VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

KNIHOVNA PRO VIZUALIZACI DYNAMICKÝCH DATOVÝCH STRUKTUR

LIBRARY FOR DYNAMIC DATA STRUCTURE VISUALIZATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **TOMÁŠ ZAHRADNÍČEK**
AUTHOR

VEDOUCÍ PRÁCE — **Dr. Ing. PETR PERINGER**
SUPERVISOR

BRNO 2017

Zadání bakalářské práce/20167/2016/xzahra24

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Zahradníček Tomáš**
Obor: Informační technologie
Téma: **Knihovna pro vizualizaci dynamických datových struktur**
Library for Dynamic Data Structure Visualization
Kategorie: Algoritmy a datové struktury

Pokyny:

1. Seznamte se s problematikou vizualizace datových struktur jazyka C. Prostudujte dostupnou literaturu zaměřenou na vizualizaci dynamických datových struktur typu seznam a strom.
2. Navrhněte vhodné rozhraní a knihovnu pro vizualizaci datových struktur umožňující zobrazení základních datových struktur s využitím programu Graphviz. Knihovna musí umožňovat volbu různých forem zobrazení datových struktur.
3. Knihovnu implementujte (v jazyku C nebo C++) tak aby byla použitelná na systémech Windows i Linux. Na vhodně zvolených příkladech proveďte testování knihovny.
4. Zhodnoťte dosažené výsledky a navrhněte další možná vylepšení.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění prvních dvou bodů zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Peringer Petr, Dr. Ing.**, UITS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstrakt

Tato práce se zabývá návrhem a implementací knihovny pro vizualizaci dynamických datových struktur ve formě orientovaných grafů. Zaměřuje se také na chyby, které při implementaci těchto struktur můžou vzniknout. Výsledná knihovna může být použita jako ladicí nástroj nebo pro výuku algoritmů pro práci s těmito strukturami.

Abstract

This thesis deals with design and implementation of library for visualizing dynamic data structures as directed graphs. It also focuses on errors, that can occur when implementing such structures. Resulting library can be used as debugging tool and may help understand algorithms for these structures.

Klíčová slova

Vizualizace datových struktur, ladicí nástroj, dynamické datové struktury

Keywords

Visualization of data structures, debugger, dynamic data structures

Citace

ZAHRADNÍČEK, Tomáš. *Knihovna pro vizualizaci dynamických datových struktur*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Peringer Petr.

Knihovna pro vizualizaci dynamických datových struktur

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Dr. Ing. Petra Peringera. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Zahradníček
17. května 2017

Poděkování

Děkuji svému vedoucímu práce Dr. Ing. Petru Peringerovi za poskytnutou pomoc a rady, které mi pomohly tuto práci vytvořit.

Obsah

Kapitola 1

Úvod

Dynamická datová struktura je taková organizace dat, která nemusí být v paměti souvislá, ale namísto toho jednotlivé prvky obsahují adresu ostatních prvků – ukazatele. Navíc se může v paměti zvětšovat a zmenšovat podle potřeby díky přidělování (alokaci) a uvolňování (dealokaci) paměti. V nově přidělené paměti potom vznikají nové prvky dynamické struktury a pouze se upraví existující ukazatele. Podle vzájemné organizace prvků se dynamické datové struktury dělí na různé typy, které potom mají různé vlastnosti. Mezi základní dynamické struktury patří například lineárně vázaný seznam nebo binární vyhledávací strom.

Protože nejsou jednotlivé prvky v paměti souvisle, ale odkazují se na sebe navzájem pomocí ukazatelů, je pro člověka složité rychle zjistit, co vlastně tato datová struktura obsahuje, a jakým způsobem jsou její prvky organizovány. Proto je vhodné zobrazovat tyto datové struktury v jejich skutečném významu a to pomocí grafů. Navíc mohou být zobrazeny i probíhající úpravy v dynamické datové struktuře pro lepší pochopení souvislostí.

Při implementaci algoritmů pracujících s dynamickými datovými strukturami mohou jednoduše vzniknout chyby s úpravou hodnot ukazatelů, nebo s přidělováním a uvolňováním paměti. I toto může být součástí zobrazovaného grafu.

Tato práce se zabývá návrhem a implementací knihovny, která by dynamické datové struktury zobrazovala, a zároveň upozornila na chyby, které mohou v dynamické struktuře vzniknout. V kapitole 2 budou představeny základy vizualizace grafů a některé existující nástroje, které pomocí grafů zobrazují dynamické datové struktury. Částečně podle těchto nástrojů budou v kapitole 3 analyzovány a navrhnuty potřebné části knihovny. V kapitole 4 bude podrobněji přiblížena implementace navržené knihovny. Nakonec budou v kapitole 5 předvedeny výsledky na základních dynamických strukturách a popsán způsob, jakým lze knihovnu použít.

V textu budou odlišovány výrazy struktura, dynamická struktura a dynamická datová struktura. První z nich popisuje datový typ složený z pevně daného počtu položek, kde každá položka má definovaný typ (struktura v jazyku C definovaná klíčovým slovem `struct`). Zbývající ze zmíněných výrazů budou oba popisovat spojení jednotlivých struktur pomocí ukazatelů.

Kapitola 2

Vizualizace grafů a dynamických datových struktur

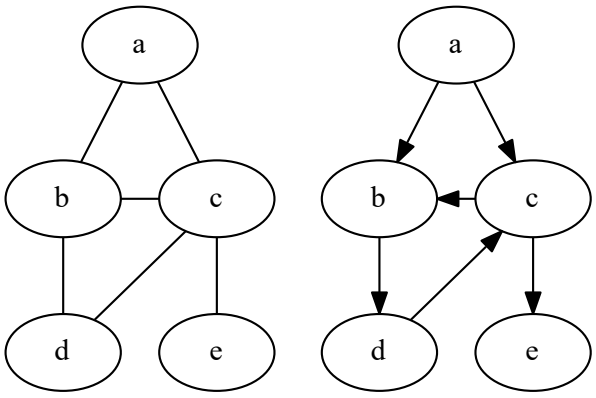
V této kapitole bude představen pojem graf a jeho součásti, způsob vizualizace dat pomocí grafů a postupy potřebné při vykreslování grafů, aby byly člověku srozumitelné. Dále budou představeny nástroje, díky kterým lze grafy a dynamické datové struktury vizualizovat.

2.1 Teorie grafů

V teorii grafů jsou definovány dva základní typy grafů: orientovaný a neorientovaný. Následující definice jsou převzaty z učebního textu *Diskrétní matematika*[9].

Definice 2.1.1. *Neorientovaný graf* G se skládá z množiny V vrcholů (uzlů) a množiny H hran tak, že každá hrana $h \in H$ je přiřazena neuspořádané dvojici (tj. dvouprvkové množině) vrcholů $u, v \in V$.

Definice 2.1.2. *Orientovaný graf* G se skládá z množiny V vrcholů a množiny H hran tak, že každá hrana $h \in H$ je přiřazena uspořádané dvojici vrcholů $(u, v) \in V \times V$.

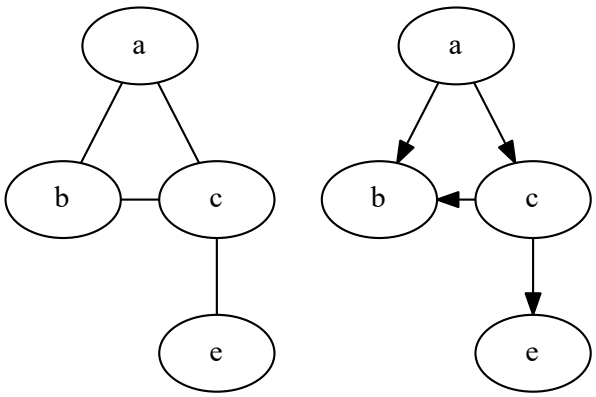


Obrázek 2.1: Příklad neorientovaného (vlevo) a orientovaného grafu (vpravo).

V obou případech může být ke stejné dvojici vrcholů přiřazeno více hran. V takovém případě se jedná o hrany *násobné*. Také mohou být grafy rozšířené o ohodnocení hran nebo vrcholů. Takové ale nebudou v této praci potřeba.

Teorie grafů definuje mnoho dalších pojmů, jako například podgraf, okolí vrcholu, stupeň vrcholu nebo minimální kostra grafu. Pro tuto práci však postačí pojmy graf, vrchol, hrana, a dále také podgraf, sled, cesta, kružnice, cyklus a izomorfizmus grafů.

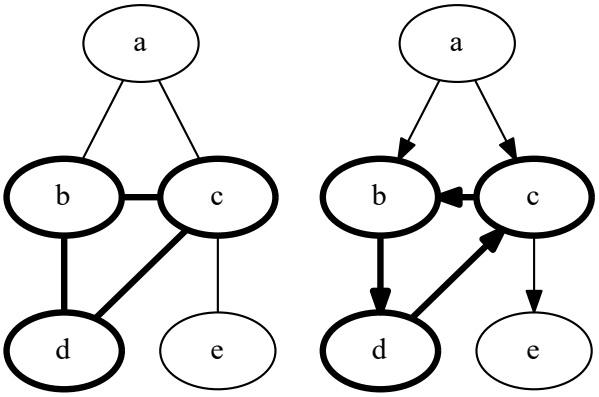
Definice 2.1.3. Nechť $G_1 = (V_1, H_1), G_2 = (V_2, H_2)$ jsou grafy. Graf $G2$ je potom podgrafem grafu $G1$, jestliže $V_2 \subseteq V_1$, $H_2 \subseteq H_1$.



