

**Genetické programování v
platformově nezávislém jazyce**
**Genetic Programming Based on a
Platform Independent Language**

Tuto stránku nahradíte v tištěné verzi práce oficiálním zadáním Vaší diplomové či bakalářské práce.

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava*.

Zde vložte text dohodnutého omezení přístupu k Vaší práci, chránící například firemní know-how. Zde vložte text dohodnutého omezení přístupu k Vaší práci, chránící například firemní know-how. A zavazujete se, že

1. o práci nikomu neřeknete,
2. po obhajobě na ni zapomenete a
3. budete popírat její existenci.

A ještě jeden důležitý odstavec. A ještě jeden důležitý odstavec. A ještě jeden důležitý odstavec. A ještě jeden důležitý odstavec. A ještě jeden důležitý odstavec. Konec textu dohodnutého omezení přístupu k Vaší práci.

V Ostravě 16. dubna 2009

+++
.....

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

V Ostravě 16. dubna 2009

+++
.....

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli, protože bez nich by tato práce nevznikla.

Abstrakt

Tohle je nějaký abstrakt. Tohle je nějaký abstrakt. Tohle je nějaký abstrakt. Tohle je nějaký abstrakt. Tohle je nějaký abstrakt. Tohle je nějaký abstrakt. Tohle je nějaký abstrakt. Tohle je nějaký abstrakt. Tohle je nějaký abstrakt. Tohle je nějaký abstrakt.

Klíčová slova: evoluční, genetický, genetické programování

Abstract

This is English abstract. This is English abstract. This is English abstract. This is English abstract. This is English abstract. This is English abstract. This is English abstract.

Keywords: evolutionary, genetic, genetic programming

Seznam použitých zkratek a symbolů

EP – Evoluční programování

Obsah

| | | |
|----------|--|-----------|
| 1 | Úvod | 5 |
| 2 | Předmluva | 6 |
| 3 | Evoluční algoritmus | 7 |
| 3.1 | Teorie evoluce | 7 |
| 3.2 | Úvod do teorie optimalizačních algoritmů | 8 |
| 3.3 | Společné rysy | 8 |
| 3.4 | Populace | 9 |
| 3.5 | Optimalizace a účelová funkce | 9 |
| 3.6 | Evoluční algoritmy | 10 |
| 4 | Genetické programování | 13 |
| 5 | Implementace | 14 |
| 6 | Testování a výsledky | 15 |
| 7 | Závěr | 16 |
| 8 | Reference | 17 |
| 8.1 | Výpisy programů | 18 |
| 8.2 | Obrázky a tabulky | 19 |
| | Přílohy | 19 |
| A | Grafy a měření | 23 |

Seznam tabulek

| | | |
|---|--|----|
| 1 | Pokusná tabulka | 21 |
| 2 | Experimental Files — Detailed Statistics | 22 |

Seznam obrázků

| | | |
|---|--|----|
| 1 | Pokusný obrázek – absolutní velikost | 19 |
| 2 | Pokusný obrázek – relativní velikost | 19 |
| 3 | Pokusný obrázek – otočený naležato | 20 |
| 4 | Nějaký graf | 24 |

Seznam výpisů zdrojového kódu

| | | |
|---|--|----|
| 1 | Program v jazyce Java | 18 |
| 2 | Program v jazyce Java, načtený z externího souboru | 18 |
| 3 | Program v Pascalu | 18 |

1 Úvod

Tento text je ukázkou sazby diplomové práce v \LaTeX u pomocí třídy dokumentů `diploma`. Pochopitelně text není skutečnou diplomovou prací, ale jen ukázkou použití implementovaných maker v praxi. V kapitole ?? jsou ukázky použití různých maker a prostředí. V kapitole 7 bude „jako závěr“. Zároveň tato kapitola slouží jako ukázka generování křížových odkazů v \LaTeX u.

2 Předmluva

Evoluční algoritmy vznikly jako řešení a jeden ze způsobů optimalizace v matematických a vědních kruzích. Celkem dlouho byl způsob optimalizace řešen dnes již klasickým matematickým aparátem, který je založen na infinitezimálním počtu, na variačních metodách aplikovaných ve funkcionálním prostoru nebo numerických metodách.

Tímto způsobem však lze nalézt optimální řešení pro problémy jednodušší charakter a pro ty složitější umožňuje nalézt pouze suboptimální řešení.

