FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Projekt

Vizualizace vybraných abstraktních datových typů

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 1802T007 – Informační technologie
Autor práce: **Václav Kožený**
Vedoucí práce: Ing. Igor Kopetschke

Liberec 2025

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2024/2025**

ZADÁNÍ BAKALÁŘSKÉHO PROJEKTU

Jméno a příjmení:	**Václav Kožený**
Název práce:	**Vizualizace vybraných abstraktních datových typů**
Zadávající katedra:	**Ústav nových technologií a aplikované informatiky**
Vedoucí práce:	**Ing. Igor Kopetschke**
Rozsah práce:	**15—20 stran**

Zásady pro vypracování:

1. Vypracujte rešerši případných již existujících řešení.
2. Na základě rešerše zdůvodněte vývoj vlastní aplikace.
3. Navrhněte a následně realizujte nástroj pro popis a vizualizaci vybraných abstraktních datových typů formou webové aplikace na platformě jazyka JavaScript.
4. Poskytněte hotové řešení skupině testovacích uživatelů a získejte od nich zpětnou vazbu.
5. Získanou zpětnou vazbu vyhodnoťte a navrhněte případné změny a vylepšení.

Seznam odborné literatury:

[1] HAVERBEKE, Marijn. Eloquent Javascript: a modern introduction to programming. 4th edition. San Francisco: No Starch Press, [2025]. ISBN 978-1-7185-0410-3.

[2] JavaScript. Online. 2025. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript. [cit. 2025-04-29].

[3] Abstraktní datový typ. Online. Wikipedia. 2025. Dostupné z: https://cs.wikipedia.org/wiki/Abstraktn%C3%AD_datov%C3%BD_typ. [cit. 2025-04-29].

V Liberci dne .....

.....
Ing. Igor Kopetschke

Prohlášení

Prohlašuji, že svůj projekt jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mého projektu a konzultantem.
Jsem si vědom toho, že na můj projekt se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.
Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mého projektu pro vnitřní potřebu Technické univerzity v Liberci.
Užiji-li projekt nebo poskytnu-li licenci k jeho využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.
Beru na vědomí, že můj projekt bude zveřejněn Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.
Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

11. 5. 2025 Václav Kožený

Vizualizace vybraných abstraktních datových typů

Abstrakt

Tato práce se zabývá návrhem a implementací webové aplikace pro popis a vizualizaci abstraktních datových typů (ADT), mezi které patří například fronta, zásobník, seznam, množina nebo strom. Cílem je přiblížit strukturu, principy a základní operace těchto datových typů formou interaktivní prezentace s důrazem na přehlednost a srozumitelnost. Každý typ je doplněn popisem, ukázkou implementace, složitostí operací a vizualizací chování v reálném čase.

Klíčová slova: ADT, JS, Zásobník, Fronta

Abstract

This thesis deals with the design and implementation of a web application for the description and visualization of abstract data types (ADT), which include, for example, a queue, a stack, a list, a set, or a tree. The aim is to present the structure, principles and basic operations of these data types in an interactive presentation with an emphasis on clarity and readability. Each type is accompanied by a description, an example implementation, the complexity of the operations and a real-time visualisation of the behaviour.

Keywords: ADT, JS, Stack, Queue

Poděkování

Rád bych poděkoval vedoucímu za to, že si mě vzal pod svá křídla a poskytl mi cenné rady během práce na tomto projektu. Dále děkuji svým spolužákům za jejich zpětnou vazbu k aplikaci.

Obsah

Seznam obrázků

Seznam zkratek

ADT	Abstraktní datový typ
JS	JavaScript
LIFO	Last in, First Out
FIFO	First In, First Out
BFS	Breadth-first search
BST	Binary search tree
AVL	Adelson-Velsky-Landis
DSV	Data Structure Visualizations
DOM	Document Object Model
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
SCSS	Sassy CSS
SVG	Scalable Vector Graphics
ALD	Algoritmy a datové struktury

1 Úvod

Abstraktní datové typy, jako je zásobník, strom nebo seznam, jsou základem velkého množství algoritmů a dalších odvozených datových struktur. Jejich pochopení je proto pro vývojáře velmi důležité, protože mohou být použity pro zefektivnění jejich softwaru. Projekt „Vizualizace vybraných abstraktních datových typů“ si klade za cíl vytvořit interaktivní webovou aplikaci, která by koncepty abstraktních typů zprostředkovala srozumitelnou cestou. Hlavním cílem je napomoci porozumění principům pomocí názorných vizualizací.

