Zadání 10. cvičení – Funkcionální programování v C#

V souboru **`10-cviceni.zip`** naleznete vytvořený základ pro následující úkoly. Ty vypracujte.

Úkoly 1 – 3 jsou variantou problémů, které jsme řešili v přednášce. Po jejím shlédnutí by měly být snadno zvládnutelné. ¨

Úkol 4 je mnohem těžší a vyžaduje více času.

Úkol 1 – Fronta

Implementujte datovou strukturu reprezentující frontu. Zkuste vymyslet řešení, kde fronta bude *immutable* datová struktura, podobně jako jsme měli zásobník na přednášce.

Netriviální rozšíření (pokud by jste už opravdu neměli co dělat 😊): Dále zkuste navrhnout řešení, kde operace vkládání a výběru dat mají konstantní složitost, jinými slovy, počet kroků při výběru a vložení nového prvku není závislý na počtů prvků, které jsou ve frontě aktuálně uloženy.

Úkol 2 – Monáda Identity

Jednou ze základních monád v jazyce Haskell je identita. Tato monáda v principu vůbec nic neděla, je obalí konkrétní hodnotu. Je ale důležitá v dalších rozšířeních, například pokud chceme realizovat přechody mezi monádami (v Haskellu jsou pod pojmem *state transformers*).

Řekněme že máme jednoduchou třídu:

```
public class Identity<A> : IMonad<A>
    {
        public A Value { get; init; }
    }
```

1. **Implmentujte rozhraní IMonad (definované na přednášce, je součástí připraveného archivu).**
2. **S pomocí metod rozhraní IMonad definujte metody Select a SelectMany aby bylo možné monádu Identity použít v *query syntax* jazyka C#.**

Výsledkem by mělo být to, že připravený test ve třídě Identity bude funkční.

Úkol 3 – Rozšíření Reader

V přednášce byla ukázána monáda State, která zaobalí stavový výpočet. Jednou z variant této monády v jazyce Haskell je monáda Read. Ta má podobně jako State nějaký vnitřní stav, ten ale nemodifikuje a jen z něj čte. Je možné ji vnímat tak, že zapouzdřuje nějakou konfiguraci.

Zatímco State bychom reprezentovali delegátem:

```
public delegate (TState State, T Value) State<TState, T>(TState state);
```

Tuto novou monádu Read bychom definovali jako:

```
public delegate T Reader<in TEnvironment, out T>(TEnvironment environment);
```

Rozdíle tedy je, že stav není modifikován.

Implementujte rozšiřující metody **Select** a **SelectMany** tak, abychom mohli Read používat v rámci query syntax.

Po implementaci by měl fungovat připravený test ve třídě `ReaderExtensions`.

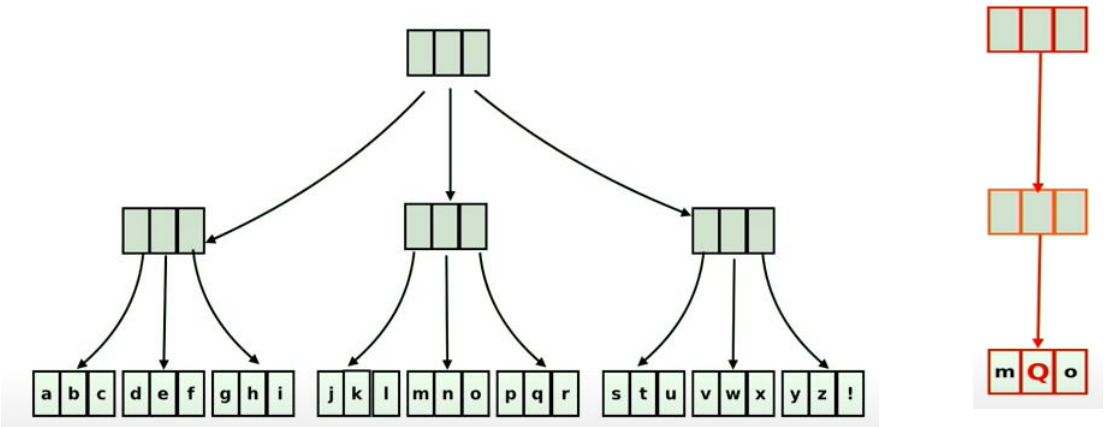
Úkol 4 – Immutable Array

Immutable datovým strukturám se taky někdy říká persistentní (https://en.wikipedia.org/wiki/Persistent_data_structure#Path_Copying, https://en.wikipedia.org/wiki/Persistent_array) .