3 Evoluční algoritmus

Před tím, než se pustíme do popisu toho, co jsou to evoluční algoritmy a na jakých principech fungují, se seznámíme s tím, co jsou to „evoluční výpočetní techniky“.

3.1 Teorie evoluce

Podle klasické Darwinové a Mendelovy teorie evoluce, je uznáváno dogma, podle něhož se jednotlivé druhy vyvíjejí tak, že jsou z rodičů plozeni potomci, kteří podléhají při svém vzniku mutacím. Rodiče a potomci nevhodní pro aktuální životní prostředí vymírají cyklicky po tzv. generacích, čímž uvolňují místo novým rodičům a jejich potomkům.

Příklad 3.1

Postup evolučního algoritmu

1. Vymezení parametrů evoluce – jako je stanovení kritéria ukončení (počet cyklu, vhodnost řešení, ...), stanovení účelové funkce, případně tzv. vhodnosti. Účelovou funkcí se rozumí obvykle matematický model, jehož minimalizace/maximalizace vede k řešení
2. Generování prvopočáteční populace (obecně matice $M \times N$, kde M je počet parametrů jedince a N je počet jedinců v populaci). Jedincem se rozumí vektor čísel s takovým počtem složek, kolik je optimalizovaných parametrů. Složky jsou nastaveny nahodile a každý jedinec představuje jedno možné řešení
3. Všichni jedinci se ohodnotí přes definovanou účelovou funkci a každému z nich se přiřadí: a) buď přímá hodnota vrácená účelovou funkcí, nebo b) vhodnost, což je upravená hodnota účelové funkce
4. Nastává výběr rodičů podle jejich kvality
5. Křížením rodičů se tvoří potomci. Proces křížení je u každého algoritmu odlišný.
6. Každý potomek je zmutován
7. Každý jedinec se ohodnotí stejně jako v kroku 3.
8. Vyberou se nejlepší jedinci
9. Vybraní jedinci zaplní novou populaci
10. Stará populace je zapomenuta a na její místo nastupuje populace nová. Dále se pokračuje krokem 4.

Evoluční algoritmy nejsou populární jen proto, že jsou moderní a odlišné od klasických, ale hlavně pro fakt, že v případě vhodného aplikování jsou schopny nahradit člověka ■

Poznámka 3.1 Následující definice a věty nedávají dohromady příliš smysl. Jsou tu jen pro ukázkou.

3.2 Úvod do teorie optimalizačních algoritmů

Optimalizační algoritmy jsou mocným nástrojem pro řešení mnoha problémů inženýrské praxe. Obvykle se používají tam, kde je řešení daného problému analytickou cestou nevhodné či nereálné. Většina problémů inženýrské praxe může být definována jako optimalizační úlohy, např. nalezení optimální trajektorie robota či optimální tloušťka tlakové nádoby. Řešení takových problémů obvykle vyžaduje práci s argumenty optimalizovaných funkcí.

Algoritmy této třídy mají svůj specifický název, a to „evoluční algoritmy“. Řeší problémy tak elegantně, že se staly velmi oblíbené a používané v mnoha inženýrských oborech. Z hlediska nejobecnějšího členění patří evoluční algoritmy k algoritmům heuristickým. Heuristické algoritmy můžeme rozdělit na deterministické a stochastické. Algoritmy druhé skupiny se liší v tom, že některé jejich kroky využívají náhodné operace, a to znamená, že výsledné řešení, které jimi získáme, se v jednotlivých bězích programu mohou lišit.

Stochastické heuristické metody poskytují pouze obecný rámec a vlastní operace algoritmu je třeba zvolit (operace křížení, mutace, ...). Protože tyto metody se často inspiřují přírodními procesy, jsou také nazývány evoluční algoritmy. Podle jejich strategie je lze rozdělit do dvou tříd

3.3 Společné rysy

Evoluční algoritmy mají několik společných rysů:

1. Jednoduchost, protože tyto algoritmy lze naprogramovat obvykle velmi jednoduše.
2. Hybridnost čísel, se kterými algoritmus pracuje. Bez jakýchkoliv problémů lze kombinovat čísla typu integer, real, případně jen vybrané množiny čísel.
3. Používání dekadických čísel. Jedinec se nemusí převádět do binárního kódu, který je běžně používán u genetických algoritmů. U binárního zápisu mohou totiž mutace způsobit skokovou změnu čísla, což nemusí být dobrý dopad na průběh evoluce. Tyto nerovnoměrnosti se sice dají odstranit použitím Grayova kódování, nicméně práce s reálnými čísly je stále výhodnější.
4. Rychlost. Díky své relativní jednoduchosti, zvláště při porovnání s klasickými metodami, lze říci, že požadované řešení naleznou mnohem rychleji.
5. Chopnost nalézt extrém i u funkcí, které jsou v grafickém slova smyslu ploché a extrém je jen dírou v této rovině. Lze to označit jako hledání jehly v kupce sena. Ovšem u těchto problémů je účinnost jakýchkoliv algoritmů, včetně evolučních, velmi nízká.
6. Schopnost dát vícenásobné řešení. Výsledkem evoluce je nejlepší jedinec – jedno řešení. Pokud je ovšem v globálních extrémech více, pak lze očekávat, že budou rovněž evolučním procesem nalazeny. Jinými slovy, evoluční algoritmy jsou vhodné pro

hledání extrémů funkcí trpících takovými patologiemi, jako např. šum, vysoký počet dimenzí, „multimodalita“

3.4 Populace

Typickým rysem evolučních algoritmů je, že jsou založeny na práci s populací jedinců. Populace může být znázorněna jako matice $N \times M$, kde sloupce představují jednotlivé jedince. Každý jedinec představuje aktuální řešení daného problému. S každým jedincem je navíc spojena hodnota účelové funkce, která říká, jak vhodný je jedinec pro další vývoj populace.

Hlavní činnosti evolučních algoritmů je cyklické vytváření nových populací, tedy náhrada starých populací novými. To vše pomocí přesně definovaných matematických pravidel.

K vytvoření populace je třeba nadefinovat tzv. vzor, podle kterého se generuje celá počáteční populace. Ve vzoru jsou pro každý parametr konkrétního jedince definovány tři konstanty, a to typ proměnné a hranice intervalu, v němž může pohybovat hodnota parametru. Volba hranice je velmi důležitý krok, protože při jejich nevhodném zvolení se může stát, že budou nalezena řešení, která nebudou možné fyzikálně realizovat nebo nebudou mít opodstatnění. Další neméně důležitý význam hranic souvisí se samotným evolučním procesem. Může se stát, že daný problém bude reprezentován plochou, která bude nabývat lokálních extrémů stále větších hodnot se vzrůstající vzdáleností od počátku. To způsobí, že evoluce bude nacházet stále nová řešení až do nekonečna. Je to způsobeno tím, že evoluční proces směřuje do stále hlubších a vzdálenějších extrémů.

Populace je na základě vzorového jedince vygenerována podle vzorce.

$$\Theta(1 + \alpha).$$

Tento vztah zajišťuje, že všechny parametry jedinců budou náhodně vygenerovány uvnitř povolených hranic prostoru možných řešení.

Zobrazení o tom, jak kvalitně proběhla evoluce, se provádí pomocí tzv. historie vývoje hodnoty účelové funkce ve formě jednoduchého grafu. Na něm je vykreslena závislost vývoje účelové funkce na aktuálním počtu jejich ohodnocení. Jde o sekvenci nejhorších a nejlepších řešení z jednotlivých populací. Výhodnější je však zobrazení závislosti hodnoty účelové funkce na aktuálním počtu jejich ohodnocení. To proto, že u evolučních cyklů se provádí u jednotlivých algoritmů různý počet ohodnocení účelové funkce. U prvního případu může být pomalejší konvergence hodnoty účelové funkce zobrazena jako rychlejší a naopak. Skutečná informace o kvalitě evoluce je pak zkreslená. U druhého způsobu můžeme objektivně porovnávat různé typy algoritmů bez ohledu na jejich vnitřní strukturu. Kromě vývoje nejlepšího jedince je pak vhodné zobrazovat vývoj i nejhoršího jedince z populace, a to do jednoho grafu.

Vývoj populace musí být vždy konvergentní k lepším hodnotám, což znamená, že nemůže nikdy vykazovat divergenci. V daném algoritmu funguje tzv. „elitismus“, který slouží jako jakýsi jednosměrný filtr, jenž propouští do nové populace pouze ta řešení, která jsou lepší či stejně dobrá jako ta ze staré populace.