Obrázek 2.2: Příklad neorientovaného (vlevo) a orientovaného podgrafu (vpravo) z grafů v obr. 2.1.

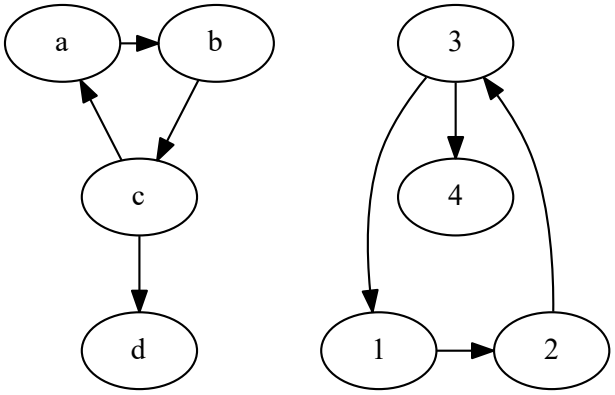
Definice 2.1.4. *Sledem* délky n nazýváme poslouplnost vrcholů v_i a hran h_j grafu G tvaru $v_0h_1v_1h_2...v_{n-1}h_nv_n$ kde hraně h_i je přiřazena dvojice vrcholů $\{v_{i-1}, v_i\}$ (resp. (v_{i-1}, v_i) v orientovaném grafu). Sled se nazývá *uzavřený*, jestliže $v_0 = v_n$.

Definice 2.1.5. *Cesta* v grafu G je sled, v němž se každý vrchol grafu vyskytuje nejvýše jednou. Uzavřený sled se nazývá *uzavřenou cestou*, pokud se v něm vyskytuje každý vrchol s vyjímkou počátečního vrcholu, který je současně i koncový, nejvýše jednou. *Kružnice* je neorientovaná uzavřená cesta a *cyklus* je uzavřená cesta orientovaná.



Obrázek 2.3: Příklad kružnice (vlevo) a cyklu (vpravo) v grafech z obr. 2.1.

Definice 2.1.6. Grafy $G_1 = (V_1, H_1)$ a $G_2 = (V_2, H_2)$ se nazývají izomorfní, jestliže existuje bijekce $f : V_1 \to V_2$ a $g : H_1 \to H_2$ takové, že libovolné hraně $h \in H$ jsou přiřazeny vrcholy $x, y \in V_1 \iff$ hraně $g(h) \in H_2$ jsou přiřazeny vrcholy $f(x), f(y) \in V_2$.



Obrázek 2.4: Příklad izomorfních grafů.

2.2 Vykreslování grafů

Pomocí grafu se dá vyjádřit mnoho různých informací, například propojení počítačové sítě, tok řízení programu nebo organizace datové struktury. Rozmístění vrcholů grafu tak, aby byla výsledná vizualizace co nejvíce přehledná však není triviální problém. Pro přehlednost grafu pomáhají následující vizuální vlastnosti grafu[11]:

- Hrany směřující v jednotném směru
- Krátké hrany
- Přímé hrany
- Jednotně vzdálené vrcholy
- Co nejméně křížení hran

Existuje mnoho algoritmů pro rozmisťování vrcholů. Některé jsou určeny pro obecné grafy bez bližší informace o organizaci prvků v grafu, jiné jsou specializovány na některý typ grafů[7].

Příklady algoritmů jsou Walkerův algoritmus pro grafy typu strom[12], nebo algoritmus pro hierarchické orientované grafy, které navrhl Sugyiama[10].

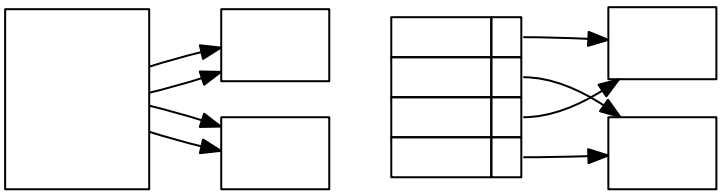
Algoritmus podle Sugyiama se skládá z 5 fází:

- Odstranění cyklů: Pokud graf obsahuje cyklus, musí být některá hrana z tohoto cyklu otočena, aby byl orientovaný cyklus přerušen
- Přiřazení úrovní: Každému vrcholu je přiřazena úroveň, přičemž vrcholy na stejné úrovni budou mít ve výsledném zobrazení stejnou souřadnici y. Také jsou vytvářeny pomocné vrcholy tak, aby každá hrana spojovala jen vrcholy na sousední úrovni.

- Snižování počtu překřížení: Vrcholy v každé úrovni jsou přeuspořádány tak, aby byl snížen celkový počet překřížení hran.
- Přiřazení souřadnic: Vrcholům v každé úrovni jsou přiřazeny souřadnice x tak, že je zachováno pořadí vrcholů v rámci úrovně.
- Směrování hran: Hrany jsou vykresleny jako lomená čára, kde zlomy jsou vytvářeny dalšími pomocnými vrcholy. Existují další způsoby vykreslování hran, jako například pomocí spline křivek.

Dále mohou být zaváděny požadavky, které jsou vhodné při vykreslování dynamických datových struktur, jako například propojování hran na stejné úrovni, určení seřazení některých párů vrcholů ve stejné úrovni, nebo určení která z hran bude použita pro přerušení cyklu[7]. Tyto úpravy mohou některé dynamické struktury zpřehlednit, ale vyžadují úpravu algoritmu, aby byly možné.

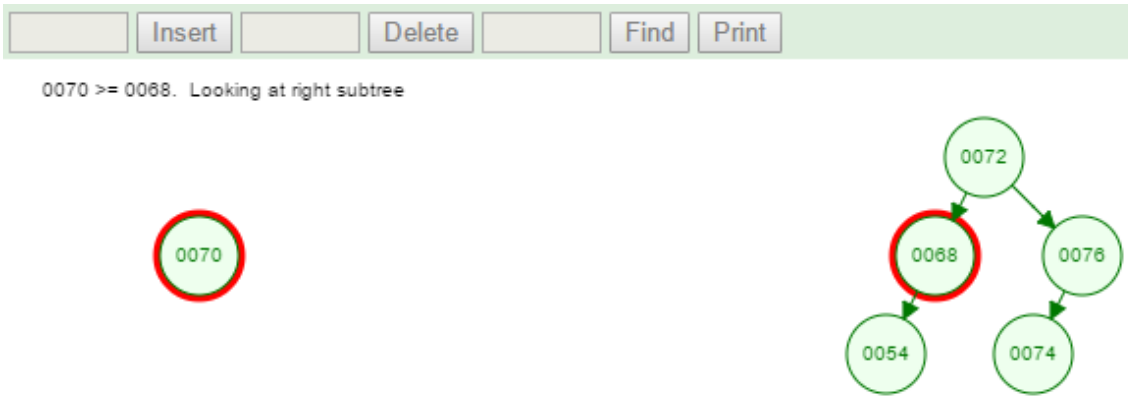
Při vykreslování dynamických datových struktur obsahují její jednotlivé prvky několik položek. Každá z těchto položek může být ukazatel, a tedy z ní může směřovat hrana do dalšího vrcholu. Na rozdíl od obyčejného grafu tedy záleží, ze které části vrcholu hrana vychází. Při vykreslení prvků dynamické struktury tedy může nastat křížení hran, se kterým uvedený algoritmus nepočítá (viz obr. 2.5). Původní algoritmus totiž předpokládá, že křížení hran může nastat pouze mezi hranami spojující rozdílné páry vrcholů.



Obrázek 2.5: Porovnání křížení hran dvou stejných grafů, pokud se nebere v potaz vnitřní struktura jednotlivých vrcholů (vlevo) nebo ano (vpravo).

2.3 Online výukové nástroje

Pro výukové účely je možné na internetu najít mnoho nástrojů zobrazujících práci s dynamickými strukturami (např. [1]). Tyto nástroje nabízí interaktivní vyzkoušení operací nad předpřipravenými dynamickými strukturami (viz obr. 2.6). To může být ale zároveň nevýhoda, protože tyto dynamické struktury jsou pevně dané, nelze je tedy upravovat a nelze experimentovat na nižší úrovni implementace, jako je správné propojování jednotlivých prvků pomocí ukazatelů.



