VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

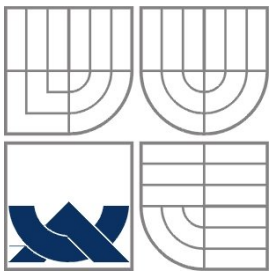
ALGORITMY VYHLEDÁVÁNÍ ŘETĚZCŮ V TEXTU A ALGORITMY REKURZE V JAZYCE C

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

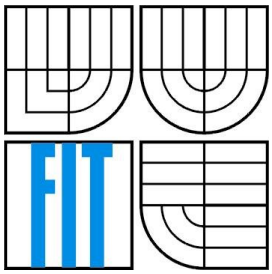
AUTOR PRÁCE
AUTHOR

JIŘÍ PORČ

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ALGORITMY VYHLEDÁVÁNÍ ŘETĚZCŮ V TEXTU A ALGORITMY REKURZE V JAZYCE C

STRING SEARCHING AND RECURSIVE ALGORITHMS IN C LANGUAGE

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — JIŘÍ PORČ
AUTHOR

VEDOUCÍ PRÁCE — prof. Ing. Jan M. Honzík, CSc.
SUPERVISOR

BRNO 2007

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2006/2007

Zadání bakalářské práce

Řešitel: **Porč Jiří**
Obor: Informační technologie
Téma: **Algoritmy vyhledávání řetězců v textu a algoritmy rekurze v jazyce C**
Kategorie: Alg. a datové struktury

Pokyny:

1. Seznamte se detailně s textem kapitol 6 a 7 o vyhledávání podřetězců v řetězcích a o rekurzívních algoritmech ve studijní opoře pro předmět IAL a s dalšími algoritmy v dostupné literatuře.
2. Modifikujte text kapitoly zapsané pro pascalovský jazyk tak, aby zachoval co nejvíce (i v detailech) původní obsah, ale aby se vztahoval k zápisu příkladů a alforitmů v jazyce C
3. Převeďte všechny příklady a algoritmy do jazyka C a odladťe je v prostředí vytrvořeném pro tento účel.
4. Vytvořte program pro animovanou demonstraci vybraných operací s využitím nástrojů pro grafickou reprezentaci.
5. Navrhněte další vhodné kontrolní otázky a příklady.
6. Navrhněte příklady vhodné pro formulářově orientovanou písemnou zkoušku ze znalostí tohoto okruhu.

Literatura:

- Honzík,J.M. Algoritmy. Studijní opora pro předmět Algoritmy. Elektronický text. FIT VUT v Brně

Při obhajobě semestrální části projektu je požadováno:

- Přepsané a odladěné algoritmy v jazyce C

Program pro animovanou reprezentaci funkce vyhledáváaích metod s využitím grafického rozhraní.

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

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

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

Vedoucí: **Honzík Jan M., prof. Ing., CSc.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2006
Datum odevzdání: 15. května 2007

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

L.S.

doc. Ing. Jaroslav Zendulka, CSc.
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami:

1. **Pan**

Jméno a příjmení:	Jiří Porč
Id studenta:	84976
Bytem:	Doubravice nad Svitavou 16, 679 11 Doubravice nad Svit.
Narozen:	25. 12. 1984, Brno

(dále jen „autor“)

a

2. **Vysoké učení technické v Brně**

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen „nabyvatel“)

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
   bakalářská práce

Název VŠKP:	Algoritmy vyhledávání řetězců v textu a algoritmy rekurze v jazyce C
Vedoucí/ školitel VŠKP:	Honzík Jan M., prof. Ing., CSc.
Ústav:	Ústav informačních systémů
Datum obhajoby VŠKP:	.....

VŠKP odevzdal autor nabyvateli v:

tištěné formě	počet exemplářů 1
elektronické formě	počet exemplářů 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizovaní výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti
   □ ihned po uzavření této smlouvy

   □ 1 rok po uzavření této smlouvy

   □ 3 roky po uzavření této smlouvy

   □ 5 let po uzavření této smlouvy

   □ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: ……

……... ……

Nabyvatel Autor

Abstrakt

Úloha nalézt v textu zadaný textový vzorek patří v oblasti počítačů k jedné z nejčastěji využívaných. V mé práci se zabývám problematikou vyhledávacích a rekurzivních algoritmů. Cílem mé práce je vytvoření programů pro animovanou demonstraci vybraných operací a přepsání studijní opory předmětu Algoritmy do jazyka C. Vytvořené programy pro animovanou demonstraci řeší problémy spojené s vyhledáváním textových řetězců v textu a rekurzí.

Klíčová slova

Vyhledávací algoritmy, Karp-Rabinův algoritmus, Knuth-Morris-Prattův algoritmus, Boyer-Mooreův algoritmus, rekurze, hanojské věže, osm dam

Abstract

The problem of finding a specified pattern in a text is one of the most frequently used tasks in the computers field. My work deals with the problems of searching and recursive algorithms.
The objective of my work is to create programs for animated demonstration of selected algorithms and rewriting of the Study supporting materials for the course Algorithms into the C language.
The created demonstrational programs are dealing with the problems of searching text strings in a text and recursion.

Keywords

Searching algorithms, Karp-Rabin algorithm, Knuth-Morris-Pratt algorithm, Boyer-Moore algorithm, recursion, tower of Hanoi, eight queens

Citace

Jiří Porč: Algoritmy vyhledávání řetězců v textu a algoritmy rekurze v jazyce C, bakalářská práce, Brno, FIT VUT v Brně, 2007
Algoritmy vyhledávání řetězců v textu a algoritmy rekurze v jazyce C

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením prof. Ing. Jana M. Honzíka, CSc.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jiří Porč
8.5.2007

Obsah

Úvod

Původní studijní opora předmětu Algoritmy (*IAL*) byla psána jazykem *PASCAL*. Vzhledem k tomu, že se v dnešní době více využívá jazyka *C* než jazyka *PASCAL*, začal se na Fakultě informačních technologií od roku 2004 vyučovat jazyk *C*. Převedením studijní opory předmětu Algoritmy do jazyka C se studentům usnadní studium problematiky algoritmů. Studenti budou moci studovat algoritmy v obou jazycích. Při přepsání studijní opory se také předpokládá kontrola a případné doladění všech algoritmů. Mým úkolem je přepsat kapitoly 6 a 7 ze studijní opory o vyhledávání podřetězců v řetězci, rekurzi a dále také vytvoření programů pro animovanou demonstraci. Tyto programy mají sloužit budoucím studentům k jednoduššímu a hlavně názornějšímu pochopení problematiky vyhledávání podřetězců v řetězci a rekurze. Moje práce je rozdělena na dvě hlavní části. V první části se věnuji vyhledávání podřetězců v řetězci. Na začátku je nejprve krátký teoretický úvod, pak vysvětlení Karp-Rabinova vyhledávacího algoritmu, příklad vyhledávání a na konci první části je popis prvního demonstračního programu SubSearchString. V druhé části mé práce se zabývám rekurzivními algoritmy. Tato část také začíná teoretickým úvodem, následuje popis programu pro animovanou demonstraci. Celá práce je zakončena zhodnocením mé činnosti a práce.
Součástí přílohy jsou převedené kapitoly 6 a 7 ze studijní opory předmětu Algoritmy, odladěné algoritmy z kapitol 6 a 7, vyhotovené programy pro animovanou demonstraci vyhledávacích algoritmů a rekurze, vzorové otázky a příklady pro formulářově orientovanou zkoušku.

