Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Systém na podporu výučby kódovania

Diplomová práca

2021 Bc. Peter Becza

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Systém na podporu výučby kódovania

Diplomová práca

Študijný program:	Informatika
Študijný odbor:	2508 Informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	doc. RNDr. Daniel Olejár, PhD

Bratislava, 2021 Bc. Peter Becza

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Peter Becza
Študijný program: informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Systém na podporu vyučby kódovania
Coding theory e-learning system

Cieľ: Vytvoriť sofvérový nástroj umožňujúci demonštrovať priebeh výpočtov pri kódovaní a dekódovaní samoopravných kódov, funkcie na ručné vykonávanie výpočtov a experimentovanie so samoopravnými kódmi. Cieľom je umožniť výučbu výpočtovo náročných algoritmov kódovania a dekódovania.

Kľúčové slová: samoopravné kódy, e-learning

Vedúci: doc. RNDr. Daniel Olejár, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: prof. RNDr. Martin Škoviera, PhD.

Dátum zadania: 14.12.2016

Dátum schválenia: 16.12.2016 prof. RNDr. Rastislav Kráľovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Prehlásenie: Čestne prehlasujem, že som túto diplomovú prácu vypracoval samostatne s použitím uvedenej literatúry.

Poďakovanie: Ďakujem môjmu diplomovému vedúcemu doc. RNDr. Danielovi Olejárovi PhD., za trpezlivosť a odborné rady.

Abstrakt

V tejto práci sme navrhli softvérový nástroj, pomocou ktorého je možné prezentovať priebeh výpočtov pri kódovaní a dekódovaní samoopravných kódov. Rozhranie nástroja a spôsob jeho návrhu umožňuje používateľovi použiť nástroj aj ako platformu pre implementáciu vlastných algoritmov s možnosťou ich zapojenia do súvislých celkov za účelom obojsmerného krokovania jednotlivých komponentov celku, ako aj postupného zobrazovania priebehu algoritmu jedného komponentu. Nástroj obsahuje rozhranie pre podporu náročných výpočtov pri ručnom vykonávaní výpočtov. Nástroj sme navrhli tak, aby bol ľahko upravovateľný a rozšíriteľný so zachovaním kľúčového princípu komponentovej modularity.

Kľúčové slová: samoopravné kódy, e-learning

Abstract

In this work, we have designed a software tool with it is possible to present the process of calculations for the coding and decoding of error correcting codes. The interface of the software tool and the method of its design, allows the user to use the tool as a platform for implementing their own algorithms with the possibility of their involvement in continuous units for two-way stepping of individual components of the unit, as well as gradually displaying the algorithm of one component. The tool includes an interface to support complex calculations when performing calculations manually. We designed the tool to be easily customizable and extensible while maintaining the key principle of component modularity.

Keywords: error-correcting codes, e-learning

Predhovor

Táto práca je určená záujemcom o problematiku teóriu kódovania a komunikačného kanála. Už počas môjho štúdia informatiky na univerzite vznikla otázka, či existuje nástroj, ktorý by dokázal umožniť tvorbu vlastného komunikačného kanála a simulovanie priebehov výpočtov použitých algoritmov. Ukázalo sa, že v akademickej sfére neexistuje voľne dostupný nástroj, ktorý by dokázal pokryť požiadavky na flexibilnú simuláciu algoritmov. Ak aj niekto takýto nástroj vytvoril, nie je voľne dostupný, čo študentom k nemu komplikuje prístup. Teória kódovania a pridružených tém (napr. aplikácie algebrických štruktúr) je v súčasnosti vyučovaná na viacerých vysokých školách, či univerzitách na Slovensku, a v Česku. Stránky vyučovaných predmetov sú skúpe na nástroje, a ak aj niektorý z webov predmetov obsahuje odkaz na aplikáciu, či simuláciu algoritmu, ide o krokovanie triviálnych prípadov, algoritmy obsahujú pevne zadané vstupy, prípadne ide o algoritmy s prednastavenými parametrami. Prostredníctvom tejto práce sa snažím o analýzu a implementáciu prototypu softvérového nástroja, ktorý by dokázal používateľovi poskytnúť implementáciu vlastných komponentov komunikačného kanála a ich flexibilné zapájanie do simulačných zostáv. Implementácie jednotlivých predmetných algoritmov z teórie kódovania je možné nájsť na internete v podobe voľného softvéru, preto som v tejto práci nekládol veľký dôraz na potrebu implementácie konkrétnych algoritmov. Jednou z výhod nástroja je pohodlné integrovanie algoritmov vytvorených tretími stranami prostredníctvom importu modulu, ktorý ich realizuje. Od používateľa, ktorý má záujem o viac ako len prezentačnú formu nástroja, očakávam, že je oboznámený s obsahom prednášok úvodu do teórie kódovania na niektorom z dostupných kurzov a tiež, že má absolvované základy programovania v niektorom z programovacích jazykov. Na základe týchto vedomostí by mal byť používateľ schopný pohotovo doplniť do nástroja novú funkcionalitu, prípadne zapracovať import modulov tretích strán.

Prácu som vypracoval sám, na základe rád školiteľa, vlastných skúsenosti z obdobia štúdia na univerzite, aj z praxe v zamestnaní. Počas tvorby nástroja som pri dizajne dbal na spätnú väzbu testovacích používateľov z akademického prostredia.

Obsah

Zoznam obrázkov

Kapitola 1

Úvod

"Doba kamenná sa neskončila pre nedostatok kameňa, a doba ropná sa skončí dávno predtým ako svetu dôjde ropa"[1]. V súčasnosti žijeme v období, ktoré je mnohými nazvané informačný vek[2], a vôbec nám nehrozí, že by nám došlo množstvo informácií v dátovej podobe. Práve naopak.

Kvalita a dostupnosť technológií používaných v súčasnosti sa každoročne znásobuje [3] [4], čo zapríčiňuje pomerne rýchle zmeny z nápadov na produkty a služby. Príkladom sú technológie zberu a analýzy veľkých objemov dát, hlasový asistenti rozpoznávajúci hlas používateľa transformujúci na príkazy, či rozmach sociálnych sietí, priemyslu 4.0, alebo vesmírneho priemyslu.

Všeobecné vnímanie situácie naznačuje, že sme denne zahltení veľkým množstvom rôznych údajov, a prirodzene sa preto vyvíja záujem o rozširovanie schopností uchovávania, spracovania, analýzy, interpretácie, či rozhodovania sa za pomoci týchto údajov. Viacero nezávislých zdrojov predpokladá, že trend exponenciálneho rastu množstva údajov bude pokračovať, čo so sebou prináša zvýšené nároky na spôsob ukladania údajov, ich prenos a vyhodnocovanie. Časti tejto problematiky sa venuje skupina vedných disciplín teória informácie, kódovania, komunikácie, či šifrovania.

Ďalej sa zameriame iba na komunikáciu, ako výmenu informácii medzi dvoma alebo viacerými entitami. Aby bola komunikácia efektívna a spoľahlivá, je nutné, aby prebehla vo vymedzenom čase a s vynaložením ohraničeného úsilia (energie). Spoľahlivosť informácie znamená, že odoslaná správa je prijatá v nepozmenenej podobe, resp. prijímateľ dokáže odoslanú správu presne zrekonštruovať. Hodnovernosťou správy a ďalšími bezpečnostnými aspektami sa zaoberá disciplína informatická bezpečnosť, ktorej sa v tejto práci nebudeme venovať.

Za základ teórie kódovania a informácie je mnohými považovaný článok A Mathematical Theory of Communication [5], ktorý bol publikovaný v roku 1948 a ktorého autorom je Claude Shannon. Od tohto momentu vzniklo veľké množstvo rôznej literatúry venujúcej sa týmto dvom oblastiam (napr. [6][7][8][9]). V súčasnosti sú tieto teórie prednášané na technických, či prírodovedných vysokých školách a univerzitách v rôznych podobách, buď ako samostatné predmety, alebo sú súčasťou prednášok pridružených predmetov (napr. súčasť predmetu aplikácie algebrických štruktúr).

Aj napriek pomerne rozšíreným možnostiam výučby a výskumu je náročné nájsť dobrý nástroj na simuláciu systému komunikačného kanála. Existujú rôzne implementácie predmetných algoritmov, existujú bohaté materiály na teóriu a spôsoby konštrukcie kódov, avšak nepodarilo sa nám nájsť nástroj, ktorý by bol voľne dostupný, a ktorý by umožňoval flexibilnú tvorbu vlastných simulácií komunikačného kanála či tvorbu vlastných komponentov komun. kanála. V kontexte hľadania vyhovujúceho nástroja sme navštívili web stránky fakúlt, katedier, predmeto a prednášajúcich popredných slovenských a českých univerzít vyučujúcich problematiku teórie kódovania (napr. [10][11][12][13][14]). Pre rozsiahlosť zoznamu uvádzame iba predmetné vysoké školy a univerzity: Univerzita Komenského V Bratislave, Žilinská univerzita v Žiline, Slovenská technická univerzita v Bratislave, Univerzita sv. Cyrila a Metoda v Trnave , Univerzita Pavla Jozefa Šafárika v Košiciach, Univerzita Mateja Bela v Banskej Bystrici, Univerzita Konštantína Filozofa v Nitre, Paneurópska vysoká škola v Bratislave, Masarykova univerzita v Brne a Univerzita Karlova v Prahe.

Ak sme aj našli nejaký nástroj, išlo o vizualizáciu algoritmu jedného konkrétneho kódu, resp. kódu s pevne zadanými parametrami. Toto je jedným z dôvodov vzniku tejto práce. Počas písania tejto práce sa jej pôvodné zameranie mierne vyvinulo. Práca doplňujúca a implementujúca obsah prednášok sa zmenila na všeobecnejší softvérový nástroj poskytujúci platformu pre sebarozvoj a experimentovanie z implementovanými algoritmami.

Práca je určená najmä poslucháčom prednášok teórie kódovania, ale aj študentom iných pridružených oblastí (napr. šifrovania, prenosu signálu), či laickej verejnosti so záujmom o uvedenú problematiku. Na univerzitách, resp. vysokých školách prenášajú výskumníci a odborníci na problematiku teórie informácie a kódovania a dokážu poslucháča v prípade potreby nasmerovať na ďalšiu odbornú literatúru, preto naďalej odporúčame navštevovať prednášky. Úlohou vytvoreného nástroja je poskytnúť používateľom rozhranie na prácu s komponentami a vytvorenými komunikačnými kanálmi na účely lepšieho pochopenia problematiky prednášanej na prednáškach a tiež poskytnúť platformu na skúšanie vlastných algoritmov s možnosťou jednoduchej vizualizácie.

1.1 Cieľ práce

Cieľom tejto práce je vytvoriť softvérový nástroj umožňujúci demonštrovať priebeh výpočtov pri kódovaní a dekódovaní samoopravných kódov, funkcie na ručné vykonávanie výpočtov a experimentovanie so samoopravnými kódmi. Cieľom je umožniť výučbu výpočtovo náročných algoritmov kódovania a dekódovania.

1.2 Členenie práce

Práca je rozdelená do niekoľkých kapitol, pričom každá kapitola vystupuje ako samostatná ucelená časť. V nasledujúcich odsekoch stručne popisujeme obsah každej z kapitol. V prípade potrebných odkazov z jednej kapitoly do inej je táto referencia označená doplňujúcim textom, napr. v zátvorkách.

Každá kapitola vo svojom závere obsahuje text odkazujúci na dodatočné informácie a ponúka informáciu o tom, čo sa čitateľ dozvie v nasledujúcej kapitole. Ak je v záujme čitateľa iba vybraná časť práce, v nasledujúcich odstavcoch sa môže dozvedieť čo je predmetom každej z kapitol práce.

V kapitole 2 sa venujeme stručnej teórii problematiky teórie kódovania. Kapitola obsahuje vymedzenie základných pojmov v kontexte teórie kódovania, komunikačného kanála či pojmov použitých pri realizácii praktickej časti tejto práce, teda pri vývoji softvérového nástroja s príslušnou terminológiou. V tejto kapitole popisujeme stručný úvod do oblasti komunikačného kanála a jeho vybraných komponentov.

V kapitole 3 popisujeme vybrané algoritmy použité pri kódovaní a dekódovaní a uvádzame k nim nevyhnutný matematický základ, potrebný k ich porozumeniu. Definujeme požiadavky na konštrukciu týchto algoritmov, ich parametre a časovú, resp. priestorovú zložitosť.

Kapitola 4 je venovaná funkčnej špecifikácii softvérového nástroja a úvodu do implementácie praktickej časti práce. Definuje požiadavky na aplikáciu, a zahŕňa popis princípov programovania a prístupu k problémom. Zdôvodňuje výber softvérových nástrojov. Obsahuje popis, ako je vyvinutý softvér členený na pozadí - popisuje usporiadanie zdrojového kódu a adresárovej štruktúry. Opisuje hierarchiu a rozloženie grafického používateľského rozhrania.

V kapitole 5 je uvedený popis implementácie jednotlivých komponentov softvérového nástroja. Popisuje ako aplikácia funguje pre lepšie pochopenie funkcionalít a

rozšírení.

Kapitola 6 obsahuje stručný návod na inštaláciu a slúži ako krátka príručka pre používateľa na zobrazenie už implementovaných komponentov komunikačného kanála a postup, ako si používateľ môže pridať vlastný komponent.

Práca obsahuje dodatok v podobe priloženého zdrojového kódu softvérového nástroja zaznamenaného na SD karte spolu s obsahom kapitoly 6 vo formáte PDF.

1.3 Vlastný prínos

V práci sme predstavili koncept softvérového nástroja, ktorý pokrýva niektoré z identifikovaných potrieb používateľa pre pohodlnejšie pochopenie problematiky teórie komunikácie, resp. modelu komunikačného kanála. Nástroj umožňuje svojvoľné zapájanie vopred definovaných komponentov komunikačného kanála do súvislých celkov a tiež zostavovanie vlastných modelov.

Softvérový nástroj obsahuje grafické rozhranie, ktorého potrebná funkcionalita je dostupná z hlavného okna aplikácie. Funkcie hlavného okna sú obohatené o doplňujúce informácie alebo funkcionalitu prostredníctvom vyskakovacích okien, ktoré sú samostatnými ucelenými celkami a vznikajú reakciou na akcie používateľa (napr. vloženie vstupu pre komunikačný kanál, krokovanie vizualizácie algoritmu).