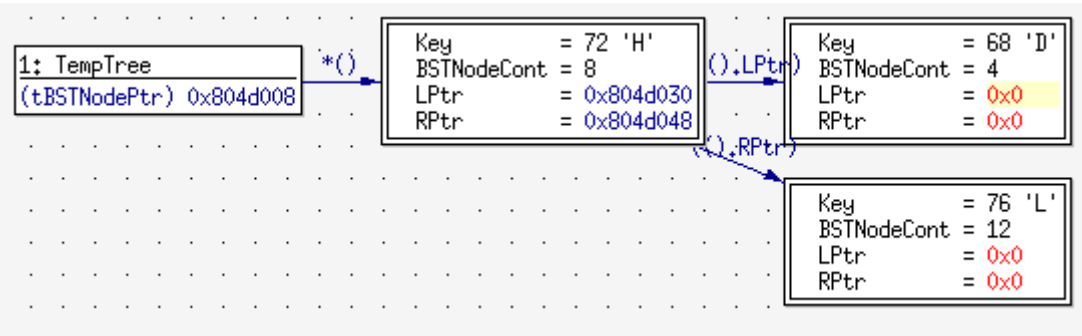
Obrázek 2.6: Zobrazení dynamické struktury typu binární vyhledávací strom s právě vkládaným prvkem v [1]

2.4 DDD

Základním nástrojem pro ladění programů v jazyku C je GDB (GNU project debugger)[3]. Je dostupný na operačních systémech unixového typu i Windows. Nabízí pouze textové rozhraní, a proto může být obtížné vyznat se v informacích, které zobrazuje. To platí hlavně pro hodnoty typu ukazatel, kdy GDB zobrazí pouze hodnotu ukazatele, tedy adresu v paměti. Je sice možné nechat si postupně tyto části paměti vypsat, ale po několika zanoření je tento přístup pomalý a nepřehledný.

Tento nedostatek dokáže vyřešit nástroj Data Display Debugger (DDD)[2], který je grafickou nadstavbou nad nástrojem GDB. V DDD je možné využívat základní schopnosti GDB bez znalosti textových příkazů. Přesto, aby bylo možné využít všechny možnosti GDB, je dostupná konzole, kde je možné příkazy zadávat textově.

Důležitou součástí DDD je přehlednější zobrazení datových struktur graficky. V případě, že struktury obsahují ukazatele, dokáže zobrazit obsah paměti, kam ukazatel směřuje. Tímto způsobem je možné celou dynamickou datovou strukturu zobrazit ve formě orientovaného grafu (viz obr. 2.7).



Obrázek 2.7: Zobrazení dynamické struktury typu binární vyhledávací strom v DDD

2.5 GraphViz

GraphViz je sada nástrojů pro zobrazení a práci s obecnými grafy. Využívá textovou reprezentaci grafů zapsanou v jazyce DOT [6], které lze následně převést jedním z několika programů na obrázek v některém z běžných formátů (PNG, PDF, SVG, PostScript). Jedním ze základních nástrojů pro vykreslování grafů je `dot` [8], který je primárně určen pro zobrazení hierarchických orientovaných grafů. Další nástroje ze sady jsou zaměřeny na různé typy grafů a využívají jiných způsobu rozmísťování vrcholů [4].

K dispozici jsou knihovna s hlavičkovým souborem v jazyku C, díky kterým lze grafy vytvářet přímo v programu, namísto v textové reprezentaci.

Graph Description Language, zkráceně DOT, je jazyk popisující grafové struktury. Graf je popisován pomocí základních elementů, kterými jsou graf, vrchol a hrana [6]. Každý z těchto elementů má definovanou svoji množinu možných atributů, kterými lze změnit vzhled výsledného grafu, ať už jde jen o změnu barvy nebo stylu některého elementu, nebo o vzájemné rozložení jednotlivých elementů. Pro zobrazení tabulky jako obsahu vrcholu lze využít buď zápis typu `record` nebo formát html table, a tím je možné přehledně zobrazit jednotlivé prvky dynamické datové struktury. Ačkoliv zápis typu record je kratší, html nabízí více možností, jako je například změna barvy jednotlivých buněk.

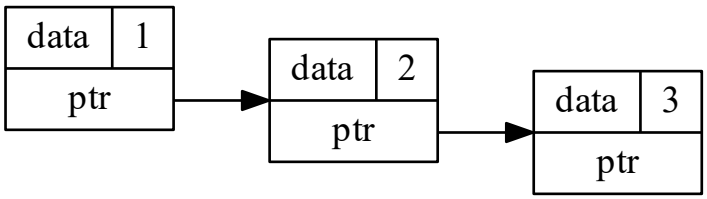
Hrana se může definovat nejen mezi celými vrcholy, ale také mezi jednotlivými buňkami tabulky ve vrcholu, což je vhodné právě pro zobrazení ukazatelů (viz obr. 2.8).

```
digraph g {
  // atribut grafu
  rankdir=LR;
  // vychozi atribut vrcholu
  node [shape=record];

  // definice vrcholu pomoci record
  n1[label="{data | 1}|<p1> ptr"]
  n2[label="{data | 2}|<p1> ptr"]
  n3[label="{data | 3}|<p1> ptr"]

  // definice hran
  n1:p1->n2
  n2:p1->n3
}
```

Výpis 2.1: popis grafové reprezentace lineárního seznamu

```
...
node [shape=plaintext]

n1[label=<<TABLE BORDER="0"
  CELLBORDER="1" CELLSPACING="0">
  <TR><TD>data</TD><TD>2</TD></TR>
  <TR><TD COLSPAN="2" PORT="p1">ptr</TD></TR>
</TABLE>>];
...
```

Výpis 2.2: popis vrcholu `n1` z výpisu 2.1 pomocí html table



Obrázek 2.8: příklad výstupu nástroje dot výpisu 2.1

Kapitola 3

Analýza a návrh knihovny

V této kapitole bude popsán postup návrhu knihovny. Ten bude vycházet z požadované funkcionality knihovny, kterou je zobrazení stavu dynamické datové struktury v jazyku C.

Pro přehlednou práci s obecnými dynamickými strukturami vhodné vytvořit jejich interní reprezentaci, se kterou bude knihovna pracovat.

Protože knihovna je zamýšlena hlavně pro ladicí a výukové účely, bude potřeba se zaměřit také na běžné chyby, které se v této oblasti vyskytují.

3.1 Vstup knihovny

Aby bylo možné získat hodnoty v dynamické struktuře, je potřeba znát typy struktur jednotlivých prvků. Ladicí nástroje jako gdb to dokáží zjistit z ladicích informací, které mohou být do výsledného spustitelného souboru vloženy překladačem. Bez ladicích informací dokáží ladicí nástroje zobrazit jen informace na úrovni strojového kódu.

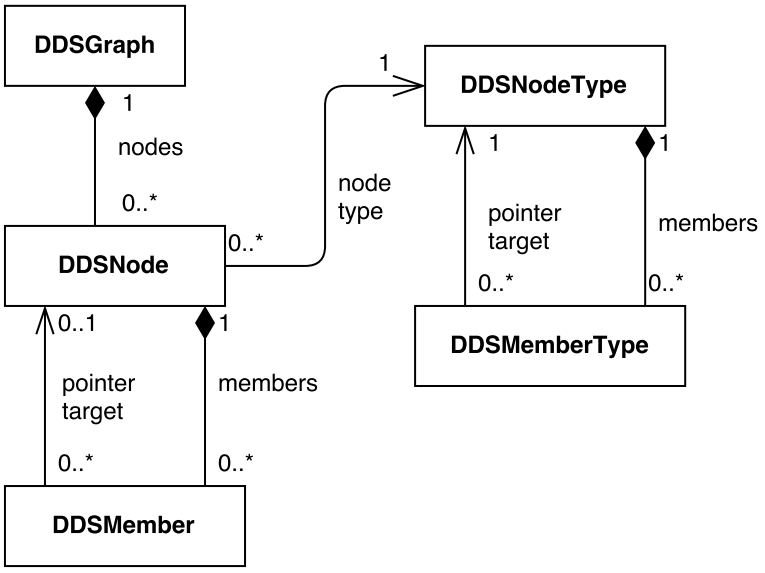
Knihovna musí zobrazovat celé struktury. Může tedy využít ladicí informace stejně jako je využívají ladicí nástroje. Další možností je, aby popis struktury byl přímo v kódu programu, který knihovnu využívá. V takovém případě bude potřeba, aby součástí rozhraní knihovny byla možnost definovat struktury, které se v tomto programu mohou nacházet.

Definovat bude potřeba jednotlivé struktury a jejich členy. U každého členu je potřeba znát datový typ pro odpovídající zobrazení, a jeho umístění ve struktuře. V případě, že položkou struktury je ukazatel, který má význam ve spojování dynamické struktury, je potřeba znát typ struktury, na který může ukazovat.

Po definici struktur bude moci program předat knihovně ukazatel na kořenový prvek dynamické datové struktury. Tento ukazatel musí být předán zároveň s informací o typu struktury, na který ukazatel směřuje.

3.2 Interní reprezentace dynamických datových struktur

Pro vytvoření reprezentace grafových struktur je vhodné se inspirovat jazykem DOT. Jak bylo zmíněno v kapitole 2.5, graf se v jazyku DOT popisuje elementy graf, vrchol a hrana. Interní reprezentace by však měla brát v úvahu i specifičnost uložené struktury – každý vrchol, který reprezentuje jeden element v dynamické datové struktuře, musí mít konkrétní typ. Tento typ odpovídá získaným informacím popsané v kapitole 3.1.



Obrázek 3.1: Návrh interní reprezentace dynamických datových struktur v knihovně

Ačkoliv z jednoho vrcholu může vést více hran, každá hrana je určena jednoznačně ukazatelem, který ukazuje na další vrchol, a tedy pro hranu není potřeba vytvářet vlastní třídu. Z těchto úprav potom vychází následující návrh interní reprezentace (viz obr. 3.1).

