VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

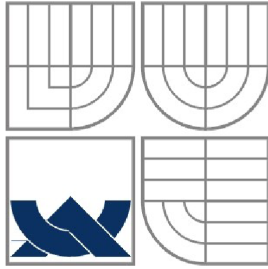
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

DEMONSTRAČNÍ PROGRAM VYHLEDÁVÁNÍ ŘETĚZCŮ V TEXTU

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

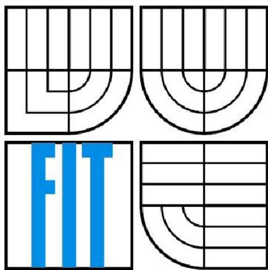
AUTOR PRÁCE PETR ŠATKA
AUTHOR

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

DEMONSTRAČNÍ PROGRAM VYHLEDÁVÁNÍ ŘETĚZCŮ V TEXTU

DEMONSTRATION PROGRAM OF SEARCHING OF STRING IN TEXT

BAKALÁŘSKÁ PRÁCE

BACHELOR´S THESIS

AUTOR PRÁCE PETR ŠATKA

AUTHOR

VEDOUCÍ PRÁCE Ing. ROMAN LUKÁŠ, Ph.D.

SUPERVISOR

BRNO 2008

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů — Akademický rok 2007/2008

Zadání bakalářské práce

Řešitel: **Šatka Petr**
Obor: Informační technologie
Téma: **Demonstrační program vyhledávání řetězců v textu**
Kategorie: Alg. a datové struktury

Pokyny:

1. Seznamte se podrobně s metodami pro vyhledávání řetězců v textu probíranými v rámci kurzu Algoritmy.
2. Nastudujte potřebné kapitoly z počítačové grafiky.
3. Seznamte se s vhodnou multiplatformní knihovnou pro tvorbu uživatelských rozhraní.
4. Proveďte návrh implementace demonstračního programu, který bude didakticky ilustrovat vyhledávání řetězců v textu.
5. Daný program naimplementujte.
6. Diskutujte přednosti, nedostatky a možný další vývoj vašeho řešení projektu.

Literatura:

- Honzík, J. M., Hruška, T., Máčel, M.: Vybrané kapitoly z programovacích technik, Vysoké učení technické v Brně, Brno, 1991.
- Žára, J., Beneš, B., Sochor, J., Felkel, P.: Moderní počítačová grafika, Computer Press, 2005.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 - 4

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

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

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

Vedoucí: **Lukáš Roman, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2007
Datum odevzdání: 14. května 2008

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: **Petr Šatka**
Id studenta: 78735
Bytem: Družba 1214, 763 31 Brumov-Bylnice
Narozen: 16. 08. 1986, Zlín
(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: Demonstrační program vyhledávání řetězců v textu
Vedoucí/školitel VŠKP: Lukáš Roman, Ing., Ph.D.
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 pořizování 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

V mé práci se zabývám problematikou vyhledávacích algoritmů. Cílem bylo vytvořit program pro demonstraci principu algoritmů pro vyhledávání v textu uvedených v opoře pro předmět Algoritmy. Vytvořený program tyto algoritmy názorně demonstruje pomocí animací.

Klíčová slova

vyhledávání, výukový program, naivní algoritmus, Knuth-Morris-Prattovův algoritmus, Bayer-Mooreovův algoritmus, Baeza-Yates-Gonnetův algoritmus, algoritmus Quicksearch, Karp-Rabinův algoritmus

Abstract

My work deals with the problems of strings searching algorithms. The objective of this work is to create a program for demonstration of the strings searching algorithms described in the mimeographed for course Algorithms. The algorithms are demonstrated by the animations.

Keywords

searching, demonstration program, naive algorithm, Knuth-Morris-Pratts algorithm, Bayer-Moores algorithm, Baeza-Yates-Gonnets algorithm, Quicksearch algorithm, Karp-Rabins algorithm

Citace

Šatka Petr: Demonstrační program vyhledávání řetězců v textu, bakalářská práce, Brno, FIT VUT v Brně, 2008

Prohlášení

Prohlašuji,

že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Romana Lukáše Ph.D.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

....

Petr Šatka

8.5.2008

Poděkování

Rád bych poděkoval panu Ing. Romanovi Lukášovi Ph.D. za vedení mé bakalářské práce.

© Petr Šatka, 2008.

Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

Úvod 10
1 Vyhledávání v textu 11
1.1 Algoritmy 11
1.1.1 Naivní algoritmus 11
1.1.2 Knuth-Morris-Prattův algoritmus 13
1.1.3 Boyer-Mooreův algoritmus 16
1.1.4 Quick search 19
1.1.5 Baeza-Yates-Gonnetův algoritmus 20
1.1.6 Karp-Rabinův algoritmus 21
1.1.7.1 Hashovací funkce a hash 21
1.1.7.2 Hashovací tabulka 21
2 Postup práce 23
2.1 Analýza 23
2.1.1 Vývojové prostředky a platforma 24
2.1.2 Návrh grafického rozhraní 25
2.1.3 Objektový návrh 26
2.2 Implementace 29
2.3 Testování a kompatibilita 31
3 Práce s programem 33
4 Závěr 36
Zdroje 37
Seznam příloh 38
Přílohy 39

Úvod

Cílem mé bakalářské práce bylo vytvořit program, který bude názorně didakticky demonstrovat činnost algoritmů pro vyhledávání v textu. Při tvorbě jsem se zaměřil především na jednoduchost ovládání a praktičnost použití. Vzhledem k tomu, že nebylo specifikováno, kdo bude program využívat, pokusil jsem se jej vytvořit tak, aby byl použitelný jak vyučujícími při přednáškách a na cvičeních, tak i samotnými studenty. Princip znázornění činnosti jednotlivých algoritmů vychází z ilustrací v opoře k předmětu algoritmy. To by mělo zaručit, že bude rychle pochopitelný pro většinu studentů. Má práce by měla sloužit pro jednodušší a názornější pochopení problematiky vyhledávání vzorků v textu. Práce je rozdělena na část teoretickou, kde se zabývám principy vyhledávání v textu a podrobně popisuji jednotlivé vyhledávací algoritmy. V této části se zaměřuji především na algoritmy, které můj program demonstruje, okrajově také zmíním některé další známé, nebo zajímavé algoritmy. V další části se věnuji samotné tvorbě výukového programu. Zmiňuji zde požadavky kladené na vytvářený program, pak následuje analýza úkolu sestávající z volby cílové platformy, výběru vývojového prostředí a objektového návrhu. Následuje část popisující samotnou implementaci. V další části popisuji způsob a výsledky testování. Poukazuji zde na možné problémy, které mohou nastat při používání programu, a zabývám se zde kompatibilitou s různými operačními systémy. Poslední část je věnována popisu práce s programem. V této části se zabývám ovládáním programu a zmíním zde možnost tvorby jazykových překladů uživatelského rozhraní.

1 Vyhledávání v textu

S vyhledáváním v textu se každý setkává dnes a denně. Protože vyhledávání v textu je dnes v době rozmachu informatiky, kdy se většina dokumentů vede v elektronické podobě, jedna z nejčastějších činností počítačů. Pod pojmem vyhledávání v textu si většinou představíme funkci textového editoru, pomocí které vyhledáme, případně nahradíme vzorek v dokumentu. Problematika vyhledávání v textu se však nezužuje pouze na vyhledání vzorku v textovém dokumentu, ale je mnohem rozsáhlejší. Stejné principy jsou také používány v databázích a v operačních systémech například pro nalezení souboru podle jeho názvu nebo obsahu. Algoritmy vyhledávající vzorky v řetězci se také používají v antivirových programech, kde se vyhledává škodlivý kód viru v napadených souborech. S touto problematikou se můžeme setkat dokonce i v jiných oborech než v informatice. Příkladem může být biologie, kde se metody vyhledávání v textu uplatňují při analýze DNA. Vyhledávání v textu je dnes jistě velmi užitečné a vzhledem k tomu, že je mnohdy potřeba prohledávat velké množství dat v co nejkratším čase, je důležité, aby použité algoritmy byly efektivní a rychlé.

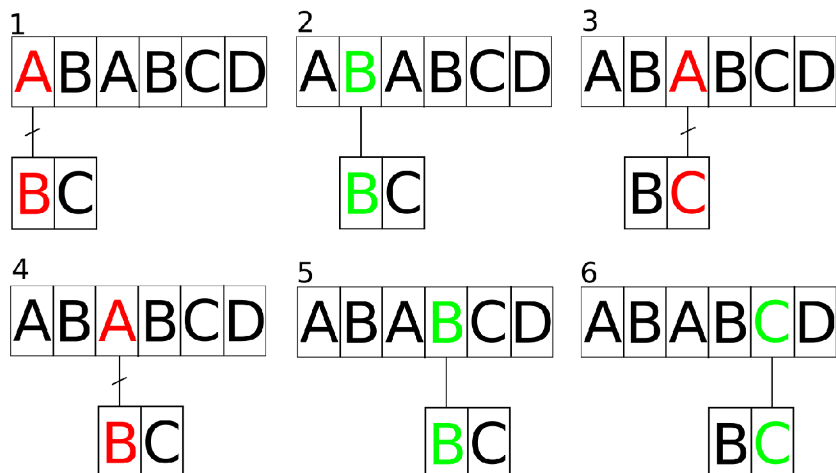
1.1 Algoritmy

Ve své práci se zabývám především třemi algoritmy uvedenými v opoře pro předmět Algoritmy: naivním algoritmem, Knuth-Morris-Prattovým (KMP) algoritmem a Bayer-Mooreovým algoritmem (BMA). Tyto algoritmy slouží k vyhledání pouze jednoho vzorku. Každý z těchto algoritmů využívá zcela odlišného přístupu a jsou také různě efektivní. KMP a BMA jsou si podobné v tom, že potřebují předzpracovat vyhledávaný vzorek. Vedle těchto algoritmů existují další a mnohdy zajímavé algoritmy (Baeza-Yates-Gonnet, Quicksearch, Karp-Rabin). Vzhledem k tomu, že demonstrační program, který je předmětem mé bakalářské práce, má za úkol vysvětlit základní a nejvýznamnější algoritmy, budu se těmito ostatními algoritmy zabývat jenom okrajově. Nicméně pokusil jsem se program navrhnout tak, aby jej bylo možno někdy v budoucnu, pokud to bude potřeba, jednoduše doplnit o další algoritmy.

