Datové struktury s klíčem, indexem a hashované

S klíčem

C#

- Dictionary
- SortedDictionary
- ConcurrentDictionary
- ListDictionary
- SortedList
- HashTable

Dictionary
Generický typ

Na haldě je uložen jako objekty v poli, objekt obsahuje key a value.

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

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

Key nemůže být null, ale value může. Důvod: z null nejde udělat hashcode

Value není povinně objekt, může být třeba hodnotový datový typ. Na rozdíl od HastTable, kde jsou value hodnoty objekty, proto může být dictionary rychlejší

Použití slovníků má jedno logické omezení – ve slovníku nemohou být duplicitní klíče, nesmí mít stejný stejný hashcode.

Objekty použity jako key musí mít metody GetHashCode() a Equals(), jinak by výpočet hashcode při vložení jiného objektu se stejnými proměnami měl jiný hashcode.

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

Velikost slovníku roste podle potřeby, protože je dynamický. Roste podle “prime numbers“ – 3,7,11,17,23,29,37,.. zvětší se, vždy když počet prvků přesáhne aktuální číslo

Dané číslo reprezentuje počet bucketů

Princip dictionary – kolekcích založených na hashtablu (hashset, hashtable)

Dictionary je asociativní pole, které vyhledává value na základě key, je implementovaný aby zajistilo nejrychlejší získání value pomocí key, ideálně konstantní O(1).

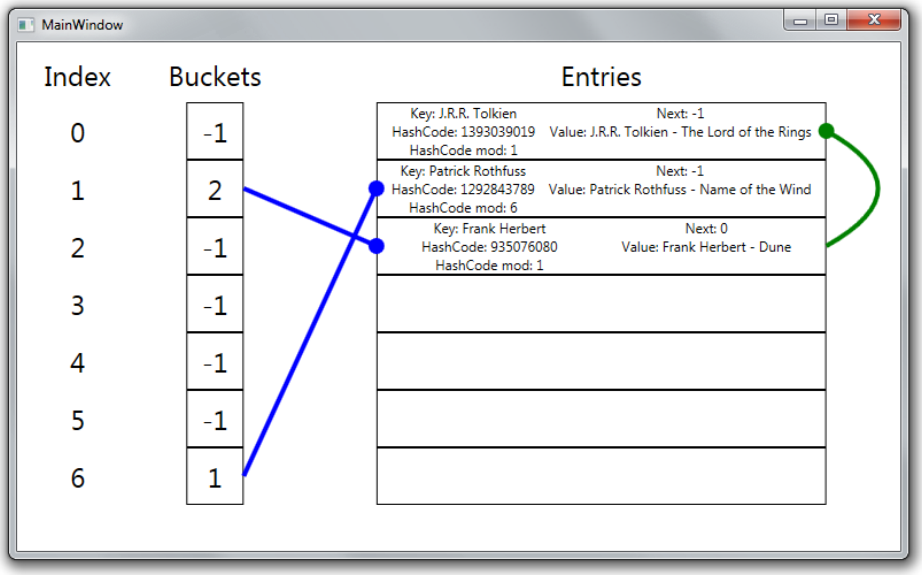
Dictionary obsahuje list bucketů, které mají v sobě uložené jednu nebo více values.

Když se vloží key a value do dictionary, tak se vezme hashcode key a vypočítá se podle počtů bucketů index bucketu kam se value a key uloží. (výpočet indexu bucketu – hashtable se 3 buckety: hashcode 1 – 1 % počet bucketů 3 = 1 index, hashcode 5654123 % 3 = 2)

Když podle key budeme chtít získat hodnotu, tak se vypočítá hashcode key a poté index bucketu a ten bude obsahovat referenci na pole entries kde se uloží (hashcode, key, next, value)

Je možné, aby bucket měl v sobě více hodnot. Díky výpočtu může z jiných hachcodů vzniknout stejný index bucketu. V tom případě index, který byl do bucketu vložen jako poslední obsahuje referenci na item který byl před ním (této referenci se říka next).

Když počet prvků při dalším přidání by měl překročit aktuálním prime number tak se vytvoří nové dictionary s velikostí dalšího prime number. A budou vypočítané nové indexy bucketů z hashcodů.



V C# debug takže asi ve struktuře dictionary, nejsou -1 u bucketů ani u next. Místo nich je 0.

Struktura: pole bucketů (zvané hashtable), které uchovává prvky

Zachované pořadí: ano

Označení místa uložení: key

Dvě stejné hodnoty: key 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“.

Metody

Add() – O(1), s kolizí v poli bucketů nebo při zvýšení kapacity O(n)

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

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

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

ContainsValue() – O(n) projde ValueCollection

Vlastnosti: Values (kolekce hodnot), Keys (kolekce klíčů), dic[key], Count

SortedDictionary

Generická

Hodně ve funkčnosti podobný Dictionary, jen je seřazený.