1 Postup práce

Před zahájením vlastní práce bylo nutné pečlivě prostudovat celé zadání. Po prostudování zadání následovalo rozvrhnutí postupu práce.

Na začátku jsem prostudoval kapitoly 6 a 7 o vyhledávání podřetězců v řetězcích a o rekurzivních algoritmech ze studijní opory předmětu Algoritmy. Při studování těchto kapitol jsem narazil na několik drobných překlepů jak ve vlastním textu studijní opory, tak i v algoritmech. Překlepy a nejasnosti jsem si poznamenal, aby je bylo možné opravit i v původním textu studijní opory. Jako další cíl jsem si zvolil modifikaci textu kapitol z jazyka *PASCAL* do jazyka *C*, ale brzy jsem zjistil, že nejprve bude nutné přepsat a odladit všechny algoritmy ze studijní opory a až poté modifikovat texty kapitol. Proto jsem nejdříve přistoupil k odladění všech algoritmů. Některé algoritmy nebyly zcela funkční, protože obsahovaly chyby vzniklé překlepy nebo například kopírováním.

Odladění algoritmů jsem prováděl v prostředí operačního systému *LINUX*. Pro překlad algoritmů ze studijní opory v jazyce *PASCAL* (z důvodu porovnání funkčnosti s algoritmy mnou přeloženými do jazyka *C*) jsem využil překladač jazyka *PASCAL* (Free Pascal Compiler). Odladěné algoritmy v jazyce *C* jsou součástí přílohy mé bakalářské práce. U všech algoritmů je soubor *makefile*, který se postará o překlad (po zadání příkazu *make*). Při odlaďování a převádění algoritmů jsem se snažil, aby převedené algoritmy co nejvíce (i v detailech) zachovaly původní obsah, ale aby se vztahovaly k zápisu příkladů a algoritmů v jazyce *C*.

Po odladění a dosažení funkčnosti algoritmů jsem se vrátil zpět k modifikaci textu kapitol studijní opory. Kapitoly studijní opory jsem přepsal do jazyka *C*. Dále jsem se začal věnovat tvorbě programů pro animovanou demonstraci. Nejprve bylo nutné vybrat vhodný nástroj, který by byl dostatečně jednoduchý, jednoduše se v něm dalo vytvořit grafické uživatelské prostředí, byl použitelný vzhledem k algoritmům, které mám implementovat a byl nenáročný na pochopení pro případné zájemce ze strany studentů. Až jsem měl vybráno vývojové prostředí, mohl jsem začít s vlastní implementací.

Jako první jsem se rozhodl vytvořit program pro animovanou demonstraci znázorňující problém hanojských věží a osmi dam. Protože se jedná o dva rekurzivní problémy, zahrnul jsem je oba do stejného demonstračního programu. Program je popsán a jeho princip vysvětlen níže. Jako další demonstrační program jsem vytvořil program na demonstrování vyhledávání podřetězců v řetězci. Tento program implementuje naivní algoritmus, Knuth-Morris-Prattův a Boyer-Mooreův algoritmus. Také tento program je popsán a vysvětlen níže.

Po dokončení programů pro animovanou demonstraci jsem začal se studiem další dostupné literatury. Po prostudování literatury jsem se rozhodl přidat do programu demonstrujícího vyhledávání podřetězců algoritmus Karb-Rabinův. Princip algoritmu je vysvětlen níže.

Po přidání Karp-Rabinova algoritmu do demonstračního programu jsem navrhl kontrolní otázky a příklady a následně také příklady vhodné pro formulářově orientovanou zkoušku ze znalostí tohoto okruhu.

1.1 Výběr vývojového prostředí

Pro vytvoření aplikací pro animovanou demonstraci bylo nejprve nutné vybrat vhodné vývojové prostředí. Vhodnost vývojového prostředí jsem posuzoval podle několika následujících kritérií.

Vybrané prostředí muselo být dostatečně jednoduché (abych se mohl zabývat pouze problematikou algoritmů), jednoduše se v něm dalo vytvořit grafické uživatelské prostředí (aby výsledná aplikace měla příjemné uživatelské rozhraní), aby bylo použitelné vzhledem k algoritmům, které mám implementovat a nenáročné na pochopení pro případné zájemce ze strany studentů.

Rozhodoval jsem se mezi jazykem *C* s využitím knihovny GLUT, Windows API, Microsoft Visual Studiem a flash animací. Při použití jazyka *C* s knihovnou GLUT by bylo značně komplikované vytvořit kvalitní grafické uživatelské rozhraní. Při použití Windows API by se sice vytvoření uživatelského rozhraní usnadnilo, ale i v tomto případě by to bylo ještě stále dost složité. Využití flash animací jsem zamítl, protože jsem nemohl předpokládat, že všichni studenti budou znát problematiku flash animací.

I z tohoto posledního zmíněného důvodu jsem se rozhodl pro využití C++ .NET v Microsoft Visual Studiu. Lze v něm velmi jednoduše a přehledně vytvořit propracované grafické uživatelské prostředí, je použitelné pro algoritmy ze studijní opory. Dalším neméně důležitým aspektem je to, že pro studenty, kteří ovládají jazyk *C*, není příliš komplikované pochopit zápisy, algoritmy a zdrojové kódy zapsané v jazyce C++. Je také velmi pravděpodobné, že mnoho studentů (v době studia předmětu Algoritmy) již má osobní zkušenost s Microsoft Visual Studiem a bude pro ně proto snadnější pracovat ve známém a vyzkoušeném prostředí.

2 Vyhledávání v textu

Vyhledávání v textu je akce, při které zjišťujeme, zda se hledané slovo - *vzorek* nachází v prohledávaném textu. Úloha nalézt v textu zadaný textový vzorek patří v oblasti počítačů k nejčastějším. Algoritmy využívající funkce pro hledání vzorku se používají například v textových editorech (pohyb po editovaném textu, nahrazování slov, zvýrazňování syntaxe), jádrech operačních systémů, vyhledávání souborů podle názvu, v antivirových programech, při studiu DNA, při analýze obrazu a zvuku, atd. Velikost prohledávaných dat může být velmi různá. Může se pohybovat například od desítek kilobajtů (v textových souborech) až po stovky megabajtů (jádra systémů, informační systémy,vědecké práce,...) i mnohem více. Při velkých objemech prohledávaných dat hraje velmi významnou roli efektivnost vyhledávacích algoritmů, která může velmi výrazně ovlivnit dobu hledání a také efektivnost celého systému. Proto je velmi důležitá volba správného vyhledávacího algoritmu.

2.1 Rozdělení vyhledávacích algoritmů

Vyhledávací algoritmy můžeme rozdělit podle mnoha kritérií. Všechna tato kritéria jsou ale vztažena buď k vyhledávanému vzorku nebo k prohledávanému textu.

Jedním z možných rozdělení algoritmů je rozdělení podle toho, zda potřebují předzpracování vzorku nebo předzpracování textu. Následující tabulka ukazuje, jaké předzpracování potřebují použité algoritmy.