1.1.1 Naivní algoritmus (někdy také klasický)

Jedná se o algoritmus, který napadne po chvíli asi každého programátora, pokud dostane za úkol navrhnout algoritmus, který vyhledá vzorek v textu. Je to jednoduchý, ale značně neefektivní algoritmus. Jeho princip spočívá v tom, že postupně znak po znaku prochází prohledávaný text a kontroluje, jestli se znak na aktuální pozici shoduje s prvním znakem ve vzorku. Pokud dojde ke shodě, zvětší se porovnávaná pozice vzorku. Pokud dojde k neshodě před dosažením konce vzorku, porovnávaná pozice vzorku se nastaví na první znak. Algoritmus končí úspěchem, pokud se dostane na konec vyhledávaného vzorku a zároveň porovnání na této pozici je úspěšné. Pokud dojde na konec

prohledávaného textu před dosažením konce vzorku, končí hledání neúspěchem.



Obr. Průběh naivního algoritmu

Jak je vidět na obrázku, vzorek se jakoby pod prohledávaným textem posouvá a porovnávají se znaky, které se nachází pod sebou. Hledáme tedy vzorek BC v textu ABABCD. V prvním kroku dochází k neshodě, proto se vzorek posune. Ve druhém kroku se znaky shodují, vzorek zůstává tedy na místě a srovnají se znaky na další pozici (krok 3), zde dochází k neshodě, vzorek se tedy posouvá a začíná se opět srovnávat od prvního znaku vzorku, jak je patrné v kroku 4. V pátém a šestém kroku se znaky shodují. Algoritmus tedy končí úspěchem.

Klasický algoritmus by v jazyce C++ mohl vypadat následovně:

```
(1) int ClassicSearch(string srcText, string searchedText)
(2) {
(3)    int n=srcText.length();
(4)    int m=searchedText.length();
(5)    int srcIndex=0;
(6)    int searchedIndex=0;
(7)    int pomZac=0;
(8)    while ((srcIndex<n) && (searchedIndex<m))
(9)    {
(10)       if (srcText[srcIndex]==searchedText[searchedIndex])
(11)       {
(12)          srcIndex++;
(13)          searchedIndex++;
(14)       }
```

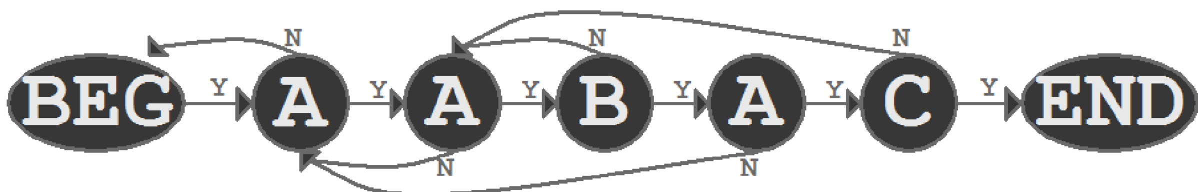
```
(15)     else
(16)     {
(17)       pomZac++;
(18)       srcIndex=pomZac;
(19)       searchedIndex=0;
(20)     }
(21)   }
(22)   if (searchedIndex>m)
(23)     return(PomZacT); //Nalezeno
(24)   else
(25)     return(srcIndex); //Nenalezeno, vrací se n+1
(26) }
```

Analýza algoritmu

V nejlepším případě, pokud by byl vyhledávaný vzorek na začátku řetězce, by se provedlo *m* porovnání. Není-li ve vyhledávaném textu obsažen první znak vzorku, provede se *n* porovnání. V nejhorším případě dojde k *m-1* shodám na každé startovací pozici, v tomto případě se provede *mn* porovnání a algoritmus má tedy složitost *O(mn)*. V přirozených jazycích se však takovéto extrémní případy vyskytují jen zřídka, a proto zde algoritmus pracuje průměrně dobře. Problémem tohoto algoritmu by v některých aplikacích mohl být návrat v prohledávaném řetězci (řádek 18).

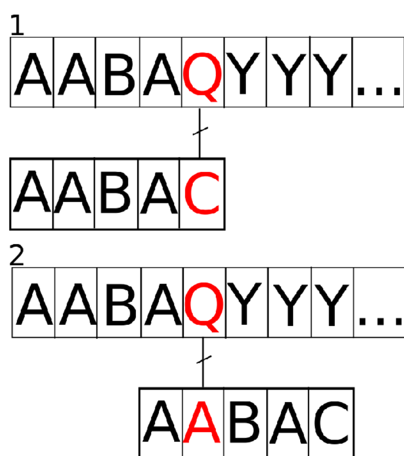
1.1.2 Knuth-Morris-Prattův algoritmus

Knuth-Morris-Prattův algoritmus řeší nevýhodu návratu v prohledávaném řetězci u naivního algoritmu a také zefektivňuje posun vyhledávaného vzorku. Využívá informací, které poskytuje vyhledávaný vzorek k posunu vzorku o víc než jeden znak. Toto činí Knuth-Morris-Prattův algoritmus výrazně rychlejší než naivní algoritmus. Díky tomu, že není nutné se v prohledávaném textu vracet, sníží se tedy počet přístupů na disk, které by systém musel vykonávat, kdyby se algoritmus vracel na pozici, která už není v paměťovém bufferu. Vzhledem k tomu, že potřebujeme o vyhledávaném vzorku zjistit dostatečné množství informací ještě před samotným vyhledáváním, je nutné vzorek předzpracovat. Výsledkem tohoto předzpracování je konečný automat.



Obr. Automat KMP

Na obrázku je KMP automat pro vzorek AABAC. Při vyhledávání se pak v případě shody postupuje po hraně Y, naopak v případě neshody po hraně N. Je zde vidět smysl algoritmu KMP, nemusí se totiž v případě neshody vzorek prohledávat vždy od prvního znaku. Například dojde-li k neshodě na pozici znaku C, tak se vzorek zarovná tak, že znak v prohledávaném textu na pozici, kde došlo k neshodě se bude nacházet nad druhým znakem A ve vzorku, ke kterému směřuje hrana N ze stavu C. Je totiž zřejmé, že pokud došlo k neshodě až na znaku C, musela mu předcházet shoda na znaku A, při novém porovnávání vzorku můžeme tedy první znak A vynechat. Posun vzorku ukazuje následující obrázek.



Obr. Posun vzorku

Funkce pro sestavení automatu a vyhledávací algoritmus budou vypadat následovně. Automat je zde reprezentován vektorem *vect*.

```
(27) vector SetVect(string searchedText)
(28) {
(29)    int k,r;
(30)    vector<int> vect (searchedText.Length(),0);
(31)    vect.at(0)=-1;
(32)    for (k=1; k<searchedText.Length(); k++)
(33)    {
(34)      r=vect.at(k-1);
(35)      while ((r>=0) && (searchedText.GetChar(r)!=searchedText.GetChar(k-1)))
(36)      {
(37)        r=vect.at(r);
(38)      }
(39)      vect.at(k)=r+1;
(40)    }
```

```
(41)      return(vect);
(42) }

(43) int KMPSearch(string srcText, string searchedText)
(44) {
(45)    int m=searchedText.length();
(46)    int n=srcText.length();
(47)    int searchedIndex=0;
(48)    int srcIndex=0;
(49)    while ((srcIndex<n) && (searchedIndex<m))
(50)    {
(51)      if ((searchedIndex==-1) &&
(52)      (srcText.GetChar(srcIndex)==searchedText.GetChar(searchedIndex)))
(53)      {
(54)        searchedIndex++;
(55)        srcIndex++;
(56)      }
(57)      else
(58)      {
(59)        searchedIndex=vect.at(searchedIndex);
(60)      }
(61)    }
(62)    if (searchedIndex>m)
(63)      return (srcIndex-m); //Nalezeno
(64)    else
(65)      return (searchedIndex); //Nenalezeno, vrací se n+1
(66) }
```

Při vyhledávání pak algoritmus inkrementuje pozici v prohledávaném textu a v případě neshody posunuje vzorek o počet pozic uložených ve vektoru *vect* na indexu pozice vzorku (řádek 59). V případě, že vektor *vect* na daném indexu obsahuje *-1* (řádek 51), tak se vzorek zarovná prvním znakem na pozici v prohledávaném textu (řádek 54).

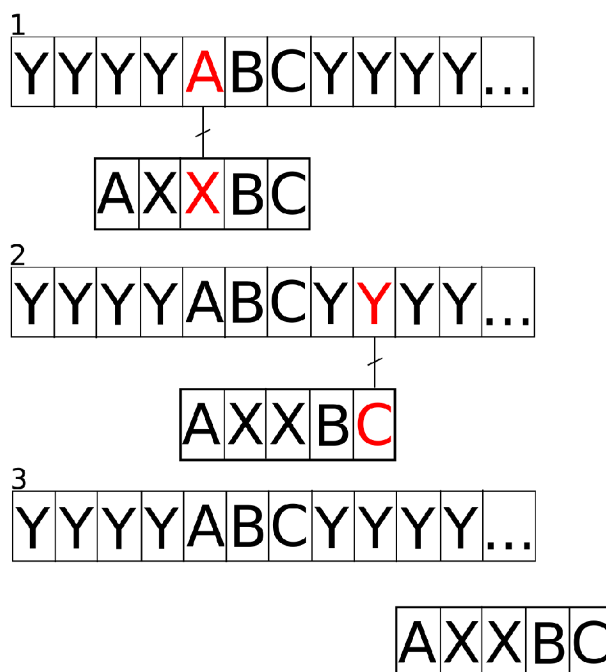
Analýza algoritmu

Celková složitost algoritmu se skládá ze složitosti funkce *SetVect* a složitosti samotného algoritmu *KMPsearch*. Vzhledem k tomu, že při každém průchodu vnějšího cyklu (*m-1* krát) ve funkci *SetVect* se proměnná *r* zvětší o jedna (řádek 34 a 39) a ve vnitřním cyklu se hodnota *r* zmenšuje, protože

vect.at(r)<r, nemůže se tedy *r* snižovat více než *m-2* krát. Z toho plyne lineární časová složitost *O(m)*. Samotný algoritmus provede maximálně *2n* srovnání, takže jeho časová složitost je *O(n)*. Celková složitost je tedy *O(m + n)*. Jak je vidět, je tento algoritmus výrazně rychlejší než naivní algoritmus se složitostí *O(mn)*.