`DDSGraph` představuje celý jeden stav dynamické datové struktury. Těch může v průběhu úprav dynamické struktury vzniknout více. `DDSNodeType` obsahuje informace popsané definicí datového typu struktura v jazyku C a popis jejích položek. Neobsahuje tedy ještě žádná data konkrétní struktury. Informace o jednotlivých položkách struktury jsou potom uloženy v `DDSMemberType`. `DDSNode` je prvkem celé dynamické struktury, a musí být vytvořena na základě typu (`DDSNodeType`). Díky tomu jsou u každého vrcholu `DDSNode` známy potřebné informace o jednotlivých položkách. `DDSMember` je potom samotnou hodnotou jedné položky, ať už se jedná o přímou hodnotu nebo ukazatel, a odpovídá typu položky (`DDSMemberType`) struktury.

3.3 Čtení dynamické struktury z paměti

Po tom, co jsou známy informace o struktuře, musí knihovna analyzovat dynamickou strukturu. Ta se bude ukládat do interní reprezentace dynamických struktur knihovny (popsáno v kapitole 3.2), a až z této reprezentace se bude vytvářet výstup v jazyce DOT nebo případně ještě provádět potřebné úpravy.

Při čtení dynamické struktury z paměti je potřeba si dávat pozor na ukazatele, které mohou ukazovat na neplatné místa v paměti. Tím nejsou myšleny ukazatele s hodnotou `NULL`, které označují konec dynamické struktury, nýbrž ukazatele, které by se mohly jevit jako platné, ale přístup na cílovou adresu by mohl způsobit pád programu. Ačkoliv pád programu může být při ladění programu pracujícího s dynamickými strukturami indikací místa chyby, v případě samostatné knihovny je toto chování nežádoucí. Namísto toho by se chybná práce s ukazateli měla ukázat ve výsledném grafu. Způsob řešení tohoto problému je dále popsán v kapitole 4.5.

3.4 Výstup knihovny

Protože datové struktury s ukazateli vytvářejí orientované grafy a o tvaru takového grafu nejsou známy další informace (není předem známý typ dynamické struktury), bude v knihovně využit nástroj `dot` ze sady GraphViz. Případných úprav ve výsledném grafu lze docílit využitím některých z mnoha vlastností jazyka DOT.

Po zpracování dynamické struktury získané z programu využívající knihovnu bude potřeba vytvořit výstup v jazyku DOT. Interní reprezentace však nemá jednoznačnou reprezentaci vizuální a tedy ani v jazyku DOT. Uživatel, neboli program využívající knihovnu, by měl mít možnost zvolit způsob jakým bude finální podoba vypadat.

Prvním z těchto způsobů zobrazení by mohla být jednoduchá orientace grafu (zleva doprava nebo shora dolů), což také patří mezi atribut grafu v jazyku DOT.

Ačkoliv v nástroji DDD, kterým je tato knihovna z hlediska vizualizace dynamických struktur inspirována, jsou vždy zobrazené číselné hodnoty ukazatelů, tato reprezentace nemá z hlediska dynamické struktury význam. Proto bude knihovna vždy ukazovat pouze spojení jednotlivých vrcholů grafu, bez číselné hodnoty. Navíc lze využít možnosti jazyka DOT směrování hrany přímo z buňky tabulky, která odpovídá jednomu ukazateli (oproti DDD, kde hrana vychází z celého vrcholu). Díky tomu je viditelné, ke kterému ukazateli hrana patří, a není potřeba pojmenovávat samotnou hranu.

Při ladění práce s dynamickými strukturami nemusí být chyba viditelná přímo z jednoho stavu dynamické struktury, ale při zobrazení změny mezi dvěma následujícími stavy může být jasnější co se vlastně stalo, a tedy i výskyt případné chyby. Knihovna by tedy měla nabízet možnost zobrazit nejen samotný stav, ale stejně tak i změny, které k tomuto stavu vedly.

Knihovna je sice zamýšlena pro jednoduché a málo obsáhlé dynamické struktury, ale přesto by mohlo pomoci nezobrazovat všechny prvky dynamické struktury, ale jen ty, kterých se týká změna, nebo ty, které si vybere uživatel. Ostatní prvky by potom byly sjednoceny do jednoho vrcholu v grafu nebo vynechány, a tím by se celý graf zjednodušil a zpřehlednil.

Kapitola 4

Implementace knihovny

Ačkoliv je knihovna určena pro jazyk C, implementace v jazyku C++ nabízí mnoho usnadnění. Jednou z nich je standardní knihovna, která nabízí jednodušší způsob implementace často řešených problémů než v jazyku C, a to hlavně díky šablonám (templates) z STL (Standard Template Library). Dále je možné v C++ definovat třídy pro objektovou orientaci. Také Nabízí způsob vytvoření funkcí, které je možné volat z jazyka C, a tím je možné vytvořit požadované rozhraní.

V této kapitole bude navázáno na požadavky popsané v předchozí kapitole. Bude popisována implementace jednotlivých částí v jazyku C++ a problémy, které z implementace vzešly.

4.1 Definice struktur

Před vytvořením jakékoliv dynamické struktury je potřeba znát popis jednotlivých prvků, ze, kterých se skládá. K tomu slouží třída `DDSNodeType`, která obsahuje informace o struktuře, a její prvky `DDSMemberType`, které obsahují informace o jednotlivých položkách struktury. Popis struktury se musí zadávat v kódu voláním funkcí knihovny. Pro zjednodušení je možné použít makra, která využívají operátoru `sizeof` a makra `offsetof`, které zjistí velikost a umístění položky ve struktuře (viz výpis. 4.1). S využitím pomocné funkce potom předá tyto informace knihovně, společně se jménem struktury nebo položky pro pozdější vyhledávání nebo zobrazení. U datových položek je také potřeba uvést datový typ, podle kterého se budou hodnoty této položky zobrazovat. U položek typu ukazatel není potřeba zjišťovat velikost (velikost datového typu ukazatel je při překladu vždy známa), ale je potřeba znát typ cílové struktury.

Struktury lze potom definovat v kódu následovně (viz výpis. 4.2).

4.2 Vytvoření interní reprezentace dynamické struktury

Když uživatelský program zavolá funkci knihovny pro vykreslení dynamické struktury (viz výpis. 4.3), knihovna nejdříve vytvoří interní reprezentaci této dynamické struktury. K tomu využije ukazatel na začátek dynamické struktury, a protože jsou v tuto chvíli známy všechny jednotlivé struktury i s jejími položkami, můžou se postupně číst paměťová místa, na které položky ve strukturách postupně ukazují, i s potřebnými datovými položkami. V dynamické struktuře může existovat cyklus, takže je potřeba si ukládat, která místa už byla přečtena.

```
// makro pro definici struktury
#define newStruct(STRUCTNAME) \
  newStruct_f(#STRUCTNAME, sizeof(STRUCTNAME))

// makro pro definici datove polozky ve strukture
#define newMemberData(STRUCTNAME, MEMBERNAME, TYPE) \
  newMemberData_f(#STRUCTNAME, #MEMBERNAME, TYPE, \
  sizeof(((STRUCTNAME *)0)->MEMBERNAME), \
  offsetof(STRUCTNAME, MEMBERNAME))

// makro pro definici polozky typu ukazatel ve strukture
#define newMemberPtr(STRUCTNAME, MEMBERNAME, TARGETNAME) \
  newMemberPtr_f(#STRUCTNAME, #MEMBERNAME, #TARGETNAME, \
  offsetof(STRUCTNAME, MEMBERNAME))
```

Výpis 4.1: Zápis maker pro definici struktur a položek

```
// definice struktury v jazyku C - prvek stromu
struct treeNode {
  int Key;
  struct treeNode * LPtr;
  struct treeNode * RPtr;
}

int main(void) {
  // definice samotne struktury pro knihovnu
  newStruct(struct treeNode);
  // definice jednotlivych polozek struktury pro knihovnu
  newMemberData(struct treeNode, Key, UNSIGNED);
  newMemberPtr(struct treeNode, LPtr, struct treeNode);
  newMemberPtr(struct treeNode, RPtr, struct treeNode);
```

Výpis 4.2: Příklad použití maker pro definici struktury a položek

```
// treePtr je ukazatel na typ struct treeNode
drawDDS(struct treeNode, treePtr);
```

Výpis 4.3: Použití makra pro vykreslení definované dynamické struktury

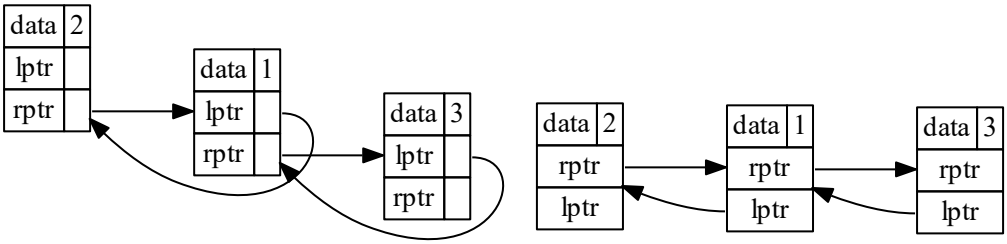
Také je vždy kontrolováno zda na dané paměťové místo je bezpečné přistoupit (popsané v kapitole 4.5)

Výsledkem je objekt třídy `DDSGraph`, který popisuje celou dynamickou strukturu, která byla přečtena z paměti. Jednotlivé prvky, neboli vrcholy grafu, jsou jednoznačně identifikovány jejich původním paměťovým místem a jsou uloženy v objektu třídy `DDSNode`. Jednotlivé hodnoty položek uložené v `DDSMember` lze rozeznat mezi ukazateli a datovými hodnotami pomocí odkazu na `DDSMemberType`.