Navrhnutý nástroj poskytuje možnosť ako prezentovať vyvinuté objekty vo vopred zadefinovanej forme, a taktiež môže slúžiť ako platforma pre tvorbu nových komponentov komunikačného kanála. Nástroj bol navrhnutý tak, aby zachovával niekoľko princípov tvorby softvéru. Uplatnením princípu modularity zabezpečujeme, že každý prvok aplikácie vystupuje ako uzavretý modul, ktorý je možné meniť bez vplyvu na iné moduly. Princípom otvorenosti a flexibility poskytujeme používateľovi, ktorý sa rozhodne upravovať zdrojový kód nástroja, robiť malé zmeny na už implementovaných objektoch aj bez nutnosti hlbokých programátorských vedomostí, resp. bez nutnosti poznania konkrétnych nástrojov, ktorými bola aplikácia vyvinutá.

Kapitola 2

Základné pojmy

V prvej časti tejto kapitoly, sme definovali základné pojmy využité v nasledujúcich kapitolách práce. Pojmy a výrazy, ktoré sme použili sú väčšinou zaužívanými názvami v oblasti disciplín matematickej algebry, formálnych jazykov, teórie kódovania, programovania či dizajnu softvérových aplikácii. Mnohé uvedené pojmy sú zároveň aj zaužívanými pojmami bežného jazyka, avšak bez presne zadefinovaného významu. Aby sme predišli nedorozumeniam, resp. rôznym interpretáciam, je potrebné každý pojem vysvetliť.

V druhej časti tejto kapitoly predstavujeme komunikačný kanál ako formu výmeny správ medzi zdrojom a príjemcom. Venujeme sa popisu jeho komponentov, spôsobu ich zapojenia a ich významu.

2.1 Základné pojmy, operácie a označenia

V tejto kapitole definujeme základné pojmy využívané najmä v kontexte matematického zápisu, popisu algoritmov a popisov komponentov komunikačného kanála.

Definícia 1 (Abeceda a symboly)**.** *Pod pojmom abeceda nazývame ľubovoľnú neprázdnu konečnú množinu, ktorej prvkami sú znaky, resp. symboly. Abecedu budeme označovať symbolom* Σ *(sigma). V prípade potreby použitia viacerých abecied, budeme ich rozlišovať rôznym spodným indexom. Napr.* Σ_1, Σ_2. *Symboly budeme označovať malými písmenami zo začiatku abecedy.*

Definícia 2 (Slovo)**.** *Slovo nad abecedou* Σ *bude ľubovoľná konečná postupnosť znakov z abecedy* Σ. *Pre zjednodušenie zápisu a čítania, budeme jednotlivé symboly jedného slova zapisovať bez medzier a rozdeľovacích znakov. Napr.* $u = a_1, a_2, ..., a_n$, $v = abcde$, $w =$

011001. *Slová budeme označovať malými písmenami z konca abecedy, napr. u,v,z. Ak postupnosť neobsahuje žiadne symboly, postupnosť je nazývaná prázdne slovo, budeme ho označovať symbolom ϵ.*

Definícia 3 (Dĺžka slova)**.** *Nech w je ľubovoľné slovo nad abecedou Σ, potom dĺžka slova je počet symbolov, ktoré dané slovo obsahuje. Dĺžku slova w budeme označovať ako $l(w)$, resp. $len(w)$. Prázdne slovo má dĺžku 0.*

Definícia 4 (Podslovo, prefix, suffix, reverz)**.** *Podslovo slova $w = a_1, a_2, ..., a_n$ je ľubovoľná súvislá podpostupnosť $a_i, a_{i+1}, ..., a_j$, pričom platí $1 <= i <= j <= n$. Ak $i = 1$ hovoríme, že ide o počiatočné podslovo - prefix, a pre prípad $j = n$ hovoríme, že ide o koncové podslovo - suffix. Ak preusporiadaním prvkov slova $u = a_1, a_2, ..., a_{n-1}, a_n$ dostaneme postupnosť $v = a_n, a_{n-1}, ..., a_2, a_1$ s opačným usporiadaním prvkov, hovoríme, že v je reverz slova u, označíme ho ako u^R a platí $u^R = v$, $v^R = u$.*

Definícia 5 (Jazyk)**.** *Jazykom L nad abecedou Σ budeme označovať ľubovoľnú množinu slov vytvorenú nad abecedou Σ.*

2.1.1 Operácie nad slovami a jazykmi

Definícia 6 (Zreťazenie)**.** *Zreťazenie slov $u = a_1, a_2, ..., a_{n-1}, a_n$, a $v = b_1, b_2, ..., b_n$ je také slovo w, ktoré vznikne zapojením postupnosti symbolov slova v za postupnosť symbolov slova u, do súvislej postupnosti w. Teda $w = uv = a_1, a_2, ..., a_n, b_1, b_2, ..., b_n$. Novo vzniknutá postupnosť je tiež slovo a platí, že dĺžka slova w je súčet dĺžok slov u a v, $l(w) = l(u) + l(v)$. Neutrálnym prvkom vzhľadom na operáciu zreťazenia je prázdne slovo, a ktoré platí vzťah $w\epsilon = \epsilon w = w$.*
Nech L_1 a L_2 sú jazyky, u je ľubovoľné slovo jazyka L_1 a v je ľubovoľné slovo jazyka L_2, potom zreťazenie jazykov L_1 a L_2 je jazyk L, ktorého slová vznikli zreťazením všetkých možných slov u a v.

Definícia 7 (Umocnenie)**.** *Ľubovoľný jazyk L je možné zreťazovať so sebou samým. Nech k je prirodzené číslo, potom $L^0 = \epsilon$ a $L^{k+1} = L^k L$*

Definícia 8 (Iterácia)**.** *Nech L je ľubovoľný jazyk, a platí $L^* \cup_{i=0}^{\infty} L^i$, potom jazyk L^* nazývame iterácia a bude obsahovať všetky slová, ktoré je možné dostať zreťazením niekoľkých slov z L. Špeciálnym prípadom je kladná iterácia, pre ktorú platí vzťah $L^* \cup_{i \leq 1}^{\infty} L^i$*

Spomenuté operácie nad jazykmi sú viackrát použité v ďalších kapitolách tejto práce. Vzhľadom na to, že jazyk je množina slov, je možné aj nad jazykmi definovať

ďalšie pokročilejšie operácie, napr. prienik, zjednotenie, rozdiel, doplnok, a iné. Ak má čitateľ záujem o túto problematiku, odkazujeme ho na disciplínu teoretickej informatiky nazvanej formálne jazyky, štúdiom ktorej sa môže dozvedieť viac o vlastnostiach jazykov a modelov, ktoré ich opisujú.

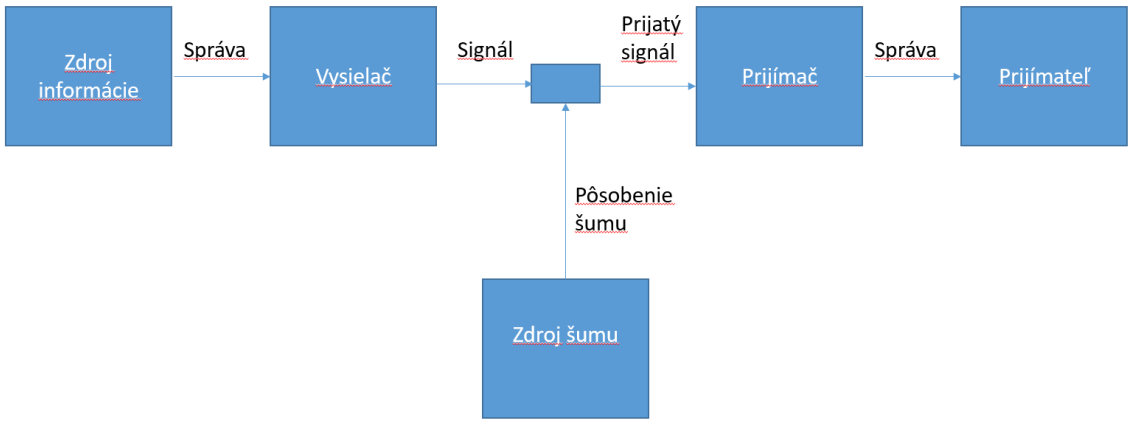
2.1.2 Komunikácia

Definícia 9 (Komunikácia)**.** *V tejto práci pod pojmom komunikácia definujeme interakciu medzi dvomi entitami, odosielateľom (zdroj) a príjemcom, prostredníctvom komunikačného kanála. Interakcia predstavuje výmenu správ, nesúcich údaje, ktorých obsahom je informácia.*

Definícia 10 (Správa)**.** *Pod správou rozumieme formu prenosu údajov. Správu je môžeme posudzovať z niekoľkých aspektov: sémantiky, syntaxe a praktického využitia. V tejto práci sa budeme venovať hlavne syntaxe, formálnemu vyjadreniu správy. Správu bude tvoriť postupnosť znakov nad abecedou, resp. slovo z jazyka.*

Definícia 11 (Kód, kódovanie a dekódovanie)**.** *Zobrazenie, ktoré množine znakov nad abecedou Σ_1, resp. slov nad jazykom L_1 jednoznačne priradí množinu znakov nad abecedou Σ_2, resp. slov nad jazykom L_2 budeme nazývať kód. Postup (algoritmus) tvorby kódu nazveme kódovanie a k nemu opačný postup označíme za dekódovanie.*

2.2 Komunikačný kanál



Obr. 2.1: Shannonov model komunikačného kanála

Vzhľadom na vysokú zložitosť problematiky teórie informácie, kódovania a komunikácie sa v tejto práci obmedzíme na zovšeobecnený model komunikačného systému, ktorý bol odvodený z pôvodného systému, ktorý publikoval Claude E. Shannon v roku 1948 [5] (obrázok 2.1).

Nami ďalej používaný model bude pracovať s existenciu nasledujúcej množiny komponentov: zdroj údajov, kodér K_1, kodér K_2, prenosový kanál, na ktorý môže pôsobiť komponent zdroj šumu, dekodér samo-opravných kódov, dekodér správy, príjemca, pričom sme si vedomí toho, že ide o veľmi zovšeobecnený model, ktorý predpokladá absenciu mnohých iných komponentov ako napríklad modulátor, vysielač, prijímač, demodulátor, komponenty zabezpečujúce šifrovanie a dešifrovanie správy, a iné.

Definícia 12 (Zdroj)**.** *Bez ujmy na všeobecnosti budeme o komponente Zdroj uvažovať ako o entite, ktorá je zdrojom informácie. Správu bude generovať v podobe postupnosti znakov, resp. slov nad abecedou zdroja* Σ*.*

Definícia 13 (Kodér K_1)**.** *Úlohou zdroja je efektívny zápis informácie. Tento zápis spočíva v úlohe priradiť symbolom zdrojovej abecedy konkrétne slová pozostávajúce z kódovacej abecedy, tak aby sa minimalizovala redundancia použitých symbolov. Tieto symboly kódujeme tak, aby symboly, ktoré sa vyskytujú častejšie boli reprezentované krajšími kódovými slovami. Spôsob akým je správa zdrojom vygenerovaná, nezaručuje, že je táto správa reprezentovaná efektívne. Pre ďalšie spracovanie je preto vhodné túto správu pred odoslaním spracovať, a to kódovaním zdrojovej informácie, pod čím budeme rozumieť redukciu redundancie znakov zdrojovej postupnosti. Výsledkom práce kodéra* K_1 *je postupnosť nad abecedou kódovej abecedy* Σ_C*, ďalej len správa. Úlohou kodéra* K_1 *je dosiahnutie efektívneho zápisu zdrojovej informácie, v optimálnom prípade ide o čo najkratšiu správu nad príslušnou kódovou abecedou. Požiadavku na efektívnosť vyjadríme ako rovnakú pravdepodobnosť výskytu každej k-tice výslednej kódovej abecedy. Na základe takéhoto kódovania by malo byť možné správu v plnom rozsahu dekódovať do pôvodnej podoby. Pri kódovaní stroja rozlišujeme bezstratovú a stratovú kompresiu. V tejto práci sa budeme venovať výlučne bezstratovej kompresii, čiže žiadna zo zdrojových informácií nebude odfiltrovaná či iným spôsobom zanedbaná.*

Definícia 14 (Kodér K_2)**.** *Úlohou komponentu kodér* K_2 *je pripraviť správu na poslanie prostredníctvom komunikačného kanála. Pri prenose hrozí, že správa bude deformovaná a preto kodér* K_2 *transformuje správu dĺžky* k *nad kódovou abecedou* Σ_C *na správu dĺžky* n *nad rovnakou abecedou* Σ_C *tak, aby sa čo najmenším počtom pridaných znakov vytvorila schopnosť odhaliť, prípadne aj opraviť prípadné chyby, ktoré môžu pri prenose prostredníctvom prenosového kanálu nastať. Ďalej budeme hovoriť o kodéri bez pamäte, čo znamená, že kodér neberie do úvahy vzťahy medzi kódovými slovami. Budeme hovoriť, že kodér* K_2 *realizuje zobrazenie* $\Sigma_C^k \to \Sigma_C^n$*. Toto zobrazenie je injektívne, čo znamená,*

že slovo zdrojovej abecedy w_z ($l(w_z) = k$) sa vždy transformuje rovnako na to isté kódové slovo w_z ($l(w_z) = n$)

Definícia 15 (Prenosový kanál a šum)**.** *Kódové slovo, ktoré je na výstupe z kodéra K_2 je pred prenosom potrebné pripraviť do podoby signálu, čo je fyzikálna reprezentácia správy. V tejto práci nebudeme rozoberať, akým spôsobom sa správa transformuje na signál, ani akým spôsobom vysielač generuje signály s dostatočnou silou, aby boli doručené do prijímača. Tejto problematike sa bližšie venuje teória spracovania signálu. O prenosovom kanály budeme hovoriť ako o abstraktnej entite, ktorou prechádza správa bez ujmy na transformácii. V reálnom svete na prenosový kanál môže vplývať okolité prostredie, či kvalita média, na ktorom je správa prenášaná. Avšak, aby sme aspoň z časti mohli simulovať vznik chyby pri prenose, a tým demonštrovať potrebu zavedenia kodéra K_2, budeme uvažovať o komponente šum, ktorý môže vplývať na prenosový kanál takým spôsobom, že niektoré zo znakov prenášanej správy pozmení. Šum pre naše potreby bude simulovať nahradenie symbolu, alebo skupiny symbolov prenášanej správy, iným symbolom zo zhodnej kódovej abecedy. Následne, po tom, čo je správa prenesená prostredníctvom prenosového kanála, je prijatá prijímačom, z ktorého putuje do demodulátora. Problematikou prijímača a demodulátora sa v tejto práci nebudeme zaoberať.*