1.1.3 Boyer-Mooreův algoritmus

Jedná se o asi nejpoužívanější algoritmus pro vyhledávání v textu. Od předchozích dvou algoritmů se zásadně liší v tom, že neprochází všechny znaky prohledávaného textu. Dokáže totiž znaky, které se nemohou rovnat, přeskočit. Tuto vlastnost mu umožňuje to, že vzorek při porovnávání s prohledávaným textem prochází zprava doleva a při neshodě neposunuje vzorek o jednu pozici vpravo, ale k výpočtu vzdálenosti, o kterou lze vzorek posunout, používá dvou heuristik a z nich pak vybírá výhodnější výsledek. Opět, stejně jako u Knuth-Morris-Prattova algoritmu, je nutné vzorek předem zpracovat, tentokrát dokonce dvakrát. Avšak díky použitým heuristikám je Boyer-Mooreův algoritmus v praxi velmi efektivní. Algoritmus končí úspěchem, pokud dojde ke shodě na prvním znaku vzorku. První heuristika se nazývá heuristika špatného znaku (bad-character shift nebo occurrence shift). Tato heuristika určuje posun vzorku podle „špatného“ znaku v prohledávaném textu, který způsobil neshodu. Pokud tedy dojde k neshodě a znak, který neshodu způsobil se ve vzorku vyskytuje, zarovná se vzorek podle nejpravějšího výskytu tohoto znaku. Naopak, pokud se znak ve vzorku nevyskytuje, zarovná se vzorek těsně za pozici, kde došlo k neshodě.

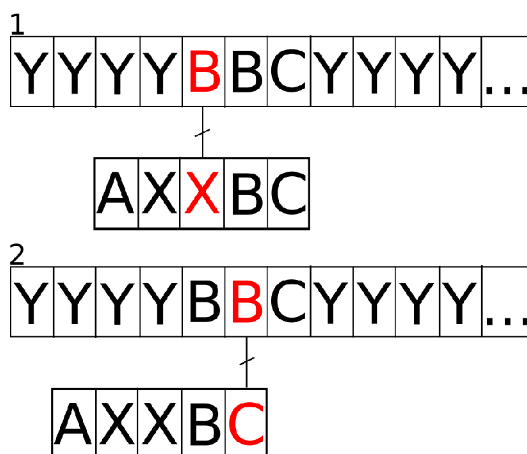


Obr. Heuristika špatného znaku

V prvním kroku na obrázku došlo k neshodě znaku A se znakem X. Připomínám, že BMA prochází

vzorek zprava doleva, tedy znaky C a B už jsou zkontrolovány. Neshodu v tomto případě způsobil znak A. Znak A se ve vzorku vyskytuje, vzorek se tedy zarovná svým nejpravějším A (v tomto případě jediným) pod znak A v prohledávaném textu. Při porovnání Y a C dojde opět k neshodě, ale Y se ve vzorku nevyskytuje. V tomto případě se vzorek posune těsně za tento znak Y.

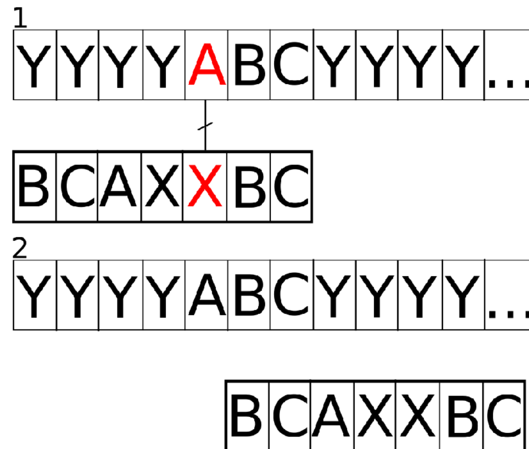
Heuristika špatného znaku algoritmus sice výrazně urychlí, ale někdy nemusí poskytovat žádnou možnost. Tato situace nastane, pokud je nejpravější výskyt „špatného“ znaku ve vzorku napravo od pozice, kde došlo k neshodě. Vzorek by se pak posunul doleva.



Obr. Nedokonalost heuristiky špatného znaku

Pokud pozměním prohledávaný text tak, aby neshodu nezpůsobil znak A tak jako v předchozím případě ale znak B, dojde k tomu, že se vzorek posune špatným směrem. Stane se to právě proto, že B se ve vzorku vyskytuje vpravo od místa, kde došlo k neshodě.

Nedokonalosti předchozí heuristiky řeší druhá heuristika, která se nazývá heuristika dobré přípony (good-suffix shift). Pokud se pravá část vzorku („dobrá přípona“) shoduje s vyhledávaným textem a až později dochází k nesouhlasu a pokud se shodná část vyskytuje ve vzorku dvakrát, dojde k posunu vzorku tak, aby další výskyt této přípony ve vzorku stál proti stejné kombinaci znaků v prohledávaném textu. Další výskyt shodné přípony se tedy bude nacházet na stejné pozici, kde se předtím nacházela shodná přípona.



Obr. Heuristika dobré přípony

Obrázek ukazuje situaci, kdy se shodná přípona BC vyskytuje ve vzorku dvakrát. Neshodu sice způsobil znak A a vzorek by mohl být zarovnán podle výsledku heuristiky špatného znaku, ale heuristika dobré přípony poskytuje v tomto případě lepší výsledek, proto se vzorek zarovná podle výsledku heuristiky dobré přípony. Tedy tak, aby znaky BC v prohledávaném textu a další výskyt znaků BC ve vzorku ležely pod sebou.

Analýza algoritmu

Složitost Boyer-Mooreova algoritmu se tedy bude skládat ze složitosti obou heuristik a složitosti samotného algoritmu.

Heuristika špatného znaku

```
(67)int m=searchedText.length();
(68) vector<int> jumpVector(256, m);
(69) for (int i=0; i<m; i++)
(70) {
(71)    jumpVector.at(Chtoi(searchedText.at(i)))=m-i-1;
(72)}
```

pzn.: Funkce Chtoi(char) převede znak char na integer. Vrací hodnoty v rozsahu 0-255.

Její složitost bude záležet na velikosti abecedy $|\Sigma|$ a délce vzorku. Bude tedy *O(* $|\Sigma|$ *+ m)*. Na první pohled nemusí být zřejmé, proč je složitost závislá na $|\Sigma|$. Je nutné si uvědomit, co se děje na řádku 68. Vytvoří se zde vektor o délce *256 = počet ASCII znaků =* $|\Sigma|$ a každá položka vektoru se inicializuje.

Heuristika dobré přípony

Tato heuristika má složitost $O(m)$. Algoritmus je uveden v příloze 1.

Celková složitost

Složitost obou heuristik je $O(|\Sigma| + m)$ a složitost samotného algoritmu je stejná jako u naivní formy ($O(mn)$). Celková složitost je tedy $O(mn + |\Sigma| + m) = O(|\Sigma| + m * (1 + n))$. Jedná se však o složitost v nejhorším případě. V praxi dosahuje tento algoritmus velmi dobrých výsledků.

1.1.4 Quick search

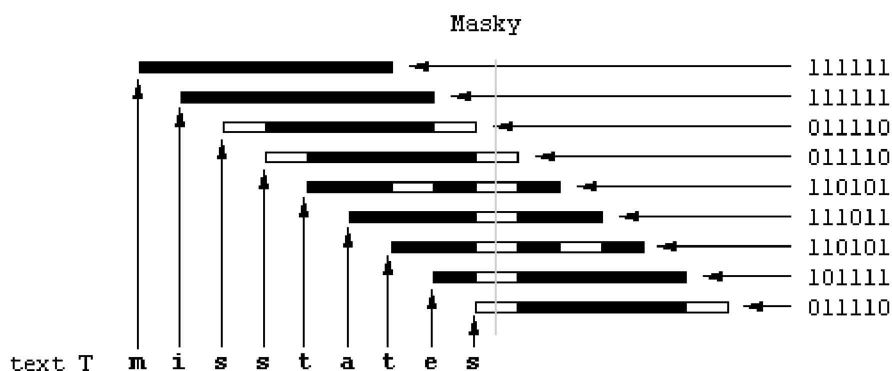
Tento algoritmus je podobný Boyer-Mooreovu algoritmu, ale na rozdíl od něho postupuje při porovnávání vzorku zleva doprava a posun vzorku je řešen jednodušším způsobem. Pokud dojde při porovnávání k neshodě, zkontroluje se znak následující za vzorkem a pokud se tento znak ve vzorku nevyskytuje, posune se vzorek až za tento znak. Pokud by se znak ve vzorku vyskytoval, posune se vzorek o nejmenší možnou vzdálenost tak, aby byly shodné znaky pod sebou. Poté se skočí na první znak vzorku a pokračuje se dále.

Obr. Princip algoritmu Quicksearch

Na obrázku dochází k neshodě hned na začátku, ověří se tedy znak následující bezprostředně za vzorkem, zda se nenachází ve vzorku. Znak C se ve vzorku vyskytuje, vzorek se tedy zarovná tak, že znak C ve vzorku a znak C v prohledávaném textu se nacházejí pod sebou. Dochází opět k neshodě, tentokrát se však znak Q, který následuje za vzorkem ve vzorku nevyskytuje, vzorek se tedy posune těsně za tento znak.

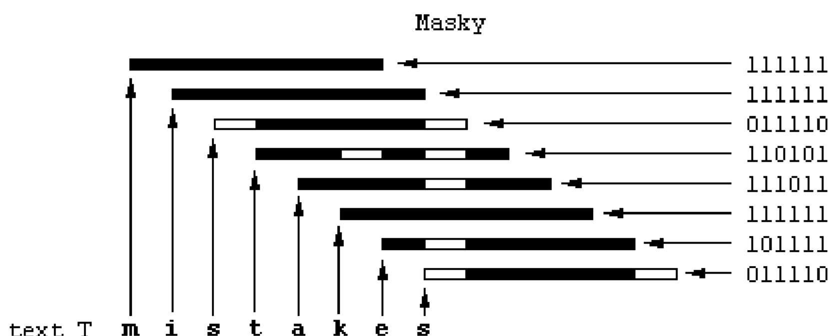
1.1.5 Baeza-Yates-Gonnetův algoritmus

Uplatňuje úplně jiný přístup k vyhledávání v textu než předchozí čtyři algoritmy. Při vyhledávání používá tabulku bitových vektorů. Pro každý znak vstupní abecedy se předem vytvoří bitový vektor, ve kterém každá bitová pozice odpovídá pozici daného znaku ve vyhledávaném vzorku. Vektor může tedy být posloupnost jedniček stejně dlouhá jako vzorek, ve které se pak nulami vyznačí pozice daného znaku ve vzorku. Při vzorku „abcda“ by vektor „a“ vypadal takto: 01110. Nejběžněji se čísluje zprava, tudíž vektor pro „b“ by byl 11101. Při vyhledávání se pak masky zarovnají k jednotlivým znakům prohledávaného textu a pokud dojde k situaci, že se nad sebou nachází počet nul odpovídající délce vzorku, znamená to úspěšné nalezení. Algoritmus si tedy vytvoří prázdnou masku skládající se jenom z jedniček. V každém kroku pak tuto masku posune o jeden bit doleva a pomocí operace *or* k ní přidá masku právě kontrolovaného znaku. Pak zkontroluje, zda se v masce na pozici (počítané zprava) odpovídající délce vzorku vyskytuje nula. Pokud se tedy nula na této pozici nachází, znamená to úspěšné nalezení hledaného vzorku.