4.3 Převedení na výstupní reprezentaci

Jakmile je vytvořena interní reprezentace dynamické struktury, je možné ji převést do jazyka DOT. Toto se provádí přímo výpisem textu do výstupního souboru (není využívána knihovna GraphViz pro vytváření grafu přímo v kódu). Následně je spuštěn nástroj dot, který tento soubor převede na formát, který si uživatel může předem zvolit.

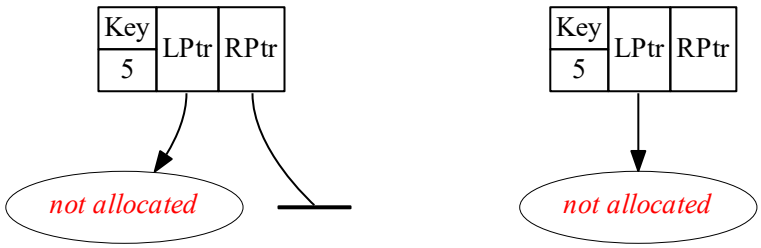
Pro popis vrcholů popisujících struktury je využíváno formátu html table, protože umožňuje formátovat jednotlivé části zobrazené struktury (například jen jednu buňku tabulky), ačkoliv se jedná o zdlouhavější zápis než alternativa. Hodnoty datových položek struktury jsou zobrazeny v tabulce, volitelně vedle názvu odpovídající položky. Ukazatele jsou zobrazeny jako orientované hrany vycházející z odpovídajícího místa v tabulce a směřující na další vrchol. Tyto hrany mohou vycházet z buňky tabulky vedle názvu ukazatele. To potom znamená, že hrany budou vycházet vždy z pravé (resp. spodní, podle orientace grafu) strany tabulky (viz. obr. 4.1). Toto ale zapříčiní zbytečně mnoho křížení hran. Je tedy lepší vytvářet co nejméně omezení pro uspořádání hran, a to úpravou, po které mohou být hrany umístěny na obou stranách tabulky.



Obrázek 4.1: Původní zobrazení ukazatelů (vlevo) a zobrazení ukazatelů v upravené tabulce (vpravo)

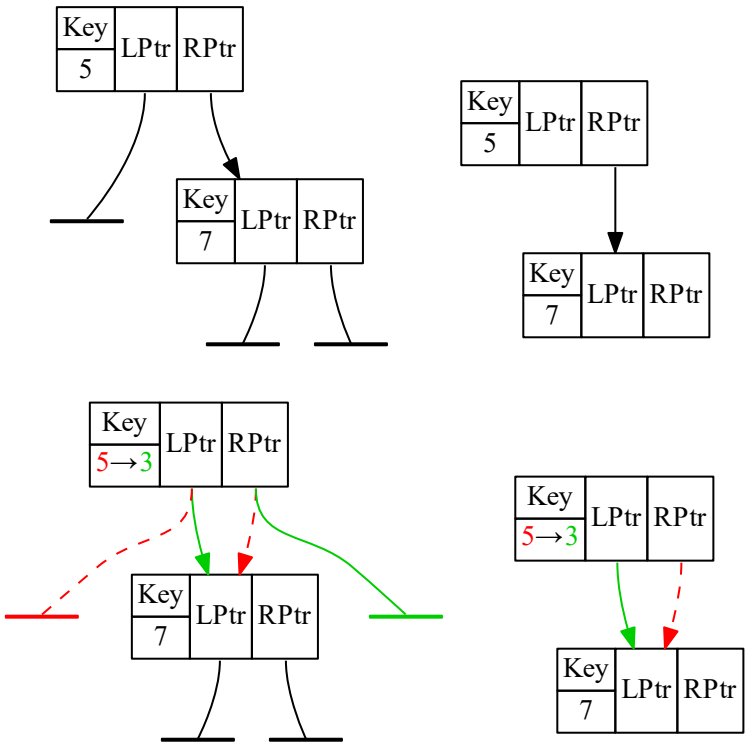
Pokud se jedná o jeden stav dynamické struktury, jsou postupně do výstupního souboru vypsány všechny vrcholy, které se následně propojí ukazateli. Zde se může také vyskytovat ukazatel, který ukazuje na nealokovanou paměť. Takový ukazatel je zobrazen hranou směřující na vrchol zvýrazněně označený jako nealokovaná paměť (*not allocated*). Ukazatele s hodnotou `NULL`, které označují konec dynamické struktury, jsou zobrazeny speciálním

vrcholem. Protože se ale v dynamické struktuře mohou vyskytovat velmi často, je možné je takové ukazatele nezobrazovat vůbec – z místa odpovídající ukazateli nevychází žádná hrana (viz obr. 4.2).

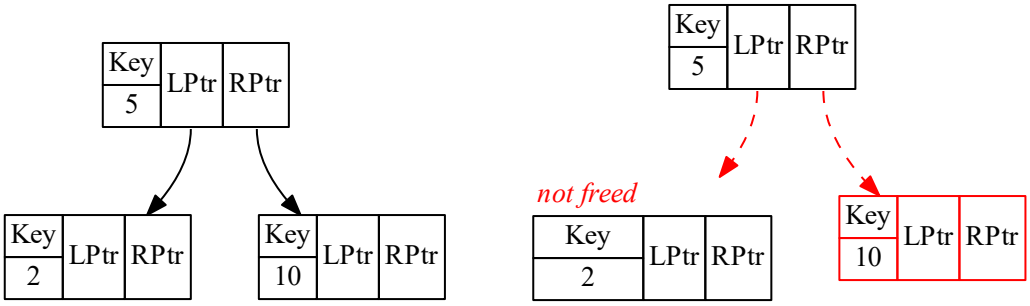


Obrázek 4.2: Zobrazení chybného ukazatele (LPtr, nenastavený na NULL) a správně ukončeného ukazatele s hodnotou `NULL` (RPtr). Vlevo se zobrazením ukazatelů s hodnotou NULL, vpravo bez něj.

Dále mohou být zobrazovány změny, které v dynamické struktuře nastaly. V takovém případě jsou nové nebo odstraněné prvky zobrazeny zvýrazněním celé tabulky. Změněné hodnoty jsou zvýrazněny v jejich buňce tabulky a změněné ukazatele jsou zobrazeny zvýrazněním hran (viz obr. 4.3). Navíc se mohou zobrazit prvky, které nebyly uvolněny (`free()`). Takové prvky jsou potom označeny zvýrazněným *not freed*(viz obr. 4.4).



Obrázek 4.3: Zobrazení změny datových hodnot a ukazatelů (dole) oproti předchozímu stavu (nahoře). Vlevo se zobrazením ukazatelů s hodnotou NULL, vpravo bez něj.



Obrázek 4.4: Zobrazení změny dynamické struktury (vpravo) oproti předchozímu stavu (vlevo) s neuvolněným prvkem (levý ukazatel) a správně uvolněným prvkem (pravý ukazatel). NULL ukazatele nejsou zobrazovány.

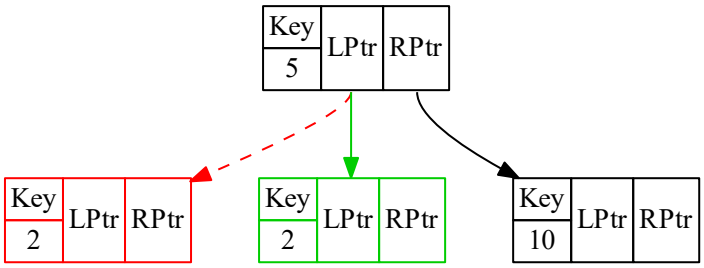
4.4 Rozdíl mezi stavy dynamické struktury

Kromě vykreslení jednoho stavu dynamické struktury nabízí knihovna také zobrazení rozdílů, které v dynamické struktuře nastaly. Třída `DDSGraph` obsahuje metodu `diff`, která ze dvou vstupních grafů vytvoří nový, který obsahuje vrcholy z obou grafů, a zaznamenává změny v jednotlivých vrcholech a hodnotách. Pokud je vrchol přítomný v obou vstupních

grafech, bude nezměněn i ve výsledném grafu. Pokud je vrchol přítomný v obou grafech, ale některá položka se změnila, ve výsledném vrcholu bude uloženy obě hodnoty (ať už datové hodnoty nebo ukazatele). Pokud je vrchol jen v jednom grafu, bude ve výsledném grafu obsahovat odpovídající příznak pro pozdější zobrazení.

Aby toto bylo možné, bylo potřeba upravit návrh interní reprezentace tak, aby každý vrchol mohl obsahovat kromě původních hodnot položek struktury i hodnoty předchozí. Také každý vrchol obsahuje informaci jestli patří do obou grafů nebo jen jednoho. Po těchto změnách je ale jak pro jeden stav tak rozdíl mezi dvěma stavy použita stejná interní reprezentace. Výchozí nastavení vrcholů a jejich položek je stejné jako při uložení samotného stavu dynamické struktury, takže zobrazení nezměněných částí dynamické struktury se provádí stejně jako při zobrazení jednoho jejího stavu.