Tabulka znázorňující předzpracování použitých algoritmů

	Předzpracování	
	Vzorku	Textu
Naivní alg.	NE	NE
KMP alg.	ANO	NE
BM alg.	ANO	NE
Karp-Rabinův	ANO	NE

2.2 Hashovací tabulky

Protože jsem do textu zahrnul i Karp-Rabinův algoritmus, který využívá ke své činnosti hashovací funkce, bude nezbytné, abych uvedl i základní princip hashovacích tabulek a funkcí.

Hashovací tabulky nabízí možnost, jak vytvářet velice efektivní tabulky, kde při splnění několika předpokladů, dosáhneme velké účinnosti. Přímo adresovatelné tabulky a hashovací tabulky jsou ve své podstatě jen rozšířením standardních polí. Přímo adresovatelné tabulky používají přímo klíče jako indexy v poli, hashovací tabulky transformují prostor klíčů o velmi velké mohutnosti pomocí hashovací funkce do relativně malého prostoru indexu pole.

2.2.1 Přímo adresované tabulky

Přímo adresované tabulky je jednoduchá technika, která dobře pracuje, pokud má množina klíčů malou mohutnost. Předpokládejme, že aplikace potřebuje ke své činnosti dynamicky se měnící množinu a množina klíčů není velká. Dále předpokládejme, že žádné dva prvky nemají shodné klíče. Pro zastoupení takové množiny použijeme pole nebo přímo adresovatelnou tabulku *T[0..m-1],* kde *m* není velké. Každá pozice náleží nějakému klíči z množiny. Pozice *k* ukazuje na prvek s klíčem *K*. Jestliže množina neobsahuje prvek s klíčem *K*, pak tato pozice má hodnotu *NULL*. Všechny operace jako je vkládání, mazaní a výběr prvku jsou velice rychlé. V mnoha případech je možné uchovávat prvky množiny přímo v tabulce. Není tedy nutné mít v tabulce jen klíče na ukazatele na prvky množiny.

2.2.2 Hashovací tabulky

Hlavním problémem přímého adresování je to, že v případě, že množina prvků je velká, je udržování tabulky *T* dost nepraktické a neefektivní. Přesto množina aktuálně uložených klíčů *K* může být relativně malá vzhledem k množině. Jestliže množina klíčů *K* je mnohem menší než množina všech možných klíčů, spotřebuje hashovací tabulka mnohem méně paměťového místa než tabulka s přímým adresováním.

V přímo adresovatelné tabulce je prvek s klíčem *K* uložen na pozici *k*. V hashovací tabulce je ale uložen na pozici *h(k)*, kde *h()* je hashovací funkce. Hashovací funkce převádí množinu klíčů na pozice hashovací tabulky. Hlavním účelem hashovací funkce je převod jednotlivých klíčů do jednotlivých, konkrétních pozic. Tím se snižují paměťové nároky. Problémem ale zůstává, pokud by se hashovací funkcí dva klíče zobrazily do jedné pozice. Pak dojde k takzvané kolizi. Existuje několik způsobů, jak uvedený problém řešit.

Nejlepším způsob jak zabránit kolizím, je jim předcházet. To znamená nalézt takovou hashovací funkci, která by kolizím zabránila úplně nebo alespoň minimalizovala jejich počet. Kvalitním návrhem hashovací tabulky a hashovací funkce lze výrazně minimalizovat počet kolizí. Další velmi důležitou vlastností hashovací funkce je, že pro jeden konkrétní klíč určí vždy za každých podmínek stejnou pozici.

2.3 Vyhledávací algoritmy

2.3.1 Karp-Rabinův algoritmus

V roce 1980 došli R. M. Karp a M. O. Rabin k závěru, že prohledávání textů není příliš vzdáleno od standardních vyhledávacích metod a přišli s algoritmem, který je určen k prohledávání textu a využívá hashovací funkci. Hashování nabízí jednoduchou možnost, jak se ve většině případů vyhnout kvadratické složitosti vyhledávání.

Místo toho, abychom pro každou možnou pozici v textu zkoumali, jestli se zde vzorek vyskytuje nebo ne, je výhodnější zkoumat jen ty pozice, které jsou podobné hledanému vzorku. K vyjádření podobnosti mezi hledaným vzorkem a částí textu (shodné délky jako vzorek) použijeme hashovací funkci.

V textu budeme hledat pouze ty úseky, které mají shodnou hashovací hodnotu jako hledaný vzorek. Algoritmus si můžeme představit tak, že máme pomyslnou hashovací tabulku, do které vkládáme části textu. Části textu, kterým hashovací funkce přidělí jiný index než vzorku, nemusíme vůbec zkoumat. Zkoumat musíme jen ty části, kterým hashovací funkce přidělí stejný index jako vyhledávanému vzorku textu tj. sledujeme jen kolidující části textu. Při nalezení kolidující části textu jej musíme porovnat znak po znaku. Dobře navržená hashovací funkce by měla kolize eliminovat.

Správná hashovací funkce je taková, která je snadno vypočitatelná, citlivá na změny řetězce, při posunu v textu by se nový stav hashovací funkce měl dát snadno odvodit z předchozího a s malou pravděpodobností kolizí. Předzpracování vzorku v Karp-Rabinově algoritmu znamená výpočet hashovací funkce pro hledaný vzorek.

Příklad prohledávání textu:

Prohledávaný text : „GCATCGCAGAGAGTATACAGTACG“
Hledaný vzorek : „GCAGAGAG“

Hashovací hodnota hledaného vzorku: hash(GCAGAGAG) = 17597.
První porovnání:

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
G	C	A	G	A	G	A	G																

hash(y[0 . . . 7]) = 17819

Druhé porovnání:

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
	G	C	A	G	A	G	A	G															

hash(y[1 . . . 8]) = 17533

Třetí porovnání:

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
		G	C	A	G	A	G	A	G														

hash(y[2 . . . 9]) = 17979

Čtvrté porovnání:

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
			G	C	A	G	A	G	A	G													

hash(y[3 . . . 10]) = 19389

Páté porovnání:

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
				G	C	A	G	A	G	A	G												

hash(y[4 . . . 11]) = 17339

Šesté porovnání:

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
					G	C	A	G	A	G	A	G											

hash(y[5 . . . 12]) = 17597 = hash(x)

Protože se hashovací hodnota vzorku rovná hashovací hodnotě části textu, je nutné pomocí n porovnání (v našem případě osmi) zjistit, zda se jedná skutečně o námi hledaný vzorek.

Sedmé porovnání:

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
						G	C	A	G	A	G	A	G										

hash(y[6 . . . 13]) = 17102

Osmé porovnání:

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
							G	C	A	G	A	G	A	G									

hash(y[7 . . . 14]) = 17117

Deváté porovnání:

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
								G	C	A	G	A	G	A	G								

hash(y[8 . . . 15]) = 17678

Desáté porovnání:

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
									G	C	A	G	A	G	A	G							

hash(y[9 . . . 16]) = 17245

Jedenácté porovnání:

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
										G	C	A	G	A	G	A	G						

hash(y[10 . . . 17]) = 17917

Dvanácté porovnání:

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
											G	C	A	G	A	G	A	G					

hash(y[11 . . . 18]) = 17723

Třinácté porovnání:

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
												G	C	A	G	A	G	A	G				

hash(y[12 . . . 19]) = 18877

Čtrnácté porovnání:

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
													G	C	A	G	A	G	A	G			

hash(y[13 . . . 20]) = 19662

Patnácté porovnání:

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
														G	C	A	G	A	G	A	G		

hash(y[14 . . . 21]) = 17885

Šestnácté porovnání:

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
															G	C	A	G	A	G	A	G	

hash(y[15 . . . 22]) = 19197

Sedmnácté porovnání:

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
																G	C	A	G	A	G	A	G

hash(y[16 . . . 23]) = 16961

Karp-Rabinův algoritmus provedl 17 porovnání hashovacích hodnot a 8 porovnání znaků.