2 Datové typy

Tato kapitola vychází převážně z přednášek předmětu ALD. [5] [6]

Datové typy definují množinu hodnot a operací, které lze s těmito hodnotami provádět. Jsou základem každého programovacího jazyka. V této práci se zaměřujeme zejména na abstraktní datové typy (ADT), které umožňují pracovat se složitějšími strukturami dat.

2.1 Základní typy

Základní (primitivní) datové typy, jako jsou celá čísla (int), reálná čísla (float), znaky (char) či logické hodnoty (bool), jsou základem pro tvorbu složitějších struktur. Tyto typy jsou implementovány přímo jazykem a jsou použity při definici abstraktních struktur, jako je zásobník (stack) nebo fronta (queue).

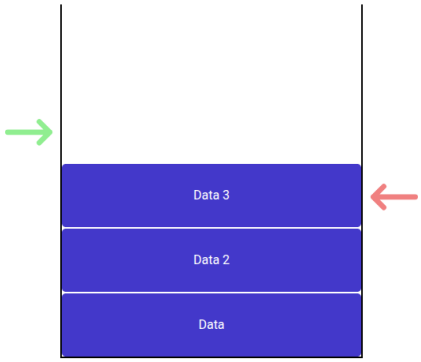
2.2 Abstraktní datový typ

Abstraktní datové typy (ADT) rozšiřují možnosti práce s daty nad rámec primitivních typů. Definují rozhraní a chování datových struktur bez ohledu na konkrétní implementaci. Jsou běžně podporovány většinou moderních programovacích jazyků a výrazně přispívají ke srozumitelnosti a udržovatelnosti kódu.

Každý ADT definuje množinu operací, typicky vytvoření kontejneru, vložení a odebrání prvku.

2.2.1 Zásobník

Jednoduchý abstraktní typ, který pracuje na principu LIFO, tedy poslední vložený prvek se odebere jako první.



Obrázek 2.1: Zásobník

Základní operace se zásobníkem jsou **create()**, **push(item)** který vkládá na vrchol zásobníku a **pop()**, který odebírá z vrcholu zásobníku. Doplňujícími operacemi mohou být **peak()** - přečtení vrcholu zásobníku bez odebrání a **isEmpty()**, která kontroluje obsazenost zásobníku.

Použití může být krátkodobé odložení informací, které potřebujeme odebírat v opačném pořadí, například když proces volá metody, odloží kontext do zásobníku a následně ho opět získá. Nebo třeba při rekurzivním volání funkcí, kdy se v opačném pořadí navracíme zpátky do místa prvního volání.

Zásobník může být implementován pomocí pole (s ukazatelem na vrchol) a nebo spojového seznamu.

2.2.2 Fronta

Fronta je abstraktní datový typ založený na principu FIFO - první vložený prvek je první odebrán.

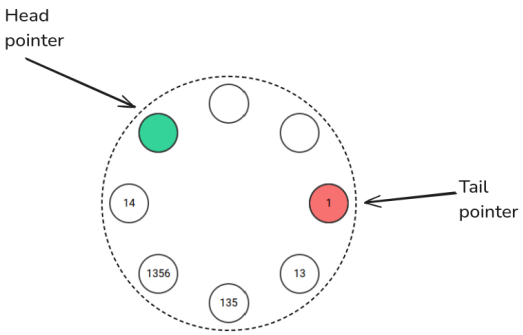
Základní operace s frontou jsou **create()**, **enqueue(item)**, která vkládá prvek na konec fronty a **dequeue()**, ta naopak odebírá ze začátku fronty.

Fronta může být realizována pomocí pole nebo spojového seznamu. Patří mezi klíčové struktury v informatice, využívá se například v plánování procesů, zpracování událostí nebo v algoritmech, jako je prohledávání do šířky (BFS).

2.2.3 Kruhový buffer

Kruhový buffer je rozšířením běžné fronty, kde jsou prvky organizovány do kruhu.

Většinou bývá implementován pomocí pole a dvou ukazatelů - jednoho pro pozici zápisu a druhého pro pozici čtení. Pokud zápisový ukazatel dosáhne konce pole, pokračuje dále od začátku. Tím může dojít k přepsání starších dat, pokud nebyla dosud přečtena. Díky tomu, že čtení i zápis probíhá přímo na pozicích ukazatelů, mají obě operace konstantní časovou složitost.



Obrázek 2.2: Kruhový buffer

Hlavní operace jsou **write(item)**, zápis na pozici zápisového ukazatele a **read()**, přečtení z ukazatele.

Kruhový buffer se často využívá jako vyrovnávací paměť, například při zpracování datových toků v reálném čase.