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)

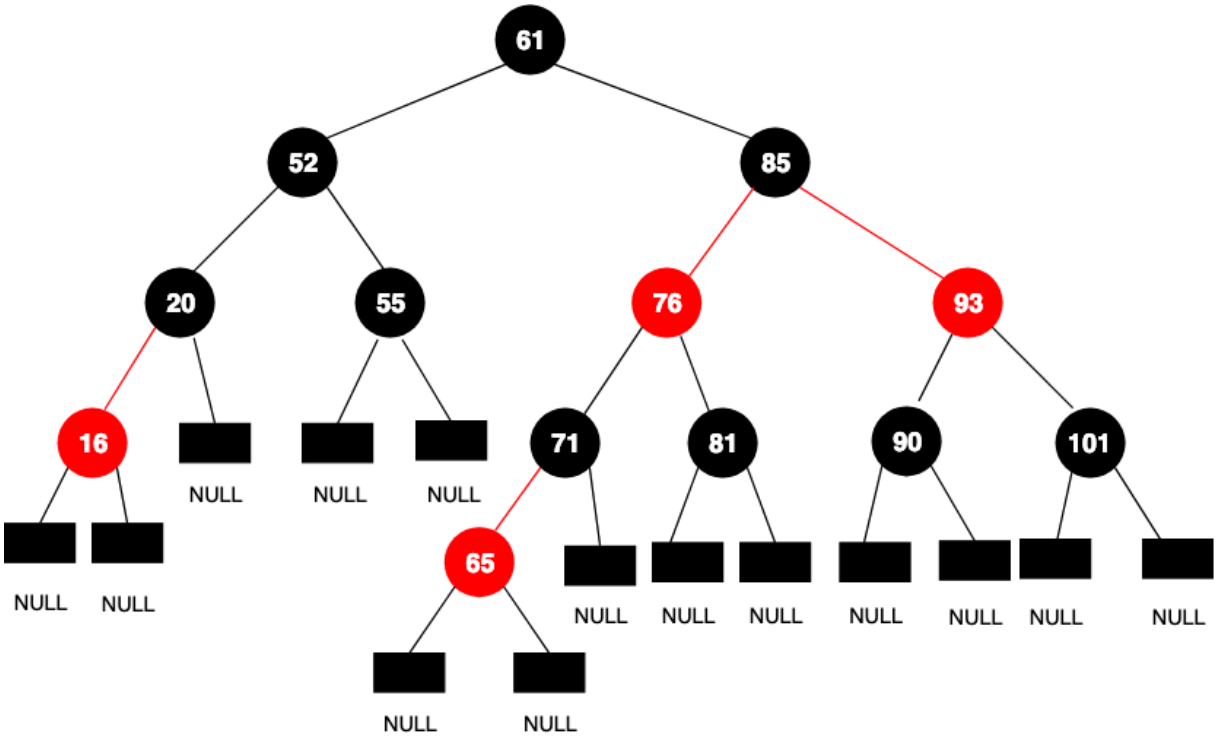
Struktura: Red-Black tree

Je podobný Balanced Binary Search Tree, takže dělá rotace (root nezůstává stejný)

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)

log_2 (n) protože při přidávání, hledání neprojde celou kolekci, hled na rootu jednu větev, tím, že se vydá buď do prava nebo do leva a tak se to opakuje

ListDictionary

Není generický

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)

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

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

Hlavní rozdíl: Negenerický umožňuje získání prvků pomocí indexů

```
SortedList sl = new SortedList();
SortedList<int, string> sl2 = new SortedList<int, string>();
sl.Add(51,"nongeneric");
sl2.Add(12,"generic");
Console.WriteLine(sl.GetByIndex(0));
Console.WriteLine(sl2[12]);
```

Struktura: 2 dynamická 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)

ConstainValue() – O(n)

ConstainKey() – O(log n)

SorterList

Negenerický

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

Výhoda je možnost získávání hodnot pomocí indexu

GetByIndex – O(1)

Metody: Clear(), Clone(),GetByIndex(int), IndexOfValue(int), IndexOfKey(int), RemoveAt(int), SetByIndex(int, value)

Podtržený jsou metody pracující s indexem

Vlastnosti: Capacity - Získá nebo nastaví kapacitu SortedList objektu.

SortedList<Key,Value>

Generický

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

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

ConcurrentDictionary

Generický

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

= thread-safe

Interně spravuje zamykání a poskytuje snadné rozhraní pro přidávání / aktualizaci položek
Procházení pomocí KeyValuePair<Key,Value>

Prostě jako dictionary jen thread-safe

Procházení

Generické

pomocí KeyValuePair<key,value>

```
foreach (var item in sl2)
{
    Console.WriteLine(item.Key+" "+item.Value);
}
```

```
foreach (KeyValuePair<int, string> x in dic)
{
    Console.WriteLine(x.Key);
    Console.WriteLine(x.Value);
}
```

Negenerické

```
foreach (DictionaryEntry item in ld)
{
    Console.WriteLine(item.Key +   " "+item.Value);
}
```

Var neumí DictionaryEntry

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.