Aby se poznalo, které vrcholy se při vytváření rozdílu mají porovnávat, porovnává se adresa původní struktury. Díky tomu se není potřeba zabývat porovnáváním grafů nebo jejich izomorfizmem. Také to znamená, že pokud dva stavy dynamické struktury obsahují stejná data, ale na různých paměťových místech, budou označeny jako rozdílné (viz obr. 4.5).



Obrázek 4.5: Zobrazení změny dynamické struktury s prvkem, který byl odstraněn a přidán

Pokud se provádí rozdíl mezi dvěma grafy, je také možné zjistit, zda byla struktura odpovídající některému vrcholu správně odstraněna – byla zavolána funkce `free()`, a paměťové místo pro tuto strukturu už není alokováno. K tomu je využíváno sledování alokované paměti popsané v kapitole 4.5. V případě, že je paměť odstraněného prvku stále alokována, je vrchol označen odpovídající chybou. Odstraněným prvkem je chápán takový, ke kterému už nevede posloupnost ukazatelů od ukazatele na strukturu, který byl předán knihovně.

4.5 Sledování alokace paměti

V jazyku C se pro dynamickou alokaci paměti a její uvolňování používají funkce `malloc`, `calloc`, `realloc` a `free`. Aby bylo možné ve vytvářené knihovně zjistit, které části jsou těmito funkcemi alokované, je potřeba každé volání těchto funkcí zachytit, a podle vrácené hodnoty označit části paměti, ke kterým je bezpečné přistoupit (dereferencí). Díky tomuto přístupu je možné zobrazit i takové dynamické struktury, které nejsou správně ukončeny ukazatelem s hodnotou `NULL`, což je častá chyba při práci s dynamickými strukturami.

Prvek dynamické struktury může však začínat na zásobníku – např. vázaný seznam, který začíná strukturou s ukazateli ukazující na důležité prvky (první, poslední, aktivní). Paměťové místo, kde je tato počáteční struktura tedy nebyla vrácena alokační funkcí. Když je ukazatel na takovou strukturu předán knihovně, musí se k ní přistoupit i přesto, že odpovídající paměť nebyla alokována. Každý ukazatel, který je předán knihovně je tedy označen za platný, a pokud odkazuje na nealokovanou paměť, nastane nedefinované chování, stejně jako v obvyklé práci s ukazateli v jazyku C. Tento přístup jsem zvolil aby byla zachována jednoduchost použití knihovny bez potřeby dalších funkcí v rozhraní.

Na linuxu je možné využít memory allocation hooks [5], což umožňuje při každém zavolání alokační funkce spustit vlastní funkci. Tato funkce může po zavolání původní alokační funkce poznačit, které části paměti byly alokovány. Na ostatních operačních systémech podobná jednoduchá možnost není, takže jsem se rozhodl tuto funkcionalitu implementovat definováním maker se stejnými jmény jako alokační funkce. Místo původních alokačních funkcí se tedy zavolají vlastní, které opět poznačí alokované části paměti. Nevýhodou tohoto přístupu je nutnost zahrnutí hlavičkového souboru knihovny do každého souboru, který využívá alokačních funkcí, ačkoliv samotné rozhraní knihovny nevyužívá.

Toto zachytávání alokačních funkcí je implementováno v `DDSvizMemory.cpp`. Aby nebyly zbytečně zachytávány alokace paměti v rámci knihovny, v případě Linuxu je na začátku každé funkce rozhraní knihovny zachytávání zrušeno a na jejím konci zase obnoveno. V případě ostatních operačních systémů nedojde vůbec k expanzi maker, protože v kódu knihovny jsou tato makra zrušena.

Po zachycení některé z alokační funkce se vrácený ukazatel uloží jako klíč do kontejneru `std::map`, který obsahuje páry klíč – hodnota, uspořádané podle klíče. Jako hodnota se uloží velikost alokované paměti k danému ukazateli. V kontejneru jsou potom tedy páry začátek alokované paměti – velikost alokované paměti od začátku. Při pokusu o čtení je potom možné zjistit, zda paměť ze které se bude číst patří do některého takového intervalu. Díky uspořádání položek v použitém kontejneru je možné rychle zjistit nejbližší začátek alokované paměti, i pokud se nejedná přesně o stejnou adresu, a potom jen otestovat, zda je alokovaná velikost dostatečná.

4.6 Závislosti

Knihovna využívá sadu nástrojů GraphViz, konkrétně nástroj dot pro zobrazení výsledného grafu. K vytvoření zobrazitelného grafu je tedy potřeba mít tento balík nainstalovaný[1]. Cesta k nainstalovaným spustitelným souborům (tedy k nástroji `dot`) musí být potom obsažena v proměnné prostředí `PATH`, aby bylo možné je spustit z příkazové řádky.

Dále je pro sestavení knihovny potřeba nástroj make. Na Linux je tento nástroj přítomný od instalace systému (případně je potřeba využít gmake). Na Windows je jednou z možností tento nástroj získat pomocí MinGW[2] a následně nainstalovat potřebný make společně s překladačem gcc a g++. Knihovna se potom sestaví spuštěním make v adresáři `/src`.

[1] dostupný pro Windows i Linux na <http://www.graphviz.org/Download..php>

[2] dostupný na <http://www.mingw.org/>

4.7 Použití knihovny

Pro využívání rozhraní knihovny je potřeba v kódu jazyka C zahrnout hlavičkový soubor knihovny `DDSviz.h`. Výsledný program je potom potřeba sestavit s knihovnou. K tomu slouží v překladači gcc přepínače `-L` pro určení cesty k adresáři obsahující knihovnu a `-l` pro určení názvu knihovny, tedy `-lDDSviz`.

Na Windows při spuštění vlastního programu stačí aby bylo možné knihovnu nalézt v některé z cest v proměnné PATH. Na Linuxu je jednou z možností určit cestu, kde se má sestavená knihovna hledat:

```
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:cesta/ke/knihovne
export LD_LIBRARY_PATH
```

4.8 Rozhraní knihovny

Pro komunikaci s knihovnou lze využít následující funkce a makra.

- Určení směru grafu. "tb"pro směr shora dolů, "lr"pro směr zleva doleva.

  ```
  setGraphDir("tr");
  ```

- Nastavení zobrazení jmen jednotlivých položek struktur.

  ```
  // jmena polozek se nebudou zobrazovat
  setShowNames(false);
  ```

- Nastavení zobrazení ukazatelů s hodnotou NULL.

  ```
  // ukazatele s hodnotou NULL se nebudou zobrazovat
  setShowNullPtrs(false);
  ```

- Nastavení začátku jména výstupního souboru. Zbytek jména souboru bude pořadové číslo grafu a přípona. Název může obsahovat pouze alfanumerické znaky a podtržítko.

  ```
  setFilenamePrefix("muj_graf");
  ```

- Nastavení formátu souboru zobrazitelného grafu.

  ```
  setOutputFormat("pdf");
  ```

- Definice struktury a jejích jednotlivých položek. Podrobnější příklady budou ukázány u jednotlivých testovaných dynamických struktur.

  ```
  // definice struktury
  newStruct(struct treeNode);
  // v existujici strukture definice polozky Key, ktera
  // se zobrazi jako bezznamenkove cislo
  newMemberData(struct treeNode, Key, UNSIGNED);
  // v existujici strukture definice polozky LPtr, ktera
  // ukazuje na danou strukturu
  newMemberPtr(struct treeNode, LPtr, struct treeNode);
  ```

- Při definici datových položek jsou možná následující zobrazení

```
UNSIGNED // zobrazeni jako bezznamenkove cislo
SIGNED // zobrazeni jako znamenkove cislo
REAL // zobrazeni jako realne cislo
BOOL // zobrazeni jako true nebo false
CHAR // zobrazeni jako znak
```

- Vykreslení dynamické datové struktury, která začíná daným ukazatelem

```
drawDDS(struct treeNode, ptr);
```

  Vykreslení změny od posledního zobrazeného grafu dynamické datové struktury, která začíná daným ukazatelem

```
drawDiffDDS(struct treeNode, ptr);
```

Kapitola 5

Testování knihovny

V této kapitole bude implementovaná knihovna použita v programech využívající dynamické datové struktury. Jedná se o programy použité v předmětu Algoritmy, kde je práce s těmito dynamickými strukturami vyučována. Budou předvedeny výstupy knihovny na jednotlivých základních dynamických strukturách typu lineární vázaný seznam a binární vyhledávací strom. Testy byly prováděny na operačním systému Windows 10 a Linux CentOS 6.9 nainstalovaném na školním serveru merlin.

5.1 Jednosměrně vázaný lineární seznam

Jednosměrně vázaný seznam je dynamická struktura, ve kterém každý prvek až na poslední odkazuje na následující prvek. Jedna z modifikací je, že součástí této datové struktury je i ukazatel, který značí právě aktivní prvek, se kterým se může dále pracovat.