2.2.4 Graf

Abstraktní datový typ **graf** reprezentuje množinu uzlů (vrcholů) a množinu hran, které tyto uzly spojují. Hrany mohou být orientované nebo neorientované. Graf může být sestaven pomocí seznamu sousedů, matice sousednosti nebo seznamu hran. Graf se používá k modelování vztahů mezi objekty, například v dopravních sítích nebo při vyhledávání cest. [8]

2.2.5 Množina

Množina je abstraktní datový typ vycházející z matematické teorie množin. Zajišťuje jedinečnost prvků a nepodporuje žádné specifické pořadí. Každý prvek se v množině vyskytuje nejvýše jednou.

Základní operace s množinou jsou **insert(value)**, která přidává do množiny, **remove(value)** naopak z množiny odebírá a **contains(value)**. Ta vrací, jestli se prvek v množině nachází či nikoli.

Jelikož má matematický základ, podporuje množinové operace **union(A, B)** – sjednocení množin, **intersection(A, B)**, průnik dvou množin a **difference(A, B)**, rozdíl množin (prvky v A, které nejsou v B)

2.2.6 Seznam

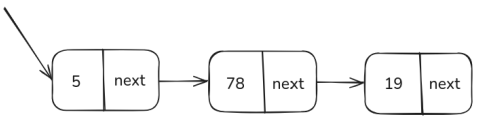
Seznam je lineární datová struktura, která uchovává prvky v definovaném pořadí. Na rozdíl od pole může být dynamický – tedy měnit svou velikost za běhu programu.

Mezi základní operace patří vytvoření a inicializace seznamu, vkládání prvků na zvolené pozice, jejich mazání a přístup k hodnotám na daných indexech. Dále může seznam podporovat operace jako hledání konkrétní hodnoty, kontrolu prázdnosti a získání počtu prvků.

Seznam lze implementovat dvěma hlavními způsoby, pomocí **pole**, nebo **spojového seznamu**.

Implementace **polem** je postavena nad polem fixní velikosti, tudíž umožňuje přístup v konstantním čase. Pokud dojde k naplnění pole, vytvoří se pole o dvojnásobné velikosti a prvky se překopírují.

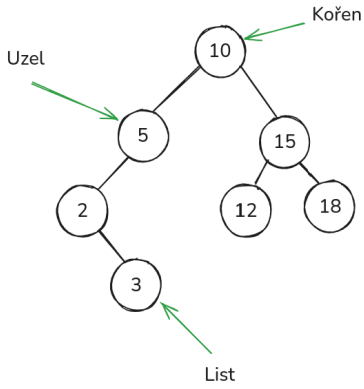
Implementace **spojovým seznamem** je postavena na principu ukazatelů. Každý prvek má svoji hodnotu a ukazatele na následující (následující a předchozí) prvek. Přístup k položkám má lineární složitost. Přidávání a odebírání je jednoduché, pouze změníme ukazatel u prvků. Tímto je také eliminován problém s fixní velikostí pole.



Obrázek 2.3: Implementace seznamu pomocí spojového seznamu

2.2.7 Strom (Tree)

Strom je hierarchická datová struktura složená z uzlů (**nodes**), kde každý uzel (kromě kořene) má právě jednoho rodiče. Počet potomků určuje **n-aritu** stromu, hloubka udává počet úrovní a vyváženost ovlivňuje efektivitu operací.



Obrázek 2.4: Binární vyhledávací strom

Mezi základní typy stromů patří **binární strom**, kde každý uzel má nejvýše dva potomky, a **binární vyhledávací strom** (BST), který zachovává uspořádání levých potomků s menší hodnotou a pravých s větší hodnotou. **AVL strom** je samovyvažující varianta BST, kde rozdíl hloubek podstromů nepřesahuje 1. Obecnější **n-ární stromy** umožňují libovolný počet potomků.

Hlavní operace zahrnují **insert(item)**, vložení podle pravidel stromu, odstranění prvku s přizpůsobením struktury pravidlem, **find(item)**, které vyhledá zadaný prvek a procházení stromu. Existují tři základní druhy procházení **preorder** kdy se prochází v pořadí rodič → levý podstrom → pravý podstrom. **Inorder** prochází strom v pořadí levý → rodič → pravý a **postorder**, levý → pravý → rodič.

Stromy nacházejí široké využití v informatice, například v databázových indexech, parserech nebo v organizaci hierarchických dat.