2.3.2 Knuth-Morris-Prattův algoritmus

Knuth-Morris-Prattův algoritmus vychází z analýzy Morris-Prattova algoritmu. Zlepšení spočívá v prodloužení délek posunů. Jeho princip funkčnosti a získání předzpracování hledaného vzorku je vysvětleno ve studijní opoře a proto se jím zde nebudu nadále zabývat.

Příklad prohledávání textu stejným ukázkovým vzorkem jako u předchozího algoritmu:

Prohledávaný text : „GCATCGCAGAGAGTATACAGTACG“

Hledaný vzorek : „GCAGAGAG“

Předzpracování vzorku:

	0	1	2	3	4	5	6	7	8
x[i]	G	C	A	G	A	G	A	G	
FAIL[i]	-1	0	0	-1	1	-1	1	-1	1

První porovnání:

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
G	C	A	G	A	G	A	G																

Posun o 4

Druhé porovnání:

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
				G	C	A	G	A	G	A	G												

Posun o 1

Třetí porovnání:

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
					G	C	A	G	A	G	A	G											

Posun o 7

Čtvrté porovnání:

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
												G	C	A	G	A	G	A	G				

Posun o 1

Páté porovnání:

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
													G	C	A	G	A	G	A	G			

Posun o 1

Šesté porovnání:

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
														G	C	A	G	A	G	A	G		

Posun o 1

Sedmé porovnání:

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
															G	C	A	G	A	G	A	G	

Posun o 1

Osmé porovnání:

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
																G	C	A	G	A	G	A	G

Posun o 1

Knuth-Morris-Prattův algoritmus provedl celkem 18 porovnání znaků.

2.3.3 Boyer-Mooreův algoritmus

Boyer-Mooreův algoritmus je považován za jeden z nejefektivnějších vyhledávacích algoritmů pro běžné aplikace. Zjednodušená verze tohoto algoritmu je základem implementace funkce „najdi“ a „nahrad“ v textových editorech.

Boyer-Mooreův algoritmus je představitelem protisměrných vyhledávacích algoritmů. To znamená, že prochází znaky ve vzorku zprava doleva. Má několik vylepšení vůči metodě Knuth-Morris-Pratta. Pokud během porovnávání zjistíme, že požadovaný znak není součástí hledaného vzorku, můžeme přeskočit dopředu o celou délku vzorku. Algoritmus se tedy věnuje méně porovnávání shod a více porovnává neshody (neshody se statisticky vyskytují mnohem častěji). Proto je posun v porovnávání velmi často značně větší než jedna.

Algoritmus je tedy docela jednoduchý a hlavně rychlý. Stejně jako u Knuth-Morris-Prattova algoritmu, i zde musíme provést předzpracování vzorku, abychom vytvořili tabulku posunů. I princip a výpočet předzpracování vzorků je ve studijní opoře a proto je nebudu dále popisovat.

Příklad prohledávání textu:

Prohledávaný text : „GCATCGCAGAGAGTATACAGTACG“

Hledaný vzorek : „GCAGAGAG“

Předzpracování vzorku:

c	A	C	G	T
CharJump[c]	1	6	2	8

i	0	1	2	3	4	5	6	7
x[i]	G	C	A	G	A	G	A	G
MatchJump[i]	7	7	7	2	7	4	7	1

První porovnání:

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
G	C	A	G	A	G	A	G																

Posun o 1

Druhé porovnání:

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
	G	C	A	G	A	G	A	G															

Posun o 4

Třetí porovnání:

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
					G	C	A	G	A	G	A	G											

Posun o 7

Čtvrté porovnání:

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
												G	C	A	G	A	G	A	G				

Posun o 4

Páté porovnání:

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
																G	C	A	G	A	G	A	G

Posun o 7

Boyer-Mooreův algoritmus provedl celkem 17 porovnání znaků.

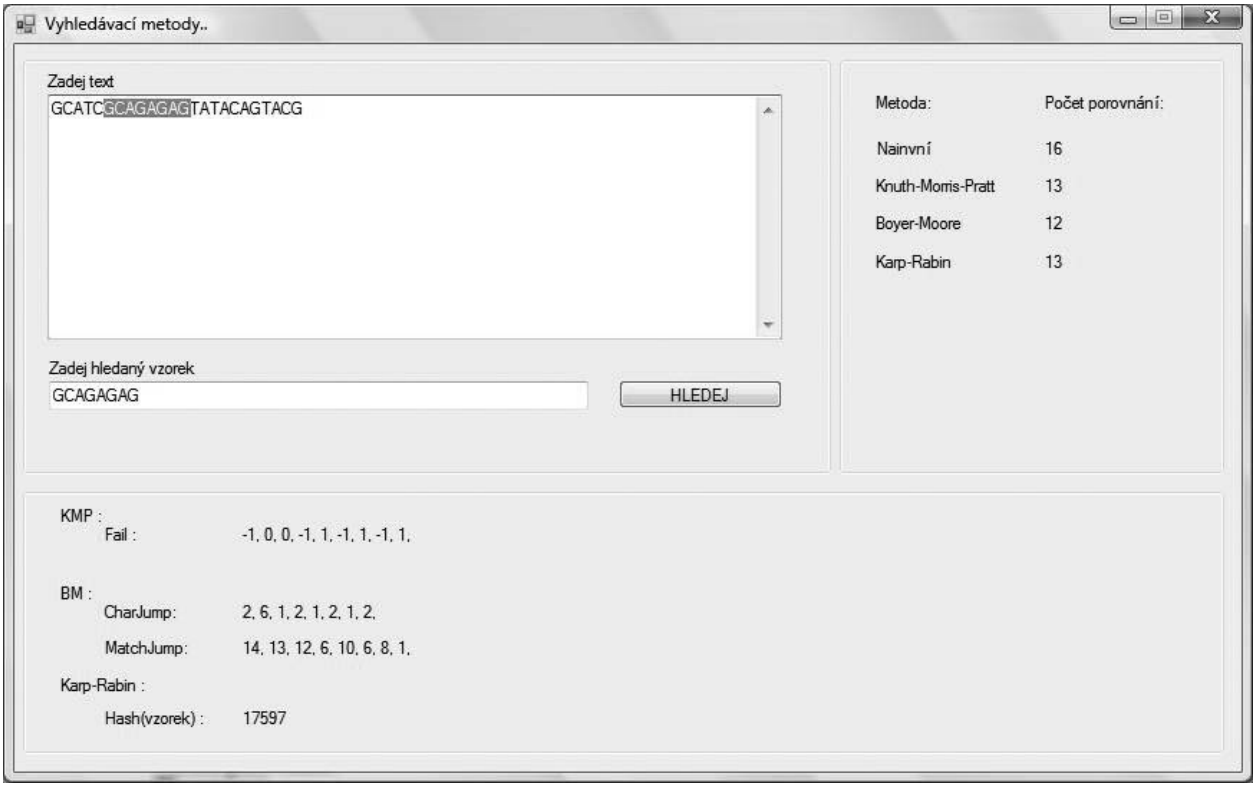
2.4 Popis programu SubSearchString

Demonstrační program SubSearchString slouží k porovnání vyhledávací schopnosti několika algoritmů. Program porovnává algoritmy jako jsou naivní algoritmus, Knuth-Morris-Prattův, Boyer-Mooreův a Karp-Rabinův algoritmus.