Definícia 16 (Dekodér D_2)**.** *Prijatú správu je potrebné čo najlepšie rekonštruovať. Za predpokladu, že poznáme všetky kódové slová u_i, všetky prijaté slová v_j a podmienené pravdepodobnosti $p_{i,j} = p(v_j|u_i)$, kde $p_{i,j}$ je pravdepodobnosť s akou bolo prijaté slovo v_j po odvysielaní slova u_i). Komponent K_2 dekóduje na základe maximálnej pravdepodobnosti, to znamená, že prijaté slovo v_j dekódujeme na také slovo u_i, pre ktoré platí, že podmienená pravdepodobnosť $p_{i,j}$ je najväčšia. Výsledkom takéhoto dekódovania je tzv. úplné dekódovanie, pretože zabezpečuje, že výsledkom je existujúce kódové slovo. Neúplné dekódovanie nastane v prípade, že dekodér nie je schopný chybu opraviť, aj napriek tomu, že ju našiel. Ak nastala taká chyba, že zmenila jedno kódové slovo na iné, tak ju dekodér nie je schopný odhaliť, preto sa ďalej budeme venovať dekodérom realizujúcim neúplnému dekódovanie na základe maximálnej pravdepodobnosti, čo znamená, že prijaté slovo v_j dekódujeme na také slovo u_i, pre ktoré platí, že podmienená pravdepodobnosť $p_{i,j}$ je najväčšia, alebo na symbol ∞.*

Vzhľadom na vyššie popísané okolnosti nemôžeme garantovať správne dekódovanie prijatej správy. Ak dekodér nesprávne dekódoval prijaté slovo, alebo interpretoval prijaté slovo ako iné kódové slovo, budeme hovoriť o chybe dekodéra. Tento prípad zahŕňa taktiež situáciu, kedy dekodér odhalil chybu, ale nebol schopný ju opraviť.

Definícia 17 (Dekodér D_1)**.** *Úlohou dekodéra D_1 je transformovať dekódovanú správu do vhodnej podoby pre príjemcu, tak aby čo najlepšie odzrkadľovala zdrojovú informáciu.*

Definícia 18 (Príjemca). *Pod pojmom príjemca budeme rozumieť entitu, ktorej bola zdrojová informácia určená, a ktorá je prijímateľom prenesenej správy.*

2.3 Ďalšie pojmy

Definícia 19 (Rozhranie). *Pojem zadefinujeme ako hranicu medzi dvoma prostrediami. V prípade softvérového nástroja pod pojmom používateľské rozhranie (UI - User Interface) rozumieme spôsob, ako používateľ komunikuje so softvérom, čiže ako je realizovaná výmena informácie medzi používateľom a aplikáciou. V prípade grafického používateľského rozhrania hovoríme o okennej aplikácii, ktorá obsahuje grafické prvky, ktoré intuitívne poznáme z bežnej práce so softvérom - textové polia, tlačidlá, menu, či plátno, na ktoré je možné vykresľovať geometrické prvky. Medzi ďalšie používateľské rozhrania patrí napríklad príkazový riadok (príkazy sú písané do riadku, výstup je realizovaný výpisom textu na obrazovku) alebo menu (celá aplikácia je ovládaná menu rozhraním a všetky situácie, ktoré môžu nastať sú obmedzené možnosťami menu, príkladom je rozhranie bankomatov alebo automat na lístky).*

Definícia 20 (Entropia). *Pojem entropia zaviedol (v súvislosti s teóriou informácie) Claude E. Shannon v roku 1948, preto je občas v literatúre uvádzaná aj ako shannonovská entropia. Vyjadruje strednú hodnotu množstva informácie pripadajúceho na jeden znak, resp. symbol generovaným zdrojom dát. Mieru entropie vypočítame vzťahom* $-\sum_{i=1}^{n} p_i + log(p_i)$ *, kde* p_i *je pravdepodobnosť výskytu symbolu* i*. Entropiu zvyčajne označujeme symbolom* H*.*

Definícia 21 (Cena kódu). *Nech* P *je* $P = p_0, p_1, ..., p_{m-1}$ *rozdelenie pravdepodobnosti znakov zdrojovej abecedy* $\Sigma_S = a_0, a_1, ..., a_{m-1}$ *a nech* V *je kód kódujúci znaky kódovej abecedy* $a_i \rightarrow v_i$ *pre* $i = 0, 1, ..., m-1$*, potom cenou kódu* V *označíme vzťah* $L(P, V) = \sum_{i=0}^{m-1} l(a_i)p_i$*. Cenu kódu môžeme interpretovať aj ako strednú hodnotu dĺžky kódového slova.*

Definícia 22 (Kraft-McMillanova nerovnosť). *Nech* $l_0, l_1, ..., l_{m-1}$ *sú ľubovoľné nenulové prirodzené čísla. Potom rozdeliteľný kód* $V = v_0, v_1, v_{m-1}$ *s dĺžkami slov* $l_i = l(v_i)$*, pre* $i = 0, 1, ..., m-1$ *existuje práve vtedy, keď platí nasledujúca nerovnosť* $\sum_{i=0}^{m-1} 2^{-l_i} \leq 1$ *.*

Definícia 23 (Prefixový kód). *Kód* $V = v_0, v_1, ..., v_{m-1}$ *nazývame prefixovým kódom, ak pre ľubovoľné* v_i *a* v_j *z* V*, kde* $v_i \neq v_j$*, platí, že* v_i *nie je prefixom slova* v_j*. Inými slovami, platí, že žiadne z kódových slov nie prefixom iného kódového slova.*

Definícia 24 (Hammingova váha). *Pod pojmom Hammingova váha vektora budeme rozumieť počet nenulových symbolov daného vektora.*

Definícia 25 (Hammingova vzdialenosť)**.** *Pod pojmom Hammingova vzdialenosť budeme rozumieť počet symbolov, v ktorých sa dve slová, rovnakej dĺžky, líšia.*

Kapitola 3

Vybrané algoritmy

V tejto kapitole spomenieme vybrané algoritmy, ktoré sme v súvislosti s touto prácou implementovali. Ide o algoritmy prednášané na väčšine kurzov teórie kódovania. Predpokladom pre pochopenie problematiky tejto kapitoly je, že čitateľ bol oboznámený so základmi teórie kódovania (napr. Kraftova-McMillanova nerovnosť, cena kódu, a iné), rozlišuje rozdiely medzi rovnomernými a nerovnomernými kódmi. Pre jednoduchosť predpokladáme, že kódová abeceda bude binárna.

3.1 Kódovanie zdroja

V tejto kapitole si predstavíme 2 známe kódy, pri tvorbe ktorých sa využíva znalosť pravdepodobnosti výskytu jednotlivých kódovaných znakov. Fanov a Huffmanov kód. Oba kódy pracujú so zoznamom symbolov a počtom ich výskytov, resp. pravdepodobnosťou ich výskytu.

3.1.1 Fanov kód

Fanov kód patrí do triedy kvázi-optimálnych rozdeliteľných kódov. Kód je pomenovaný po Robertovi Fanovi a prvýkrát bol publikovaný v roku 1949. Výstup algoritmu nemusí byť vždy nutne optimálny, teda najkratší možný. Kód sa v praxi používa napríklad pri bezstratovej kompresii údajov do formátu ZIP.

Parametre

Algoritmus neobsahuje žiadne parametre.

Konštrukcia kódu

Pre konštrukciu kódu predpokladáme, že na vstupe je dané rozdelenie pravdepodobnosti $P = p_0, p_1, p_{m-1}$. Ak toto rozdelenie nie je k dispozícii, pre správu vytvoríme zoznam symbolov a pre každý symbol vypočítame pravdepodobnosť jeho výskytu (napríklad z počtu výskytov daného symbolu)

Kroky konštrukcie kódu:

1. Usporiadame pravdepodobnosti výskytu jednotlivých znakov zostupne, do nerastúcej postupnosti $p \geq p_0 \geq p_1 \geq p_{m-1} \geq 0$. Do prvého stĺpca tabuľky pre každý symbol priradíme prázdny znak λ.
2. Ak tabuľka obsahuje aspoň 2 riadky, rozdelíme ju na 2 časti tak, aby súčet pravdepodobností v hornej časti tabuľky bol čo najviac podobný so súčtom pravdepodobností v spodnej časti tabuľky. Ak tabuľka obsahuje iba jeden riadok, výpočet ukončíme.
3. Kódové slová zreťazme sprava v hornej časti tabuľky so symbolom 0, slová v spodnej časti tabuľky so symbolom 1. Rekurzívne pokračujeme krokom 2.

Časová a pamäťová zložitosť algoritmu

Fáza získania početnosti výskytov znaku je lineárne závislá od dĺžky vstupu, usporiadanie znakov je závislé na algoritme usporiadania a generovanie výstupu je opäť lineárne závislé od dĺžky vstupu. Časová zložitosť algoritmu je teda $O(n + |\Sigma| * log|\Sigma|)$. Pamäťová zložitosť algoritmu: $O(|\Sigma|)$.

3.1.2 Huffmanov kód

Huffmanov kód je stromový rozdeliteľný optimálny kód, ktorý dostal názov podľa svojho tvorcu David. A. Huffmana, ktorý ho publikoval v roku 1951 počas štúdia na MIT. Huffmanovou konštrukciou dostaneme vždy optimálny (najkratší možný) kód).

Parametre

Algoritmus neobsahuje žiadne parametre.

Konštrukcia kódu

Pre konštrukciu kódu predpokladáme, že na vstupe je dané rozdelenie pravdepodobnosti $P = p_0, p_1, p_{m-1}$. Ak toto rozdelenie nie je k dispozícii, pre správu vytvoríme zoznam symbolov a pre každý symbol vypočítame pravdepodobnosť jeho výskytu (napríklad z počtu výskytov daného symbolu)

Kroky konštrukcie kódu:

1. Usporiadame pravdepodobnosti výskytu jednotlivých znakov zostupne, do nerastúcej postupnosti $p \geq p_0 \geq p_1 \geq p_{m-1} \geq 0$.
2. Posledným dvom znakom s najmenšími pravdepodobnosťami priradíme kódový znak 0, druhému 1. Následne tieto 2 hodnoty sčítame, odstránime zo zoznamu a novú hodnotu priradíme do zoznamu.
3. Krok 2 opakujeme, kým neostal posledný znak s pravdepodobnosťou 1.
4. Kódové slová dostaneme tak, že zreťazíme všetky kódové znaky v opačnom poradí ako sme ich priraďovali

Časová a pamäťová zložitosť algoritmu

Fáza získania početnosti výskytov znaku je lineárne závislá od dĺžky vstupu, následné usporiadanie znakov je závislé na algoritme usporiadania a generovanie výstupu je opäť lineárne závislé od dĺžky vstupu. Časová zložitosť algoritmu je teda $O(n + |\Sigma| * log|\Sigma|)$ Pamäťová zložitosť algoritmu: $O(|\Sigma|)$

Dekódovanie

Keďže oba kódy sú prefixové (pre každé slovo platí, že nie je prefixom iného slova), dekódovanie je priamočiare a jedinou nutnou podmienkou je vedomosť o kódovacej tabuľke obsahujúcej informáciu. ktoré kódové slovo zodpovedá akému symbolu. Postupným iterovaním cez postupnosť slov je možné priamo zapisovať na výstup príslušné symboly

p;vodnej správy preto má algoritmus časovú aj pamäťovú zložitosť lineárnu v závislosti od dĺžky vstupu.

3.2 Samoopravné kódy

Princíp samoopravných kódov je založený na tom, že k pôvodnej správe pripojíme tzv. kontrolné bity, ktoré síce pre prijímateľa nemajú informačnú ani obsahovú hodnotu, umožňujú nám však overiť, či posielaná správa bola pozmenená. Ak prijatá správa svojou štruktúrou nezodpovedá systému kódovania, vieme detegovať chybu, prípadne zistiť, aká bola najpravdepodobnejšia správa.

3.2.1 Paritný kód

Pridáme ku každému vektoru dĺžky n, jeden bit navyše, tak aby počet bitov vo vektore n+1 bol párny. Kódové slová budú rozšírené o jeden bit, ktorý bude prispôsobený na základe n informačných bitov. Doplnený bit nazveme paritným, resp. kontrolným bitom. Takéto kódovanie dokáže odhaliť chyby nepárnej dĺžky, resp. nepárny počet chýb. Ak pri prenose nastane párny počet chýb, algoritmus nezhodu neodhalí a pokladá prijaté slovo za správne.

Kroky konštrukcie kódu:

1. Iteruj cez postupnosť symbolov vektora a počítaj paritu postupnosti.
2. Ak je vektor párny, pridaj na koniec vektora symbol 0, inak pridaj symbol 1.

Dekódovanie

Postupnou iteráciou symbolov vektora počítaj paritu vektora. Ak je výsledná parita párna, vektor pokladáme za správny. V prípade nepárnej parity hovoríme, že sme detegovali chybu.

Algoritmus nedokáže odhaliť presnú pozíciu chyby, dokáže len informovať, že chyba nastala (ak je počet chýb nepárny). Ak by nastal párny počet chýb, algoritmu chybu neodhalí.

Časová a pamäťová zložitosť algoritmu

Výpočet parity postupnosti symbolov je lineárne závislá od dĺžky vstupu. Rovnaký postup na určenie parity je uvedený pri kódovaní aj dekódovaní. Taktiež pamäťové nároky na uloženie postupnosti symbolov sú priamo závislé od dĺžky vstupu. Časová zložitosť algoritmu je teda $O(n)$. Pamäťová zložitosť algoritmu: $O(n)$.