2.3 Existující řešení

2.3.1 Data Structure Visualizations

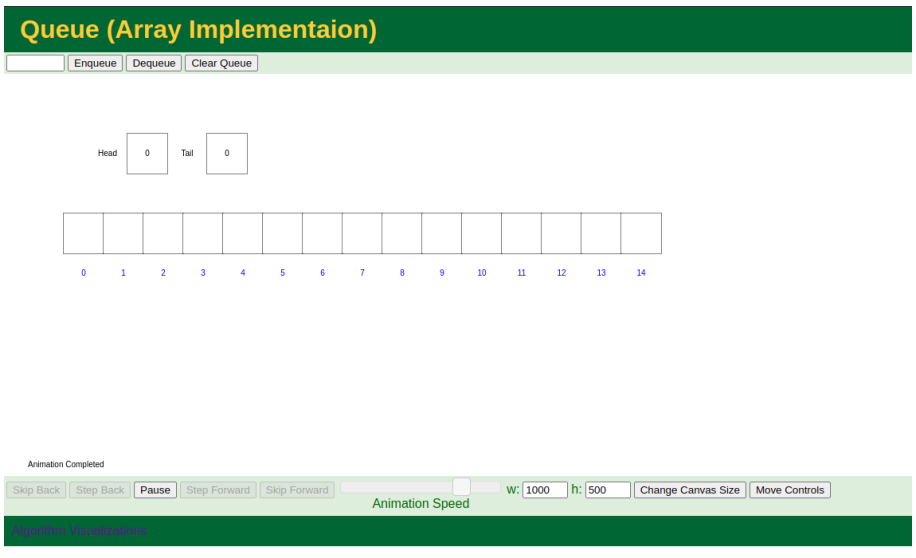
https://www.cs.usfca.edu/~galles/visualization/Algorithms.html

Aplikace pro vizualizaci algoritmů a datových struktur byla vyvinuta profesorem Davidem Gallesem z University of San Francisco. [3] Aplikace obsahuje velké množství datových struktur a algoritmů, které jsou rozděleny do několika kategorií:

- **Basic**, v této kategorii je vizualizace fronty a zásobníku pomocí pole a spojového seznamu.
- **Indexing** obsahuje stromové struktury a hashování.
- **Sorting** se zaměřuje na třídící algoritmy.
- V sekci **Graph Algorithms** se vyskytují algoritmy s grafy jako například hledání nejkatší cesty.
- **Recursion**, **Heap-like Structures** a další – pokročilejší témata včetně rekurze a haldových struktur.

Aplikace nabízí poměrně široké možnosti interakce:

- Uživatel si může zobrazit krok po kroku jednotlivé fáze algoritmu.
- Je umožněno zpomalit, nebo zrychlit animace dle potřeby.
- Lze se vracet o krok zpět a znovu sledovat konkrétní fáze.



Obrázek 2.5: Ukázka DSV [3]

Výhody:

- Široké pokrytí různých typů struktur a algoritmů.
- Webová dostupnost bez nutnosti instalace.

Nevýhody:

- Zastaralý design a uživatelské rozhraní, které nemusí být přívětivé pro současné uživatele.
- Absence české lokalizace nebo podrobnějších popisů algoritmů.

2.3.2 VisuAlgo

https://visualgo.net/en

Za touto aplikací stojí profesor Steven Halim a jeho tým z National University of Singapore, kteří postupem času vyvinuli webovou aplikaci primárně pro studenty tamní univerzity. [4]

Aplikace nabízí široké spektrum datových struktur a algoritmů. Na rozdíl od předchozí aplikace 2.3.1 zde není strukturované dělení do kategorií – hlavní důraz je kladen na grafové struktury a související algoritmy.
I u tohoto řešení jsou velké možnosti interakcí, jedná se o:

- Krokování jednotlivých operací, jak vpřed, tak i vzad
- Možnost zpomalení, nebo zrychlení podle potřeby

Výhody:

- Široká nabídka struktur a algoritmů
- Moderní design aplikace
- Výpis stavu při každém kroku, což napomáhá pochopení
- Ukázka kódu s vizuálním zvýrazněním aktuálního kroku
- Režim E-Lecture s podrobným popisem struktur a operací

Nevýhody:

- Méně intuitivní ovládání
- Absence české lokalizace

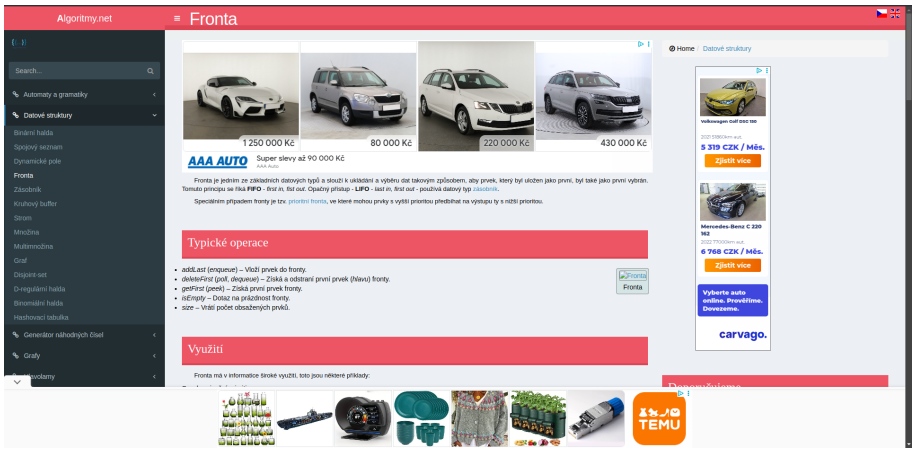
2.3.3 algoritmy.net

https://www.algoritmy.net/

Algoritmy.net je česká webová stránka zaměřená na algoritmy a datové struktury, jak napovídá už její název. Podle příspěvku na fóru Webtrh.cz za projektem původně stojí Pavel Mička.[7] Na samotné stránce je jako držitel autorských práv uveden Jan Neckář, avšak další informace o autorech či historii projektu zde chybí.[1]

Obsahově je stránka velmi rozsáhlá. Pokrývá širokou škálu témat — od základních datových struktur, přes grafové algoritmy a kryptografii až po třídicí algoritmy. Právě třídicí algoritmy jsou doplněny animovanou vizualizací, která napomáhá pochopení jejich fungování. Celkově lze obsah považovat za kvalitní výukový materiál.

Za hlavní nevýhodu považuji velké množství reklam, které mohou rušit uživatelský dojem. Působí to dojmem, že stránka byla odprodána a novému provozovateli slouží primárně jako reklamní prostor. Jako další nevýhodu bych uvedl nedostatek animovaných vizualizací.



Obrázek 2.6: Ukázka algoritmy.net [1]

2.3.4 Proč vlastní řešení?

Na základě rešerše existujících nástrojů je zřejmé, že i když aplikace jako VisuAlgo či DSV nabízejí široké možnosti vizualizace, často jsou buď příliš komplexní, nebo zaměřené úzce na konkrétní typ datových struktur. Rozhodl jsem se pro vývoj vlastní aplikace, která by byla jednodušší, intuitivnější, přehlednější a bez reklam.

Cílem bylo vytvořit nástroj, který nabídne základní vizualizaci vybraných abstraktních datových typů s důrazem na uživatelskou přívětivost a srozumitelnost a zároveň mi umožní prakticky si vyzkoušet návrh a implementaci interaktivního webového rozhraní.

3 Praktická část

Tato část práce se věnuje návrhu a implementaci vlastní webové aplikace pro vizualizaci vybraných abstraktních datových typů. V kapitole je popsán výběr použitých technologií, knihoven a principy, na kterých aplikace funguje.

3.1 JavaScript

Výběr nástrojů a jazyka byl jednoduchý, jelikož jde o jednoduchou webovou aplikaci bez backendu padl výběr na JavaScript, interpretovaný programovací jazyk určený zejména pro jednoduché skriptování webových aplikací, který ale zejména díky node.js nachází uplatnění i na straně backendu. Čistý JavaScript jsem doplnil o knihovny.

3.1.1 GSAP

GSAP (GreenSock Animation Platform) je JavaScriptová knihovna pro tvorbu plynulých a přesných animací na webu. Umožňuje snadno animovat prvky DOM, a to s širokými možnostmi konfigurace. [9] V mé aplikaci je použita na všechny animace.

3.1.2 PhysicsJS

PhysicsJS je JavaScriptová knihovna pro fyzikální simulace ve webovém prostředí. Umožňuje simulovat tělesa, jejich pohyb, kolize a interakce podle fyzikálních zákonů. [10] V mé aplikaci využívám této knihovny v modulu Množina, protože mi dovolí ukázat nesetříděnost množiny.

3.1.3 Vis.js

Vis.js je open-source JavaScriptová knihovna určená pro vizualizaci dat, zejména v podobě sítí (grafů), časových os a diagramů. Knihovna umožňuje interaktivní a dynamické vykreslování grafových struktur s možností přizpůsobení vzhledu i chování uzlů a hran. [2] V mé aplikaci využívám modul **Network** ve vizualizaci Grafu a Stromu.

3.2 CSS

Pro tvorbu vzhledu aplikace využívám preprocesor **SCSS**, který je následně překládán do běžného CSS. Dále je použita knihovna **Tailwind CSS**, moderní framework, který umožňuje rychlé a konzistentní vytváření stylů přímo v HTML pomocí předdefinovaných tříd.

