Fronta a zásobník

Tyto datové struktury si lze vyzkoušet na této stránce.

Fronta i zásobník jsou 2 principiálně velmi podobné způsoby dočasného uložení dat v rámci programu. Používají se v případech, kdy je třeba postupně projít všechny prvky dat, a to právě jednou.

Fronta a zásobník jsou centrální datové struktury algoritmů vyhledávání do šířky (BFS) a vyhledávání do hloubky (DFS). Jediným rozdílem mezi těmito algoritmy je použitá datová struktura.

Základní funkce

- **push(objekt)** – vkládá nová data do řady
- **pop()** nebo také **pull()** – získá data, která jsou právě na řadě

Fronta (queue)

Funguje na principu **FIFO** (first in first out), to znamená, že si frontu můžeme představit třeba jako frontu na úřadu práce. Lidé (data) se řadí po příchodu do fronty a ve stejném pořadí se i dostanou na řadu (data jsou tedy dříve vrácena).

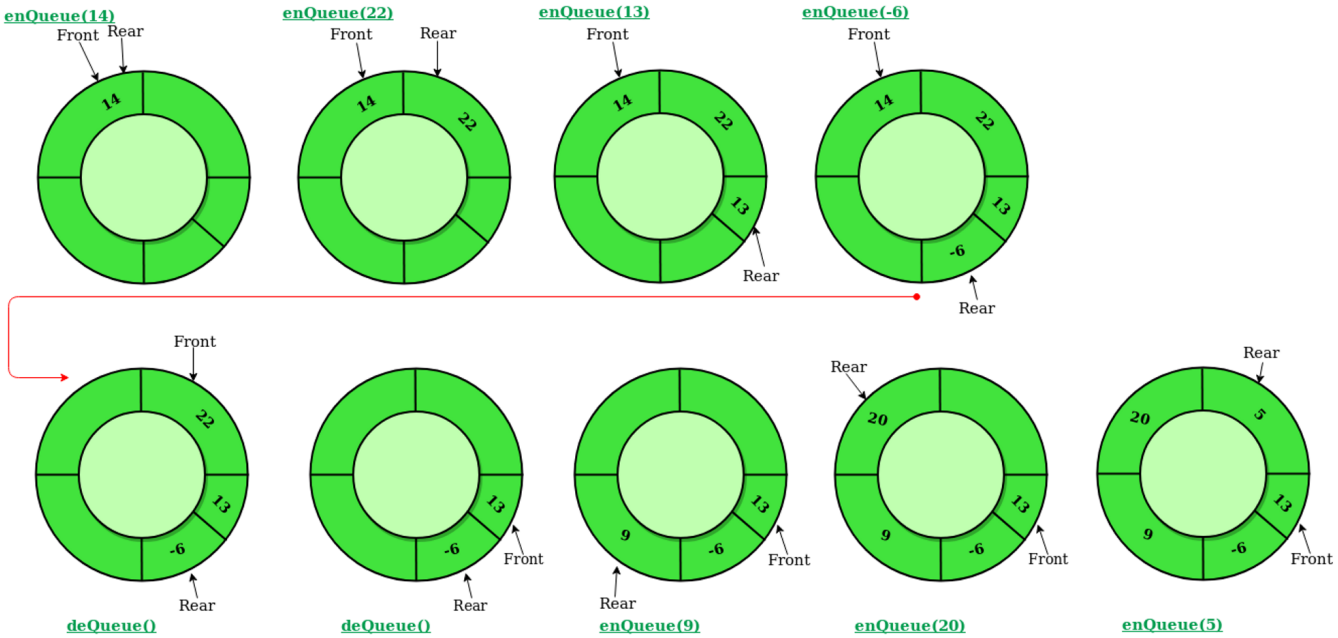
Jsou potřeba dva ukazatele: front (zde jsou odebírána data) a rear (zde jsou data vkládána)

