# Vlastnosti datových struktur - Seřazenost a opakování prvků, Indexace, hashování a klíče prvků

## Seřazené datové struktury:

* SortedSet
* SortedList
* SortedDictionary

Jsou takové datové struktury, které své elementy specificky řadí. Vzestupně nebo sestupně s ohledem na porovnávající funkci. Tyto struktury běžně umožňují rychlejší vyhledávání elementů naopak vkládání do těchto struktur může trvat více času než u jiných neseřazených struktur.

using System;

using System.Collections.Generic;

namespace collections

{

internal class Program

{

static void Main(string[] args)

{

SortedSet<MojeKolekce> set = new SortedSet<MojeKolekce>();

set.Add(new MojeKolekce(14));

set.Add(new MojeKolekce(4));

set.Add(new MojeKolekce(3));

foreach (MojeKolekce i in set)

{

Console.WriteLine(i.Age);

}

//3

//4

//14

Console.WriteLine();

SortedList<int,int> list = new SortedList<int,int>(new ComperableInt ());

list.Add(7,2);

list.Add(5, 4);

list.Add(6,3);

foreach (int i in list.Values)

{

Console.WriteLine(i);

}

//2

//3

//4

Console.WriteLine();

SortedDictionary<int,int> dictionary = new SortedDictionary<int,int>();//defaultne vz estupne

dictionary.Add(7,2);

dictionary.Add(5,4);

dictionary.Add(6,3);

foreach (int i in dictionary.Values)

{

Console.WriteLine(i);

}

//4

//3

//2

}

}

class ComperableInt : IComparer<int>

{

public int Compare(int first,int second)

{

return first < second ? 1 : first > second?-1:0;//sestupne

}

}

class MojeKolekce : IComparable<MojeKolekce>

{

public int Age { get; set; }

public MojeKolekce(int age)

{

this.Age = age;

}

public int CompareTo(MojeKolekce obj)

{

return this.Age < obj.Age ? -1 : this.Age > obj.Age ? 1 : 0;//vzestupne

}

}

}

## Neseřazené datové struktury:

* List
* Queue
* Stack
* HashSet
* Dictionary
* LinkedList
* ConcurrentBag
* ConcurrentQueue
* ConcurrentStack

using System;

using System.Collections.Generic;

using System.Collections.Concurrent;

namespace collections

{

internal class Program

{

static void Main(string[] args)

{

Queue<string> queue = new Queue<string>();

queue.Enqueue("a");

queue.Enqueue("b");

Console.WriteLine(queue.Dequeue());//a

Console.WriteLine();

Stack<string> stack = new Stack<string>();

stack.Push("a");

stack.Push("b");

Console.WriteLine(stack.Pop());//b

Console.WriteLine();

LinkedList<string> list = new LinkedList<string>();

list.AddLast("a");

list.AddLast("b");

Console.WriteLine(list.First.Value);//a

Console.WriteLine();

ConcurrentBag<string> bag = new ConcurrentBag<string>();

bag.Add("a");

bag.Add("b");

string item;

bag.TryPeek(out item);

Console.WriteLine(item);//b

}

}

}

Jsou takové datové struktury, kterých elementy nejsou ukládány v žádném specifickém pořadí. Tyto struktury mohou a nemusí povolovat duplikace. Nalezení prvku může být pomalejší, než u seřazené struktury naopak vložení bude rychlejší.

Výjimkou je například datová struktura HashSet, která využívá hash funkci díky které můžeme vyhledávat v funkci mnohem rychleji než v ostatních neseřazených funkcí. Navíc nedovoluje duplikáty ve struktuře.