3.5 Optimalizace a účelová funkce

3.5.1 Vybrané pojmy z optimalizace

Úlohy vedoucí k výpočtu extrémů funkce jsou obvykle úlohy praktické činnosti člověka a vyžadují analytické a mnohem častěji numerické výpočty extrémů funkcí více proměnných. Existují však komplikace, které ztěžují optimalizaci některých problémů, mající tyto zdroje:

- Prostor možných řešení je příliš velký. To může být například u numerativního přístupu (procházení všech možností) dosti velkým problémem
- Samotný problém je natolik složitý, že jeho matematický model použitý při optimalizaci vrací výsledky, které sice odpovídají příslušnému modelu, ale nesouhlasí se skutečně řešeným problémem. Model je vůči realitě nepřesný.
- Účelová funkce použitá k měření kvality aktuálního nalezeného řešení může podléhat šumu nebo se může měnit v čase. Proto nestačí pouze jedno řešení, ale množina řešení reprezentující vývoj toho nejlepšího.
- Množina řešení podléhá striktnímu omezení, že nalezení optimálního řešení je extrémně složitý problém.

Historický původ optimalizačních úloh sahá do antiky. Z té doby známe úlohy, jako je Didonina úloha, která měla od krále přislíbenou zemi, kterou by ohraničila volskou kůží. Didonina kůží nařezala na tenké pásy a ohraničila území, které se stalo základem Kartága.

Impulz pro rozvoj optimalizačních metod po druhé světové válce přinesly úlohy z oblasti ekonomie, které lze shrnout pod název problému optimalizace výrobních programů. Typicky se v nich předpokládá, že podnik má technické možnosti pro výrobu n druhů výrobků a hledá výrobní program x . Matematicky toto můžeme zapsat jako $h(k)$ je $\Theta(1 + \alpha)$, kde udává počet výrobků tohoto druhu. Takový výrobní program přináší podniku užitek. O této funkci mluvíme jako o funkci účelové. Podnik však musí pro konkrétní program respektovat řadu omezení (výrobní zdroje) a rovněž omezení odbytu výrobků.

Jinou skupinou optimalizačních úloh pak může být problém úloh z dopravního hospodářství. V těchto problémech jde o to, aby byl navržen plán přepravy zboží s míst výroby do míst spotřeby. Respektují se omezení na množství přepravovaného zboží, požadavky spotřebitelů a příp. další. Výsledkem by pak měl být plán na omezení nákladů na přepravu a nebo čas přepravy.

3.6 Evoluční algoritmy

Seznam několika evolučních algoritmů

3.6.1 Stochastic Hill Climbing

Stochastický horolezecký algoritmus je verze tzv. horolezeckého algoritmu obohaceného o stochastickou složku. Patří mezi gradientní metody, tzn. že prohledávají prostor možných řešení ve směru největšího spádu. Díky své gradientní povaze velmi často uvízne v lokálním extrému. Vždy se vychází z náhodného bodu v prostoru možných řešení. Pro momentálně navržené řešení se pomocí konečného souboru transformací navrhne určité okolí a daná funkce se minimalizuje jen v tomto okolí. Získané lokální řešení se pak použije jako střed pro výpočet nového okolí.

3.6.2 Simulované žíhání

Je vylepšená verze horolezeckého algoritmu, u něhož vylepšení spočívá v tom, že je do horolezeckého algoritmu zavedena tzv. krátkodobá paměť, jejímž úkolem je pamatovat si ty transformace, které sloužily pro vypočítání aktuálního středu. To má v konečném důsledku ten efekt, že se algoritmus nezacyklí, díky zakázanému použití těchto transformací.

Na rozdíl od algoritmu horolezeckého, Tabu Search tak často neuvízne v lokálním extrému.

3.6.3 Evoluční strategie

Evoluční strategie patří mezi první úspěšné stochastické algoritmy v historii. Byl navržen počátkem šedesátých let Rechenbergem a Schwefelem. Vychází z principů přirozeného výběru podobně jako genetické algoritmy. Na rozdíl od jiných stochastických algoritmů pracuje evoluční strategie přímo s reálnými hodnotami. Jejím jádrem je práce s řešením ve formě vektoru x , které je mutováno pomocí vektoru náhodných čísel.

