Tato část bakalářské práce se bude zabývat implementací datové struktury fronty, a to konkrétně její real-time implementací. Nejprve si řekneme, co to fronta vlastně je.

Fronta je datová struktura, která je určená pro ukládání a vybírání prvků v takovém pořadí, ve kterém prvek, který byl přidán nejdříve, bude také nejdříve vybrán. Toto dobře vystihuje anglická zkratka FIFO tedy first in, first out.

Definujme pro frontu následující operace:

1. Enqueue – přidání prvku na konec fronty

2. Dequeue – odebrání prvku ze začátku fronty

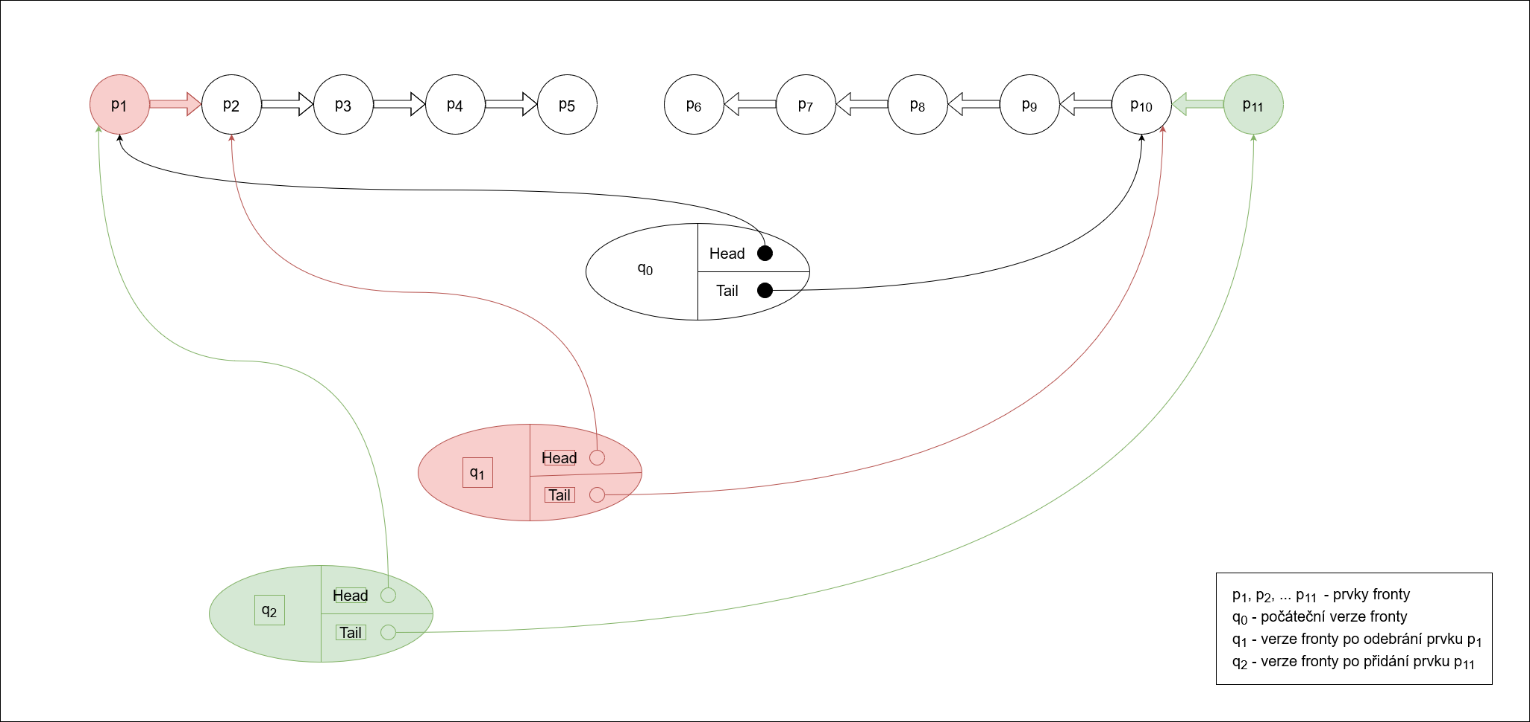
Pro implementaci real-time fronty potřebujeme, aby obě tyto operace probíhaly v konstantním čase. Pokud bychom reprezentovali frontu jedním seznamem, inhed narazíme na následující problém.