Obr. Baeza-Yates-Gonnetova algoritmu – úspěšně nalezeno

(http://htmltolatex.sourceforge.net/samples/sample3.html)



Obr. Baeza-Yates-Gonnetova algoritmu – nenalezeno

(http://htmltolatex.sourceforge.net/samples/sample3.html)

Na prvním obrázku je vidět vyhledání vzorku states. V maskách umístěných nad sebou se nachází

nepřerušený sloupce nul. Jsou zde také vidět masky pro jednotlivé znaky, například znak m se ve vzorku nenachází, proto se jeho maska sestává ze samých jedniček. Naproti tomu znak s se nachází ve vzorku hned dvakrát, a to na prvním a posledním místě, jeho maska tedy obsahuje na těchto pozicích nulu.
Druhý obrázek ukazuje situaci, kdy se vzorek na dané pozici nenachází. Maska znaku k neobsahuje na správném místě nulu. Přesněji řečeno, neobsahuje nulu vůbec a nevznikne tedy nepřerušený sloupce nul nad sebou.

U tohoto algoritmu se uvádí [1] časová složitost $O(m+|\Sigma|)$ části pro předzpracování vzorku a $O(n)$ pro vyhledávací část. Vzhledem k použití bitových operací pracuje velmi rychle, ale nevýhodou může být omezení délky masky, dnes většinou na 32 nebo 64 bitů.

1.1.6 Karp-Rabinův algoritmus

Vzhledem k tomu, že Karp-Rabinův algoritmus při své práci využívá hashovací funkce, vysvětlím nejprve pojmy hashovací funkce a hashovací tabulky.

1.1.6.1 Hashovací funkce a hash

Hashovací funkce je dle [2] reprodukovatelná metoda pro převod vstupních dat do (relativně) malého čísla, které vytváří jejich otisk (můžeme ho označit jako charakteristika dat). Výsledný otisk se označuje také jako výtah, miniatura, fingerprint či hash.

U hashovací funkce je důležité, aby pro libovolné množství dat poskytovala stejně dlouhý hash, dále aby při malé změně vstupních dat došlo k velké změně na výstupu. A musí poskytovat vysokou pravděpodobnost, že data se stejným hashem jsou stejná. Už z principu hashovací funkce plyne, že může a určitě bude docházet ke kolizím. Tomu se nelze vyhnout (kromě speciálních případů např. předem známé množiny vstupních dat), lze však snižovat pravděpodobnost, že nastane kolize pro podobná vstupní data.

1.1.6.2 Hashovací tabulka dle [3]

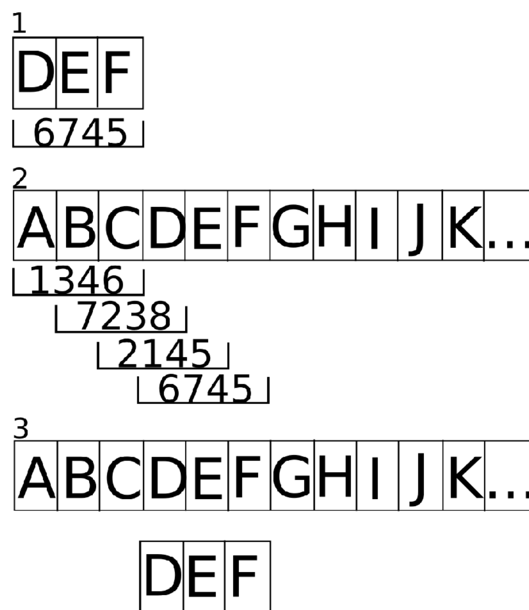
Hashovací tabulka je datová struktura, která asociuje hashovací klíče s odpovídajícími hodnotami. Hodnota klíče je spočtena z obsahu položky podle takového pravidla hashovací funkce, aby klíč byl co nejjednoznačněji určen, tj. aby pravděpodobnost přiřazení stejného klíče dvěma a více rozdílným položkám byla co nejnižší a aby rozptyl hodnot klíčů pro dvě obsahově blízké položky byl co nejvyšší.

Pomocí hashovací funkce přiřazujeme hodnotě klíče index (ukazatel) do homogenní datové struktury. Při zápisu obsahu položky zapíšeme položku na odpovídající místo, pokud je obsazeno pomocí vhodného algoritmu, přiřadíme pro položku další volný index. Při vyhledávání položky,

spočteme s pomocí klíče index hledané položky. Pokud již bylo odpovídající místo přepsáno položkou s jiným klíčem, opět podle vhodného algoritmu prohledáváme další položky.

Princip Karp-Rabinova algoritmu

Tento algoritmus využívá toho, že stačí porovnávat pouze části textu, které mají stejný výsledek hashovací funkce jako vzorek. Nejprve se tedy spočítá hash vzorku. Pak se začne procházet prohledávaný text a postupně se počítá hash částí prohledávaného textu. Částí textu tedy můžeme ukládat do hashovací tabulky, což usnadní příští vyhledávání. Pokud je výsledek hashovací funkce shodný s hashem vzorku, pak se znak po znaku porovná vzorek s podezřelou částí textu. Je to proto, že hashovací funkce není ideální a může docházet ke kolizím.



Obr. Karp-Rabinův algoritmus

V prvním kroku se vypočítá hash vzorku, poté se prochází prohledávaný text, dokud se nenajde část se stejným hashem. V posledním kroku se vzorek zarovná pod část se stejným výsledkem hashovací funkce a otestuje se shodnost.

2 Postup práce

Před zahájením práce bylo nejprve nutné pečlivě prostudovat zadání a dostatečně se zorientovat v problematice vyhledávání v textu. Ujasnil jsem si také, které algoritmy zahrnu do výukového programu. Analyzoval jsem tedy vybrané algoritmy, abych zjistil, jak pracují. Po seznámení s jednotlivými algoritmy jsem začal zvažovat, jakým způsobem budu demonstrovat jejich činnost. Rozhodl jsem se pro demonstraci pomocí animací podobných nákresům v opoře pro předmět Algoritmy. Poté jsem začal vybírat vývojové prostředí a programovací jazyk, ve kterém vytvořím program k přehrávání jednotlivých animací. Nakonec jsem se rozhodl, že budu program vytvářet v jazyce C++ s použitím knihoven WxWidgets pro tvorbu uživatelského rozhraní a TinyXML pro práci se soubory XML. Pro práci s WxWidgets jsem si zvolil editor WxDEV C++, který umožňuje jednoduše navrhnout grafické rozhraní aplikace. Pro WxWidgets jsem se rozhodl díky předchozím pozitivním zkušenostem s touto knihovnou, musel jsem si však ověřit, jak se ve WxWidgets pracuje s animovanou grafikou. Poté jsem si důkladně rozmyslel, jak bude výsledný program vypadat. Po tomto kroku jsem vytvořil objektový návrh aplikace a začal s implementací. Po vytvoření funkčního grafického rozhraní jsem naimplementoval naivní algoritmus. Na tomto algoritmu jsem si pak ověřil správnost návrhu celé aplikace. Provedl jsem několik úprav v návrhu a v grafickém uživatelském rozhraní a naimplementoval ostatní algoritmy. Nakonec jsem přidal možnost uživatelské volby jazyka grafického rozhraní aplikace. Poté následovala fáze testování. Pro otestování jsem dal program vyzkoušet několika studentům. Odezvy byly vesměs kladné, padlo jenom několik připomínek k automatickému přizpůsobování velikosti fontu animací podle délky zadaného textu.

2.1 Analýza

Při analyzování problému bylo třeba si vyjasnit několik otázek. K čemu má program sloužit? Kdo ho bude používat? Jaké algoritmy budou demonstrovány? Jak budou tyto algoritmy demonstrovány? Bude někdy program někdo rozšiřovat? Program by měl sloužit k výuce, musí tedy poskytovat informace, které pomohou problém pochopit. Nebude se tedy jednat o program, který by vypisoval například statistické informace o jednotlivých algoritmech, ale bude přímo demonstrovat průběh činnosti těchto algoritmů. Je pravděpodobné, že program budou využívat jak studenti, tak i vyučující. Každá skupina uživatelů však bude program používat odlišným způsobem. Studenti budou program většinou využívat na svých počítačích v případě, že si budou chtít ujasnit algoritmus, který nepochopili. Budou se jim tedy hodit dodatečné základní informace o algoritmech, proto by měl program tyto informace poskytovat. Vyučující zase ocení rychlost a jednoduchost použití při cvičeních a přednáškách. Bylo by tedy vhodné, aby grafické prostředí vyhovovalo zobrazení na obrazovce osobního počítače a zároveň by byl program zobrazitelný dataprojektorem. Aplikace také podporuje přidávání jazyků uživatelského rozhraní, tuto funkci ocení především zahraniční studenti.

Jak už jsem uvedl výše, program bude demonstrovat algoritmy obsažené v opoře pro předmět Algoritmy, tedy naivní algoritmus, Knuth-Morris-Prattův algoritmus a Boyer-Mooreův algoritmus. Boyer-Mooreův algoritmus je v programu implementován ve dvou verzích. První verze obsahuje pouze heuristiku špatného znaku, druhá verze obsahuje obě dvě heuristiky. To umožní studentům lépe pochopit základní myšlenku Boyer-Mooreova algoritmu, protože při použití obou heuristik není činnost algoritmu příliš přehledná a animace by se bez znalosti základního principu fungování tohoto algoritmu mohla jevit zmatená. Vzhledem k tomu, že by bylo vhodné, aby byl princip znázornění činnosti algoritmů dostatečně názorný a pro uživatele pochopitelný bez zdlouhavého přemýšlení, rozhodl jsem se navázat na ilustrace v opoře, které budou studentům známé.

Je pravděpodobné, že v budoucnu přibudou nové vyučované algoritmy, proto jsem tuto skutečnost zohlednil už v návrhu programu. Snažil jsem se tedy program navrhnout tak, aby byl jednoduše rozšiřitelný o nové animace. Formát pro ukládání konfiguračních údajů programu a formát pro uložení jazykových překladů uživatelského rozhraní jsem zvolil XML.

2.1.1 Vývojové prostředky a platforma

Program je napsán v objektově orientovaném jazyce C++. K tvorbě uživatelského prostředí byl zvolen framework WxWidgets. Pro práci s XML byla použita knihovna TinyXML. Jako operační systém, ve kterém probíhal vývoj programu, jsem zvolil Microsoft Windows Vista. K překladu zdrojových kódů byl použit překladač MingW 3.4.2.