## Opakovanost

### Neopakovatelné

Bez opakování

* HashSet
* SortedSet

using System;

using System.Collections.Generic;

namespace collections

{

internal class Program

{

static void Main(string[] args)

{

HashSet<int> hash = new HashSet<int>();

hash.Add(1);

hash.Add(2);

hash.Add(1);

foreach (var item in hash)

{

Console.WriteLine(item);

}

//1

//2

Console.WriteLine();

SortedSet<int> sorted = new SortedSet<int>();

sorted.Add(1);

sorted.Add(2);

sorted.Add(1);

foreach (var item in sorted)

{

Console.WriteLine(item);

}

//1

//2

}

}

}

Při pokusu vložení prvku, který již kolekce obsahuje, se prvek nevloží.

Bez opakování, key/value

### S klíčem neopakovatelné

U těchto kolekcí, se nemůže opakovat pouze key, value může

* Dictionary
* SortedDictionary
* ConcurrentDictionary
* ListDictionary
* SortedList
* HashTable

using System;

using System.Collections.Generic;

using System.Collections;

namespace collections

{

internal class Program

{

static void Main(string[] args)

{

Dictionary<int, int> map = new Dictionary<int, int>();

map[0] = 1;

map[1] = 2;

map[0] = 3;

foreach (int i in map.Keys)

{

Console.WriteLine(i);

}

//0 -> 3

//1 -> 2

Console.WriteLine();

Hashtable openWith = new Hashtable();

openWith.Add(1, 2);

openWith.Add(2, 3);

//openWith.Add(1, 4); throw excpetion

foreach (int i in map.Keys)

{

Console.WriteLine(i);

}

}

}

}

### Opakovatelné

Při pokusu vložení prvku, který již kolekce obsahuje, se prvek nevloží a bude vyhozena exception.

S opakováním

* List
* Queue
* Stack
* LinkedList
* ConcurrentBag
* ConcurrentQueue
* ConcurrentStack

## Indexem

* List, ArrayList
* Dictionary
* String[] (Array)

using System;

using System.Collections.Generic;

namespace collections

{

internal class Program

{

static void Main(string[] args)

{

string stringString = "a";

Console.WriteLine(stringString[0]);

string[] stringField = new string[1];

stringField[0] = "a";

stringField[1] = "b";

Console.WriteLine(stringField[0]);

List<string> stringList = new List<string>();

stringList.Add("a");

Console.WriteLine(stringList[0]);

}

}

}

## Hashované

Výhody hashovaných kolekcí oproti listu je rychlejší najití prvku, list musí iterovat (procházet jednu za jednou), ale u hashe se spočítá hashcode poté se půjde do bucketu a tam se pokusí najít stejný hashcode. Ze stejného důvodu je rychlejší i Remove() List má rychlejší přídání prvku, u hashe se musí vypočítat hashcode a bucket, list ho prostě přidá nakonec

1. **Hashování klíče nebo prvku:** Každý prvek nebo klíč je převeden na unikátní hodnotu pomocí hashovací funkce. Tato hodnota se používá pro rychlé vyhledávání v tabulce nebo sadě.
2. **Porovnání hodnoty klíče nebo prvku s existujícími položkami:** Pokud již v hashovací tabulce nebo sadě existuje prvek s touto hodnotou klíče nebo prvku, operace vložení selže a prvek nebo klíč nebudou vloženy.
3. **Určení indexu:** Na základě hashované hodnoty je určen index v hashovací tabulce nebo sadě, kam bude prvek nebo klíč vložen.
4. **Vložení prvku nebo klíče do hashovací tabulky nebo sady:** Pokud neexistuje prvek s touto hodnotou klíče nebo prvku, bude prvek nebo klíč vložen na příslušný index v hashovací tabulce nebo sadě.

* HashTable
* HashSet

using System;

using System.Collections.Generic;

using System.Collections;

namespace collections

{

internal class Program

{

static void Main(string[] args)

{

Hashtable hashtable = new Hashtable();

hashtable.Add("key1", "value1");

hashtable["asd"] = "asd";

hashtable["key1"] = "asd";

//hashtable.Add("asd", "value1");throw error

HashSet<string> keys = new HashSet<string>();

keys.Add("key1");

keys.Add("ke2");

keys.Add("key1");//nevlozi se znovu

}

}

}