3.6.4 Optimalizace mravenčí kolonií

Optimalizace mravenčí kolonií je algoritmus, jehož činnost napodobuje chování mravenců v kolonii. Princip je následovný: Necht' existuje zdroj mravenců (mraveniště) a cíl jejich snažení (potrava). Když jsou mravenci vypuštěni, tak po nějaké době dojde k tomu, že všichni mravenci se pohybují po kratší (optimální) cestě mezi zdrojem a cílem. Tento efekt, kdy mravenci najdou optimální cestu je dán tím, že si svou cestu značkují feromonem. Pokud dorazí první mravenec k rozcestí dvou cest, které vedou ke stejnému cíli, pak je jeho rozhodnutí, po které cestě se vydá, náhodné. Ti, kteří najdou potravu, začnou cestu značkovat a při návratu jsou díky těmto značkám při rozhodování ovlivněni ve prospěch této cesty. Při návratu ji označují podruhé, což opět zvyšuje pravděpodobnost rozhodnutí dalších mravenců v její prospěch. Tyto principy jsou použity v ACO algoritmu.

Feromonová váha je aditivní, což umožňuje přidávat další feromony od dalších mravenců. V ACO algoritmu je zohledněn i fakt vypařování feromonů tak, že váhy jednotlivých spojů s časem slábnou. To zvyšuje robustnost algoritmu z hlediska nalezení globálních extrémů.

ACO byl úspěšně použitý na optimalizování problému Obchodního cestujícího nebo při návrhu telekomunikačních sítí.

3.6.5 Imunitní systém

Metoda imunitního systému je, jak už název napovídá, algoritmus, který je založen na principech fungování imunitního systému v živých organismech. Nahlíží se na něj jako na multiagentní systém, kde jednotliví agenti mají svůj specifický úkol. Tito agenti mají různé pravomoci a schopnosti komunikovat s jinými agenty. Na základě této komunikace a určité svobody v rozhodování jednotlivých agentů vzniká hierarchická struktura schopná řešit komplikované problémy. Může například jít o použití antivirové ochrany u velkých rozsáhlých počítačových sítí.

3.6.6 Memetický algoritmus

Memetický algoritmus. Tento pojem představuje širokou třídu metaheuristických algoritmů. Klíčovou charakteristikou těchto algoritmů je použití různých aproximačních algoritmů, technik lokálního vyhledávání, speciálních rekombinačních operátorů apod. V podstatě mohou být memetické algoritmy charakterizovány jako strategie soutěže a spolupráce projevující atributy synergetiky. Jako příklad memetického algoritmu lze uvést hybridní kombinaci genetických algoritmů a simulovaného žitání či paralelní lokální prohledávání.

S úspěchem byly použity na řešení takových problému, jako problém obchodního cestujícího, učení neuronové sítě, plánování údržby, nelineární celočíselné

3.6.7 Rozptýlené prohledávání

Rozptýlené prohledávání se svou podstatou liší od standardních evolučních algoritmů a je dost podobný algoritmu Tabu Search. Je to vektorově orientovaný algoritmus, který má za úkol generovat nové vektory na základě pomocných heuristických technik. Při startu se vychází z řešení získaných pomocí vhodné heuristiké techniky. Poté jsou generována nová řešení na základě podmnožiny nejlepších řešení ze startu. Z těchto nově nalezených řešení se opět vybere množina těch nejlepších a celý proces se opakuje. Tento algoritmus byl použit k řešení problémů, jako je řízení dopravy, učení neuronové sítě, optimalizace bez omezení, atd.

3.6.8 Rojení částic

Rojení částic je založeno na práci s populací jedinců, jejichž pozice v prostoru možných řešení je měněna pomocí tzv. rychlostního vektoru. Podle popisu nedochází v základní verzi mezi jedinci k vzájemnému ovlivňování. To je odstraněno ve verzi s tzv. sousedstvím. V rámci tohoto sousedství dochází k vzájemnému ovlivňování tak, že jedinci patří do jednoho sousedství putují k nejhlubšímu extrému, který byl v tomto sousedství nalezen.

3.7 Optimalizační problémy

Zde budou popsány některé optimalizační problémy

3.7.1 Problém obchodního cestujícího

Problém obchodního cestujícího (TSP) je úloha z množiny NP-úplných problémů.