Jazyk C++

C++ je objektově orientovaný jazyk. Není to však čistě objektový jazyk, protože podporuje více programovacích stylů. C++ vyvinul na počátku 80. let 20. století Bjarne Stroustrup rozšířením jazyka C. První norma byla přijata až v roce 1998, další pak v roce 2003. Dnes patří C++ mezi nejrozšířenější jazyky.

WxWidgets

WxWidgets je dle [4] nástroj pro vývoj aplikací s grafickým uživatelským rozhraním (GUI) jak pro desktopy, tak i pro mobilní zařízení. Je to framework, který za vás odvede spoustu rutinní práce a postará se o standardní chování vašich aplikací. Knihovna WxWidgets obsahuje velký počet tříd a metod, ať už pro přímé použití či pro další úpravy. Typické aplikace zobrazují okno se standardními ovládacími prvky, často zobrazují speciální obrázky a grafiku a reagují na vstup z klávesnice, od myši či z jiných zdrojů. Taktéž mohou komunikovat s dalšími procesy nebo je přímo ovládat. Jinak řečeno, WxWidgets dělají programátorům vývoj standardních aplikací využívajících moderních technologií relativně snazší.

V [4] je uvedeno, že ačkoliv je WxWidgets často označován jako vývojový nástroj pro tvorbu

GUI aplikací, ve skutečnosti je mnohem více a má mnoho užitečných vlastností pro různá stádia vývoje aplikací. Což je velmi vhodné, neboť veškeré WxWidgets aplikace musí být schopné běhu na různých platformách a to se netýká pouze GUI. WxWidgets poskytuje třídy pro práci se soubory, s proudy, více vlákny, nastavením aplikace, pro komunikaci mezi procesy, online nápovědu, přístup k databázi a mnoho dalšího.

WxWidgets je multiplatformní nativní framework, poskytuje tedy aplikacím nativní vzhled, to znamená že využívá většinou vlastních ovládacích prvků dané platformy. Díky WxWidgets je možné program jen s malými, případně žádnými úpravami kódu kompilovat a spouštět na různých platformách. Podporované platformy jsou Microsoft Windows, Apple Macintosh, HP OpenVMS, OS/2 a Unix/Linux (X11, Motif, GTK+). Knihovny jsou implementovány v C++, ale je možné je použít v mnoha dalších jazycích.

WxWidgets je open source projekt. S vývojem začal Julian Smart v roce 1992, tehdy se projekt jmenoval WxWindows. V roce 2004 museli vývojáři dřívější WxWindows přejmenovat na WxWidgets. Název WxWindows totiž nerespektoval registrovanou obchodní známku Microsoftu.

TinyXML

Jedná se o malou a jednoduchou knihovnu pro práci s XML soubory. Díky její jednoduchosti je snadno použitelná v projektech, kde je potřeba pracovat s XML daty. Autorem této knihovny je Lee Thomason.

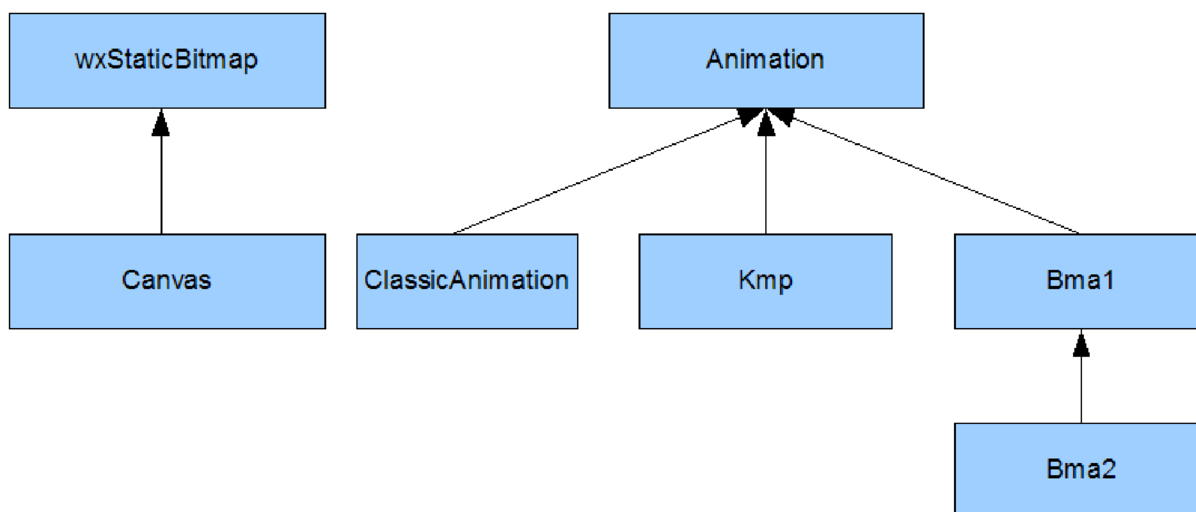
2.1.2 Návrh grafického rozhraní

Grafické rozhraní aplikace jsem navrhl tak, aby připomínalo softwarový přehrávač digitálního videa. To proto, že program vlastně jakoby přehrává animace a na ovládání přehrávačů videa jsou uživatelé zvyklí. Designově jsem rozhraní přizpůsobil vzhledu operačního systému Microsoft Windows Vista. Program umožňuje plynulé přehrávání animace s nastavitelnou rychlostí. Dále je možno zapnout přehrávání ve smyčce. Samozřejmostí je také možnost animaci kdykoliv přerušit a vrátit na začátek stisknutím tlačítka stop, nebo přehrávání pozastavit a poté pokračovat od původní pozice. Také je zde implementována možnost krokovat animaci činnosti algoritmu, což umožní důkladnější analýzu a lepší pochopení uživatelem. Původně byla zamýšlena možnost krokování i směrem zpět. Toto jsem však zavrhl, protože by to bylo implementačně náročné a přínos pro uživatele by byl minimální. Určitě by nebylo správné, kdyby program přehrával animace jen s přednastavenými texty. Proto jsou na panelu pod ovládacími prvky umístěna dvě textová pole pro zadání prohledávaných a vyhledávaných řetězců. To umožní uživateli vyzkoušet si, jak algoritmus pracuje s daty, které si sám zvolí. Pro rychlejší práci s programem zejména na přednáškách a cvičeních jsou vyhledávané a prohledávané řetězce přednastaveny na ukázkové hodnoty. Pokud je tedy uživatelské pole prázdné, bude použita přednastavená hodnota. Vedle těchto textových polích se zde také nachází menu, kde si

uživatel zvolí požadovaný algoritmus. Po zvolení algoritmu program vypíše základní informace o vybraném algoritmu v pravé části této oblasti. Grafické rozhraní je optimalizováno tak, aby bylo dobře zobrazitelné na dataprojektoru. To znamená, že prvky animací jsou kontrastní vzhledem k barvě pozadí a velikosti písma použitého v jednotlivých animacích se přizpůsobují tak, aby byla využita co největší plocha obrazovky a písmo bylo co nejlépe čitelné. Z toho však plyne jedno omezení, a to omezení délky zadávaného textu. Program tedy při zadávání omezuje délku vkládaného textu tak, aby zůstaly animace přehledné. Pokud by toto nedělal, musel by se obraz při přehrávání posouvat a tím by utrpěla přehlednost a zároveň by byl narušen design přehrávače videa. Pokud je potřeba zvětšit místo pro animaci a využít tak větší plochu, je možno skrýt spodní panel pro zadávání řetězců. Tuto možnost lze využít zejména pro zpřehlednění animace Knuth-Morris-Prattova algoritmu, kde je potřebná větší výška obrazovky. Při promítání na dataprojektoru lze podle podmínek zlepšit čitelnost animací přepnutím barevného schématu. Je zde možnost zvolit buď tmavý podklad a světlé popředí, nebo světlý podklad a tmavé popředí. Aplikace dovoluje nastavit jazyk uživatelského prostředí. Je zde možnost také přidávat nové jazyky. Díky formátu XML je přidávání nových jazykových voleb jednoduchou záležitostí bez nutnosti zásahu do zdrojových kódů aplikace (podrobnosti v kapitole Práce s programem). Formát XML je zde použit také pro ukládání nastavení aplikace.

2.1.3 Objektový návrh

Vzhledem k tomu, že jsem aplikaci vyvíjel v objektově orientovaném jazyce, bylo nutné si před započetím implementace vytvořit objektový návrh. Aplikace se tedy skládá z několika základních tříd pro vytvoření jednotlivých oken grafického rozhraní a tříd pro zobrazení a řízení běhu animací. Pro vykreslování animací se používá třída Canvas, které se předá ukazatel na instanci některé ze tříd tvořící jednotlivé animace. Aby mohla třída Canvas pracovat s jednotlivými třídami animací, bylo nutné využít polymorfismu a vytvořit bázovou třídu Animation, od které jsou potom třídy jednotlivých animací zděděny.



Obr. Třídy pro práci s animacemi - dědičnost

Základní třídy aplikace

class bakuleFrmApp : public wxApp

Jedná se o třídu, která tvoří vlastní aplikaci. Má za úkol vytvořit a zobrazit hlavní okno aplikace.

class bakuleFrm : public wxFrame

Toto je třída hlavního okna. Zabezpečuje všechny činnosti grafického rozhraní a chod aplikace. Instance ostatních tříd jsou vytvářeny právě metodami této třídy. Hlavní okno aplikace zpracovává většinu událostí vyvolaných uživatelem, obsahuje základní ovládací prvky pro ovládání animací, vstupní pole pro zadávání zpracovávaných řetězců a jiné. Třída hlavního okna zpracovává události vyvolané uživatelem při práci s těmito ovládacími prvky a reaguje na ně zasíláním zpráv ostatním třídám, případně vytvářením instancí dalších tříd (dialogová okna a jednotlivé animace). Instance této třídy je tvořena v metodě OnInit třídy wxApp hned po startu aplikace.

class SettingsDialog : public wxDialog

Třída SettingsDialog tvoří formulář sloužící k nastavování parametrů aplikace. Umožňuje měnit nastavení jazyka uživatelského rozhraní, rychlosti animací a barevného schématu. Nové nastavení ukládá do konfiguračního souboru ve formátu XML. Při vytvoření instance této třídy je jí předán ukazatel na hlavní okno aplikace. Hlavní okno je pak uzamčeno, aby uživatel musel před pokračováním v práci zavřít okno nastavení. Při zavírání se pak okno nastavení samo postará o odemčení hlavního okna aplikace.