3.3 Struktura aplikace

3.3.1 /

V kořenové složce se nachází soubor `index.html`, který slouží jako výchozí (landing) stránka aplikace. Z této stránky vedou odkazy na jednotlivé moduly. Soubor `main.js` obsahuje společnou funkcionalitu využívanou napříč moduly, zejména:

- ovládání dropdown menu,
- přepínání horizontálních záložek (tabů),
- zobrazování a skrývání chybových hlášek.

Dále se zde nachází soubor se stylem `main.scss` a jeho zkompilovaná verze `main.css`.

3.3.2 pages/

Složka `pages/` obsahuje HTML stránky jednotlivých modulů (např. `stack.html`, `queue.html`, `list.html` apod.). Každá stránka představuje samostatnou část aplikace zaměřenou na konkrétní abstraktní datový typ.

3.3.3 scripts/

Ve složce `scripts/` se nachází JavaScriptové soubory, které implementují specifickou funkcionalitu pro jednotlivé moduly. Každý modul má vlastní script (např. `stack.js`, `queue.js` atd.), který obsahuje logiku potřebnou k vizualizaci a práci s daným datovým typem.

3.3.4 images/

Tato složka obsahuje obrázky, v mém případě to jsou SVG soubory šipek, které slouží k ukazování na místo přidání a odebírání prvku.

3.4 Funkce jednotlivých modulů

3.4.1 stack.js

Tento skript obsahuje dvě základní funkce, **stackPush()** a **stackPop()**, a pomocnou funkci **changeValue()**, která aktualizuje proměnnou **inputval** na základě vstupu od uživatele. Při načtení DOM stromu se spouští anonymní funkce, která nastavuje počáteční pozice šipek označujících místa pro přidání a odebrání prvků.

Funkce **stackPush()** nejprve ověřuje, zda již neprobíhá animace — v takovém případě se nic neprovede. Dále kontroluje, zda zásobník nepřesahuje svou kapacitu a zda je proměnná **inputval** platná (není prázdná). Pokud tyto podmínky nejsou splněny, je uživateli zobrazena chybová hláška. V opačném případě je hodnota **inputval** přidána do pole **stack**, je vytvořen nový DOM prvek, kterému jsou přiřazeny odpovídající vlastnosti, a následně je spuštěna animační posloupnost pomocí knihovny GSAP. Nový prvek se vizuálně přesune z horní části obrazovky dolů na správnou pozici v zásobníku a šipky se posunou o jednu úroveň výš.

Funkce **stackPop()** obdobně nejprve kontroluje, zda aktuálně neprobíhá animace. Poté ověřuje, zda zásobník obsahuje alespoň jeden prvek — pokud ne, zobrazí se chybové hlášení. Pokud zásobník obsahuje data, je poslední DOM prvek nalezen a odpovídající hodnota odstraněna z pole **stack**. Následuje animace: prvek se vizuálně vysune nahoru, šipky se posunou zpět o jednu úroveň dolů a prvek je následně odstraněn z DOMu.

3.4.2 queue.js

V tomto scriptu jsou opět dvě základní funkce pro interakci s frontou, **enqueue()** a **dequeue()**. I zde je funkce **changeValue()**. Navíc zde je funkce **resizeArrows()**, která se spouští při načtení DOM stromu. Slouží pro změnu velikosti šipek při mobilním zobrazení.

Funkce **enqueue()** ověřuje stejné věci jako **stackPush()**3.4.1. Následně je vytvořen DOM element a jsou mu přiřazeny požadované vlastnosti, a spouští se GSAP timeline. První se na místo určení nasune zelená šipka, následně přijede prvek z vrchu za frontu a následně je horizontálně nasunut na svou pozici.

Funkce **dequeue()** opět kontroluje neprázdnost a probíhající animaci. Pokud jsou podmínky splněny, odjíždí první prvek před frontu a následně nahoru mimo stránku. Prvek je z DOMu odebrán.

3.4.3 list.js

Tento script je strukturálně podobný předchozím dvěma. Pracuje se strukturou reprezentovanou jako pole. Je doplněn o pomocnou funkci **createDataElement()**, která vytváří DOM element.

Funkce **listAdd()** nejprve provede kontrolu platnosti vstupních dat. Následně se rozhoduje podle hodnoty proměnné **inputPosition**, kam má být nový prvek vložen. Pokud je **inputPosition** prázdná, prvek se přidá na konec seznamu. Pokud není, dochází k větvení:

- Pokud je pozice platná (nachází se v rozsahu indexů pole), prvky od daného indexu se odsouvají, zobrazí se zelená šipka a nový prvek se vloží na požadovanou pozici.
- Pokud je pozice mimo rozsah pole, je uživateli zobrazena chybová hláška.

Funkce **listDelete()** umožňuje dvě varianty mazání: podle hodnoty nebo podle pozice.