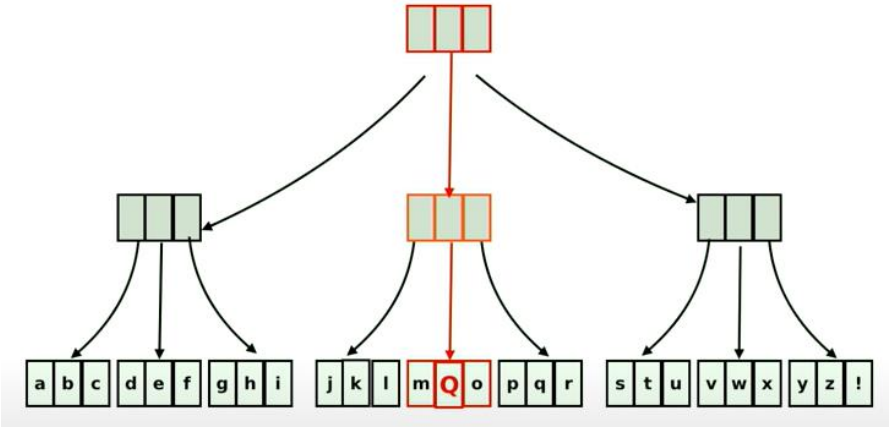
Jedna z možností, jak implementovat pole, které má tyto vlastnosti, je reprezentovat pole jako strom, kde, pokud chci změnit prvek v poli, vybuduji nový strom, kde maximálně využiji již existujících uzlů. Vlastně vybuduji znovu pouze cestu k měněnému uzlu.

Pěkně je to vysvětleno například tady: https://youtu.be/0if71HOyVjY?t=1370 (čas 23:00).

Řekněme, že v našem řešení budeme používat trojice. V budovaném stromě budou větve tohoto stromu sloužit pro navigaci, listy budou sloužit pro uložení dat. Všechny větve ve stromě budou mít stejnou hloubku. Toto je jednodušší na implementaci, ale strom bude mít teoreticky skoro 3 * více elementů, než je nutné, pro uložení požadovaného počtu prvků, protože velikost roste po násobcích tří. Ale s tím se smíříme, ono to bude těžké naprogramovat i tak.



Nyní, pokud chci změnit nějakou hodnotu, například Q v našem ukázkovém stromě, pak vybuduji znovu celou cestu od kořeně až k poslední trojici hodnot. Tu konkrétní trojici budu muset udělat celou znovu (pokud chci zachovat původní pole), ale to není takový problém. Pro všechny ostatní, referenční, uzly na této cestě, ale můžu *využít již existující uzly*! Protože data vlastně nemůžu měnit, nic tím nepokazím (sdílení nevadí).



Pro usnadnění řešení je pro vás připravená abstraktní třída Triplet, reprezentující trojici a dvě konkrétní třídy ReferenceTriplet pro referenční větev stromu a ValueTriplet pro list stromu.

V hlavní třídě ImmutableArray byl připraven konstruktor, který udělá výše zmíněný strom. Kromě vlastního stromu při vytváření spočítá skutečnou délku uložených dat.

Vaším úkolem bude:

1. **Doplnit indexer, který vrátí n-tý prvek.**
2. **Implementovat metodu Set, která vrátí nové pole, kde bude podle mechanismu popsaného výše, nahrazen konkrétní prvek za nový.**
3. **Implementovat enumerátor, pro snadné procházení touto kolekcí.**

Všechny tyto body jsou obsaženy v ukázkové metodě Test. Po jejich splnění by tato metoda měla „fungovat“.