class AboutDialog : public wxDialog

Tato třída tvoří formulář sloužící k zobrazení základních informací o programu. Využívá stejného principu uzamykání hlavního okna jako třída SettingsDialog.

class Canvas : public wxStaticBitmap

Tato třída slouží k zobrazování a řízení běhu jednotlivých animací. Třída wxStaticBitmap, od které tato třída dědí, slouží k zobrazení bitmapové grafiky. Není však vhodná pro vykreslování animací. Třída Canvas tedy rozšiřuje třídu wxStaticBitmap o metody pro přehrávání animací a upravuje vykreslování pomocí dvojitého bufferovaní tak, aby nedocházelo k rušivým jevům při překreslování jednotlivých snímků animace. Instance třídy je tvořena při startu aplikace. Jedná se o plochu v hlavním okně aplikace, na kterou se vykreslují animace. Obsahuje vlastní časovač, dokáže tedy přehrát animaci bez toho, aby muselo být přehrávání animace řízeno hlavním oknem. Metody této třídy jsou volány na základě událostí v hlavním okně aplikace. Jedná se zejména o události vyvolané uživatelem při požadavku přehrání animace, při změně velikosti a pozice hlavního okna a všechny události vedoucí k překreslení obsahu okna.

class Animation

Abstraktní třída, ze které pak dědí třídy jednotlivých animací. Obsahuje virtuální metody pro práci s animací. Instance tříd jednotlivých animací tvoří třída bakuleFrm jako reakce na událost, kdy uživatel zvolí požadovanou animaci. Ukazatel na objekt této třídy je pak předán objektu třídy Canvas. Metody této třídy jsou volány jak v metodách třídy Canvas, tak také v metodách třídy bakuleFrm.

class ClassicAnimation : public Animation

Třída představující animaci naivního algoritmu.

class Bma1 : public Animation

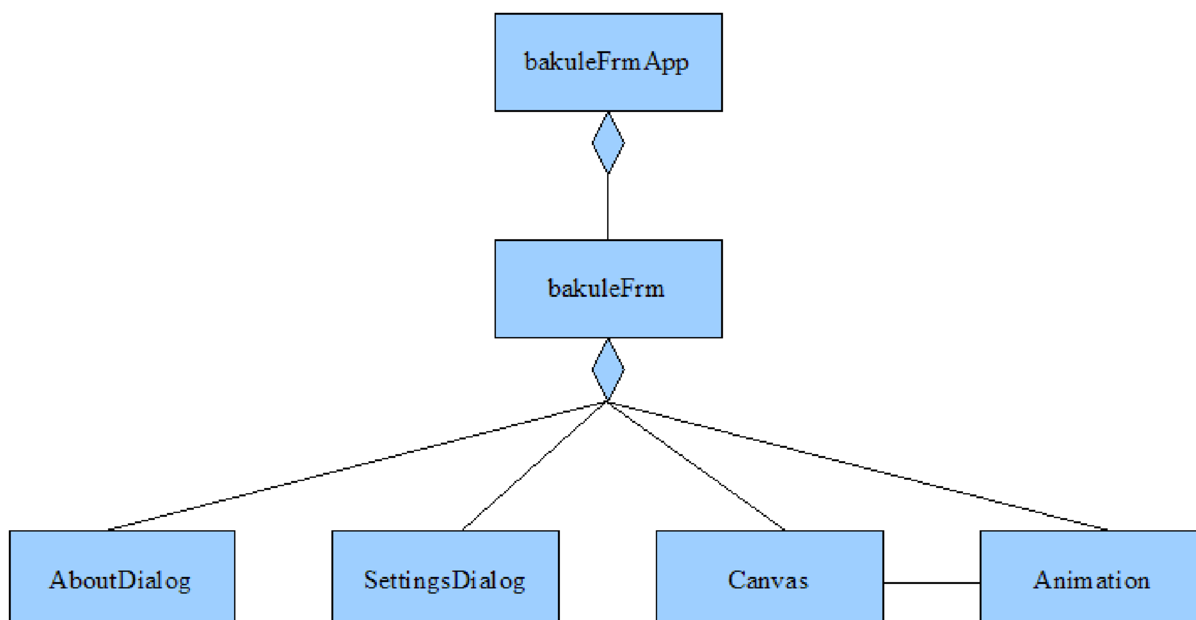
Tato třída reprezentuje animaci Boyer-Mooreova algoritmu s heuristikou špatného znaku.

class Bma2 : public Bma1

Tato třída reprezentuje animaci Boyer-Mooreova algoritmu s oběma heuristikami. Je to tedy třída Bma1 pouze rozšířená o heuristiku dobré přípony.

class Kmp : public Animation

Tato třída reprezentuje animaci Knuth-Morris-Prattova algoritmu.



Obr. Vztahy mezi základními třídami

BakuleFrmApp obsahuje bakuleFrm. BakuleFrm obsahuje AboutDialog, SettingsDialog, Canvas a Animation. Canvas pracuje s objektem třídy Animation.

Aplikace tedy pracuje tak, že je nejprve vytvořena instance třídy bakuleFrmApp, ta vytvoří instanci třídy bakuleFrm. Třída bakuleFrm pak kromě instancí tříd ovládacích prvků z knihoven WxWidgets vytvoří instanci tříd Canvas a ClassicAnimation (výchozí animace po spuštění). Pokud uživatel vyžaduje zobrazení některého z dialogových oken, je vytvořena instance jedné ze tříd SettingsDialog, nebo AboutDialog. Při volbě animace je vytvořena instance třídy příslušné animace a ukazatel na ni je předán objektu třídy Canvas. Předchozí animace je odstraněna z paměti.

2.2 Implementace

Protože by popisování celé implementace programu bylo příliš zdlouhavé a především zbytečné, zaměřím se v této kapitole pouze na základní a nejdůležitější části. Popis tříd a jejich metod je uveden v příloze číslo 2.

Třída Canvas

Jak jsem už zmínil v návrhu, jedná se o plochu, na kterou se vykreslují jednotlivé animace. Při vytváření instance této třídy je nutné v konstruktoru předat ukazatel na objekt, na kterém bude Canvas zobrazen, ukazatel na hlavní okno aplikace, pozici s rozměry a informaci o tom, zda bylo zvoleno barevné schéma s tmavým podkladem. Předání ukazatele na hlavní okno je nutné z toho důvodu, že třída Canvas řídí průběh animace a při ukončení animace musí tuto skutečnost oznámit hlavnímu

oknu, aby mohlo dojít ke změně ikon na ovládacích tlačítkách. Kromě několika soukromých metod, kterými se zde nebudu zabývat, obsahuje třída Canvas veřejné metody, které zde popíšu.

void SetCanvasSize(wxSize Size)
Tato metoda slouží k nastavení nové velikosti kreslící plochy.

void SetCanvasBColor(bool Black)
Tato metoda nastavuje barvu podkladu kreslící plochy. Přesněji řečeno, určuje, zda bylo zvoleno barevné schéma s tmavým podkladem.

void SetAnimation(Animation *pAnimation)
Metoda SetAnimation je volána v hlavním okně aplikace a je jí předán ukazatel na aktuálně zvolenou animaci.

void Play(int Fps, bool Loop)
Při zavolání Play je animace přehrána rychlostí Fps a pokud je Loop pravdivé, tak bude přehrávána stále dokola.

void Pause()
Tato metoda zastaví přehrávání animace.

void Stop()
Tato metoda zastaví animaci a vrátí ji na začátek.

Třída Animation
Obsahuje několik základních metod, které umožní nastavovat rozměry a barevné schéma animací. Její nejdůležitější metody jsou však metody pro řízení průběhu a vykreslování animace. Tyto metody popíšu podrobněji.

virtual bool SetToBegin()=0
Tato metoda nastaví animaci na první snímek.

virtual bool GetCurrentFrame(Canvas *pCanvas)=0
Tato metoda vykreslí aktuální snímek animace na kreslící plochu. Tuto metodu volá Canvas, pokud se potřebuje překreslit. Při zavolání předává ukazatel sám na sebe.

virtual bool ToNextFrame()=0

Tato metoda posune animaci o jeden krok. Avšak změny uživatel zpozoruje, až když je překreslen Canvas a zavolána metoda GetCurrentFrame. Metoda ToNextFrame pouze přepočítá hodnoty, které se vykreslují.

virtual wxString GetInfo(int lang)=0
Tato metoda vrací informaci o algoritmu v požadovaném jazyce lang.

virtual int SetSrcText(wxString SrcText)=0
Tato metoda umožňuje nastavit prohledávaný text. Je volána hlavním oknem aplikace.

virtual int SetSearchedText(wxString SearchedText)=0
Tato metoda umožňuje nastavit vyhledávaný text a je také volána hlavním oknem aplikace.

virtual void RefreshDimensions(Canvas *pCanvas)=0
Tato metoda umožňuje přepočítat rozměry a pozice jednotlivých vykreslovaných prvků. Je volána pokaždé, kdy se změní obsah animace, nebo rozměry plochy, na kterou se animace vykresluje.

Třídy ClassicAnimation, Bma1, Bma2, Kmp
Třídy jednotlivých animací jsou zděděny od bázové třídy Animation a liší se samozřejmě obsahem veřejných a soukromých metod. Všechny však obsahují důležitou soukromou metodu WriteTexts, která je volána veřejnou metodou GetCurrentFrame. Metoda WriteTexts má na starost výpočet grafického snímku. Může se zdát, že funkci metody WriteTexts mohla implementovat metoda GetCurrentFrame. Kvůli zpřehlednění kódu jsem však výpočet vykreslovaného snímku rozdělil mezi tyto dvě metody. Metoda WriteTexts vykonává vlastně hrubou práci pro metodu GetCurrentFrame.

Jednotlivé třídy pak ještě obsahují další soukromé pomocné metody. Jejich popis je uveden v příloze číslo 2.