3.3 Šum

Šum pre nás v tejto práci predstavuje realizáciu dôsledkov vplyvu vonkajších faktorov na prenosový kanál. Dôsledok vplyvu bude pre nás znamenať, že niektoré zo znakov prenášanej správy budú pozmenené. Šum pre naše potreby bude simulovať nahradenie symbolu, alebo skupiny symbolov prenášanej správy, iným symbolom zo zhodnej kódovej abecedy.

Parametre

Pre algoritmus budeme rozlišovať 2 parametre. Celkový počet chýb, ktoré chceme vyvolať na prenášanej správe (označíme n) a boolovskú hodnotu zhluk (označíme burst) chyby. Ak bude požadovaný zhluk chýb, tak sa v správe zmení postupnosť za sebou idúcich n znakov. V prípade, že zhluk chýb nie je vyvolaný, budú pozmenené symboly na náhodných pozíciach správy.

Kroky konštrukcie komponentu:

1. Načítaj vstupnú postupnosť a parametre šumu.
2. Ak je požadovaný zhluk chýb, získaj index náhodnej pozície správy a uprav postupnosť nasledujúcich n symbolov,tak, že hodnoty budú nahradené iným znakom kódovej abecedy, inak opakuj n krát vyhľadanie náhodnej pozície správy a symbol na tejto pozícii nahraď iným znakom kódovej abecedy.
3. Vráť na výstupe pozmenenú správu.

Dekódovanie

Pre komponent šumu neexistuje inverzná implementácia. každá transformácia vstupu je nenavrátiteľná.

Časová a pamäťová zložitosť algoritmu

Časová zložitosť priamo závisí od počtu chýb, ktoré používateľ požaduje vykonať na vstupnej postupnosti symbolov. Algoritmus určovania náhodnej pozície symbolov určených na zmenu je $O(1)$[15]. Časová zložitosť algoritmu je teda $O(n)$. Pamäťová zložitosť algoritmu: $O(n)$.

Kapitola 4

Funkčná špecifikácia nástroja

V tejto kapitole sme sa venovali požiadavkám a funkčnej špecifikácii softvérového nástroja. Táto špecifikácia popisuje používateľské a funkčné požiadavky prvej verzie softvérového nástroja na podporu výučby teórie kódovania. Špecifikácia je určená hlavne pre toho, kto nástroj implementoval, alebo bude mať v budúcnosti záujem o pochopenie fungovania nástroja, resp. o pochopenie toho, na základe čoho boli vykonané rozhodnutia, ktoré nasmerovali prácu do jej finálnej podoby.

4.1 Rozsah a funkcie systému

Systém pre podporu výučby kódovania, bude vo svojej prvej verzii poskytovať funkcie na pridávanie vopred implementovaných komponentov komunikačného kanála do zoznamu dostupných elementov. Zo zoznamu komponentov si používateľ bude môcť vybrať ľubovoľnú podmnožinu komponentov, ktoré následne dokáže v ľubovoľnom poradí zaradiť do náväznej postupnosti (komunikačného kanála). Používateľ by mal mať v dispozícii rozhranie na pridávanie vlastných komunikačných kanálov do zoznamu všetkých kanálov, resp. možnosť ľubovoľný kanál zo zoznamu odstrániť.

Úlohou tejto verzie je vytvoriť potrebný základ pre nasledujúce verzie systému, ktorých smerovanie bude zamerané na prídavnú funkcionalitu, rozšírenie možností pridávania viacerých druhov komponentov komun. kanála, či efektívnejšie vizualizovať priebeh výpočtov jednotlivých komponentov.

Používateľ, ktorý má záujem len o krokovanie a vizualizáciu vopred implementovaných algoritmov (komponentov), bude pracovať s používateľským rozhraním aplikácie výlučne prostredníctvom grafických prvkov ako napríklad menu, tlačidlá či textové pole

na zadanie textovej vstupu konála, prípadne parametrov komponentov kanála.

Používateľ, ktorého cieľom je pridať novú funkcionalitu, resp. rozšíriť aplikáciu o nové algoritmy a komponenty, je vyzvaný k úprave zdrojového kódu aplikácie. Jeden z predpokladov je, že sa používateľ bude riadiť princípmi implementácie popísanými v nasledujúcich kapitolách tejto práce.

4.2 Opis systému

V tejto kapitole sme na základe popisu rozsahu nástroja jeho funkcií pomenovali potrebné súčasti systému s doplnením požiadavky na ich vzájomné prepojenie. Vzhľadom na potrebu vizualizácie, rozhodli sme sa pre jednoduché používateľské rozhranie, ktoré by malo obsahovať niekoľko prvkov popísaných v nasledujúcich samostatných podkapitolách.

4.2.1 Menu

Menu, slúži pre navigáciu v rámci aplikácie. Menu sme navrhli tak, aby obsahovalo len takú časť funkcionality, ktorú si myslíme, že používateľ bude intuitívne očakávať. Napr. myslíme si, že funkcie na pridanie nového komponentu či komun. kanála sú primeranou funkcionalitou obsiahnutou v menu, avšak funkcie na krokovanie algoritmov alebo navigáciu medzi jednotlivými komponentami konkrétneho komunikačného kanála nie sú v menu potrebné. Menu by taktiež malo slúžiť na navigáciu ku súčastiam nástroja, ktoré nie sú iným spôsobom pre používateľa dosiahnuteľné, napríklad O práci, Dokumentácia.

4.2.2 Komunikačný kanál

Pre skupinu prvkov komunikačného kanála sme identifikovali, že by mala obsahovať niekoľko elementov, ktoré budú nielen svojou funkcionalitou, ale jednoznačne aj svojím usporiadaním, podporovať prácu s komunikačným kanálom. Túto skupinu sme rozdelili na 2 časti. Úložnú a vizualizačnú.

Úložná skupina prvkov komun. kanála predstavuje také elementy, ktoré sú nevyhnutné na pridanie nového kanála, elementy pre odstránenie jedného existujúceho kanála, a tiež by mala obsahovať zobrazenie zoznamu dostupných kanálov. V tejto sku-

pine sme identifikovali potrebu elementu, ktorým bude realizované zobrazenie jedného vybraného kanála, do skupiny vizualizačných prvkov na to určených.

Vizualizačná skupina prvkov komun. kanála bude zoskupovať elementy pre vizualizáciu konkrétneho vybraného kanála. Pre vizualizáciu sme identifikovali niekoľko nevyhnutných prvkov. Jedným z nich je plátno, na ktoré bude celý kanál vykreslený, a ktoré by malo zobrazovať všetky elementy vybraného kom. kanála a tiež prvok pre identifikáciu aktuálnej pozície v rámci tohto kanála. Identifikovali sme potrebu elementu pre vloženie vstupu pre konkrétny kanál, elementy pre krokovanie medzi komponentami kanála a tiež element pre výpis. Výpis môže predstavovať zobrazenie podporných informácii o komunikačnom kanále ako aj porovnanie vstupu a výstupu komunikačného kanála.

4.2.3 Komponenty kanála

Skupinu prvkov komponentov komun. kanála sme tiež rozdelili na vizualizačné a úložné.

Úložná skupina prvkov predstavuje také elementy, ktoré sú potrebné pre dosiahnutie pridania nového komponentu do zoznamu komponentov, resp. pre odstránenie už existujúceho komponentu. Pre pridanie komponentu je potrebné zobraziť samostatné okno, kde si používateľ bude môcť vybrať konkrétny typ komponentu z ponuky už implementovaných komponentov, aj s príslušnými parametrami. Ak používateľ neuvedie parametre, nástroj použije prednastavené hodnoty. Novo pridaný komponent by sa mal objaviť v ponuke všetkých dostupných komponentov, a následne byť zvoliteľný v ponuke skladania komunikačného kanála.

Pre vizualizačnú skupinu prvkov sme identifikovali, podobne ako pre vizualizačnú skupinu kanála, potrebu plátna pre vykreslenie priebehu algoritmu. To ako sa konkrétny komponent vykreslí, je vzhľadom na rôznorodosť jednotlivých prvkov komunikačného kanála a použitých algoritmov, v plnej moci a zodpovednosti implementátora konkrétneho komponentu. Pre pohyb v rámci vizualizácie komponentu sme uvažovali o potrebe štyroch kontrolných elementov: dva elementy pre postupné krokovanie oboma smermy výpočtu vizualizácie, element pre skok na začiatok výpočtu a element pre skok na posledný krok algoritmu.

Tieto skupiny môžu byť prípadne rozšírené o podporné elementy ako popis skupiny prvkov, pre zvýšenie prehľadu v rámci okna nástroja.

4.2.4 Kontext systému

Nástroj bude pôsobiť ako samostatná aplikácia, bez potreby, integrovania do iných systémov. Pomerne jednoduchými úpravami je, v prípade potreby, možné napojiť na zdroj komunikačného kanála iný spôsob zadávania vstupu, napríklad čítanie vstupu zo súboru, výber vstupného súboru z adresárovej štruktúry. Podobným prístupom je možné upraviť komponenty komunikačného kanála, pre vykonanie výpočtov tretími stranami, či integrovaním / importom modulov docieliť novú funkcionalitu.

4.2.5 Používateľské rozhranie

Používateľské rozhranie by malo obsahovať ovládacie prvky pre väčšinu elementov popísaných v podkapitole Opis systému. Druh použitých grafických prvkov, ich grafická reprezentácia a implementácia je ponechaná výlučne na programátora.

4.2.6 Hardvérové rozhranie

Nástroj neobsahuje hardvérové rozhranie, ide výlučne o softvérový nástroj, aplikáciu.

4.2.7 Softvérové rozhranie

Nástroj musí byť vyvinutý podľa opisu systému a v súlade rozsahom a definovanými funkciami systému.

4.3 Použité nástroje

Vzhľadom na vyššie spomenutý opis a funkcie systému, bolo potrebné rozhodnúť, akým spôsobom bude náš softvérový nástroj navrhnutý a zostavený. Z charakteru funkcií sme zvoli implementáciu prostredníctvom objektovo orientovaného programovania (medzi iné paradigmy patrí napríklad čisto logické, štruktúrované či modulárne programovanie).

4.3.1 Objektovo orientované programovanie

Objektovo orientované programovanie je paradigma programovania, založená na využívaní dátových štruktúr, tzv. objektov. Každý objekt má definované svoje atribúty, metódy a udalosti, prostredníctvom, ktorých dokážu objekty medzi sebou komunikovať. Vzhľadom na funkcie nástroja a vyučované programovacie jazyky na univerzitách spomenutých v úvode tejto práce, pripadali do úvahy 4 programovacie jazyky. C++, C Sharp, Java, Python. Všetky tieto programovacie jazyky spĺňajú predpoklady na to, že nami implementovaný softvérový nástroj je možné nimi vytvoriť. Každý z uvedených programovacích jazykov, má svoje výhody a nevýhody. My sme si na účely tejto práce zvolili programovací jazyk Python.

4.3.2 Python

Python[] je skriptovací vysoko-úrovňový programovací jazyk, navrhnutý svojim autorom Guido van Rossumom v roku 1991. Tento jazyk podporuje dynamickú kontrolu dátových typov a tiež rôzne programovacie paradigmy. Vzhľadom na to, že Python je počítačový softvér s otvoreným zdrojovým kódom, je voľne dostupný a použiteľný dokonca aj na komerčné využitie. Otvorenosť tohto jazyka má za následok nárast jeho popularity[16] naprieč rôznymi sférami ako výroba, umelá inteligencia, spracovanie obrazu, a iné. Jazyk Python je vyučovaný na väčšine predmetných univerzít z úvodu tejto práce, a tiež na svetových popredných univerzitách ako napríklad MIT, Berkeley či Caltech.

Jazyk je podporovaný na viacerých platformách operačných systémov. Programy vyvinuté v Pythone pod operačným systémom Windows je možné bez nutnosti ďalšieho prispôsobovania spúšťať na inom operačnom systéme. V kontexte distribúcií systémov GNU/Linux je Python priamo integrovaný do rozhrania tejto distribúcie.

Na rozdiel od jazykov ako C++, C Sharp či java je Python interpretovací, čo znamená, že sa počas spustenia programu nevytvára samostatný spustiteľný kód, ako napr. exe súbor (C++, C Sharp) či .jar súbor (Java).

Medzi ďalšie vlastnosti jazyka Python patrí jeho dobre čitateľná syntax. Rovnaká funkcionalita napísaná v inom jazyku je dlhšia. Syntax jazyka Python neobsahuje veľa syntaktických symbolov, ktoré sa štandardne vyskytujú v ostatných programovacích jazykoch (ostatných vzhľadom na vymenované jazyky v predchádzajúcom odseku), hierarchické členenie je realizované odsadením (zvyčajne 4 medzery) od vyššej úrovne.

Ako sme už vyššie spomenuli, jazyk nie je dynamicky typovaný, čo znamená, že nie je pre každú premennú nutné deklarovať, akého je typu, čo umožňuje pohodlnejšie zapisovanie algoritmov a pridáva nástroju generickosť. Preto je tento jazyk vhodný aj pre programátorov, ktorý sú oboznámený s programovaním, avšak používajú iné programovacie jazyky.

4.3.3 Použité moduly

Pre grafické prostredie sme použili knižnicu tkinter[17], ktorá je súčasťou modulového vybavenia základnej inštalácie Pythonu. V prípade, že ju inštalácia neobsahuje, je možné si tento modul štandardným spôsobom nainštalovať. Tento modul sme použili aj z dôvodu výučby jeho základov na predmetných univerzitách z úvodu tejto práce. Modul tkinter umožňuje tvorbu jednoduchých, ale aj pokročilejších grafických aplikácii prostredníctvom objektovo orientovaného prístupu. Základom každej aplikácie je top-level okno, do ktorého sú vkladané prvky grafického rozhrania (Widgets), ako napríklad Menu (Menu, stavebný prvok vytvárania menu), Tlačidlo (Button, element obdĺžnikového tvaru, ktorému je možné priradiť akciu - udalosť), Plátno (Canvas, element pre vykresľovanie textu a grafických objektov ako obdĺžnik, ovál či úsečky), Štítok (Label, element na zobrazenie textu), a iné. Prvky grafického rozhrania je možné zgrupovať pod rodičovským elementom Rám (Frame, elemement, pod ktorý je možné priradiť objekty prostredníctvom vzťahu rodič-potomok) alebo Štítkový rám (LabelFrame, podobne ako Rám, len s tým rozdielom, že okolo použitých elementov sa graficky vytvorí ešte orámovanie s nadpisom. Nutným predpokladom je existencia aspoň jedného potomka). Inými modulmi na tvorbu grafického rozhrania sú napr. PyQT, KIVY, WxPython, PySide či PySimpleGUI. My sme zvolili tkinter z dôvodu toho, že tento modul je v základnom modulovom vybavení väčšiny inštalácii Pythonu, a preto je pomerne ľahko dostupný. Z dôvodu zachovania jednoduchosti a ľahkej prístupnosti aj používateľom, bez znalosti programovacieho jazyka Python, sme sa chceli vyhnúť inštalácii dodatočných modulov.