3.7.2 Problém sudoku

Jedná se o problém nalezení řešení k danému schématu sudoku mřížky, pokud máme k dispozici pouze několik vyplněných čísel.

4 Genetické programování

Sekce, která obsahuje genetické programování

5 Implementace

Tohle je sekce zabývající se implemntaci aplikace

6 Testování a výsledky

Tohle je sekce zabývající se implemntaci aplikace

7 Závěr

Tak doufám, že Vám tato ukázka k něčemu byla. Další informace najdete v publikacích [1, 2].

Zdeněk Gold

8 Reference

- [1] Goossens, Michel, *The L^AT_EX companion*, New York: Addison, 1994.
- [2] Lamport, Leslie, *A document preparation system: user's guide and reference manual*, New York: Addison-Wesley Pub. Co., 2015.

Definice 8.1 *Binární strom je struktura definovaná nad konečnou množinou uzlů, která:*

- *neobsahuje žádný uzel,*
- *je složena ze tří disjunktních množin uzlů: kořene, binárního stromu zvaného levý podstrom a binárního stromu tzv. pravého podstromu.*

Pak by se taky mohla hodit nějaká věta a k ní důkaz.

Věta 8.1 *Průměrná časová složitost neúspěšného vyhledání v hashovací tabulce se separátním zřetěžením je $\Theta(1 + \alpha)$, za předpokladu jednoduchého uniformního hashování.*

Důkaz. Za předpokladu jednoduchého uniformního hashování se každý klíč k hashuje se stejnou pravděpodobností do libovolného z m slotů tabulky. Průměrný čas neúspěšného hledání klíče k je proto průměrný čas prohledání jednoho z m seznamů. Průměrná délka každého takového seznamu je rovna faktoru naplnění $\alpha = n/m$. Tudíž lze očekávat, že budeme nuceni prozkoumat α prvků. Z toho plyne, že celkový čas pro neúspěšné hledání (plus navíc konstantní čas pro výpočet $h(k)$) je $\Theta(1 + \alpha)$. ■

Příklad 8.1

Mějme napsat funkci, která spočítá uzly ve stromu. Předpokládejme, že binární strom je definován způsobem uvedeným v definici 8.1 na straně 17. Naše úloha se výrazně zjednoduší uvědomíme-li si její rekurzivní charakter a předpokládáme, že aktuální uzel je R .

- Je-li R prázdný strom (tj. $R = \text{NULL}$), pak počet jeho uzlů je pochopitelně nula. Tím máme problém vyřešen.
- V opačném případě víme, že ve stromu určitě jeden uzel existuje (R) a počty uzlů v levém a pravém podstromu se dají určit obdobným způsobem rekurzivně. To znamená, že počet uzlů ve stromu s kořenem R je $1 + \text{pocet_uzlu}(A) + \text{pocet_uzlu}(B)$

Počty uzlů pro jednotlivé podstromy se předávají jako výsledky volání funkcí prostřednictvím zásobníku programu, nejsou tudíž potřeba žádné pomocné proměnné. ■

Poznámka 8.1 Program z příkladu 8.1 pochopitelně chybí, ale můžete se podívat třeba na program uvedený ve výpisu 1.

8.1 Výpisy programů

Tato diplomová práce má nastaven výchozí jazyk Java, jak je vidět z výpisu 1. Výpis kódu 1 zároveň demonstuje možnost přímého vložení zdrojového kódu programu do textu práce. Druhou možností je načtení zdrojového kódu programu z externího souboru, viz výpis 2. Pokud potřebujeme změnit programovací jazyk pro konkrétní výpis kódu, můžeme jeho to provést přímo v záhlaví prostředí `lstlisting`. Výpis 3 je v jazyku Pascal. Všimněte si zvýraznění klíčových slov.