Struktura

Je na haldě o velikosti 4,8,16,…

Získání pomocí indexu

Ukazatel na pole ukazuje na index 0, číslo napsané do [] řekne o kolik se má ukazatel posunout

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

procházení

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

Třída Array(C#) Metody pro práci s polem

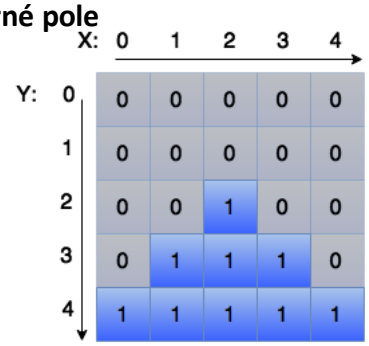
C#

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

```
int[] pole = new int[2];
pole[0] = 0;
pole[1] = 1;
Array.Resize(ref pole, 10);
pole[2] = 2; //bez změny by byla vyhozena IndexOutOfRangeException
pole[3] = 58;
foreach (var item in pole)
{
    Console.WriteLine(item);
}
```

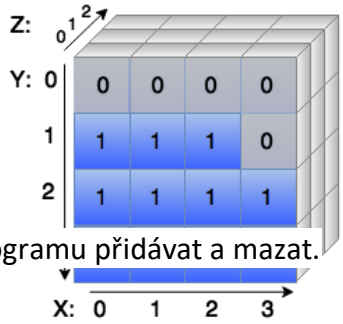
Dvourozměrné pole

index je složen z X a Y



více rozměrná pole

index je složen z X a Y a Z



Seznamy

Seznamy jsou kolekce, které umožňují prvky za běhu programu přidávat a mazat.

List

Generická kolekce

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), pokud je potřeba zvětšit pole kvůli kapacitě, tak se vytvoří nové větší pole, do kterého se překopíruje staré a přidá se nový prvek – O(n)

Remove() - O(n)

List[key] - O(1)

Insert() – O(n)

RemoveAt() – O(1)

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

ArrayList

Negenerická kolekce

Podobný listu.

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

ArrayList nemůže používat Linq na rozdíl od listu

Collection

Generická

Oproti listu nemá pevné metody a ty se dají přepsat.

ObservableCollection

generická

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

= obsahuje eventy

Hash

Hashtable

negenerická

Skoro stejný tejný jako dictionary

Key nemůže být null, ale value může

Key a value můžou být jakéhokoliv datového typu

Oproti dictionary jsou všechny jeho value objekty.

Asi tak hlavní rozdíl při programování je, že pokud v dictionary se pokusím získat hodnotu pomocí klíče, který tam neexistuje, tak dostanu error, proto se používá před získáním if s Constains nebo TryGetValue. V Hashtablu při stejným pokusu dostanu null.

Nenechává klíče ve stejným pořadí podle vkládání

Hashtable obsahuje vlastnost jestli byl použít jako thread safe.

Při přidávání z více vláken zamyká přidávání, ale není to na stejné úrovni jako thread safe dictionary.

Získání value z hashtablu je vrátí objekt, proto se pokeždé dělá unboxing

Je pomalejší protože dělá boxing a unboxing (musí protože value jsou objekty), dictionary to nedělá.

Boxing - object o = i

Unboxing - int j = (int)o

HashSet

Generická

Tato kolekce je množina, která nemá duplicitní prvky.

Pokud bude pokus o vložení prvku, jehož hashcode je už v kolekci, tak prvek nebude vložen, ale nebude vyvolána exception.

Objekty by měly mít metody GetHashCode() a Equals().

Třída poskytuje operace s vysokým výkonem.

Princip

Je podobný jako u dictionary obsahuje buckety a jeden bucket může mít i více hodnot

Ale bucket obsahuje (hashcode, next, value), neobsahuje key

Prvky nejsou seřazené, ani je nelze seřadit pomocí nějaké metody

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

Struktura: pole bucketů (zvané hashtable), které uchovává prvky

Zachované pořadí: ano

Označení místa uložení: hash

Dvě stejné hodnoty: ne

Add() – O(1), s kolizí v poli bucketů nebo při zvýšení kapacity O(n)

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

get - O(1) s kolizí O(n)

Operace HashSet –	Matematický ekvivalent
UnionWith	Sjednocení nebo nastavení sčítání
IntersectWith	Průnik
ExceptWith	Nastavit odčítání
SymmetricExceptWith	Symetrický rozdíl

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

Souhrn

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