Ďalším z použitých modulov je modul *random*[**?**]. V tomto module sú implementované metódy na výber pseudonáhodných čísel z rôznych distribúcií. Tento modul sme použili najmä pri implementácii komponentu Šum, v ktorom sme potrebovali náhodným spôsobom vybrať index symbolov určených na zmenu. Táto zmena mala simulovať vplyv šumu na prenosový kanál a tým transformovať prenášanú správu.

Na vykonávanie automatických testov sme použili modul *unittest*[18] prostredníctvom, ktorého sme v projektovom priečinku tests načrtli spôsob vytvárania testov už implementovaných súčastí aplikácie. Výhoda takéhoto prístupu spočíva v pravidelnom

testovaní existujúcej funkcionality, ako potvrdenie toho, že novo pridané funkcie a metódy neovplyvnili nežiadúcim spôsobom fungujúce súčasti nástroja.

4.4 Princípy implementácie

Pred popisom samotnej implementácie sme sa rozhodli uviesť ešte niekoľko princípov, ktorých sme sa v práci snažili držať počas vytvárania nástroja.

4.4.1 Modularita

Pod modularitou rozumieme dizajnový prístup, ktorý kladie dôraz na oddelenie funkčnosti. Každý prvok aplikácie a každú funkcionalitu sme pokladali za samostatný uzavretý modul, pre ktorý z pohľadu iných modulov nie je dôležitá vnútorná implementácia, iba rozhranie na komunikáciu (metódy). Takýto prístup dovoľuje neskoršie zmeny a úpravy zdrojového kódu bez ujmy na zmene architektúry či iných modulov. Snahou je uľahčiť výstavbu veľkých a komplexných (zložitých) programov.

Takéto členenie práce úzko súvisí s objektovo orientovaným a štruktúrovaným programovaním a povoľuje skladanie modulov do väčších modulov, resp. rozloženie jedného modulu a viac samostatných pod-modulov tak, aby zložitejší problém bolo možné rozdeliť na viac menších. Štruktúru modulárneho programu je možné reprezentovať orientovaným acyklickým grafom, ktorého vrcholy su jednotlivé moduly a hrany určujú prechod od modulu na vyššej úrovni ku modulu na nižšej alebo rovnakej úrovni. V takomto modely nie je vylúčená existencia spätných väzieb, dbali sme však na to, aby ich bolo čo najmenej.

4.4.2 Konzistenia

Využitím programovacieho jazyka Python sme sa snažili dodržať štandard PEP 8, ktorý bol pre Python publikovaný za účelom vytvorenia konvencie pre zápis zdrojového kódu. Taktiež sme sa snažili rozhodovať v súlade s princípmi "Zen of Python". Ide o 19 princípov, ktoré boli publikované v roku 1999, a ktoré ovplyvňujú dizajn kódu písaný v programovacom jazyku Python.

Kapitola 5

Implementácia

V tejto kapitole popisujeme implementáciu softvérového nástroja. Nástroj nebol pomenovaný. V nasledujúcich kapitolách vysvetlíme vytvorenú adresárovú štruktúru, programové objekty a ich zapojenie do výsledného softvéru.

5.1 Popis prostredia

Adresárová štruktúra prostredia je pod adresárom projektu členená nasledovne:

- súbor main.py,
- priečinok src,
- priečinok tests,
- priečinok codes.

5.1.1 Súbor main.py

Súbor main.py je hlavný súbor aplikácie, ktorým sa spúšťa softvérový nástroj. Súbor nemá implementovanú prácu s argumentami, a svoju úlohu plní prostredníctvom volania modulu zabezpečujúceho vykreslenie grafických prvkov aplikácie (Trieda Applicaion v súbore src/gui.py).

5.1.2 Priečinook src

Adresár src obsahuje súbory zdrojových kódov nástroja rozložené do viacerých súborov pre zvýšenie prehľadnosti aplikácie a zachovanie princípu modularity. Jednotlivé súbory popíšeme v nasledujúcich podkapitolách.

Channel.py

Obsahuje triedu Channel, ktorá je abstraktným reprezentantom jedného komunikačného kanála. Táto trieda obsahuje atribúty ako napríklad názov, ktorý si používateľ zvolí pri vytváraní objektu, a zoznam komponentov, ktorý reprezentuje zoznam pridaných komponentov do konkrétneho komunikačného kanála.

Táto trieda obsahuje metódu na pridanie nového komponentu, do existujúceho kanála a tiež niekoľko pomocných metód napríklad na výpis komponentov kanála, či na zistenie ich počtu.

Netriviálnou metódou je metóda *vizualize*(), ktorej vstupnými argumentami sú plátno a pozícia. Na základe týchto parametrov a atribútu components (zoznam komponentov) je metóda schopná vykresliť na ponúknuté plátno všetky svoje komponenty. Podľa parametra *position* označí konkrétny komponent tak, aby bolo používateľovi jasné, na ktorý komponent tento parameter práve odkazuje.

Component.py

V tom to súbore definujeme abstraktnú triedu Component. Trieda obsahuje jediný atribút, názov (name), v ktorom je uložené meno komponentu. Toto meno používateľ zadáva pri vytvorení inicializácie konkrétneho komponenetu. Okrem toho trieda obsahuje povinné metódy - *run*(), *vizualize*() zabezpečujúce ďalšiu funkcionalitu nástroja.

Každá implementácia kódu, vzhľadom na to, že je potomkom tohto elementu, má prednastavenú existenciu uvedených metód. Metóda *run*() má za úlohu vykonanie celého kódu algoritmu. Počas behu algoritmu vznikajú dáta potrebné pre vizualizáciu, ktoré je možné následne zobraziť na objekte plátno prostredníctvom metódy *vizualize*().

Hlavná myšlienka vizualizácie komponentu je krokovanie na vopred vypočítanej dátovej štruktúre, aby sa predišlo neočakávaným situáciám počas krokovania komponentu, resp. krokovania prechádzania komunikačným kanálom. Každá implementácia

konkrétneho kódu má voľnosť pozmeniť spôsob, akým je zobrazovaná a tiež aké parametre obsahuje.

Takýto prístup zabezpečuje vždy úspešnú vizualizáciu. Z charakteru algoritmov komunikačného kanála je zrejmé, že nie je možné všetky algoritmy vizualizovať zhodným spôsobom. Algoritmy sa líšia v použitých dátových štruktúrach, ako aj v obsahu toho, čo je potrebné zobraziť, a taktiež aj v počte krokov.

Niektoré vybrané algoritmy ako napr. Huffmanov kód, môžu byť zobrazené v plnej dĺžke, pretože sa všetky kroky výpočtu zmestia na obrazovku (pre vhodne veľké vstupy). Pri niektorých iných algoritmoch, je potrebné zvoliť správne dáta na zobrazenie, pretože algoritmus môže obsahovať veľké množstvo triviálnych výpočtov, ktoré nemá zmysel na obrazovku zobrazovať. Jedným z jednoduchších spôsobov vizualizácie je zobrazenie výsledku medzikrokov výpočtu v textovej podobe. Ak je takýto prístup nedostatočný, používateľ smie túto funkcionalitu rozšíriť vlastnou interpretáciou medzivýsledkov.

Containers.py

Tento súbor obsahuje dve triedy.

Trieda ComponentContainer slúži ako spoločné úložisko na zachytenie všetkých implementovaných komponentov. Obsah tohto kontajnera je zobrazovaný v komponenete na zoprazenie dostupných komponentov. Tieto komponenty sú ďalej prístupné pri výbere do zostavy komunikačného kanála. ComponentContainer má implementovanú iba jednu metódu, a to *Add*(), ktorá zabezpečuje pridanie nového komponentu do kontajnera. Tento kontajner smie obsahovať iba komponenty s unikátnymi názvami. Unikátnosť je kontrolovaná pri pridávaní nového komponentu.

Trieda ChennelContainer funcuje analogicku ku triede ComponentContainer. Sú v nej uchovávané všetky definové kanály a používateľ smie pridávať iba také, ktoré majú unikátne meno.

Unikátnosť oboch kontajnerov je dizajnovou nutnosťou. Týmto prístupom sme chceli predísť duplicitným názvom a tým zjednodušiť vyhľadávanie komponentu, resp. kanála na základe jeho názvu. Ak by existovalo viacero komponentov, resp. kanálov so zhodným názvom, nebolo by jednoznačné, ktorý komponent si vyberáme, resp. ktorý kanál chceme zobraziť na plátnach. Jedinečnosť je možné dosiahnúť napríklad pridaním atribútu ID, ktorý by každému objektu (komponent, kanál) priradil jedinečné identifikačné číslo.

Settings.py

Trieda Settings obsahuje atribúty, ktoré sú dostupné naprieč celým nástrojom. Objekt settings zabezpečuje jednotnosť nastavení, a akonáhle existuje dva a viac elementov, ktoré by mali použiť nejaký zhodný atribút či parameter, mal by byť tento parameter predmetom tejto triedy. Takýto prístup umožňuje hromadné zmeny z jedného miesta aplikácie. Typickým príkladom je napr. podfarbenie aplikácie, či veľkosť komponentov na plátne komunikačného kanála, ale aj iné parametre vykresľovania, ako napríklad štandardizácia rozmerov tlačidiel, či fontov využívaných naprieč nástrojom.

Rozsiahla dostupnosť objektu *settings* umožňuje použiť tento objekt aj na také operácie, akými sú napríklad výmena mapovacích dátových štruktúr medzi kodérom a príslušným dekodérom. Túto myšlienku výmeny dát medzi objektami je možné oddeliť do samostatného objektu, avšak pre náš nástroj je toto nastavenie postačujúce.

Utility.py

Ide o jediný súbor neobsahujúci triedu.

Súbor utility obsahuje funkciu *load_data*() na načítanie dát predimplementovaných komponentov komunikačného kanála, ktoré sú načítané hneď pri otvorení aplikácie a sú zobrazené pri otvorení hlavného okna. Tento prístup zabezpečuje, že implementované súčasti sú používateľovi dostupné na používanie ihneď ako začne nástroj používať. Okrem toho je tento súbor použitý ako zhromaždisko pomocných funkcií a procedúr, ktoré sú nezávislé od objektov nástroja, a ktoré majú podpornú funkciu naprieč celým nástrojom.

Gui.py

Táto trieda slúži na vykreslenie grafického rozhrania aplikácie.

Inicializáciou triedy Application sú načítané objekty už implementovaných komponentov a kanálov aplikácie a taktiež sú vytvorené grafické prvky aplikácie samotnej.

Pre zvýšenie prehľadu, ako sú grafické prvky organizované na obrazovke, je vytváranie týchto prvkov zapracované do samostatných metód.

Metóda *create_widgets*() zabezpečuje postupné definovanie šírky a výšky každej bunky aplikácie. Bunky aplikácie vznikajú z dôvodu postupného vytvárania menu a

podporných rámov (frames).

Vytvorené menu obsahuje tri súčasti, Súbor, Nástroje a Pomoc.

Menu položka obsahuje len jediný prvok, Skončiť. Prvok skončiť má za úlohu riadne ukončenie aplikácie.

Položka Nástroje, nesie veľkú časť funkcionality aplikácie. Pod nástrojmi sú implementované prvky Pridaj Kanál, Odstráň Kanál a Zobraz Kanál. Prvok Pridaj Kanál vyvolá okno, pre tvorbu nového kanála a jeho pridanie do zoznamu kanálov. Prvok Odstráň Kanál zruší aktuálny, resp. označený kanál zo zoznamu. Ak žiaden kanál nebol označený, vyvolá upozorňujúce okno, s textom, že používateľ najprv musí vybrať kanál, ktorý je požadované odstrániť zo zoznamu všetkých kanálov. Po kliknutí na možnosť Zobraz kanál, je zobrazený aktuálny kanál, resp. Používateľ je vyzvaný na označenie kanálu a následne opätovné vykonanie akcie Zobrazenia kanála. Zobrazenie kanála spôsobí načítanie vybraného kanála a vykonanie metód run() všetkých jeho elementov v postupnej následnosti, ako sú komponenty daného kanála usporiadané. Ak usporiadanie je v zmysle platného zapojenia komunikačného kanála, komunikačný kanál by mal byť vyhodnotený ako správny, a následnosť jednotlivých komponentov bude vyhodnotená a zobrazená podľa očakávania.

V prípadne neplatného zapojenia kanál pri spustení vráti na niektorom z krokov výnimku, týkajúcu sa neschopnosti simulovať celý komunikačný kanál. V zmysle logického zapojenia komponentov, napríklad zapojenie dekodéra pred tým, ako je použitý kodér je technicky možné, ale logicky to nedáva zmysel.

Pre prácu z Komponentami menu obsahuje prvok Pridaj Komponent a Odstráň Komponent. Pre odstránenie musí byť niektorý z komponentov vopred označený.

Prvok Pridaj komponent vyvolá okno pre pridávanie komponentov do zoznamu. Okno pridania komponentov obsahuje tlačidlá pre pridanie jednotlivých typov komponentov, ktoré je implementované v samostatnej metóde. Tomúto oknu sme zatiaľ zadefinovali 5 tlačidiel, menovite: Kodér1, Kodér2, Dekodér1, Dekodér2, a Šum. Každé z týchto tlačidiel vyvolá samostatné okno pre bližšiu špecifikáciu komponentu, napríklad výber parametrov daného komponentu. Príklad 1: Pridanie komponentu Kodér1 vyvolá okno, kde si používateľ môže zvoliť z implementácie Fanovho alebo Huffmanovho kódu. Povinným prvkom tohto okna je uvedenie názvu komponentu. Príklad 2: Voľba komponentu dekodér neponúkne okrem uvedenia unikátneho názvu komponentu žiadne ďalšie polia na výber. Je to z dôvodu zhodnej implementácie dekodéra pre oba predmetné kódy (Fano, Huffman). Ak by bola požadované zo strany používateľa iné dekódovanie, potom je nutné toto dekódovanie doplniť do nástroja.