V jazyku C je takovou dynamickou strukturu definovat podle výpisu 5.1. Při použití knihovny je potom potřeba na začátku programu zavolat odpovídající funkce (viz výpis. 5.2).

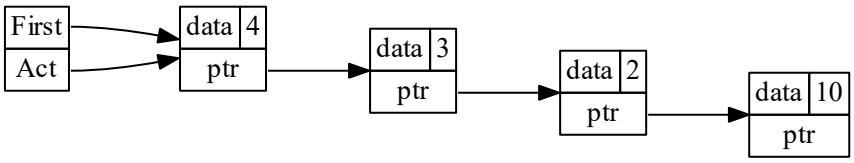
Výsledná dynamická struktura se správně zobrazí jako posloupnost prvků. Zároveň je na ní možné sledovat libovolnou operaci. Jako příklad bylo vybráno odstranění prvního prvku, kde pozici prvního prvku musí nahradit další, ale aktivitu už nepřevezme (viz obr. 5.2)

```
// prvek seznamu
typedef struct tElem {
    struct tElem *ptr;
    int data;
} *tElemPtr;

// zacatek seznamu
typedef struct {
    tElemPtr Act;
    tElemPtr First;
} tList;
```

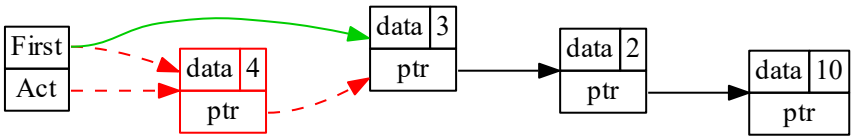
Výpis 5.1: Definice struktur pro jednosměrný seznam v jazyku C

```
newStruct(struct tElem);
newStruct(tList);
newMemberData(struct tElem, data, SIGNED);
newMemberPtr(struct tElem, ptr, struct tElem);
newMemberPtr(tList, First, struct tElem);
newMemberPtr(tList, Act, struct tElem);
```

Výpis 5.2: Definice struktur z výpisu 5.1 pro knihovnu



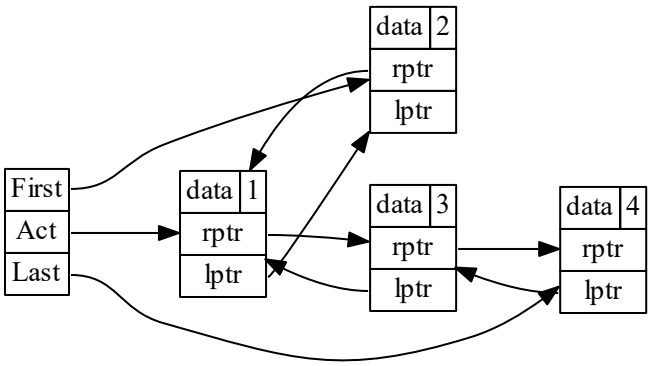
Obrázek 5.1: Zobrazení stavu jednosměrně vázaného lineárního seznamu.



Obrázek 5.2: Zobrazení změny jednosměrně vázaného lineárního seznamu (oproti předchozímu stavu z obr. 5.1), kde byl odstraněn první prvek.

5.2 Dvousměrně vázaný lineární seznam

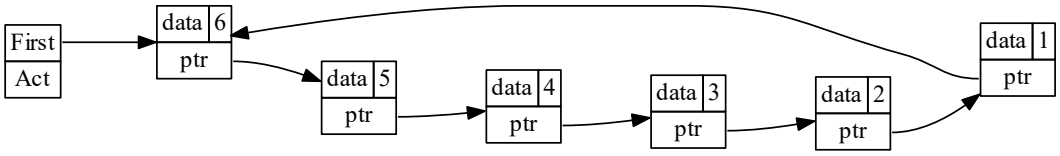
Propojení dvousměrně vázaného seznamu je velmi podobné jednosměrně vázanému seznamu. Každý prvek kromě prvního obsahuje navíc ukazatel na předchozí prvek. Použitý seznam obsahuje i odkaz na poslední prvek. Kvůli dalším ukazatelům nemusí být jasné, který prvek seznamu je první, a při zobrazení se jako první zobrazí nesprávný, což zapříčiní vizuální rozdělení seznamu (viz obr. 5.3). Pro vyřešení tohoto problému by bylo potřeba v knihovně implementovat, jakým způsobem by se jednotlivé dynamické struktury zobrazovaly, a při každé definici dynamické struktury by bylo potřeba také definovat, kterým z implementovaných způsobů se má zobrazit.



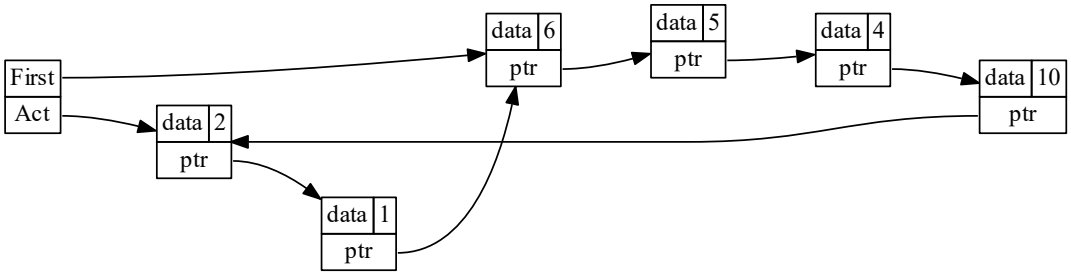
Obrázek 5.3: Zobrazení dvousměrně vázaného lineárního seznamu.

5.3 Cyklický seznam

Jednosměrný lineární seznam je možné upravit na cyklický seznam, pokud bude poslední prvek odkazovat opět na první. Při zobrazení této dynamické datové struktury se může vyskytnout malý počet křížení hran, hlavně pokud do cyklického seznamu směřuje více ukazatelů (viz obr. 5.4 a 5.5).



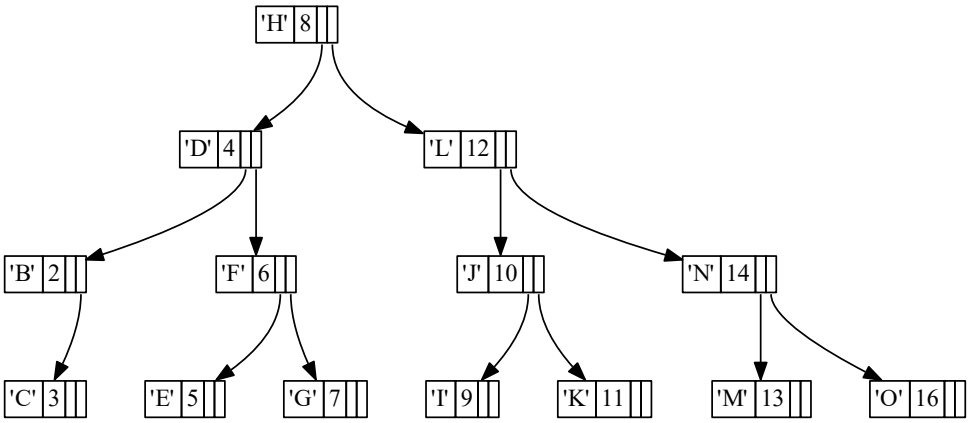
Obrázek 5.4: Zobrazení cyklického seznamu bez křížení hran.



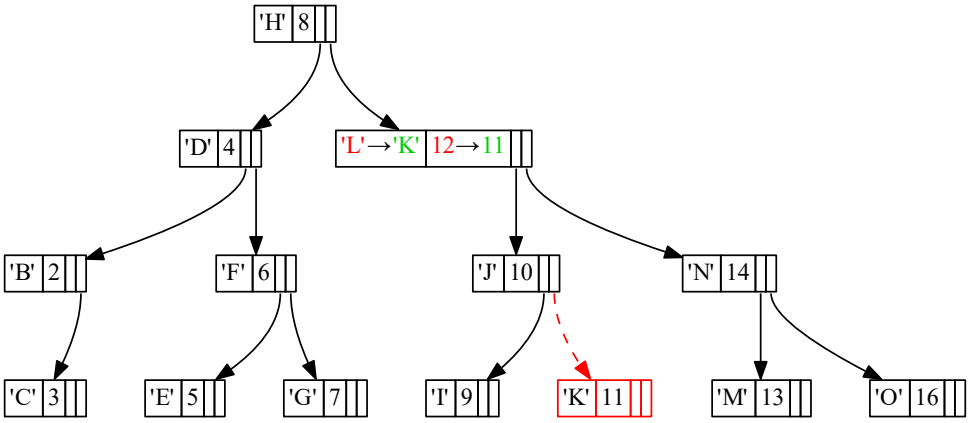
Obrázek 5.5: Zobrazení cyklického seznamu s křížením hran kvůli dalšímu ukazateli (Act).

5.4 Binární vyhledávací strom

Binární vyhledávací strom je dynamická datová struktura, ve které každý prvek obsahuje levý a pravý podstrom. Tyto podstromy můžou být i prázdné, v takovém případě se prvku říká list. Každý prvek navíc obsahuje klíč, pro který platí že je větší než každý klíč v levém podstromu a zároveň menší než každý klíč v pravém podstromu. Díky tomu je možné v této dynamické struktuře rychle vyhledávat prvky s daným klíčem. Pro zachování těchto vlastností je však potřeba například při odstraňování prvků postupovat opatrně. V obrázku 5.7 je vidět, jakým způsobem musí být upraveny jednotlivé prvky při odstraňování prvku s oběma podstromy.