Po spuštění programu se zobrazí okno, které je rozděleno na několik částí. V první části, která je umístěna vlevo, se nachází textové pole pro zadání textu, který má být prohledáván, textové pole pro vzorek, který má být v textu hledán a tlačítko, které vyhledávání spustí. V další části okna, která je umístěna vlevo od textového pole, se nachází sloupec určující metodu hledání a sloupec, který určuje, kolik porovnání daná vyhledávací metoda musela provést k prvnímu úspěšnému nalezení vzorku. Program SubSearchString hledá jen první výskyt vzorku v textu, proto vždy zcela neodpovídá počet vyhledávání složitosti vyhledávací metody. V poslední, spodní části, jsou zobrazeny vzorky předzpracování, které algoritmy využívají k urychlení vyhledávání v textu. Tyto vzorky jsou spočítány před zahájením vlastního hledání a jsou vypočítány ze vzorku, který má být hledán.

Text, který chceme prohledávat, zapíšeme to textového pole pod nápisem *Zadej text*. Pokud nebude text zadán, není možné vyhledávat a hledání nemůže pokračovat. Vzorek, který má být v textu hledán, zadáme do textového pole pod nápisem *Zadej hledaný vzorek*. Ani bez zadaného vzorku nebude hledání úspěšné. Pokud jsme zadali prohledávaný text i hledaný vzorek, můžeme začít

s vyhledáváním stiskem tlačítka *Hledej*. Po stisku tlačítka se provede kontrola, jestli jsou zadány jak text, ve kterém se hledá, tak vyhledávaný vzorek. V případě, že text nebo vzorek nebyl zadán, budeme upozorněni na nemožnost zahájení hledání. Pokud byl vzorek i text zadán, proběhne nejprve předzpracování vzorku a následně vyhledání vzorku v textu. Pří úspěšném nalezení vzorku v textu, bude tento vzorek zvýrazněn. Pokud hledaný vzorek nalezen nebude, bude uživatel upozorněn.



Obr. 1 Program SubSearchString

3 Rekurze

3.1 Hanojské věže

Hanojská věž je jedním z asi nejznámějších a nejpopulárnějších algoritmů, demonstrujících princip rekurze. Je to jednoduchý hlavolam, založený na principu přemisťování různě velkých dřevěných kotoučů na třech kolících. Na začátku jsou na jednom z kolíků kotouče uspořádány od nejmenšího po největší a nejmenší je nahoře. Proto se také Hanojské věži někdy říká Hanojská pyramida.

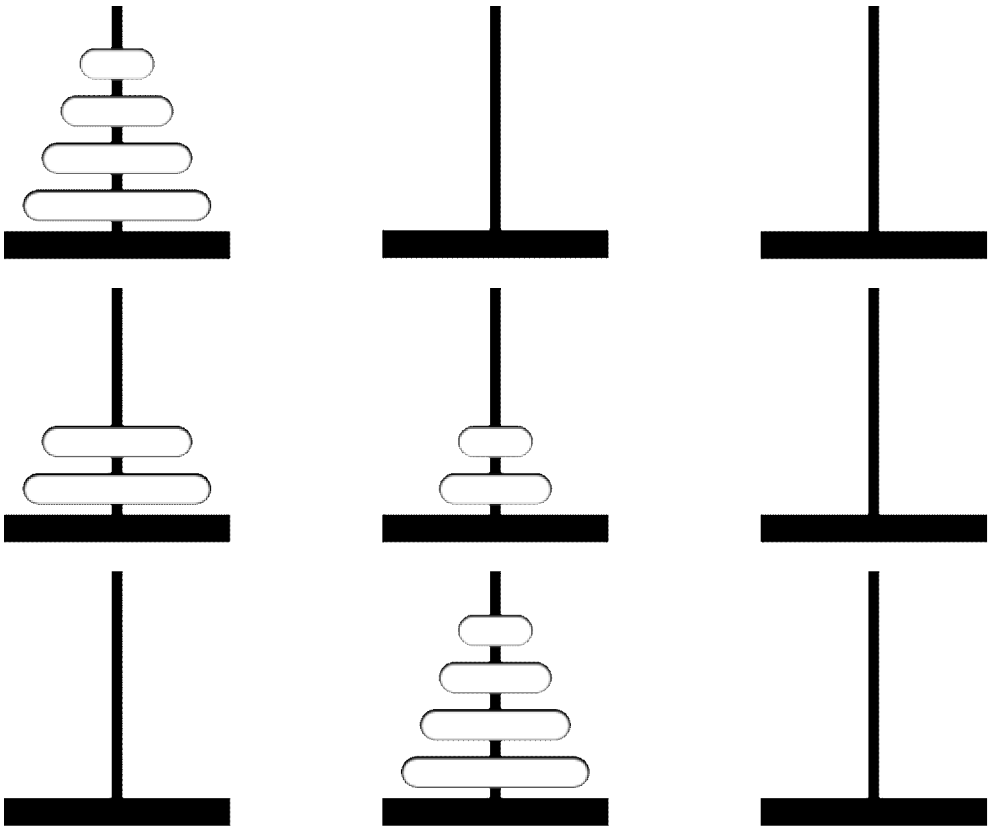
Úkolem je přeskládat kotouče tak, aby vytvořily stejnou pyramidu na jednom ze dvou zbývajících kolíků. Nesmí se přemisťovat více než jeden kotouč a nikdy nesmí být větší kotouč na menším.

Možná to vypadá jednoduše, ale s počtem kotoučů počet tahů exponenciálně roste. Zároveň roste složitost. Vypočítat minimální počet tahů pro vyřešení úlohy lze pomocí rovnice $x=2^n-1$ přičemž n je počet pater kotoučů, se kterými hrajeme a x je minimální dosažitelný počet tahů.

Tabulka závislosti počtu pater na počtu tahů

Počet pater	3	4	5	6	7	8	9	10	11
Počet tahů	7	15	31	63	127	255	511	1023	2047

Hanojská věž byla vynalezena pravděpodobně v Tibetu a zdejší mniši dospěli k názoru, že vyřešit věž s 64 patry bude trvat do konce světa. Řekněme, že přesun jednoho patra a uvažování trvá 3 sekundy. Proto na vyřešení věže se 64 patry je potřeba 18 446 744 073 709 551 615 tahů (2^{64}-1). Tedy 55 340 232 221 128 654 845 sekund (počet tahů * 3). To je skoro 1 800 000 000 000 let (jeden bilion osm set miliard let). Tato teorie o tibetských mniších je ale pravděpodobně legenda, kterou si vymyslel roku 1883 francouzský matematik Edouard Lucas, který přišel s problémem hanojských věží.