Prvok Odstráň Komponent vymaže komponent zo zoznamu komponentov (odstráni daný komponent z kontajnera komponentov).

Okrem Menu, inicializáciou sa zavolá metóda $create\_frames()$, ktorá zabezpečí zobrazenie, resp. vykreslenie ďalších súčastí grafického rozhrania. Nasledujúce prvky rozhrania sú rozdelené do nasledujúcich metód:

- $create\_frame\_channels$,
- $create\_frame\_components$,
- $create\_frame\_viz\_channel$, a
- $create\_frame\_viz\_component$.

Metóda $create\_frame\_component$ zabezpečí vytvorenie Rámu pre prvky grafického rozhrania týkajúce sa manažmentu komunikačných kanálov.

Rám obsahuje Štítok s popisom "Kanály", pod ktorým je prvok ListBox zobrazujúci zoznam komunikačných kanálov z kontajneru kanálov. Pod týmto prvkom sme zaradili tlačidlá Pridaj Kanál, Odstráň kanál a Zobraz kanál, ktoré sú svojou funkcionalitou identické s rovnomennými prvkami obsiahnutými v Menu.

Metóda $create\_frame\_components$ má za úlohu vytvorenie Rámu pre prvky grafického rozhrania súvisiacich s manažmentom implementovaných komponentov. Rám obsahuje Štítok "Komponenty", pre jednoznačnú identifikáciu príslušnosti, pod ktorým sa vyskytuje ListBox obsahujúci zoznam komponentov z kontajnera komponentov. Okrem toho, Rám obsahuje tlačidlo Pridaj Komponent a Odstráň Komponent, ktorých úloha je totožná s funkcionalitou rovnomenných prvkov z Menu.

V metóde $create\_frame\_viz\_channel$ je definovaný Rám, obsahujúci zadefinované prvky grafického rozhrania majúce za úlohu zobrazenie konkrétneho komunikačného kanála z kontajnera kanálov. Tento Rám obsahuje okrem Plátna, na ktoré je vykresľovaný aktuálne zobrazovaný komunikačný kanál ešte 4 tlačidlá.

Po kliknutí na tlačidlo Zdroj, je používateľ vyzvaný na zadanie vstupu pre komunikačný kanál. V závislosti od toho, čo je predmetom komunikačného kanálu, používateľ môže zvoliť ľubovoľný text, alebo postupnosť symbolov nad nejakou abecedou. Vstup nie je nijakým spôsobom validovaný, čím má používateľ plnú kontrolu nad tým, čo je na vstupe. Nevýhodou je, s tým súvisiaca absencia validačných pravidiel. V prvotnej verzii nástroja žiadne validačné pravidlá nie sú vedome aplikované z dôvodu prene-

chania plnej kontroly nad aplikáciou používateľovi. Po tlačidle Zdroj nasleduje dvojica tlačidiel s navzájom opačnou funkcionalitou.

Tlačidlo "Nasledujúci Komponent"zabezpečí postúpenie v zozname komponentov o jeden komponent smerom ku koncu komunikačného kanála, zatiaľ čo tlačidlo Predchádzajúci Komponent posunie pozíciu v rámci kanála o jedno miesto naspäť. Pri posune pozície medzi komponentami sa zároveň reštartuje pozícia v rámci komponentu. Viac o tejto pozícii v nasledujúcej podkapitole.

V rámci zobrazenia komponentov sme implementovali ešte tlačidlo Príjemca. Možno očakávanou udalosťou, ktorá vznikne kliknutím na toto tlačidlo je porovnanie výstupu posledného výpočtového komponentu so vstupom do aktuálneho komunikačného kanála. Cieľom by mohlo byť docieliť, aby vstup bol kódovaný a následne dekódovaný do rovnakej podoby. Zhodná rekonštrukcia kódovanej správy je síce úspešným výsledkom správne implementovaného a zapojeného komunikačného kanála. Avšak, flexibilita nástroja povoľuje také zapojenie komponentov, aby bola simulovaná iba časť komunikačného kanála. V tom prípade automatické porovnanie vstupu a výstupu komunikačného kanála by nedávalo zmysel, preto tlačidlo Príjemcu v prvotnej verzii násjtroja iba vypíše výstup posledného výpočtového komponenetu. Samotné vyhodnotenie vypísaného výstupu je prenechané používateľovi.

create_frame_viz_component V tejto metóde sme opäť zadefinovali Rám, obsahujúci vopred zadefinované prvky grafického rozhrania, tentokrát slúžiace na zobrazenie jedného konkrétneho komponentu kanála. Podmienkou pre úspešnú vizualizáciu je implementácia iterovateľnej dátovej štruktúry do metódy visualize() triedy Component.

Každý vizualizovaný komponent je na začiatku zobrazený vo svojej prvej pozícii (z pohľadu indexovania nultý index). Pohyb v rámci tejto štruktúry je realizovaný prostredníctvom 4 tlačidiel.

Tlačidlo Počiatok presunie zobrazovaciu funkciu na začiatok vizualizácie komponentu. Tlačidlá Predchádzajúci Krok a Nasledujúci Krok zabezpečujú krokovanie v smere a protismere behu algoritmu. Tlačidlo Koniec presunie zobrazenie na posledný krok vizualizácie.

5.1.3 Priečinok codes

Adresár codes sme navrhli dat, aby obsahoval zdrojové kódy pre jednotlivé implementované kódy. Rodičom týchto kódov je z pohľadu implementácie trieda Component, ktorá sa nachádza pod adresárom src v súbore component.py. Vzhľadom na túto dedičnosť

je zabezpečené, že každý kód/komponent komunikačného kanála bude mať vytvorené potrebné metódy run(), visualize().

každý z implementovaných kódov má slúžiť ako vzor pre pridávanie nových kódov a používateľ, vzhľadom na charakter nástroja, smie pomerne jednoducho využiť importy modulov tretích strán pre doplnenie nových kódov. S ohľadom na flexibilitu nástroja, jednotlivé komponenty nie sú nijakým spôsobom na seba nadviazané. V prípade, že je požadovaná výmena informácie medzi dvomi rôznymi modulmi, je potrebné implementovať akýsi medzičlánok, ktorý by robil sprostredkovateľa na výmenu informácie medzi komponentmi. Náznakom implementácie riešenia takéhoto problému je implementácia metód *add_mapping*(), *clear_mappings*() a *get_mapping*() v triede Settings. Pomocou týchto troch metód je možná napr. výmena mapovacej tabuľky kodéra zdroja (napr. HUffmanov kód) a dekodéra prefixového kódu.

5.1.4 Priečinok tests

Priečinok *tests* vznikol za účelom zhromažďovania zdrojových kódov tried a metód určených na podporu tzv. unit testov. Unit testy podporujú automatizáciu testovania implementovaných tried a ich metód, čím vznikne automatické potvrdzovanie, že nové zmeny nástroja neovplyvnili nežiadúcim spôsobom už implementované, a predtým funkčné, súčasti systému. Implementácia unit testov taktiež podporuje koncept objektovo orientovaného programovania.

V tejto kapitole sme si zhrnuli základné poznatky o organizácii zdrojového kódu a uviedli na čo slúžia jednotlivé súbory projektového priečinka.

5.2 Vzhľad nástroja

V tejto kapitole si popíšeme spôsob implementácie grafického rozhrania, na účel pochopenia organizácie jednotlivých prvkov rozhrania a ich prepojenia. Pre budúce rozširovanie funkcionalít nástroja je nevyhnuté poznať spôsob ich zapojenia do celku.

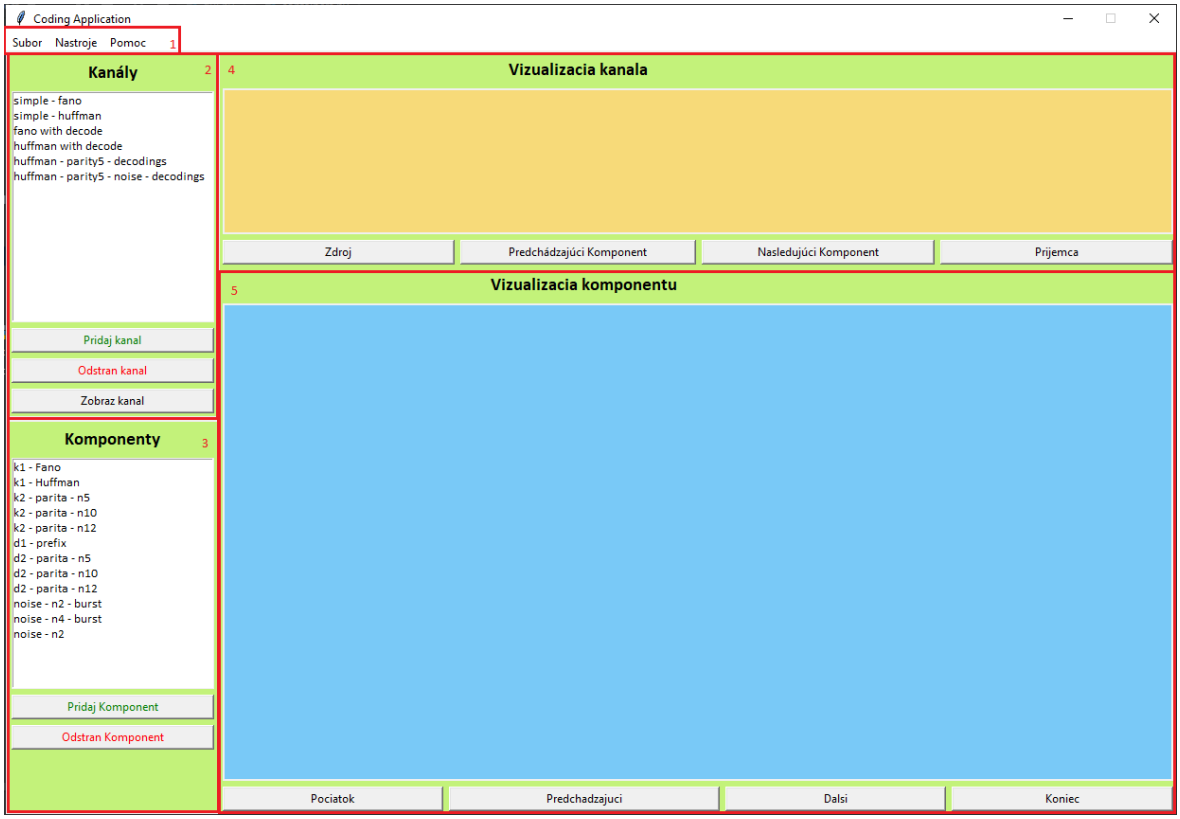
Modul tkinter slúži na tvorbu grafického rozhrania v programovacom jazyku Python. V prostredí tkinter modulu existujú tri možnosti, ako manažovať rozmiestnenie grafických prvkov (angl. widgets, ďalej aj widgety):

1. pack() - prvky sú organizované v do horizontálnych a vertikálnych obdĺžnikových rámcov limitovaných na smery hore, dole, doprava, doľava, a ktoré je nutné

definovať vzájomnou relatívnou pozíciou,

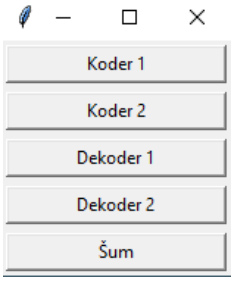
2. place() - prvky grafického rozhrania majú definované absolútne súradnice v dvojrozmernej súradnicovej sústave,

3. grid() - prvky sú umiestnené podľa indexu dvojrozmernej mriežky.

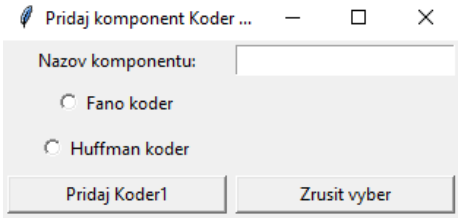


Obr. 5.1: Oblasti hlavného okna

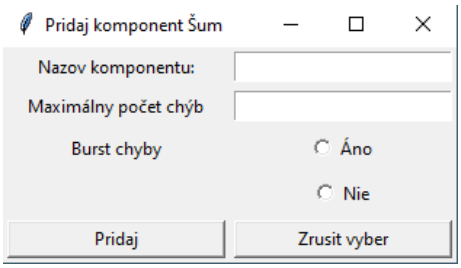
V tomto projekte sme vybrali organizáciu prvkov pomocou metódy grid(). Zadefinovali sme 5 oblastí. Menu (1), oblasť pre kontajner dostupných komunikačných kanálov (2), kontajner pre dostupné komponenety (3) komunikačného kanála, plátno pre vizualizáciu a krokovanie naprieč kanálom (4) a plátno s tlačidlami pre krokovanie aktuálne vybratého komponentu (5). Čísla v zátvorkách pri jednotlivých oblastiach súhlasia s označením oblastí na obrázku 5.1.