Odebrání prvků jsme sice schopni provést v konstatním čase, ale pro operaci přidání bychom museli nejprve projít celý seznam a až poté připojit nakonec seznamu nový prvek.

Diagram

Description automatically generated

Tento problém vyřešíme tak, že frontu budeme reprezentovat dvěma seznamy. Tyto seznamy budeme nazývat head a tail.



Díky této změně jsme již schopni provést jak operaci odebírání, tak přidání v konstantním čase.

Pro operaci dequeue budeme používat seznam head a pro operaci enqueue budeme používat seznam tail. Problém ale nastane v případě, pokud by se seznam head vyprázdnil a my bychom dále chtěli odebírat z fronty. Potom by bylo nutné přetočit prvky ze seznamu tail a až poté odebrat prvek z fronty. Toto přetáčení by zabralo n kroků, kde n je délka seznamu tail, což by ovšem porušilo podmínku provedení operací fronty v konstantním čase. Proto v implementaci dovolíme, aby byl head prázdný pouze v případě, že je prázdný i tail, tzn. když je fronta prázdná. Cílem bude tedy rozdělit přetáčení prvků z tail tak, že při operacích enqueue a dequeue provedeme i část tohoto přetáčení.

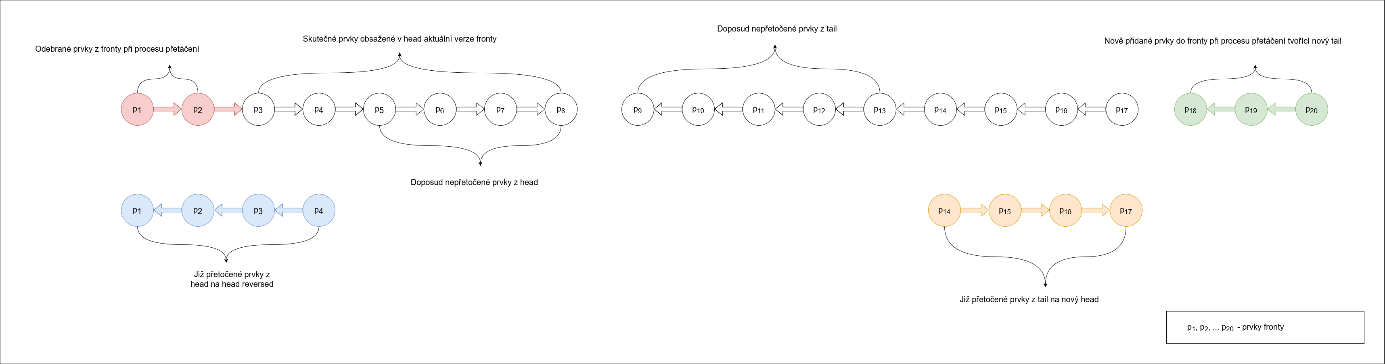
Přetáčení se bude skládat ze tří kroků:

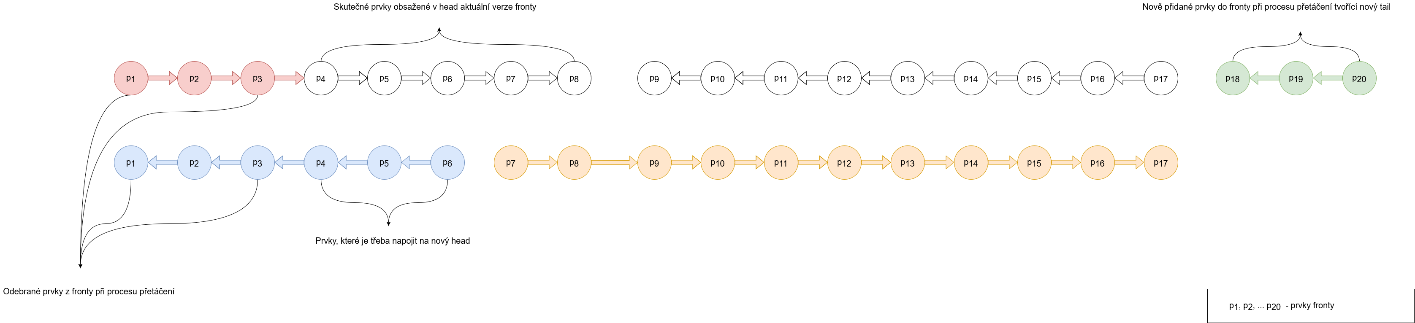
a) přetočením tail na nový seznam head

b) přetočením head na pomocný seznam head reversed

c) připojením přetočením head reversed na nový head

Všimněme si, že kroky a) a b) lze provádět souběžně. Během tohoto procesu se bude podoba fronty měnit, proto musíme zajistit, aby bylo možné zjistit, jak aktuální podobu fronty, tak aktuální podobu přetáčení. Pro přidávání prvků proto vytvoříme nový seznam tail. Při přetáčení se tedy prvky budou přidávat místo starého tailu na tail nově vytvořený. Odebírání bude řešeno pomocí dvou listů head. Zatímco jeden bude používán pro realizaci přetáčení, druhý bude sloužit pro odebírání prvků. Samozřejmě tím, že budou některé prvky při procesu přetáčení odebrány, ne všechny prvky se tak budou z head reversed na nový head napojovat. Proto budeme používat counter, který nám určí, kolik prvků je třeba při kroku c) připojit.





Dále potřebujeme určit, kdy proces přetáčení započne. S přetáčením začínáme v momentě, kdy bude velikost seznamu tail větší než velikost seznamu head. Dokud tedy fronta splňuje podmínku tail <= head, , k přetáčení nedochází a fronta je tvořena pouze seznamy head a tail. Po porušení této podmínky začínáme s procesem přetáčení.

Abychom zabránili tomu, že se head vyprázdní dříve než je proces přetáčení dokončen, je třeba vymyslet, jak rozložíme kroky přetáčení mezi n operací fronty (enqueue, dequeue). Dejme tomu, že při uvedení fronty do tohoto procesu obsahuje seznam head n prvků a seznam tail n + 1 prvků. Pro vykonání kroků a) a b) je tedy potřeba přetočení n + 1 prvků. Připojení head reversed na nový head docílíme v nejhorším případě přetočením n prvků. Musíme tedy rozdělit 2\*n + 1 kroků mezi n operací fronty. Toho docílíme tak, že při přechodu do procesu přetáčení provedeme jeden krok a při každé další operaci fronty provedeme další dva kroky přetáčení. Po dokončení celého procesu máme připravený nový seznam head a nový seznam tail. podmínka tail <= head je opět splněna, nahradíme tedy staré head a tail novými a můžeme se vrátit k reprezentaci fronty jen pomocí head a tail.

# Implementace

Jelikož fronta má různou vnitřní strukturu podle toho, zda se nachází v procesu přetáčení a také podle toho v jakém jeho kroku se nachází, rozdělíme frontu do stavů. Tyto stavy budou tři a jsou reprezentovány třídami QueueZero, QueueOne, QueueTwo.

# Element

Kod element

Pro potřeby implementace byla vytvořena třída Element, která představuje jeden prvek ve frontě.

Tato třída obsahuje value tedy hodnotu prvku a také next, což je ukazatel na další prvek.

Obrazek Element a seznam

# QueueZero



Tato třída je podobou fronty, která se nenachází v procesu přetáčení.

V tomto stavu obsahuje fronta head a tail, tedy dva seznamy tvořící vlastní frontu. Dále lendiff, který vyjadřuje rozdíl mezi velikostí seznamu head a velikostí seznamu tail a který nám bude určovat jak dlouho fronta v tomto stavu zůstane. Díky lendiff nemusíme udržovat velikosti obou seznamů. Podmínku ..... nahradíme podmínkou, kdy lendiff = 0. Dále třída QueueZero obsahuje state, který číselně vyjadřuje stav fronty. Vlastnost state bude také součástí dalších tříd a je použita k rozpoznání, ve kterém stavu se fronta nachází.

Operace enqueue vytvoří nový tail napojením nového prvku na starý tail a vrátí novou verzi fronty. Na začátku se také zkontroluje lendiff a v případě, že je roven nule, se vytvoří fronta třídy QueueOne a operace přidání prvku s value se zavolá až na tento vytvořený objekt.