2.3 Testování a kompatibilita

Jak už jsem zmínil, program jsem vyvíjel pod systémem Microsoft Windows Vista, proto testování základních funkcí probíhalo zejména pod tímto systémem. Při testování jsem se nejprve zaměřil na správnou funkci demonstrovaných algoritmů. Činnost algoritmů jsem srovnával s ukázkovými příklady v opoře pro předmět Algoritmy. Když jsem došel k závěru, že algoritmy fungují správně, začal jsem testovat funkce uživatelského rozhraní. Při testování správného překreslování, při změně velikosti hlavního okna programu jsem narazil na problém se změnou velikosti písma vykreslované animace. V průběhu zmenšování, nebo zvětšování okna se písmo animace střídavě zmenšovalo na minimální velikost a opět zvětšovalo na správnou velikost. Po analýze vlastního algoritmu, který

používám pro počítání požadované velikosti, jsem došel k závěru, že chyba v tomto algoritmu není. Pojal jsem podezření, že chyba může být v knihovnách WxWidgets, konkrétně v jedné z metod třídy wxDC. V manuálech WxWidgets jsem však zmínku o podobném chování neobjevil a porada se zkušenějšími kolegy také nic nevyřešila. Chybu jsem částečně odstranil úpravou překreslování okna při změně velikosti, dále jsem poupravil funkci pro výpočet velikosti písma. Po těchto úpravách se chyba vyskytovala jenom ojediněle. Po úpravě hodnoty nastavující minimální velikost písma se tato chyba během testování už nevyskytla. Pevně věřím, že chyba byla definitivně odstraněna. Vzhledem k tomu, že se mi nepodařilo odhalit přesnou příčinu tohoto podivného chování, pokládám za důležité informovat uživatele o možnosti, že se chyba opět projeví. Ostatní prvky a funkce grafického rozhraní se při testování jevily plně funkční. Při testování jsem se zaměřil také na to, zda bude program bez problémů použitelný i pokud bude spouštěn z adresáře nebo média, na které nemá uživatel právo zápisu, protože z praxe vím, že některý software má s tímto problém a nechtěl jsem, aby můj produkt trpěl podobnými neduhy.

Po otestování uživatelského rozhraní jsem testoval funkčnost pod jinými verzemi systému Microsoft Windows. Tak, jak jsem předpokládal, je program kompatibilní se všemi systémy Microsoft Windows od verze 98. Ze zkušeností vím, že kompatibilita knihoven WxWidgets a Microsoft Windows 95 je mírně problematická, tudíž jsem program pod tímto systémem ani netestoval. Nemyslím si totiž, že by cílová skupina uživatelů tento systém vůbec používala. Čím jsem však byl nemile překvapen, byla skutečnost, že program nelze přeložit pod systémem Linux. Vzhledem k použití multiplatformních knihoven jsem toto nepředpokládal.

Uživatelské testování

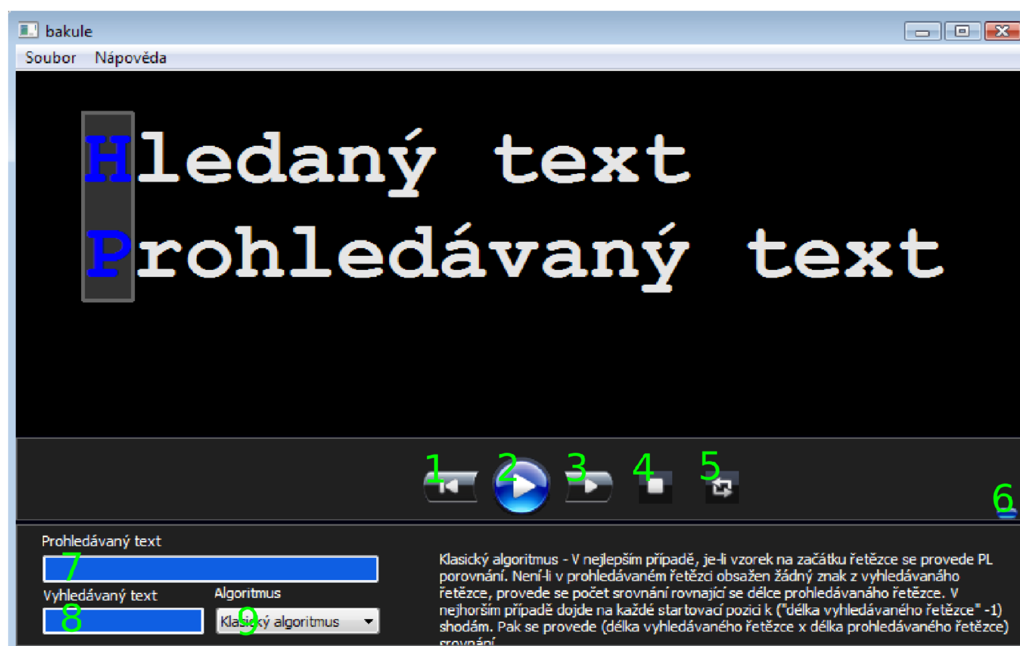
Ovládání programu se uživatelům jevilo intuitivní a přívětivé. Způsob demonstrace činnosti algoritmů jim připadal srozumitelný. Někteří uživatelé byli mírně zaskočeni automatickou změnou velikosti písma animace při zadávání vyhledávaných a prohledávaných řetězců. Přivyknutí na toto chování jim však po krátkém používání aplikace nečinilo potíže.

3 Práce s programem

V této kapitole se budu zabývat ovládáním programu. Zmíním zde doporučená nastavení a popíšu zde také formát souboru obsahující jazykové překlady uživatelského rozhraní.

Hlavní okno

Hlavní okno se dělí na tři sekce. V první sekci se zobrazuje animace, druhá sekce obsahuje ovládací tlačítka a třetí sekce umožňuje zadávání prohledávaného textu a vyhledávaného vzorku. Je zde také možno zvolit požadovaný algoritmus. Tlačítko 1 vrátí přehrávanou animaci na začátek, pokud animace běží, bude zase od začátku pokračovat. Umožňuje to uživateli rychle spustit animaci znovu, pokud se „ztratil“. Tlačítko 2 spouští animaci, pokud animace běží, tak ji pozastaví. Tlačítko 3 umožňuje krokovat animaci ručně, uživatel má pak dostatek času na pochopení jednotlivých kroků animace. Pokud animace již běží, tak stisknutím tohoto tlačítka dojde k pozastavení a dalšími stisky už uživatel může posunovat animaci po jednotlivých krocích. Tlačítko 4 zastaví animaci a vrátí jí na začátek. Poloha přepínače 5 určuje, zda bude zvolená animace přehrávána ve smyčce, nebo bude přehrána pouze jednou. Tlačítko 6 slouží ke skrytí třetí sekce, vznikne tím více místa pro animaci. Do pole 7 je možno zadat prohledávaný text, pokud je pole prázdné, je text nastaven na výchozí hodnotu. Pole 8 slouží pro zadání vyhledávaného vzorku, prázdné pole opět znamená práci s přednastaveným textem. Obě pole omezují délku zadávaného textu podle velikosti okna tak, aby zůstala zachována čitelnost animace. Roletové menu 9 vybírá požadovaný algoritmus. Vedle jsou pak zobrazeny základní informace o tomto algoritmu.

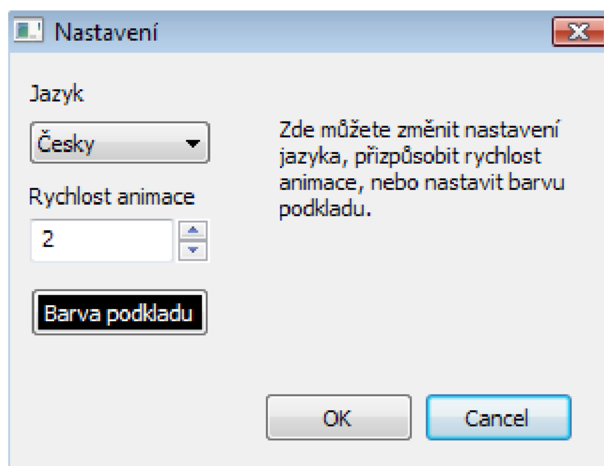


Obr. Hlavní okno aplikace

Hlavní menu

menu Soubor – podmenu Nastavení

Zobrazí okno nastavení. Zde je možno zvolit jazyk, rychlost animace a barvu podkladu. Rychlost animace se zadává v počtu kroků za sekundu. Volba barvy podkladu ovlivní také barvu vykreslovaných prvků. Lze tím zlepšit čitelnost při zobrazení dataprojektorem v závislosti na okolních podmínkách. Vzhledově optimální nastavení je ve většině případů černá barva podkladu a rychlost animace 2.



Obr. Okno nastavení

menu Soubor – podmenu Konec

Tato volba ukončí program.

menu Nápověda – podmenu Help

Zobrazí technickou zprávu ve formátu pdf.

menu Nápověda – podmenu O programu

Zobrazí základní informace o programu.

Tvorba jazykových překladů

Jazykové překlady jsou uloženy v souboru languages.xml v adresáři programu. Formát byl navržen jednoduchý a intuitivní. Při tvorbě jazyka stačí tedy přidat nový element s názvem jazyka a s parametry odpovídajícími jednotlivým řetězcům. Lze postupovat podle už hotového anglického překladu. Přidání nového překladu se plně projeví až po restartu aplikace.

Doporučení pro práci s programem

Po spuštění programu je nejlepší nejprve vybrat požadovaný algoritmus a až poté, pokud nevyhovují výchozí hodnoty zvolit vlastní prohledávaný a vyhledávaný text. Výchozí hodnoty jsou totiž zvoleny

vhodně pro každý algoritmus zvlášť.

Jestliže byl zadán velmi dlouhý text v maximalizovaném okně a okno programu poté obnovíte do původní velikosti, je vhodné zkontrolovat zda text nepřesahuje okraj okna a případně část textu odstranit.

V případě, že si program „nepamatuje“ uživatelská nastavení, zkontrolujte, zda soubor config.xml nemá nastaven atribut jen pro čtení a zda máte právo zapisovat do tohoto souboru.

Není-li možné zobrazit nápovědu, zkontrolujte, zda máte nainstalován prohlížeč souborů pdf. Program totiž nápovědu otevírá v externím prohlížeči.

Pokud se animace nezobrazují korektně, zkontrolujte, zda máte nainstalován v systému font Courier New.

Soubory potřebné ke běhu aplikace

bakule.exe
Spustitelná aplikace

config.xml
Soubor obsahující konfiguraci aplikace

languages.xml
Soubor obsahující jazykové překlady uživatelského rozhraní

4 Závěr

V mé práci jsem popsal nejznámější algoritmy pro vyhledávání v textu, jednalo se o algoritmy vyhledávající jeden vzorek. Existují i algoritmy vyhledávající množinu vzorků, ale popisování těchto algoritmů by bylo nad rámec této práce. Mým úkolem bylo především vytvořit funkční a účelnou aplikaci, která studentům umožní pochopit základní principy práce algoritmů pro vyhledávání v textu. Tato aplikace demonstruje formou animací naivní algoritmus, Knuth-Morris-Pratův algoritmus a Boyer-Mooreův algoritmus. Boyer-Mooreoův je rozdělen na dvě varianty. Jedna varianta kvůli přehlednosti obsahuje pouze heuristiku špatného znaku, druhá varianta je úplný algoritmus s oběma heuristikami. Při tvorbě programu jsem se zaměřil především na intuitivní ovládání, efektivní použitelnost a srozumitelnost jednotlivých animací. Program byl navržen tak, aby byl rozšiřitelný o jazykové překlady uživatelského rozhraní a v případě potřeby také o nové animace. Programová dokumentace potřebná pro případné rozšiřování programu je uvedena v příloze 2.