Obr. 2 Hanojské věže

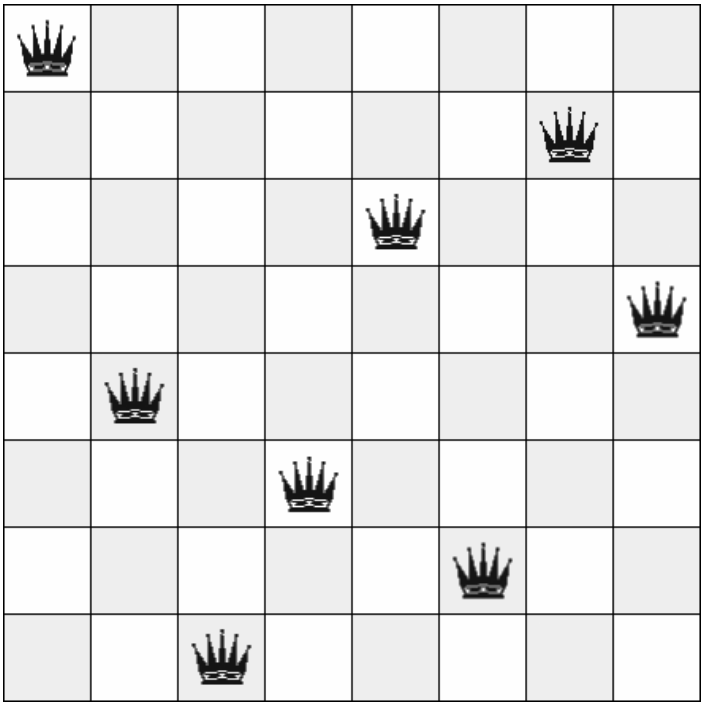
3.2 Osm dam

Problém osmi dam je další z řady populárních algoritmů. Jedná se o problém, kdy řešitel musí na šachovnici o rozměrech osm na osm polí položit osm hracích kamenů (dam) , které mají vlastnosti jako šachová dáma tak, aby se kameny navzájem neohrožovaly.

S problémem osmi dam původně přišel šachový hráč Max Bazzel roku 1848 a v průběhu let se mnoho matematiků snažilo najít co nejefektivnější řešení. Tuto úlohu lze řešit například algoritmy s návratem. To znamená, že při hledání řešení se ubíráme první možnou cestou a v případě, že dojdeme k místu, ze kterého už nalezení úspěšného řešení není možné, tak se vracíme o jednu pozici zpět a pokračujeme další možnou cestou.

Tabulka počtu řešení v závislosti na rozměru šachovnice

Rozměr	4	5	6	7	8	9	10	11	12
Počet řešení	2	10	4	40	92	352	724	2680	14200



Obr. 3 Osm dam

3.3 Popis programu Hanojské věže, osm dam

Demonstrační program *Hanojské věže, osm dam* slouží ke zjednodušenému pochopení rekurzivního problému hanojských věží a osmi dam a také názorné ukázce řešení těchto problémů.

Při spuštění je program v režimu zobrazení hanojských věží. Jsou zobrazeny tři tyče (kolíky) a na prvním jsou nasunuty kotouče. Ihned po spuštění je zobrazeno 5 kotoučů na první tyči. Počet kotoučů může uživatel programu změnit. Změnu je možné provést tažením *trackbaru* (posuvného jezdce) nazvaného *Počet kruhů*. Nejmenší možný počet nastavených kruhů jsou 3. Pro 2 kruhy tato úloha nemá význam. Největší možný počet je 10 kruhů, což pro názornou ukázku řešení problému jistě stačí. Pro provedení změny počtu kotoučů je nutné stisknout tlačítko *Inicializace*.

Pokud bylo stisknuto tlačítko *Inicializace* při zapnutém časovači, tak je časovač vypnut. Tlačítko *Inicializace* přesune na první tyč počet kruhů podle nastavení *trackbaru* a tím obnoví výchozí stav hanojských věží. Pro inicializaci programu je mimo stisku tlačítka *Inicializace* také možné použít klávesovou zkratku *ALT + I.* Po inicializaci má uživatel možnost postupně krokovat až k nalezení úspěšného řešení.

Krokování se provádí stiskem klávesy > pro krokování vpřed a stiskem klávesy < pro krokování zpět. Další možnost pro nalezení úspěšného řešení je s využitím vestavěného časovače. Časovač lze spustit tlačítkem *Spustit*, pokud je vypnut a vypnout tlačítkem *Zastavit*, pokud je zapnut. Pokud je časovač spuštěn, není možné použít klávesy > nebo < pro manuální krokování. Klávesová zkratka pro spuštění a zastavení časovače je *ALT + T.*

Rychlost běhu časovače lze ovlivnit nastavením trackbaru (posuvného jezdce) nazvaného *Rychlost*. Na tomto trackbaru je možné měnit rychlost běhu časovače od 5% do 100% po kroku 5%. Pokud je hodnota rychlosti nastavena na nižší hodnotu, běží časovač i přeskládávání kotoučů pomaleji, než když je zvolena vyšší procentuelní hodnota. Rychlost je možné měnit i za běhu časovače.

Ve spodní části okna je stavový řádek. Na něm jsou zobrazeny informace o počtu zbývajících kroků, nastavené rychlosti a celkovém počtu kotoučů k přeskládání. Tyto informace se mění plynule podle nastavení trackbarů *Rychlost* a *Počet kruhů*. Pouze informace *Počet zbývajících kroků* se mění v závislosti na počtu zbývajících kroků k vyřešení problému hanojských věží.