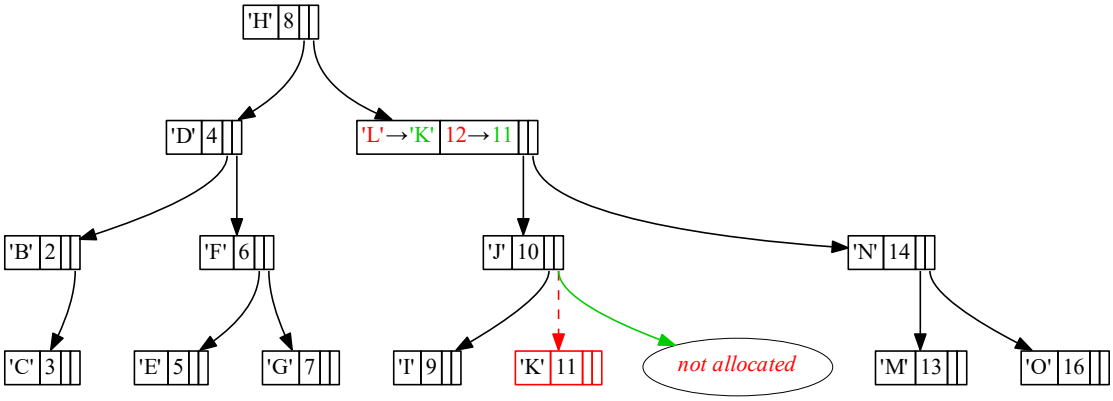


Obrázek 5.6: Zobrazení stavu binárního vyhledávacího stromu



Obrázek 5.7: Zobrazení změny binárního vyhledávacího stromu (bez názvů položek) oproti předchozímu stavu (obr. 5.6) po odstranění prvku s klíčem 'L'.

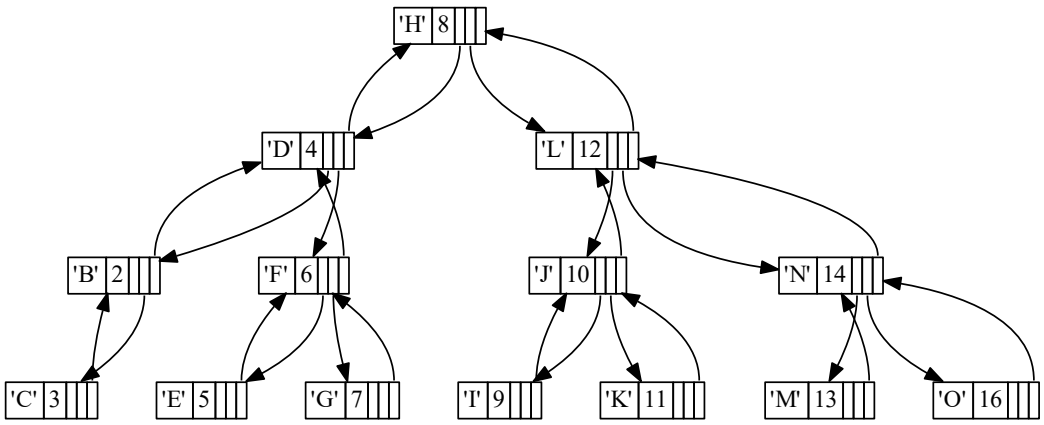
Pokud je operace odstranění chybná, například nenastaví správně ukazatele na NULL, je možné tuto chybu v zobrazeném grafu odhalit (viz obr. 5.8).



Obrázek 5.8: Zobrazení změny binárního vyhledávacího stromu (bez názvů položek) oproti předchozímu stavu (obr. 5.6) po chybném odstranění (hodnota ukazatele nenastavena na NULL) prvku s klíčem 'L'.

5.5 Binární vyhledávací strom se zpětnými ukazateli

Binární vyhledávací strom lze upravit přidáním dalších ukazatelů do každého uzlu. V této modifikaci budou přidány ukazatele na uzel předchůdce (parent pointer). Ačkoliv se kvůli většímu počtu některé hrany kříží, vzhled dynamické struktury zůstává stejný (viz obr. 5.9).



Obrázek 5.9: Zobrazení binárního vyhledávacího stromu se zpětnými ukazateli.

Kapitola 6

Závěr

Cílem práce byl návrh a implementace knihovny pro vizualizaci základních dynamických datových struktur. Inspirací podle podobně zaměřených nástrojů byl vytvořen návrh zobrazení knihovny. Knihovna dokáže přehledně zobrazit stav hlavně hierarchické dynamických datových struktur stromového typu a také různé alternativy vázaných seznamů, ale je možné zobrazit i jiné typy dynamických datových struktur, díky způsobu jejich definice v knihovně. Knihovna nabízí různé způsoby zobrazení nastavitelné v programu, podle uživatelových preferencí. Byl popsán způsob změny tohoto nastavení a také další postup pro využití knihovny v kódu jazyka C.

Kromě zobrazení stavu zmíněných dynamických datových struktur nabízí knihovna také možnost zobrazení změny, která nastala mezi dvěma stavy dynamické datové struktury. Také je díky sledování alokované paměti knihovna schopná zobrazit chybně nastavené ukazatele a neuvolněnou paměť, což jsou časté implementační chyby při práci s těmito strukturami. Díky tomu je knihovna vhodná pro pomoc při ladění programů, které dynamické datové struktury využívají.

Možným rozšířením knihovny je implementace možnosti využít ladicí informace programu pro získání popisu struktur, jako je využívají ladicí nástroje (gdb). To by zkrátilo dobu potřebnou pro nastavení knihovny a zrychlilo zobrazení nových dynamických struktur. Dalším vylepšením by byla možnost zobrazování struktur, které obsahují další složené typy – pole, union nebo další vnořené struktury. V případě, že zobrazovaný graf obsahuje části, které spolu souvisí, bylo by možné tyto části sjednotit do jednoho vrcholu, případně podgrafu, aby v případě složitějších dynamických struktur byl celý graf stále přehledný.

Literatura

[1] *Data Structure Visualizations*. [Online; navštíveno 23.04.2017].
URL https://www.cs.usfca.edu/~galles/visualization/Algorithms.html

[2] *DDD: Data Display Debugger*. [Online; navštíveno 19.04.2017].
URL https://www.gnu.org/software/ddd/

[3] *GDB: The GNU Project Debugger*. [Online; navštíveno 19.04.2017].
URL https://www.gnu.org/software/gdb/

[4] *Graphviz*. [Online; navštíveno 19.04.2017].
URL http://www.graphviz.org/

[5] *Memory allocation hooks*. [Online; navštíveno 13.04.2017].
URL
https://www.gnu.org/software/libc/manual/html_node/Hooks-for-Malloc.html

[6] *The DOT language*. [Online; navštíveno 19.04.2017].
URL http://www.graphviz.org/content/dot-language

[7] Bridgeman, S.: *Graph Drawing in Education*. Hobart and William Smith Colleges, [Online; navštíveno 12.05.2017].
URL https://cs.brown.edu/~rt/gdhandbook/chapters/education.pdf

[8] Gansner, E.; Koutsofios, E.; North, S.: *Drawing graphs with dot*. [Online; navštíveno 19.04.2017].
URL http://www.graphviz.org/Documentation/dotguide.pdf

[9] Kovár, M.: *Diskrétní matematika*. [Online; navštíveno 12.05.2017].
URL http://www.umat.feec.vutbr.cz/~kovar/webs/personal/IDA.pdf

[10] Kozo Sugiyama, M. T., Shojiro Tagawa: *Methods for Visual Understanding of Hierarchical System Structures*. *IEEE Transactions on Systems, Man, and Cybernetics*, ročník 11, 1981: str. 109–125.

[11] Patrick Healy, N. S. N.: *Hierarchical Drawing Algorithms*. University of Limerick, [Online; navštíveno 12.05.2017].
URL https://cs.brown.edu/~rt/gdhandbook/chapters/hierarchical.pdf

[12] Walker, J. Q.: *A Node-Positioning Algorithm for General Trees*. Sitterson Hall Chapel Hill, NC 27599-3175, USA: University of North Carolina at Chapel Hill, Department of Computer Science, 1989, [Online; navštíveno 23.04.2017].
URL http://www.cs.unc.edu/techreports/89-034.pdf

Přílohy

Příloha A

Obsah přiloženého paměťového média

- `/src` – zdrojové soubory knihovny a makefile pro její sestavení
- `/src/tests` – adresáře se zdrojovými soubory testovacích programů s vlastními makefile. Programy byli použiti pro ověření správné funkčnosti a zobrazování knihovny. Části zdrojových kódů a některé výstupy těchto programů byly využity v textu práce.
- `/pdf` – text této práce v elektronické podobě, barevně i verze pro tisk
- `/latex` – zdrojové soubory k vytvoření textu této práce.
- `/doxygen` – programová dokumentace vygenerovaná pomocí nástroje doxygen.
- `/README.md` – soubor se stručným popisem ostatních souborů a použití knihovny.