Poznámka 8.2 Pro správnou sazbu je třeba pro odsazování používat tabulátory, nikoliv mezery.

```
public class MyClass
{
    public int MyMethod(int a, int b)
    {
        while (a != b)
        {
            if (a < b)
                b -= a;
            else
                a -= b;
        }
    }
}
```

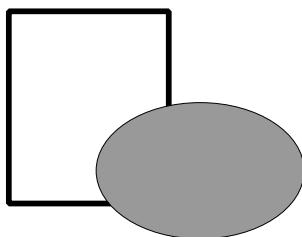
Výpis 1: Program v jazyce Java

```
public class MyClass
{
    public int MyMethod(int a, int b)
    {
        while (a != b)
        {
            if (a < b)
                b -= a;
            else
                a -= b;
        }
    }
}
```

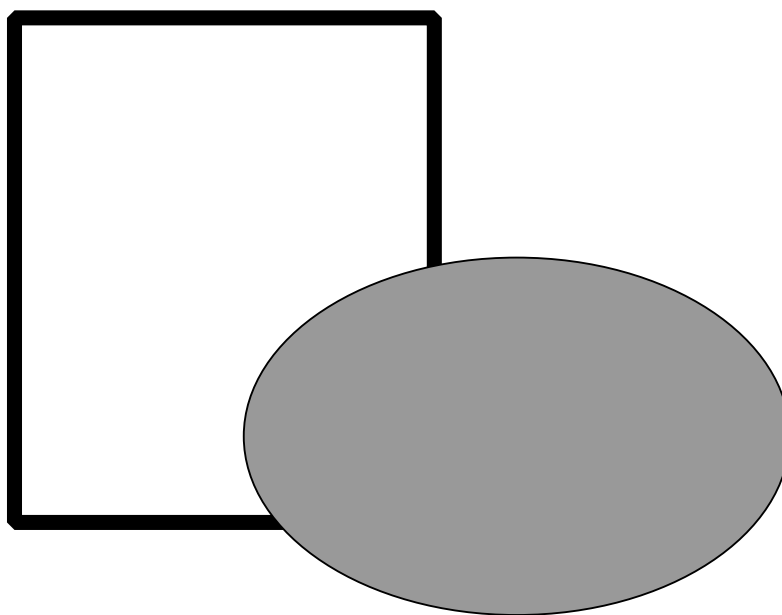
Výpis 2: Program v jazyce Java, načtený z externího souboru

```
procedure X(i : integer; var x : real);
begin
    x := i + 3;
end;
```

Výpis 3: Program v Pascalu



Obrázek 1: Pokusný obrázek – absolutní velikost



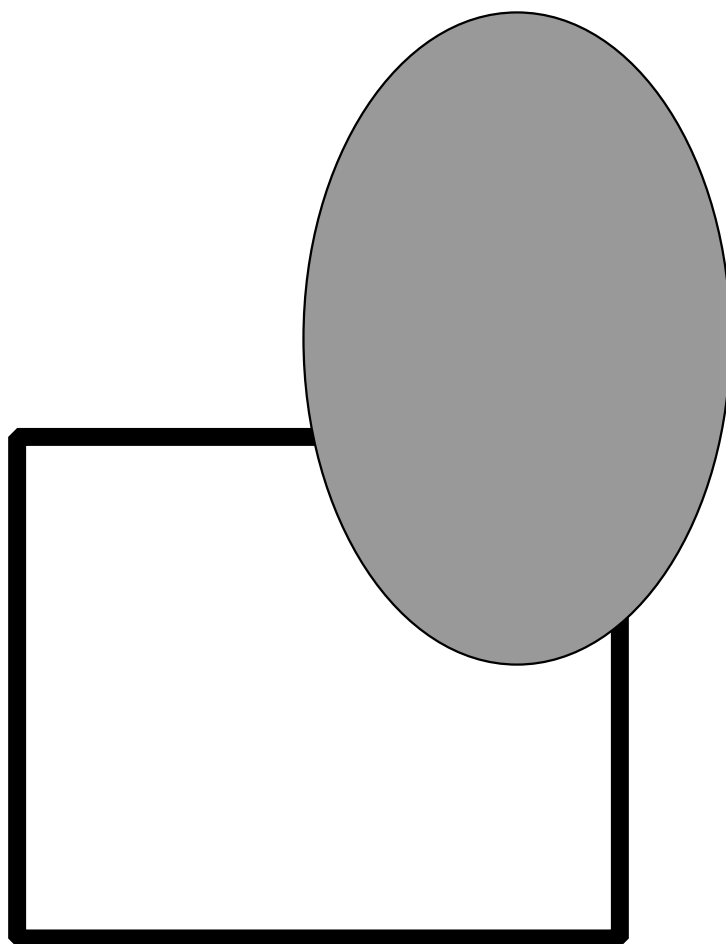
Obrázek 2: Pokusný obrázek – relativní velikost

8.2 Obrázky a tabulky

A ještě si můžeme zkusit vysázet obrázek. Obrázek 1 má určenu absolutní velikost, zatímco obrázek 2 je určen relativně vůči šířce textu.