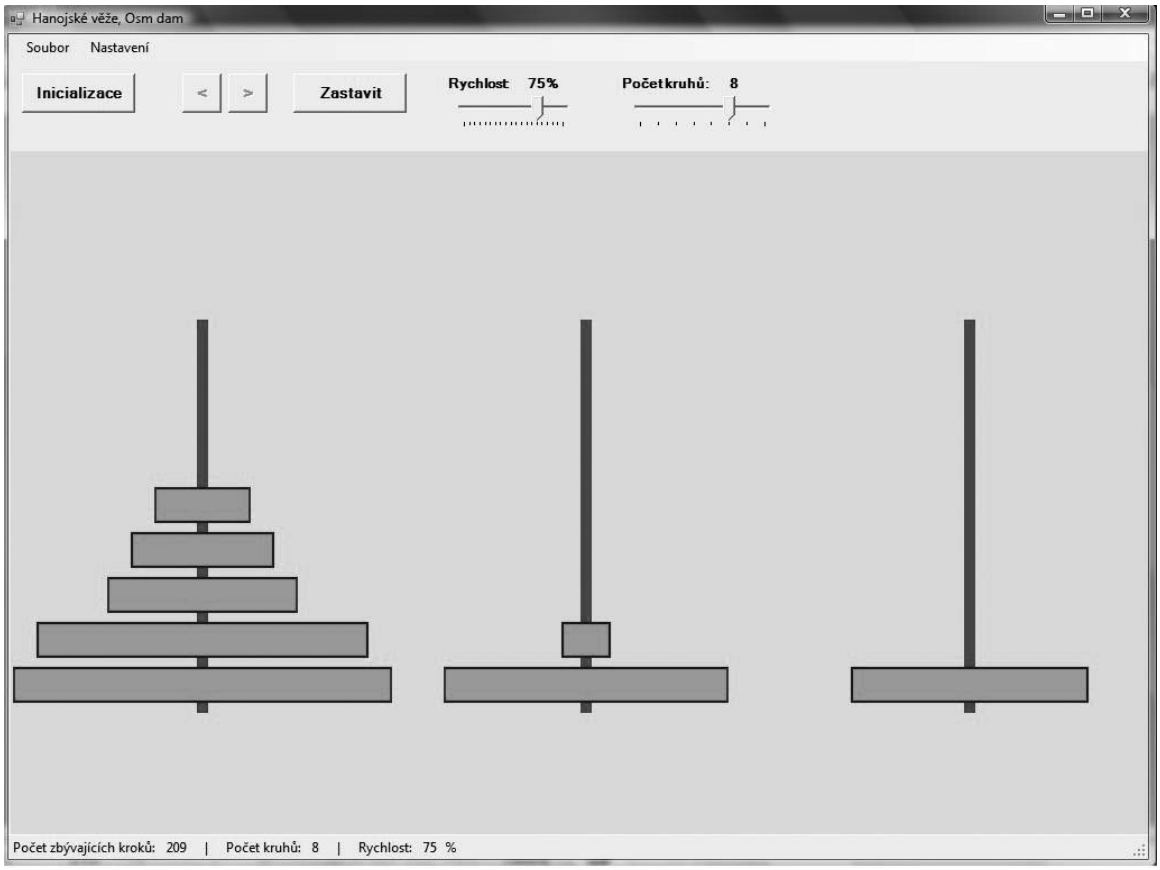
Po spuštění je program připraven řešit problém hanojských věží. Uživatel má ale také možnost jej přepnout do režimu řešení problému osmi dam. Přepnutí je možné provést v menu programu a to v *Nastavení ->Osm dam* případně zpět na hanojské věže *Nastavení ->Hanojské věže*. Pro přepnutí je také možné použít klávesovou zkratku *CTRL+ O* pro osm dam a *CTRL+ H* pro přepnutí na hanojské věže. Menu programu dále umožňuje inicializaci a ukončení programu. Inicializace se v menu skrývá pod *Soubor->Inicializace*. Inicializace v menu má stejný význam jako stisk tlačítka *Inicializace* a lze pro ni využít navíc klávesovou zkratku *CTRL+I*. Ukončení programu je možné provést v menu a to v *Soubor->Konec*. Pro ukončení lze taktéž použít klávesovou zkratku *CTRL+K*.

Při přepnutí programu do režimu řešení problému osmi dam uživatel uvidí šachovnicové pole zobrazující 8X8 čtverců. V programu jsou již položené kameny znázorněny černou barvou a aktuálně položený kámen (poslední) má barvu šedou. Z důvodu, že tento program slouží hlavně k pochopení problému osmi dam a jeho řešení, hledá algoritmus pouze první možnou kombinaci kamenů na šachovnici. Toto nalezené řešení ale není jediné, existuje jich daleko více (pro osm dam má tato úloha 92 řešení).

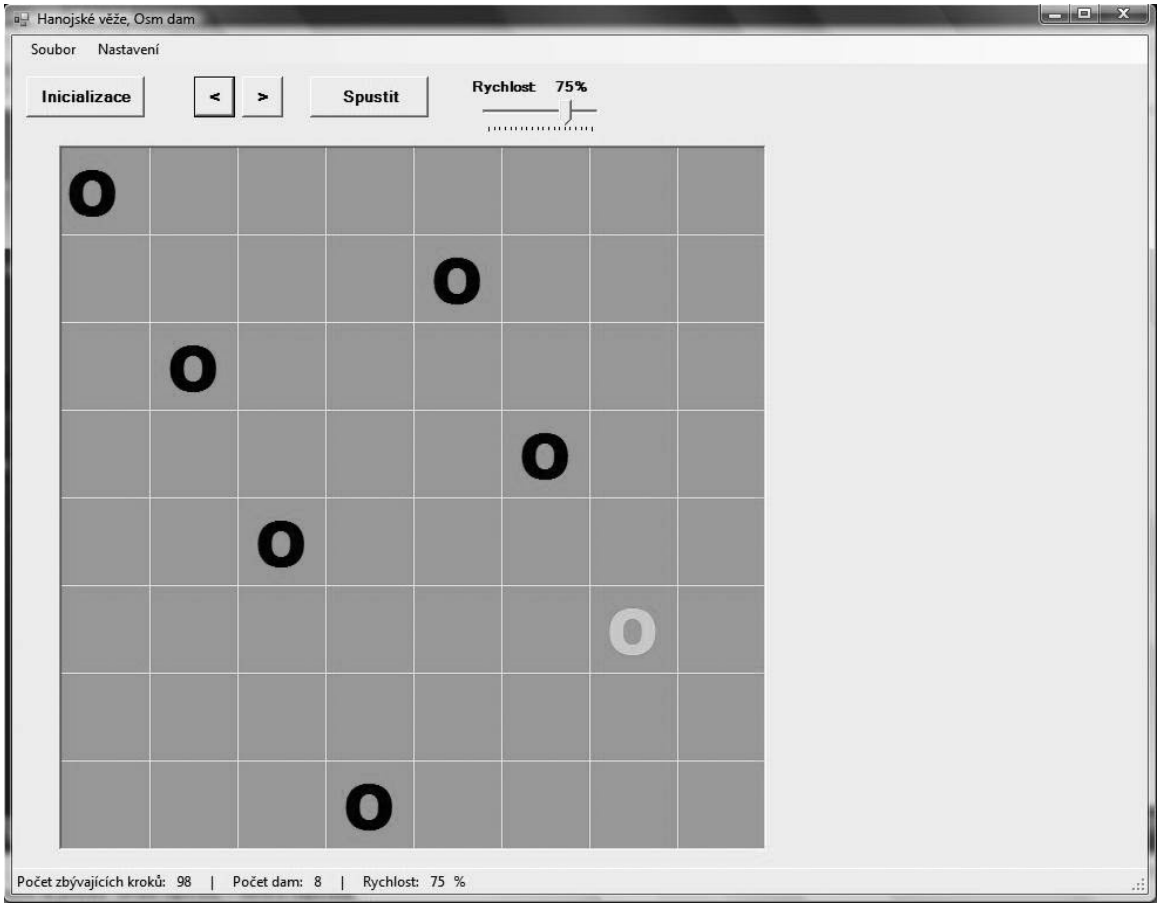
Pro nalezení požadovaného řešení je opět možné použít krokování. Krokovat lze pomocí klávesy > vpřed a klávesou < zpět stejně jako u řešení problému hanojských věží. Řešení lze taktéž nalézt pomocí časovače. Pokud je časovač spuštěn, není možné provést manuální krokování. Časovač je možné spustit stiskem tlačítka *Spustit* nebo klávesovou zkratkou *ALT+T* a zastavit, pokud je spuštěn, tlačítkem *Zastavit* nebo klávesovou zkratkou *ALT+T*. U časovače je opět možné měnit rychlost běhu posunutím hodnoty na trackbaru nazvaného *Rychlost*. Rozsah a krok je stejný jako u nastavení rychlosti u hanojských věží.

Při stisku tlačítka *Inicializace* nebo klávesové zkratky *ALT+I* se program nastaví zpět do výchozího stavu. To znamená na začátek hledání řešení, odstraní se položené kameny, zastaví časovač (pokud byl spuštěn) a položí se první výchozí kámen.

Celý program lze ukončit stiskem křížku v pravém horním rohu, stiskem *Konec* v menu (*Soubor->Konec*), nebo klávesovou zkratkou *CTRL+K*.