Metody Implementace

Cyklické pole

Při volání funkce pop() opět získáme data, ale tentokrát hodnotu rear ukazatele ponecháváme a naopak se zvyšuje hodnota ukazatele front. Celá fronta je navíc skutečně cyklická, takže v situaci kdy se uvolní prostor z původního „začátku“ fronty, může být znova zaplněn na jejím konci.

viz. diagram:



Výhody:

- O(1) přečtení libovolné hodnoty ve frontě (ne pouze na začátku nebo konci)
- rychlejší než linked list
  - výhodnější pro CPU cache
  - menší „object overhead“
- umí využít informaci o maximální velikosti (viz kód)

Nevýhody:

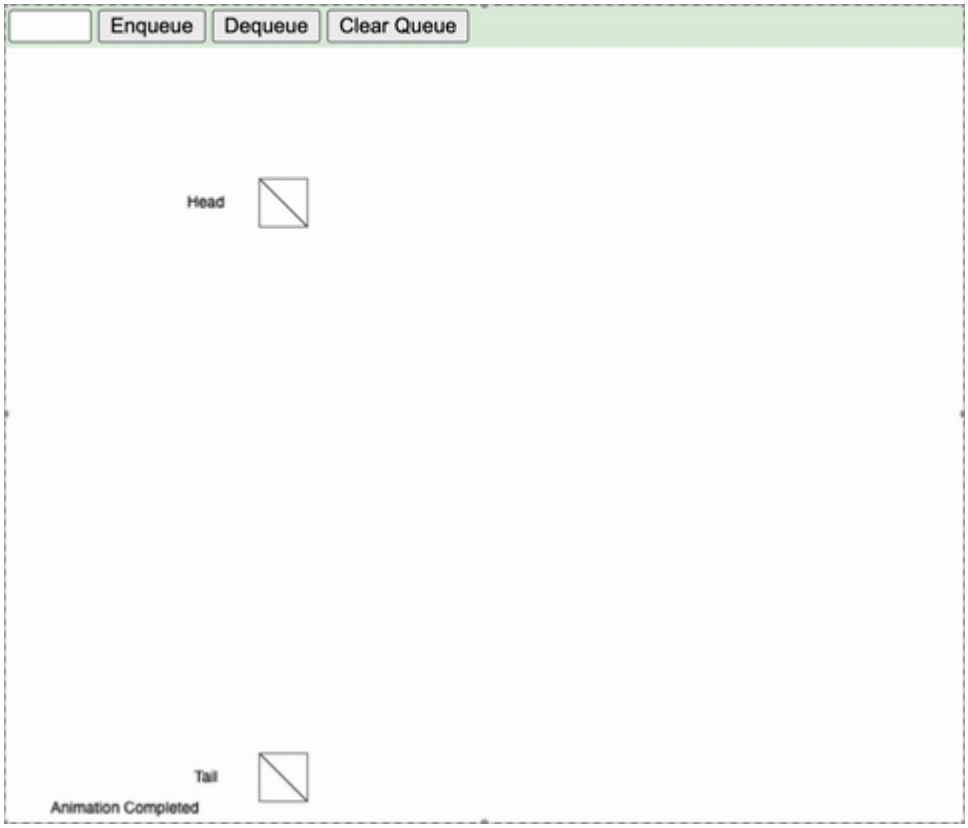
- O(1)* amortizovaná časová složitost přidání hodnoty
  - téměř vždy O(1)
  - O(n) v případě naplnění fronty - musíme celé pole zkopírovat do nového pole s větší velikostí (viz kód)
  - může být pomalejší než linked list, když musí často zvětšovat velikost pole
- neumí zmenšit svou velikost (může plýtvat pamětí)
- fronta je omezená velikostí ukazatelů - například v Javě má maximální velikost Integer.MAX_VALUE - 8 = 2 147 483 439

