjednocení) a nastavení odčítání.

**Struktura:** hashtable s linky na pole kvůli kolizi

**Zachované pořadí:** ano

**Označení místa uložení:** hash

**Dvě stejné hodnoty:** ne

Add() – O(1) s kolizí nebo přeskapacitu O(n)

Remove() - O(1) s kolizí O(n)

get - O(1) s kolizí O(n)



Metody: Add(T), Clear(), Constains(T), GetHashCode()