- V případě mazání podle hodnoty se nalezne první výskyt a začne sekvence mazání. Zobrazí se červená šipka a prvek se animuje směrem vzhůru, následně je odstaněn z DOMu.
- V případě mazání podle pozice se nejprve ověří, zda je zadaný index platný. Pokud ano, prvek se odstraní. Pokud je pozice mimo rozsah, je zobrazena chybová hláška.

3.4.4 set.js

Tento skript se liší od předchozích, jelikož využívá knihovnu PhysicsJS 3.1.2 pro simulaci fyzikálního chování objektů. Klíčovým prvkem je objekt **world**, který je zpřístupněn jako parametr callback funkce předané do **Physics()**. V rámci této funkce jsou definovány vlastnosti simulace – jako jsou okraje prostoru, detekce kolizí a gravitace.

Pomocí **world.on('step')** je zajištěna aktualizace pozice vizuálních prvků (DOM elementů) v každém kroku simulace na základě jejich fyzikálních souřadnic.

Na tlačítko **addBtn** je namapována funkce, která přidává objekty do simulace. První dochází ke kontrole vstupních hodnot. Následně je vytvořen DOM element a **Physics.body**, které dostane DOM element jako svoje view (vizuální prvek).

Tlačítko Odebrat spouští funkci odebrání. Zkontroluje vstup a odebere **Physics.body** i s jeho view.

3.4.5 buffer.js

Buffer je realizován pomocí pole a dvou ukazatelů: **head** (pozice pro zápis) a **tail** (pozice pro čtení). Skript obsahuje inicializační funkci **init()**, která vytvoří vizuální reprezentaci jednotlivých buněk bufferu a nastaví výchozí pozice ukazatelů.

Dále obsahuje funkce **bufferWrite()**, **bufferRead()**. Funkce **same()** a **movePointer()** jsou pomocné. **same()** kontroluje, zda ukazatele ukazují na stejné místo, a sjednotí je. Funkce **movePointer()** posouvá jednotlivé ukazatele o pozici vpřed.

writeBuffer() zkontroluje vstupní data a vloží do pole na pozici **head** ukazatele. Ukazatel se posouvá o jednu pozici.

Funkce **readBuffer()** pracuje na stejném principu.

3.4.6 graph.js

Tento script využívá knihovny **vis.js** 3.1.3, která dovoluje vytvářet objekt **vis.Netowrk**. Do tohoto objektu vstupují data (vlastní typ **vis.DataSet**), DOM element, uvnitř kterého se bude graf zobrazovat a **options**. Options nabízí různé možnosti nastavení, zde využívám **interaction** pro nastavení interakcí s uživatelem a **physics**, nastavení vzájemného chování uzlů.

Funkce **addNode()** provádí kontrolu vstupních dat a následně vytváří uzel se vstupní hodnotou.

Odebírání a spojování jednotlivých uzlů řeším pomocí **onClick()** eventu a **selectBoxu** se dvěma režimy (přidat hrany a odebrat uzly). Přidání hran probíhá tak, že uživatel vybere jeden uzel pomocí kliknutí myši, a následně myší klikne na další uzel a dojde k jejich propojení. Režim odebírání uzlů funguje podobně, uživatel kliká na uzly, funkce je nachází v **nodesData** a odebírá je.

3.4.7 tree.js

Tento skript opět využívá knihovnu **vis.js** 3.1.3. Středobodem skriptu je třída **BinarySearchTree**, která implementuje základní operace binárního vyhledávacího stromu. Jednotlivé uzly stromu jsou instance třídy **Node**, která obsahuje vlastnosti **data**, **left**, **right** a **visId**. Poslední slouží jako unikátní identifikátor pro vizualizaci pomocí **vis.Network**.

Přidávání uzlů do stromu zajišťuje funkce **addToTree()**, která zkontroluje vstupní hodnotu a poté zavolá metodu **insert(value)**. Pokud je strom prázdný, nový uzel se stává kořenem. Jinak se použije rekurzivní metoda pro nalezení správného místa ve stromu.

Odebírání probíhá podobně – funkce **deleteFromTree()** volá metodu **remove(value)**, která v sobě využívá rekurzivní funkci **removeNode(node, value)**. Ta podle situace odstraní uzel, nahradí ho potomkem nebo nejmenším prvkem z pravého podstromu.

Součástí skriptu je také vizualizace tří variant průchodu stromem: inorder, preorder a postorder, jak je popsáno v kapitole 2.2.7. Výsledek procházení je postup-

ně vypisován na obrazovku. Pro zlepšení názornosti je využita asynchronní funkce **highlightNode()**, která dočasně změní barvu právě navštíveného uzlu, čímž vizuálně znázorňuje pořadí průchodu.