Obr. 5 Program v režimu hanojských věží



Obr. 6 Program v režimu osmi dam

4 Závěr

Cílem mé bakalářské práce bylo usnadnit studentům, kteří se nyní místo jazyka PASCAL učí jazyk C, pochopení některých algoritmů požadovaných v předmětu IAL (Algoritmy). Má práce se týkala kapitol studijní opory zabývající se vyhledáváním podřetězců v řetězci a rekurze. Mým úkolem bylo seznámit se detailně s textem kapitol 6 a 7 o vyhledávání podřetězců v řetězci a o rekurzivních algoritmech ve studijní opoře pro předmět IAL (Algoritmy) a s dalšími algoritmy v dostupné literatuře, modifikovat text kapitol zapsaných pro pascalovský typ jazyka, aby co nejvíce – i v detailech zachoval původní obsah, ale aby se vztahoval k zápisu příkladů a algoritmů v jazyce C. Dále převést všechny příklady a algoritmy do jazyka C a odladit je v prostředí vytvořeném pro tento účel, vytvořit program pro animovanou demonstraci vybraných operací s využitím nástrojů pro grafickou reprezentaci. Na závěr jsem navrhnul další vhodné kontrolní otázky a příklady vhodné pro formulářově orientovanou písemnou zkoušku ze znalostí tohoto okruhu.

Zadané úkoly jsem splnil podle zadání. Domnívám se, že studenti, kteří se učí jazyk C, budou mít usnadněnou práci při studování studijní opory předmětu IAL (Algoritmy). Převedená studijní opora i se zdrojovými kódy algoritmů je součástí přílohy. Pro znázornění a porovnání algoritmů, které vyhledávají v textu, jsem vytvořil demonstrační program, který demonstruje použití naivního, Knuth-Morris-Prattova, Boyer-Mooreova a Karp-Rabinova algoritmu. Pro znázornění rekurzivních problémů jsem vytvořil demonstrační program, zobrazující řešení úlohy hanojských věží a osmi dam. Všechny demonstrační programy i se zdrojovými kódy jsou také součástí přílohy mé bakalářské práce. Podařilo se mě vytvořit 10 kontrolních otázek a 5 příkladů pro formulářově orientovanou zkoušku. Otázky i příklady se také nachází v příloze.

Domnívám se, že se mě a ostatním kolegům, kteří mají na starosti jiné kapitoly ze studijní opory, nepodařilo dostatečně sjednotit programy pro animovanou demonstraci. Vhodnější a přehlednější pro studenty by bylo, kdyby všechny tyto demonstrační aplikace byly vytvořeny ve stejném vývojovém prostředí a měly stejné grafické uživatelské rozhraní. I přes to si myslím, že převedení studijní opory do jazyka C a vytvoření programů pro animovanou demonstraci bude pro budoucí studenty předmětu Algoritmy přínosem.

Literatura

[1] Honzík J.M. Algoritmy. Studijní opora pro předmět Algoritmy. Elektronický text. FIT VUT v Brně.

[2] Piotr Wróblewski. Algoritmy – Datové struktury a programovací techniky. Brno, Computer Press 2004.

[3] Dr. Dobb's Essential Books on Algorithms and Data Structures. CD-ROM - Release 2 1999.

Seznam příloh

Přílohy

1.Příloha –Kontrolní otázky a příklady pro formulářově orientovanou zkoušku

Kolik předzpracování vzorku provádí algoritmy naivní, *Knuth-Morris-Prattův, Karp-Rabinův* a *Boyer-Mooreův*?

Který z algoritmů *Knuth-Morris-Pratt, Karp-Rabin* a *Boyer-Moore* je považován za nejrychlejší ?

Kolik porovnání provedou algoritmy *Knuth-Morris-Pratt, Karp-Rabin a Boyer-Moore* pokud se hledaný vzorek bude shodovat s prohledávaným textem ?

Který algoritmus využívá ke své práci konečný automat ?

Který algoritmus porovnává znaky ve vzorku zprava doleva (protisměrné vyhledávání)?

K čemu slouží hashovací funkce/tabulka ?

Co jsou to algoritmy s návratem?

O jaké funkci můžeme říci, že je efektivně vyčíslitelná?

Kolik nejvíce může být na sobě při řešení úlohy *hanojská věž* položeno kruhů?

Kolik nejvíce lze při řešení úlohy *osmi dam* položit na šachovnici kamenů? Proč?

2.Příloha – Otázky a příklady pro písemnou zkoušku z předmětu IAL

Jakou hodnotu vrátí zavolaná funkce *rek(5)* ?

```
int rek (int i)
{
    if (i<=0)
        return 0;
    else
        return i*rek(i-1);
}
```

a/ 5 b/ 120 **c/ 0** d/ nic(zacyklí se)

Pro algoritmy s návratem je typické že :

a/ algoritmus končí úspěšně při vyčerpání všech možnosti pokud nedojde ke splnění podmínek

b/ řešení se hledá metodou pokusu a omylu

c/ řešení se hledá předpisem pro jeho dosažení

d/ využívají akumulační parametr (shromažďuje informace postupně získané)

Máme úlohu hanojských věží, 3 tyče(kolíky) a na první tyči 2 kotouče. Cílem je přesunout všechny kotouče na poslední tyč. Kolik je minimálně potřeba tahů na vyřešení této úlohy?

a/ 2 **b/ 3** c/ 4 d/ úloha nemá řešení

Který z uvedených vyhledávacích algoritmů nejvícekrát předzpracovává hledaný vzorek?

a/ základní(naivní) b/ Knuth-Morris-Prattův **c/ Boyer-Mooreův** d/ všechny stejně

Který z uvedených vyhledávacích algoritmů je považován za nejefektivnější a nejrychlejší?

a/ základní(naivní) b/ Knuth-Morris-Prattův **c/ Boyer-Mooreův** d/ všechny stejně

3.Příloha – Zdrojový text vyhledávacího algoritmu Karp-Rabin

```
inline unsigned int Hash(const unsigned int a, const unsigned int b,const unsigned int h, const unsigned int d)
{///hashovaci funkce-deleni dvema bylo kvuli rychlosti nahrazeno bitovymi posuny
	unsigned int iRet ;
	iRet = ((h - (a << d)) << 1) + b;
	return iRet ;
}

int KarpRabin(String^ x, const int m, String^ y, const int n)
{///hlavni funkce Karp-Rabinova algoritmu
	unsigned int hx, hy;
	int i , j;
	hx = hy = 0;
	for ( i = 0; i < m; i++) //vytvorim hash vzorku
	{
		hx = (hx << 1) + x[i];
		hy = (hy << 1) + y[i];
	}
	i = 0;
	while ( i <= n-m)
	{

		if (hx == hy)
		{/// potencialni nalez vzorku (rovna se hashovaci fce)
			for ( j = 0; j < m; j++)
			{
				if (x[ j ] != y[ i+j]) //nenasel (vyskocim pryc)
					break;
			}
			if ( j == m)	//nasel
			{
				return(i); //vracim pozici
			}
		}
		hy = Hash(y[i ], y[ m+i], hy, m-1); //zjiskam hash textu
		i++;
	}
	return(i); // nenasel
} // konec funkce Karp-Rabin
```

4.Příloha – Studijní opora předmětu Algoritmy