Implementací cyklického pole je například kolekce ArrayDeque v jazyce Java, tady je její zjednodušená verze:

```java
public class ArrayDeque<E> {

    transient Object[] elements;
    transient int head;
    transient int tail;

    public ArrayDeque() {
        elements = new Object[16 + 1];
    }

    public ArrayDeque(int numElements) {
        elements = new Object[numElements + 1];
    }

    static final int inc(int i, int modulus) {
        if (++i >= modulus) {
            i = 0; // wrap around indices to the start when they get too big
        }
        return i;
    }

    public void offer(E e) { // enqueue
        elements[tail] = e;
        tail = inc(tail, elements.length);
        if (head == tail) {
            grow(1);
        }
    }

    public E poll() { // dequeue
        E e = (E) elements[head];
        if (e != null) {
            elements[head] = null;
            head = inc(head, elements.length);
        }
        return e;
    }

    private void grow(int needed) {
        final int oldCapacity = elements.length;
        // Double capacity if small; else grow by 50%
        int jump = (oldCapacity < 64) ? (oldCapacity + 2) : (oldCapacity >> 1);
        int newCapacity = oldCapacity + Math.min(needed, jump);
        elements = Arrays.copyOf(elements, newCapacity);
        // Exceptionally, here tail == head needs to be disambiguated
        if (tail < head || (tail == head && elements[head] != null)) {
            // wrap around; slide first leg forward to end of array
            int newSpace = newCapacity - oldCapacity;
            System.arraycopy(elements, head, elements, head + newSpace, oldCapacity - head);
            Arrays.fill(elements, head, head + newSpace, null);
            head += newSpace;
        }
    }
}
```

Pomocí ukazatelů (Linked list)

Metodu řešení pomocí **ukazatelů** můžeme vidět na níže uvedeném diagramu. Výhoda tohoto řešení je zároveň i jeho nevýhodou – jsme už sice schopni přidávat do fronty příspěvky až do konce paměti, zároveň ale (při špatné implementaci) může dojít k **přehlcení nebo až přetečení paměti**, což může vést k nestabilitě systému.



Výhody:

- O(1) složitost přidání hodnoty
- neomezená velikost
- dynamicky se zvětšuje a zmenšuje - neplýtvá pamětí

Nevýhody:

- pomalejší než cyklické pole

Zde je zjednodušená verze kolekce LinkedList v jazyce Java. (V jazyce Java je LinkedList obousměrně vázaný seznam, ale pro jednoduchost je níže implementovaný jako jednosměrně vázaný seznam)

```java
public class LinkedList<E> {

    transient Node<E> first;
    transient Node<E> last;

    public LinkedList() {
    }

    public void enqueue(E e) {
        final Node<E> l = last;
        final Node<E> newNode = new Node<>(l, e, null);
        last = newNode;
        if (l == null) {
            first = newNode;
        } else {
            l.next = newNode;
        }
    }

    public E dequeue() {
        final E element = first.item;
        final Node<E> next = first.next;
        first = next;
        if (next == null) {
            last = null;
        }
        return element;
    }

    private static class Node<E> {
        E item;
        Node<E> next;

        Node(Node<E> prev, E element, Node<E> next) {
            this.item = element;
            this.next = next;
        }
    }
}
```

Zásobník (stack)

Funguje na principu **LIFO** (last in first out), to znamená, že funguje stejně **jako zásobník v samopalu**. Při nabíjení samopalu (vkládání dat) skládáme vždy náboje na spodek zásobníku a vršíme je na sebe, při výstřelu (výběru dat) se náboje berou svrchu – poslední vložený je tedy vystřelen jako první.

Bez zásobníku by v programování nefungovalo volání funkcí. Detailně je to popsáno v tomto videu