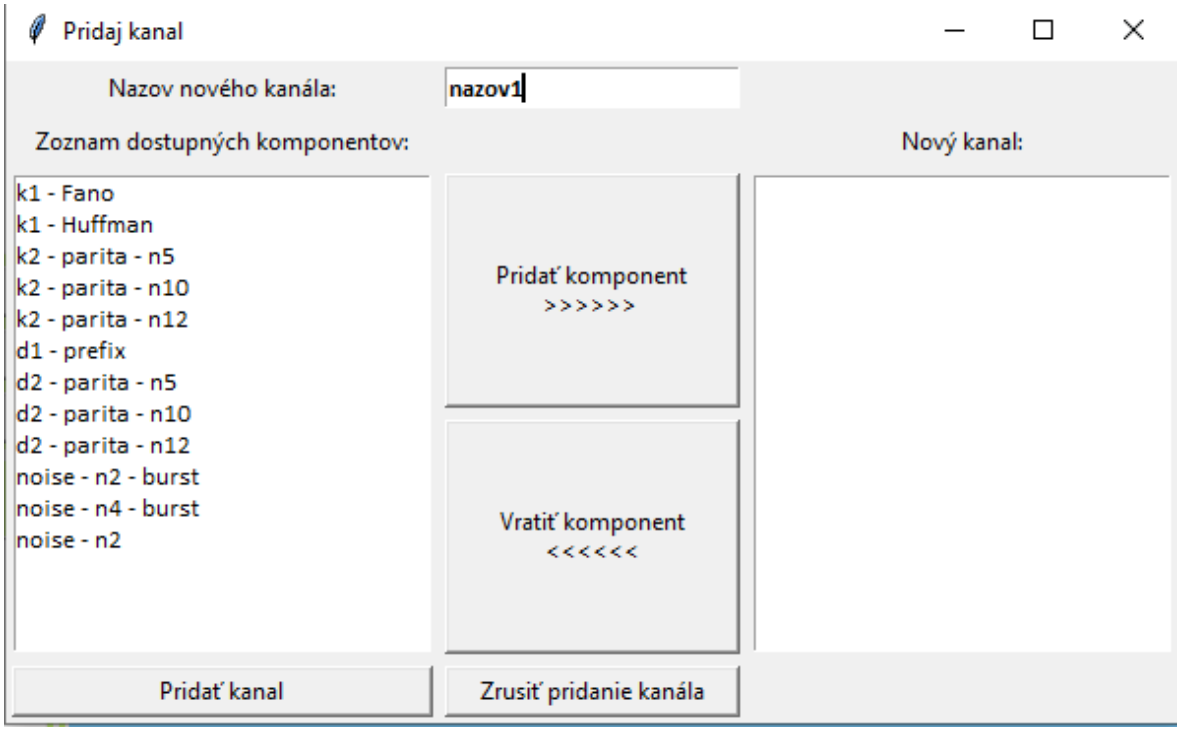
Obr. 5.2: Okno pre výber typu komponentu



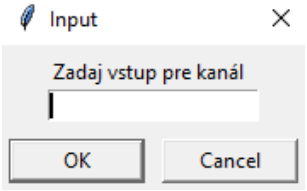
Obr. 5.3: Okno pre výber algoritmu kódovania zdroja



Obr. 5.4: Okno pre výber parametrov šumu



Obr. 5.5: Okno pre vytvorenie zostavy vlastného komunikačného kanála



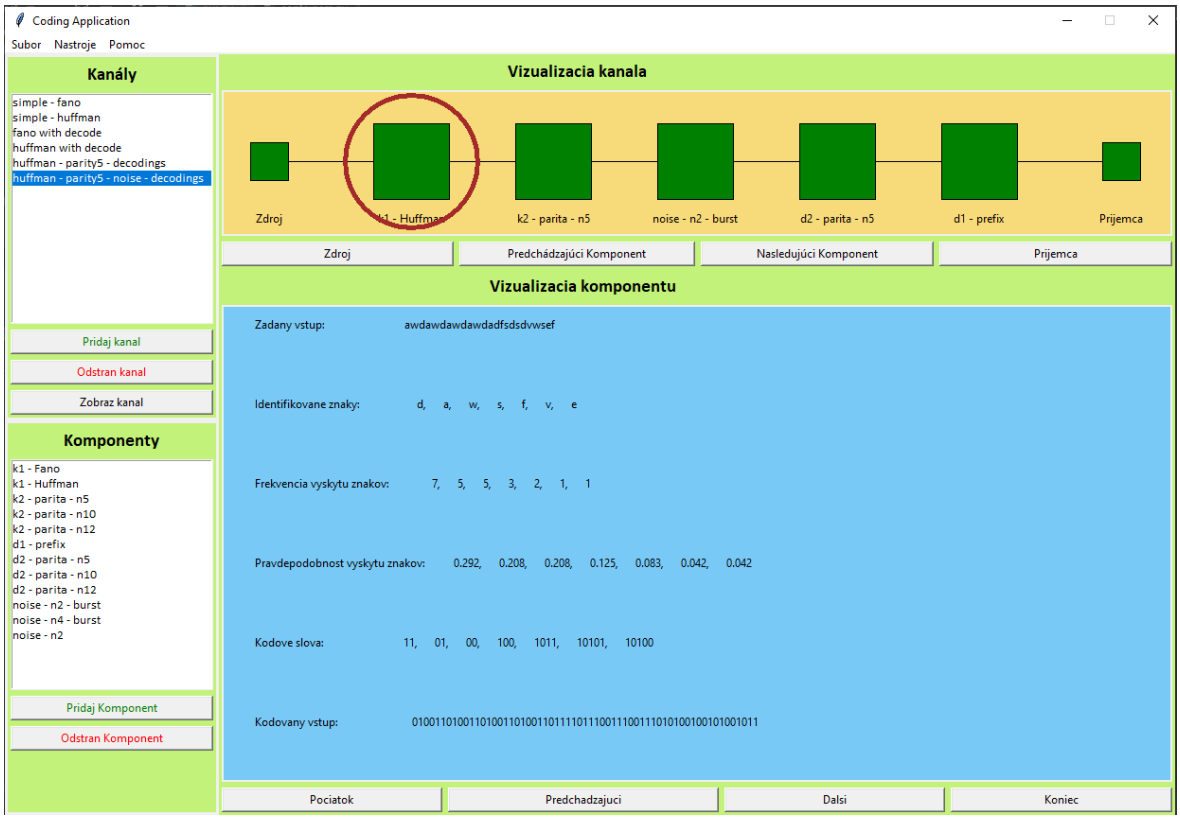
Obr. 5.6: Okno na zadanie vstupu (zdroja) pre komunikačný kanál

Po kliknutí na tlačidlo Pridaj Komponent, nástroj vyvolá okno na výber typu komponentu, ktorý chceme pridať do zoznamu komponentov. Toto okno je znázornené na obrázku 5.2. Po kliknutí na tlačidlo niektorého typu komponentu je zobrazené okno pre výber algoritmu a parametrov daného typu komponentu. Ako príklad uvádzame okno výberu algoritmu kodéra zdroja (obrázok 5.3) či výber parametrov komponentu Šum (obrázok 5.4). Po vybratí typu komponentu a jeho parametrov je daný komponent zobrazený v zozname komponentov. V prototype nástroja je implementovaných niekoľko algoritmov pre demonštráciu správnej funkčnosti nástroja. Spôsob pridania nových komponentov popisujeme v návode pre používateľa.

Pri vytvorenie vlastnej zostavy komunikačného kanála sme pripravili tlačidlo Pridaj Kanál, na ktoré keď používateľ klikne, nástroj mu zobrazí okno rozhrania na pridanie nového komunikačného kanála. Toto okno je zobrazené na obrázku 5.5. Okno v ľavej časti obsahuje identický zoznam komponentov ako oblasť (obrázok 5.1, oblasť č. 3) hlavného okna aplikácie. Používateľ si následne pomocou tlačidiel Pridať komponent a Vrátiť komponent skladá vlastnú zostavu, ktorá je zobrazená v pravej časti okna. Usporiadanie prvkov v tomto zozname zodpovedá usporiadaniu komponentov komunikačného kanála. Z dôvodu odhadovaného nízkeho počtu komponentov v prvých verziách aplikácie, sme vedome neimplementovali tlačidlá na zmenu usporiadania už vybratých komponentov nového komunikačného kanála. Ak je potrebná zmena poradia komponentov, používateľ je nútený komponenty z kanála odstrániť a pridať ich v požadovanom poradí.

Po stlačení tlačidla Pridať kanál je tento kanál zostavený pod jedinečným názvom uvedeným v hornej časti okna, a sú do neho vložené všetky, používateľom vybrané, komponenty zoznamu nového kanála.

Pre zobrazenie a simuláciu niektorého z pripravených komunikačných kanálov, je potrebné požadovaný kanál označiť v zozname kanálov a následne zobraziť prostredníctvom tlačidla Zobraz kanál nachádzajúceho sa v hlavnom okne. Zobrazením kanála je automaticky vyvolané okno pre zadanie vstupu do komunikačného kanála (obrázok 5.6). Tento vstup je simulovaním generovania zdrojových údajov. Vstup nie je v tejto verzii nástroja nijakým spôsobom overovaný a v takej podobe, v akej je zadaný



Obr. 5.7: Okno s aktívnym komponentom zobrazujúcim Huffmanov kód a jeho postupný výpočet

používateľom, je ďalej použitý.

Po zadaní zdrojových údajov sa používateľ pomocou tlačidla Nasledujúci Komponent môže presunúť na ďalší prvok aktuálneho komunikačného kanála. Na plátne je pozícia v rámci komunikačného kanála zobrazená pomocou hnedej kružnice opísanej okolo aktuálneho komponentu (obrázok 5.7). Prostredníctvom tlačidiel Predchádzajúci Komponent a Nasledujúci Komponent sa používateľ hýbe medzi komponentami aktuálne zobrazeného kanála. V časti okna Vizualizácia komponentu je realizovaná metóda visualize() príslušného komponentu, v ktorom je možné krokovať priebeh výpočtu pomocou tlačidiel Počiatok (prvý krok výpočtu), Predchádzajúci (posunie zobrazenie algoritmu o 1 pozíciu naspäť), Ďalší (posunie zobrazenie algoritmu o 1 pozíciu dopredu) a Koniec (posledný krok výpočtu).

5.3 Spracovanie chýb

Nástroj svojou flexibilitou umožňuje používateľovi využiť širokú škálu funkcií. Časť týchto funkcií má svoje implementačné obmedzenia, ktoré pri nesprávnom použití ná-

stroja môžu vyvolať neočakávané situácia, resp. chyby. Aby sme predišli neočakávaným situáciám pri používaní nástroja, boli na vybraných miestach implementované techniky na spracovanie chýb. Príkladom je výber názvu komponentu prostredníctvom funkcionality na pridanie komponentu. Používateľ pri zadávaní nového názvu ma úplnú voľnosť, avšak nástroj nedokáže pracovať s dvoma a viac objektami, ktoré majú zhodný názov. Preto pred pridaním komponentu je overená jedinečnosť jeho názvu, a používateľ je prostredníctvom vyskakovacieho okna upozornený na zhodu. Používateľ môže následne názov komponentu zmeniť. Iné situácie, ktoré môžu nastať napríklad nelogickým zapojením jednotlivých komponentov do zostavy komunikačného kanála budú mať za následok nesprávny tok dát, čo bude zobrazené na vizualizačnej vrstve, avšak nespôsobia pád nástroja.

Kapitola 6

Záver

6.1 Zhrnutie

Úvodom práce sme si definovali základné pojmy nevyhnutné pre ďalšiu prácu a predstavili sme si teóriu kódovania a pridružených tém. Na základe definovaného cieľa práce sme odvodili požiadavky na implementáciu softvérového nástroja. Analýzou požiadaviek a následnou implementáciou vznikol prototyp softvérového nástroja, ktorý spĺňa požiadavky zadania a je použiteľný v praxi, čomu odpovedá testovanie nezaujatými osobami, ktorých spätná väzba bola vzatá do úvahy počas implementácie práce.

Prototyp nástroja ponúka širokú škálu možností na jeho využitie aj mimo prostredia problematiky výučby teórie kódovania. Flexibilita a modulárny prístup umožňuje implementáciu akéhokoľvek nového komponentu práce a rozšírenie jej funkcionality.

Tento nástroj obsahuje ukážky vybraných implementovaných algoritmov, a demonštruje koncept aplikácie, ktorá môže slúžiť študentom, aj vyučujúcim. Vzhľadom na náročný rok 2020, počas ktorého sa veľká časť výučby presunula do online prostredia je nevyhnutné obzerať sa po možnostiach výučby aj iným, ako zaužívaným prístupom.

Vyvinutý nástroj demonštruje jedno z použití a relatívne jednoducho môže byť upravený pre potreby konkrétneho vyučujúceho prostredníctvom prispôsobenia nástroja, jeho grafických prvkov či vizualizačných prístupov.

6.1.1 Ukončenie

Z dôvodu rozsiahlosti problematiky teórie kódovania v tejto práci neboli implementované nasledujúce, častokrát prednášané oblasti, ako kódovanie obrazu, kódovanie zvuku, spracovanie signálu, či implementované zložitejšie algoritmy triedy lineárnych, cyklických či konvolučných kódov, a iné. Čitateľ je vyzývaný, aby tieto súčasti v prípade potreby doplnil, a umožnil tým rozvoj a propagáciu týchto tém širšiemu publiku akademickej sféry.

Literatúra

[1] Quote Investigator. *Quote Investigator.* [online, navštívené: 8.3.2021], `https://quoteinvestigator.com/2018/01/07/stone-age/#more-17685`.

[2] Wikipedia.org. *Information Age.* [online, navštívené: 8.3.2021], `https://en.wikipedia.org/wiki/Information_Age`.

[3] https://www.idc.com/. *Data Creation and Replication.* [online, navštívené: 8.3.2021], `https://www.idc.com/getdoc.jsp?containerId=prUS47560321`.

[4] Cisco. Cisco visual networking index: Forecast and trends, 2017–2022. 2018.

[5] C. E. Shannon. *A mathematical theory of communication*, volume 27. 1948.

[6] Jiri Adamek. *Foundations of coding: Theory and applications of error-correcting codes with an introduction to cryptography and information theory.* John Wiley & Sons, 2011.

[7] David JC MacKay. *Information theory, inference and learning algorithms.* Cambridge university press, 2003.

[8] Richard E Blahut. *Theory and practice of error control codes.* Addison-Wesley, 1983.

[9] Simeon Ball. *A Course in Algebraic Error-Correcting Codes.* [online, navštívené: 21.3.2021], `https://web.mat.upc.edu/simeon.michael.ball/codinglectures`.

[10] Daniela Kravecová. *Základy kódovania.* Technická univerzita v Košiciach, 2012.

[11] Daniel Olejár and Martin Stanek. *Úvod do teórie kódovania.* Fakulta fyziky, matematiky a informatiky, Univerzita Komenského, 2007.

[12] Otokar Grošek and Štefan Porubský. *Šifrovanie : Algoritmy, metódy, prax.* Grada, 1992.

[13] Jiří Ivánek. *Vybrané kapitoly z kódování informací.* Ústav informačních studi a knihovníctví, Filozocická fakulta, Univerzita Karlova, 2007.

[14] Radim Jiroušek, Ivánek Jiří, and spol. *Principy digitální komunikace.* LEDA, 2006.

[15] opengenus.org. *Different ways to select random element from list in Python.* [online, navštívené: 12.3.2020], `https://iq.opengenus.org/random-element-from-list-in-python/`.