3.5 Testování a zpětná vazba

Zpětnou vazbu jsem získal od spolužáků, kterým jsem aplikaci zaslal a poprosil je, aby si ji vyzkoušeli. Komentáře jsem sbíral přes Discord, většina reakcí byla neformální, ale přesto velmi užitečná.

3.5.1 Pozitiva

Ukázky kódu a stručné popisy u každé struktury byly často zmiňovány jako přehledné a užitečné. Uživatelé si chválili grafickou vizualizaci, zejména u kruhového bufferu a grafu. Oceňováno bylo jednoduché ovládání, které je podle komentářů vhodné i pro středoškolské prostředí nebo jako výukový nástroj do předmětu ALD.

3.5.2 Negativa

Jedna reakce kritizovala nemožnost nastavit velikost datových kontejnerů, také objevila nedostatek u modulu Množina, u kterého se prvky nepřizpůsobí velikosti obsahu. Více reakcí zmiňovalo jako negativum chybějící **localStorage** pro ukládání stavu ADT a aktuální horizontální záložky. U modulu Strom byl nalezen nedostatek ve vizualizaci průchodu, konkrétně, pokud je vybrána nějaká položka, při vizualizaci nezmění barvu. Jedna reakce zmiňovala, že by bylo dobré zvýraznit důležité informace v popisu struktury.

3.6 Zhodnocení reakcí a možná rozšíření

Zpětná vazba potvrdila, že projekt plní svůj účel jako výuková pomůcka, a vizualizace je vnímána jako srozumitelná a užitečná. Kritické připomínky byly velmi cenné.

Jako hlavní možnost rozšíření vidím přidání dalších modulů datových struktur (halda, AVL stromy atd.) a implementaci **localStorage** pro uchování stavu aplikace. Díky tomu by si uživatel mohl při přepínání mezi záložkami nebo obnovení stránky zachovat aktuální stav datové struktury i zvolenou záložku.

Závěr

Cílem této práce bylo navrhnout a implementovat webovou aplikaci sloužící k vizualizaci abstraktních datových typů a k jejich základnímu popisu. Aplikace umožňuje uživatelům interaktivně pracovat s vybranými datovými strukturami a získat tak lepší představu o jejich chování v praxi.

Během práce jsem si prohloubil znalosti v oblasti návrhu webových aplikací, práce s knihovnami pro vizualizaci grafů a v použití Tailwind CSS.

Použitá literatura

[1] *Algoritmus* [online]. [cit. 2025-05-06]. Dostupné z: https://www.algoritmy.net/.

[2] ALMENDE. *vis.js* [online]. [cit. 2025-05-03]. Dostupné z: https://visjs.org/.

[3] GALLES, David. *Data Structure Visualizations* [online]. 2011. [cit. 2025-05-03]. Dostupné z: https://www.cs.usfca.edu/~galles/visualization/Algorithms.html.

[4] HALIM, Steven. *VisuAlgo* [online]. 2011. [cit. 2025-04-28]. Dostupné z: https://visualgo.net/en.

[5] KOPETSCHKE, Igor. *Datové typy, abstraktní datové struktury* [online]. Liberec: Technická univerzita v Liberci, 2022 [cit. 2025]. Dostupné z: https://elearning.tul.cz/pluginfile.php/1020634/mod_resource/content/4/NTI-ADA-2022-datove_struktury.pdf.

[6] KOPETSCHKE, Ing. Igor. *Vyhledávání, BST* [online]. Liberec: Technická univerzita v Liberci, 2024 [cit. 2025]. Dostupné z: https://elearning.tul.cz/mod/resource/view.php?id=666346.

[7] MIČKA, Pavel. *Komentář ve vlákně: Kritika Algoritmy.net* [online]. 2010-01-30. [cit. 2025-05-06]. Dostupné z: https://webtrh.cz/diskuse/kritika-algoritmy-net/.

[8] TUTORIALSPOINT. *Graph Data Structure* [online]. [cit. 2025-05-09]. Dostupné z: https://www.tutorialspoint.com/data_structures_algorithms/graph_data_structure.htm.

[9] WEBFLOW. *GSAP* [online]. 2025. [cit. 2025-05-03]. Dostupné z: https://gsap.com/.

[10] WELLCAFFEINATED. *PhysicsJS* [online]. 2014. [cit. 2025-05-03]. Dostupné z: https://wellcaffeinated.net/PhysicsJS/.

A Přílohy

Veškeré zdrojové kódy jsou dostupné ve veřejném repozitáři na GitHubu:

- https://github.com/vaclavkozeny/ADT

Hotová webová aplikace je nasazena a dostupná na adrese:

- https://vaclavkozeny.github.io/ADT