Pevně věřím, že má práce bude pro studenty přínosem a splní svůj účel, tedy umožní jim rychleji a lépe proniknout do problematiky vyhledávání v textu a dopomůže k lepším studijním výsledkům.

Zdroje

[1] *String searching algorithm* [online]. Wikimedia Foundation, Inc., 2001 , last modified on 8 April 2008, at 01:46 [cit. 18.4.2008]. Dostupný z WWW: <http://en.wikipedia.org/wiki/String_searching_algorithm>.

[2] *Hashovací funkce* [online]. Wikimedia Foundation, Inc. 2008 , naposledy editována 10. 1. 2008 v 15:42 [cit. 22.4.2008]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Hashovac%C3%AD_funkce>.

[3] *Hashovací tabulka* [online]. Wikimedia Foundation, Inc., 2006 , naposledy editována 24. 2. 2008 v 14:32 [cit. 22.4.2008]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Hashovac%C3%AD_tabulka>.

[4] *1 - WxWidgets Czech Translate* [online]. WxWidgets Czech Translate, 2008 , naposledy editována v 19:36, 8. 4. 2008 [cit. 23.4.2008]. Dostupný z WWW: <http://www.wxwidgets.hustej.net/wiki/index.php/1>.

[5] BENEŠ, Miroslav. *Referát o vyhledávání v textu* [online]. 2001 , 31.2.2001. Dostupný z WWW: <http://htmltolatex.sourceforge.net/samples/sample3.html>.

[6] Honzík J.M. Algoritmy. Studijní opora pro předmět Algoritmy. Elektronický text. FIT VUT v Brně, 2007.

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

Seznam příloh

Příloha 1 – Heuristika dobré přípony

Příloha 2 – Programová dokumentace (třídy a jejich metody)

Příloha 3 – CD

Přílohy

Příloha 1 – Heuristika dobré přípony

```
//Funkce Min vrací menší ze dvou čísel
std::string searchedText;
std::vector<int> jumpVector2(searchedText.Length()+1,0);
std::vector<int> backJumpVector2(searchedText.Length()+1,0);
for (int i=1; i<=searchedText.length(); i++)
{
   jumpVector2.at(i)=(2*searchedText.length()-i);
}
int k=searchedText.length();
int q=searchedText.length()+1;
while (k>0)
{
   backJumpVector2.at(k)=q;
   while((q<=searchedText.length()) && (searchedText.at(k-1)!=searchedText.at(q-1)))
   {
      jumpVector2.at(q)=Min(jumpVector2.at(q),searchedText.length()-k);
      q=backJumpVector2.at(q);
   }//while
   k--;
   q--;
}//while
for (k-1; k<=q; k++)
{
   jumpVector2.at(k)=Min(jumpVector2.at(k), searchedText.length()+q-k);
}
int qq=backJumpVector2.at(q);
while (q<searchedText.length())
{
   while (q<qq)
   {
      jumpVector2.at(q)=Min(jumpVector2.at(q),qq-q+searchedText.length());
      q++;
   }
```

```
    qq=backJumpVector2.at(qq);
}
```

Příloha 2 – Programová dokumentace (třídy a jejich metody)

class bakuleFrmApp : public wxApp

Jedná se o třídu, která tvoří vlastní aplikaci. Má za úkol vytvořit a zobrazit hlavní okno aplikace.

class bakuleFrm : public wxFrame

Toto je třída hlavního okna. Zabezpečuje všechny činnosti grafického rozhraní. Tvoří instance dalších tříd.

class SettingsDialog : public wxDialog

Třída SettingsDialog slouží k nastavování parametrů aplikace. Umožňuje měnit nastavení jazyka uživatelského rozhraní, rychlosti animací a barevného schématu.

class AboutDialog : public wxDialog

Slouží k zobrazení základních informací o programu.

class Canvas : public wxStaticBitmap

Tato třída slouží k zobrazování a řízení běhu jednotlivých animací. Třída wxStaticBitmap, od které tato třída dědí, slouží k zobrazení bitmapové grafiky. Není však vhodná pro vykreslování animací. Třída Canvas rozšiřuje třídu wxStaticBitmap o metody pro přehrávání animací a upravuje vykreslování pomoci dvojitého bufferovaní tak, aby nedocházelo k rušivým jevům při překreslování jednotlivých snímků animace.

Metody třídy Canvas

Veřejné metody

Canvas(wxWindow *Parent, bakuleFrm *MainFrame, wxRect Position, bool Black);

Konstruktor

wxWindow *Parent

Ukazatel na objekt, který vytváří instanci této třídy.

bakuleFrm *MainFrame

Ukazatel na hlavní okno aplikace.

wxRect Position

Nastavuje pozici a rozměry na prvku, který vytvořil instanci této třídy.
bool Black
Určuje zda bude podklad kreslící plochy černý nebo bílý.

void SetCanvasSize(wxSize Size)
Tato metoda slouží k nastavení nové velikosti kreslící plochy.

wxSize Size
Velikost kreslící plochy.

void SetCanvasBColor(bool Black)
Tato metoda nastavuje barvu podkladu kreslící plochy.

bool Black
Určuje zda bude podklad kreslící plochy černý nebo bílý.

void SetAnimation(Animation *pAnimation)
Nastaví přehrávanou animaci.
Animation *pAnimation
Ukazatel na animaci, která bude přehrávána.

void Play(int Fps, bool Loop)
Přehraje nastavenou animaci.

int Fps
Určuje rychlost přehrávání animace.

void Pause()
Tato metoda zastaví přehrávání animace.

void Stop()
Tato metoda zastaví animaci a vrátí ji na začátek.

Soukromé metody
void Paint(wxPaintEvent& event)
Tato metoda je volána při události, která vyžaduje překreslení kreslící plochy.

void WxTimer1Timer(wxTimerEvent& event);

Tato metoda je volána vnitřním časovačem, který řídí běh animace.

class Animation

Abstraktní třída, ze které pak dědí třídy jednotlivých animací. Obsahuje virtuální metody pro práci s animací.

Metody třídy Animation

Veřejné metody

virtual void SetAnimationSize(wxSize Size)=0

Tato metoda nastavuje velikost prostoru, ve kterém se bude animace vykreslovat.

wxSize Size

Velikost plochy do které se bude animace vykreslovat. V aplikaci je stejná jako velikost kreslící plochy.

virtual void SetBlack(bool Black)=0

Tato metoda nastavuje barevné schéma s tmavým pozadím a světlým popředím, nebo se světlým pozadím a tmavým popředím.

bool Black

Určuje zvolené barevné schéma. Hodnota true znamená tmavé pozadí.

virtual bool SetToBegin()=0

Tato metoda nastaví animaci na první snímek.

virtual bool GetCurrentFrame(Canvas *pCanvas)=0

Tato metoda vykreslí aktuální snímek animace na kreslící plochu.

Canvas *pCanvas

Určuje kreslící plochu, na kterou bude snímek vykreslen.

virtual bool ToNextFrame()=0

Tato metoda posune animaci o jeden krok.

virtual wxString GetInfo(int lang)=0

Tato metoda vrací informaci o algoritmu.

int lang

Určuje jazyk poskytované informace.

virtual int SetSrcText(wxString SrcText)=0

Tato metoda umožňuje nastavit prohledávaný text.

virtual int SetSearchedText(wxString SearchedText)=0

Tato metoda umožňuje nastavit vyhledávaný text.

virtual void RefreshDimensions(Canvas *pCanvas)=0

Tato metoda umožňuje přepočítat rozměry vykreslovaných prvků.

Canvas *pCanvas

Plocha, na kterou se animace vykresluje.

class ClassicAnimation : public Animation

Třída představující animaci naivního algoritmu. Veřejné metody jsou zděděné ze třídy Animation. Proto zde a u ostatních tříd reprezentujících jednotlivé animace, budu popisovat pouze soukromé (private) nebo chráněné (protected) metody.

Soukromé metody třídy ClassicAnimation

void WriteTexts(Canvas *pCanvas, int Offset, int RectOffset, int SrcColored, int SearchedColored, int Color);

Tato metoda vykreslí aktuální snímek. Je volána metodou GetCurrentFrame(Canvas *pCanvas).

Canvas *pCanvas

Určuje kreslící plochu, na kterou se bude kreslit.

int Offset

Určuje posunutí vyhledávaného textu.

int RectOffset

Posunutí rámu vyznačující právě porovnávané znaky.

int SrcColored

Určuje znak prohledávaného textu, který bude zvýrazněn.

int SearchedColored

Určuje znak vyhledávaného textu, který bude zvýrazněn.

int Color

Určuje barvu zvýrazněného textu.

0 - výchozí

1 - zelená

2 – červená

class Bma1 : public Animation

Tato třída reprezentuje animaci Boyer-Mooreova algoritmu s heuristikou špatného znaku.

Pritected metody třídy Bma1

void WriteTexts(Canvas *pCanvas, int Offset, int RectOffset, int SrcColored, int SearchedColored, int Color);

Stejná jako ve třídě ClassicAnimation. Přibyla nová barva 3- žlutá.

int Chtoi(wxChar ch)

Tato metoda převede wxChar na integer.

wxChar ch

Převáděný znak.

class Bma2 : public Bma1

Tato třída reprezentuje animaci Boyer-Mooreova algoritmu s oběma heuristikami.

Soukromé metody třídy Bma2

int Min(int number1, int number2)

int Max(int number1, int number2)

Vrací minimum a maximum ze dvou čísel.

class Kmp : public Animation

Tato třída reprezentuje animaci Knuth-Morris-Prattova algoritmu.

Soukromé metody třídy Kmp

void WriteTexts(Canvas *pCanvas, bool YArr, int NarrFrom ,int RectOffset, int SrcColored, int SearchedColored, int Color)

Význam této metody je stejný jako v předchozích třídách. Přibyly nové parametry. Bool Yarr určuje, zda vykreslovaná hrana je hrana pro posun v před či zpět v automatu KMP. NarrFrom index, ze kterého se povede hrana k právě vykreslovanému stavu.

void DrawArrow(wxBufferedPaintDC *DC, int x1, int y1, int x2, int y2)
Obyčejná šipka z bodu x1, y1 do bodu x2, y2.

void DrawSplineArrow(wxBufferedPaintDC *DC, int n, wxPoint points[], int arrSize)
Spline šipka protínající pole bodů points[].
int ArrSize určuje velikost šipky.

void SetVect()
Sestaví vektor reprezentující automat KMP.