[16] David Robinson. *The Incredible Growth of Python.* [online, navštívené: 21.3.2020], `https://stackoverflow.blog/2017/09/06/incredible-growth-python/`.

[17] Python.org. *Tkinter.* [online, navštívené: 20.12.2020], `https://docs.python.org/3/library/tkinter.html`.

[18] Python.org. *Unittest.* [online, navštívené: 12.10.2020], `https://docs.python.org/3/library/unittest.html`.

[19] Python.org. *Python Downloads.* [online, navštívené: 21.3.2021], `https://www.python.org/downloads/`.

Dodatok A - Návody

6.2 Návod na inštaláciu nástroja

Vyvinutý softvérový nástroj je vytvorený prostredníctvom programovacieho jazyka Python. Na to, aby mohol byť nástroj spustený, je potrebné zabezpečiť inštaláciu návodom popísaným v tejto kapitole.

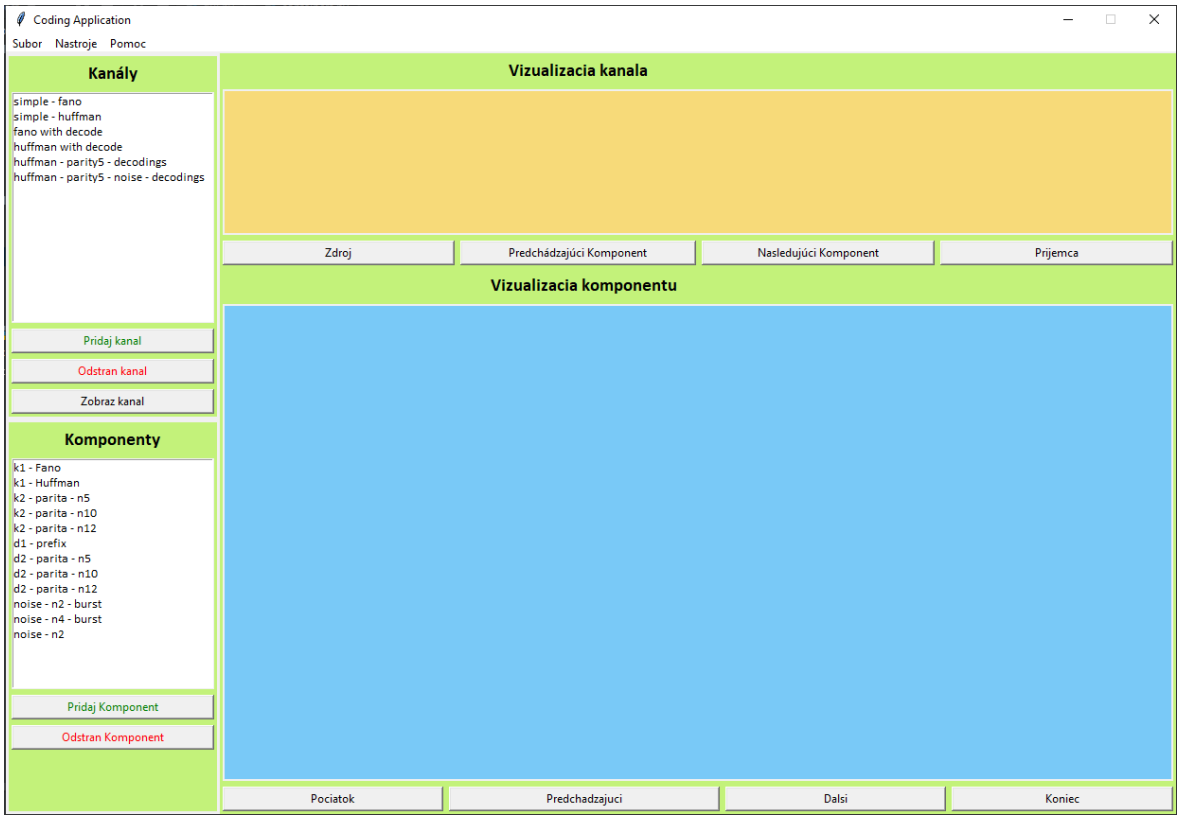
Predpokladáme, že pre distribúcie Linuxu je Python3 už nainštalovaný ako súčasť operačného systému. Preto nižšie popisujeme inštaláciu len pre operačný systém Windows.

Postup inštalácie pre operačný systém Windows je nasledovný:

1. Stiahneme inštalačný balík zo stránky `https://www.python.org/downloads/` ideálne najnovšiu verziu . najnovšia verzia v čase písania práce bola 3.9. [19]
2. Nainštalujeme stiahnutý súbor s príponou .exe. Pre menej skúseného používateľa sa stačí riadiť prednastavenou konfiguráciou a obrazovky odklikávať pomocou tlačidla Next. Pokročilejší používateľ si môže konfiguráciu počas inštalácie prispôsobiť podľa svojich požiadaviek.

Pre spustenie nie je potrebná inštalácia dodatočných modulov. V prípade problému s importom modulu tkinter, je možné inštalovať tento modul otvorením príkazového riadka, resp. terminálu a zadaním nasledujúceho príkazu (je potrebný prístup na internet bez proxy): pip install tkinter

Otvoríme priečinok projektu a spustíme súbor main.py.



Obr. 6.1: Hlavné okno nástroja

6.3 Návod pre používateľa

Od používateľa očakávame, že zvládol inštaláciu nástroja. Po otvorení aplikácie by mal používateľ vidieť obrazovku hlavn0ho okna (obrázok 6.1).

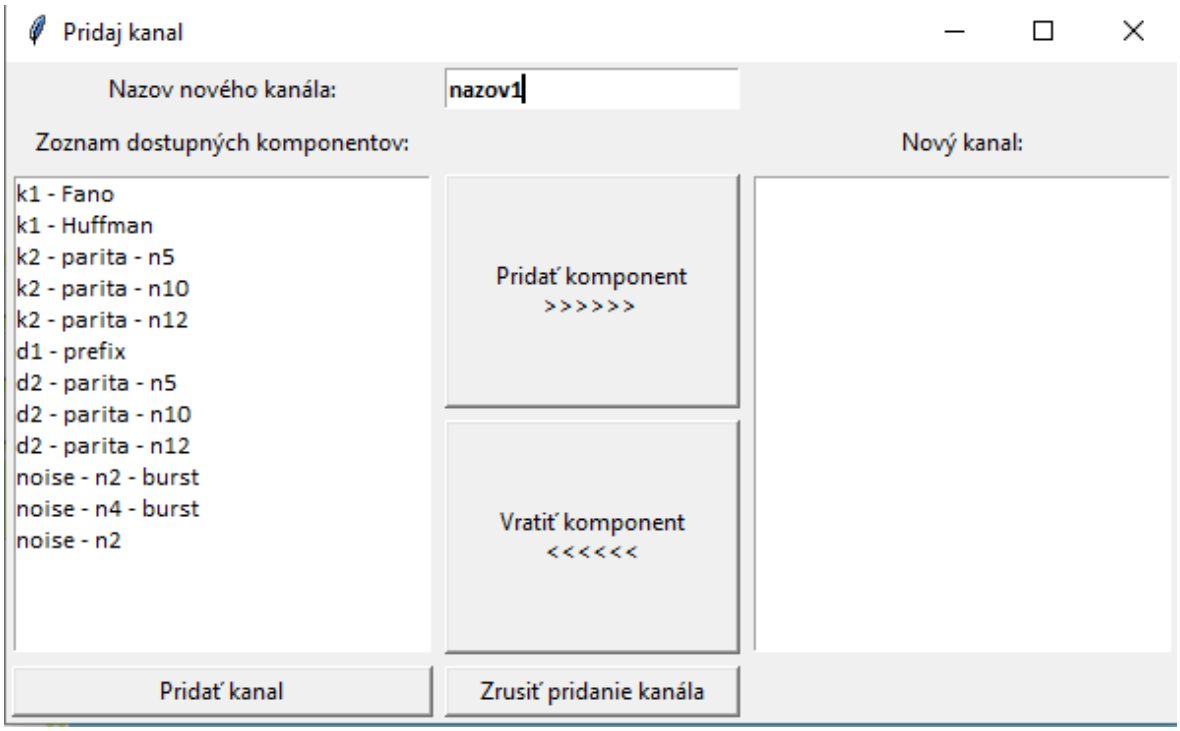
6.3.1 Postup pre prezentáciu jedného komunikačného kanála

V ľavej časti si používateľ zvolí jeden z kanálov, a klikne na tlačidlo Zobraz Kanál.

Po kliknutí na tlačidlo bol zvolený kanál načítaný a pripravený na krokovanie. Bolo vykreslené zobrazenie kanála v časti Vizualizácia Kanála. Aktuálna pozícia kanála bola nastavená na pozíciu Zdroj a zároveň bol používateľ vyzvaný na zadanie vstupu pre komunikačný kanál prostredníctvom vyskakovacieho okna s názvom Input.

Po zadaní vstupu pre kanál, a potvrdení kliknutím na tlačidlo OK, používateľ môže Zvoliť nasledujúci komponent zobrazeného kanála pomocou tlačidla Nasledujúci Komponent.

Po prechode na komponent bol zároveň v časti Vizualizácia komponentu zobrazený



Obr. 6.2: Okno pre tvorbu vlastného komunikačného kanála

prvý krok výpočtu algoritmu daného komponentu.

Prostredníctvom tlačidiel (Počiatok, Predchádzajúci, Ďalší, Koniec) v spodnej časti obrazovky sa používateľ môže presúvať medzi jednotlivými krokmi vizualizácie komponentu.

Používateľ sa pomocou tlačidiel Predchádzajúci Komponent a Nasledujúci Komponent môže pohybovať medzi jednotlivými komponentami kanála.

V ľubovoľnom stave aplikácie si používateľ môže kliknutím na tlačidlo Zdroj vyvolať nový vstup pre práve zobrazovaný kanál.

Taktiež v ľubovoľnom stave aplikácie si používateľ môže označiť, kliknutím myšou, iný predpripravený komunikačný kanál zo zoznamu kanálov a kliknutím na tlačidlo Zobraziť kanál vyvolať načítanie daného kanála.

6.3.2 Postup pre experimentovanie s komunikačným kanálom

Používateľ si môže z dostupných komponentov vytvoriť vlastný komunikačný kanál zapojením ľubovoľných komponentov do vlastnej postupnosti.

Kliknutím na tlačidlo Pridaj Kanál nástroj vyvolá okno na tvorbu nového kanála

(obrázok 6.2). Používateľ si musí zvoliť jedinečný názov nového kanála, a následne si pomocou tlačidiel Pridať komponent a "Vrátiť komponent"vytvára v pravej strane obrazovky vlastný komunikačný kanál zo zoznamu dostupných komponentov v ľavej strane okna.

Postupnosť komponentov je v plnej réžii používateľa, a medzi komponentami nie sú zavedené vôbec žiadne pravidlá, čo používateľovi poskytuje úplnú flexibilitu, avšak ak používateľ poskladá komunikačný kanál, ktorý zapojením svojich komponentov nedáva zmysel, komponenty vo vizualizačnom panely nezobrazia kroky výpočtu. Zobrazené sú iba kroky správne zapojeného komunikačného kanála. Príklad nesprávneho zapojenia je napríklad zapojenie dekodéra pred kodérom.

6.3.3 Postup pre rozšírenie funkcionality nástroja

Pod rozšírením hovoríme o rozšírení aktuálnej funkcionality nástroja, ako aj doplnenie existujúcich typov komponentov a algoritmov o nové.

V prípade doplnenia algoritmov, je nutné vytvoriť v podpriečinku codes, samostatný súbor s názvom nového algoritmu a implementovať pre tento algoritmus novú triedu, ktorá bude objektovo dediť z triedy src.Component.

Nový trieda musí obsahovať atribúty name – kde bude uložený jedinečný názov komponentu, atribút output - obsahujúci výsledok behu algoritmu a 2 metódy - run() a vizualize(). Metóda run dostane vo svojom jedinom argumente (okrem self) vstup zo zdroja, resp. Predchádzajúceho abstraktného komponentu v podobe textového reťazca. Následne v metóde vykoná predmetný algoritmu, ktorého výsledok uloží do atribútu output.

Počas vykonávania metódy run() je potrebné realizovať ukladanie priebehu výpočtu do pomocnej dátovej štruktúry, ktorá je iterovateľná.

Pre zobrazenie priebehu výpočtu na plátne "Vizualizácia komponentu"je potrebné pre funkciu vizualize() doplniť 2 argumenty (okrem self). Argument canvas je referenciou na objekt, Plátno, ktorý v hlavnom okne zobrazí priebeh výpočtu, a argument position, ktorý bude odkazovať na aktuálnu pozíciu v dátovej štruktúre uchovávajúcej priebeh výpočtu.

Následnou implementáciou tela tejto metódy ma používateľ plnú kontrolu nad tým, ktoré kroky výpočtu budú zobrazované, a akým spôsobom. Používateľ môže nechať zobraziť obsah celej dátovej štruktúry, alebo len jej časti, podľa toho, ako uzná za

vhodné.

Ak nový algoritmus pracuje s parametrami, je možné tieto parametre zadať počas inicializácie do konštruktora tohto objektu.

Pre pridanie nového komponentu do zoznamu komponentov je najjednoduchším spôsobom úprava funkcie *load_data*(), v ktorej je potrebné pridať riadok pre vytvorenie premennej, ktorá bude obsahovať inštanciu nového algoritmu, a túto inštanciu je následne potrebné pridať ako argument pre funkciu pridávania komponentov do kontajnera komponentov, pomocou príkazu *component_container.add*(). Po spustení aplikácie je komponent dostupný v zozname komponentov a následne je s ním možné pracovať v rozhraní vytvárania nového komunikačného kanála.

Dodatok B - Zdrojový kód

6.4 Zdrojový kód

Z dôvodu rozsiahlosti zdrojového kódu nie je vhodné tento kód zdieľať v písomnej podobe ako súčasť tejto práce. Preto sme zdrojový kód zaznamenali na pamäťovú SD kartu. Okrem zdrojového kódu sme na nosič umiestnili aj Dodatok A - Návody v súbore s príponou .pdf. Návody obsahujú postup inštalácie nástroja, ako aj používateľské príručky.