A ještě zkusíme vysázet několik tabulek, ale jen kvůli seznamu tabulek v úvodu. Tabulka 1 představuje jednoduchou tabulku, která se svou šířkou pohodlně vejde do šířky textu. Velké tabulky, stejně jako obrázky, můžeme vysázet naležato. Ukázkou velké, komplikované tabulky¹ je tabulka 2.

¹Pokud, ale píšete práci česky, měly by být tabulky také česky – mě se jen nechtěla předělávat do češtiny.



Obrázek 3: Pokusný obrázek – otočený naležato

| q | $\delta(q, 0)$ | $\delta(q, 1)$ |
|----------|----------------|----------------|
| q_0 | q_1 | q_0 |
| q_1 | q_1 | q_2 |
| q_2 | q_1 | q_0 |

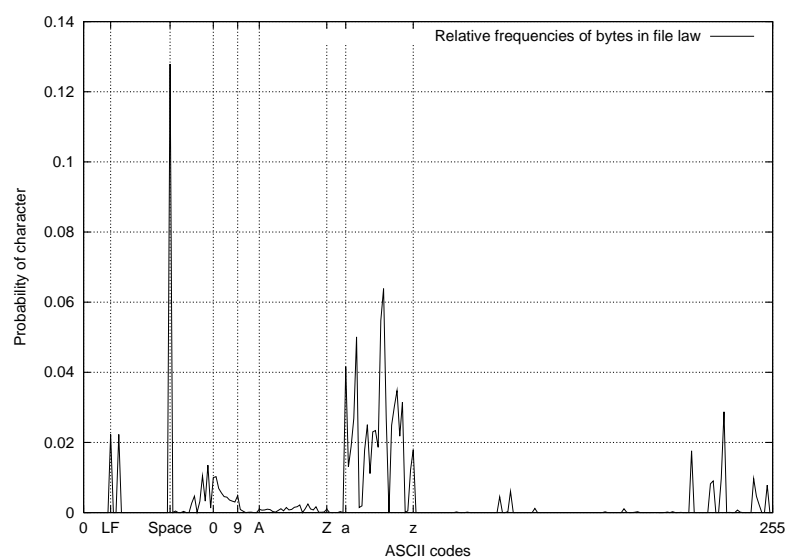
Tabulka 1: Pokusná tabulka

| File | bible.txt | world.txt | law.txt | latimes.txt |
|---------------------------|--------------------|--------------------|------------------|--------------|
| Language Format | English Plain text | English Plain text | Czech Plain text | English SGML |
| Size of file [bytes] | 4047392 | 2473400 | 64573143 | 498360166 |
| Number of tokens | 1532262 | 684767 | 19432898 | 161254928 |
| Number of words | 766131 | 342383 | 9716449 | 70766067 |
| Number of nonwords | 766131 | 342384 | 9716449 | 80619289 |
| Number of controls | | | | 9869572 |
| Number of unique tokens | 13791 | 23564 | 250570 | 529482 |
| Number of unique words | 13744 | 23082 | 246266 | 524280 |
| Number of unique nonwords | 47 | 482 | 4304 | 3079 |
| Number of unique controls | | | | 2123 |
| Word average frequency | 55.743 | 14.833 | 39.455 | 134.978 |
| Nonword average frequency | 16300.66 | 710.34 | 2257.539 | 26183.595 |
| Control average frequency | | | | 4648.88 |
| Minimal length of word | 1 | 1 | 1 | 1 |
| Maximal length of word | 18 | 27 | 41 | 58 |
| Minimal length of nonword | 1 | 1 | 1 | 1 |
| Maximal length of nonword | 4 | 56 | 700 | 253 |
| Minimal length of control | | | | 3 |
| Maximal length of control | | | | 132 |

Tabulka 2: Experimental Files — Detailed Statistics

A Grafy a měření

Tohle je příloha k práci. Většinou se sem dávají grafy, tabulky, které by vzhledem ke svému počtu překážely v textu diplomky.



Obrázek 4: Nějaký graf