Operace dequeu vytvoří novou verzi fronty pomyslným odebráním prvku ze seznamu head a vráti ji společně s hodnotou odebraného prvku. Obdobně se na začátku zkontroluje lendiff a případně se operace odebírání provede až v dalším stavu. Také se zkontroluje, zda již fronta není prázdná.



Můžeme si všimnout, že obě operace, jak enqueue, tak dequeue snižují hodnotu lendiff.

# QueueOne.



V tomto stavu začneme s přetáčením prvků. Budou se provádět následující kroky přetáčení:

a) přetáčením prvků z head na head\_reversed

b) přetáčením prvků z tail na n\_head

Dokud se v head nebo tail vyskytují nějaké prvky, fronta zůstane v prvním stavu.

Operace enqueue probíhá následovně.

V první části se zkontroluje podmínka, zda lendiff = 0. Pokud ano, víme, že nastal přechod ze stavu 0 do stavu 1, tedy že funkce byla zavolána z enqueue\_zero. Takže je potřeba po vložení prvku provést jeden krok přetáčení. Krok a) poté provedeme běžným způsobem. Krok přetáčení b) je zjednodušen. Z nově vkládaného prvku rovnou vytvoříme n\_head. Poté vrátíme novou verzi fronty ve stavu 1.

Pokud se nejedná o přechod mezi stavy 0 a 1, postupuje se následnovně, nejprve se na n\_tail napojí nový prvek a sníží se hodnota lendiff. Dále se provedou dvakrát kroky přetáčení a) a b).

V případě, že se během provedení vyprázní oba seznamy head a tail, přejde fronta do stavu 2 a my tedy vrátíme jako novou verzi objekt QueueTwo. Pokud ne, pouze vrátíme novou verzi fronty ve stavu 1.



Operace dequeue. Opět nejprve zjistíme, zda-li se jedná o přechod ze stavu 0 do stavu 1. Pokud ano, provedeme počáteční krok přetáčení. U kroku a) si musíme dát pozor na to, že je potřeba z fronty nejprve odebrat jeden prvek. Tudíž s přetáčením začneme až u následujícího prvku seznamu head tedy head.next. Krok b) potom provedeme běžným způsobem. Pokud se nejedná o přechod mezi stavy, postupuje se následovně. Nejprve vytvoříme nový head\_origin a snížíme hodnotu delta\_for\_copy. Následně provedeme kroky přetáčení a vrátíme novou verzi fronty stejně tak, jako u operace pro přidávání prvků.



# QueueTwo.



V tomto stavu dochází k poslednímu kroku procesu přetáčení, což je napojení head\_reversed na n\_head, které byly oba vytvořeny v předchozím stavu. Hodnota delta\_for\_copy, určuje kolik prvků je ještě potřeba napojit.

Přidávání prvků je řešeno jednoduše pouze připojením nového prvku na n\_tail a opět zvýšením hodnoty lendiff.



Při odebírání prvků se pouze sníží hodnota delta\_for\_copy a použijeme nový head\_origin přes ukazatel next.

Když je proces napojení dokončen, tzn. delta\_for\_copy je rovna 0, seznamy n\_head a n\_tail

jsou zcela vytvořené a mohou nahradit původní head a tail. Fronta přejde zpátky do stavu 0.

# Uživatelské rozhraní

Pro uživatele fronty byly vytvořeny následující funkce:

## Enqueue(q, value)

Tato funkce vytvoří novou verzi fronty s přidaným prvkem s hodnotou value

Vstupy:

q – objekt třídy QueueZero, QueueOne nebo QueueTwo

value – hodnota prvku

Výstupy:

objekt třídy QueueZero QueueOne nebo QueueTwo

## Dequeue(q)

Tato funkce vytvoří novou verzi fronty s odebraným prvním prvkem

Vstupy:

q – objekt třídy QueueZero, QueueOne nebo QueueTwo

Výstupy:

value -Hodnota prvku

objekt třídy QueueZero, QueueOne nebo QueueTwo

## Get\_empty\_queue()

Tato funkce vytvoří prázdnou frontu a vrátí ji uživateli.

Vstupy:

Výstupy:

objekt třídy QueueZero (prázdná fronta)