Metody Implementace

Pomocí statického pole

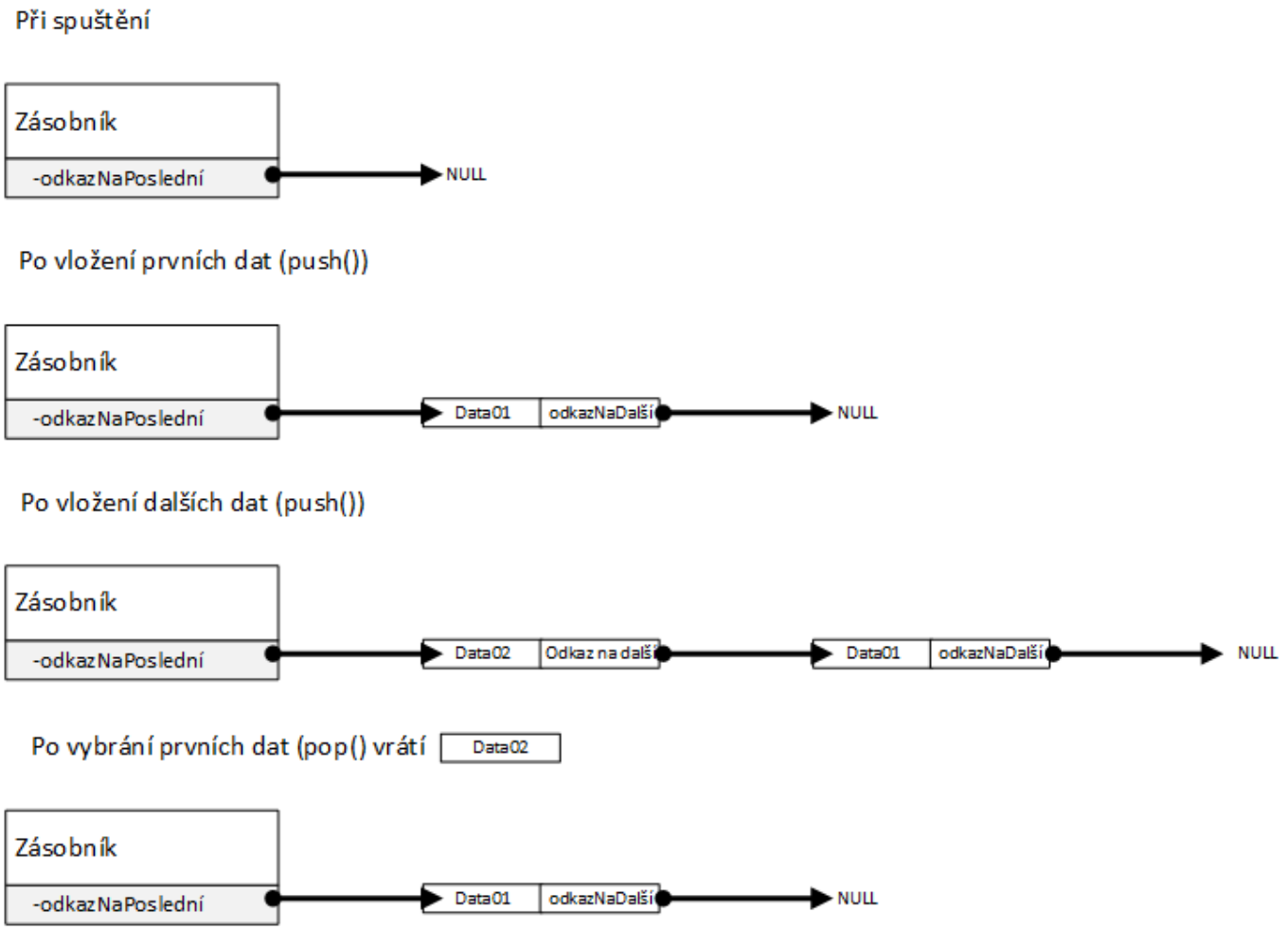
Řešení zásobníku pomocí statického pole je stejné jako u **fronty**, jediný rozdíl je v tom, že si v další proměnné musíme držet index posledních vložených dat a při zavolání funkce pop() vezmeme data z indexu posledních vložených data a index snížíme o jedna.

Výhody a nevýhody jsou stejné jako u fronty.

Pomocí ukazatelů

Řešení pomocí ukazatelů je také podobné jako u **fronty** (viz diagram níže).

Výhody a nevýhodu jsou opět naprosto stejné jako u fronty.



From:
http://wiki.gml.cz/ - **GMLWiki**

Permanent link:
http://wiki.gml.cz/informatika:maturita:21a

Last update: **17. 02. 2026, 21.03**