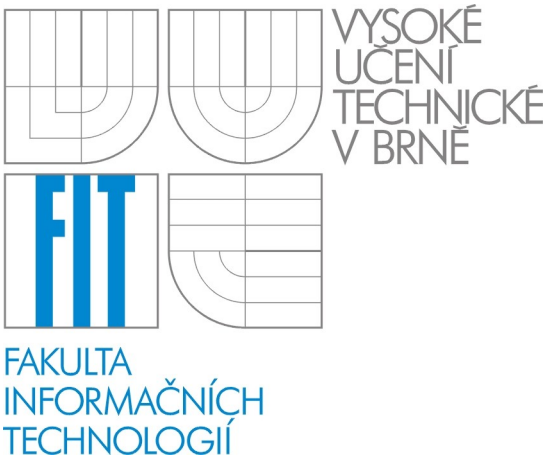

Dokumentace k projektu do předmětu GJA

Applet pro demonstraci Boyerova-Mooreova algoritmu

2007 / 2008

Autor: Jaroslav Dytrych, xdytry00@stud.fit.vutbr.cz
Fakulta Informačních Technologií
Vysoké Učení Technické v Brně

Obsah:

1. Úvod

Cílem tohoto projektu je implementace appletu pro demonstraci Boyerova-Mooreova algoritmu (dále BMA) pro vyhledávání řetězce v textu. Hlavním úkolem vytvořeného appletu bude přehledná demonstrace činnosti algoritmu a jeho datových struktur.

2. Návrh řešení

Nejprve jsem prostudoval zadaný algoritmus, vytvořil jeho vhodnou implementaci a provedl návrh uživatelského rozhraní appletu, přičemž jsem vycházel z požadavku na maximální přehlednost uživatelského rozhraní i demonstrace činnosti algoritmu. Následně jsem prostudoval možnosti jazyka Java a jeho knihoven pro tvorbu uživatelských rozhraní a dle dostupnosti navržených komponent jsem implementoval chybějící součásti uživatelského rozhraní.

Celý návrh i implementaci appletu jsem dekomponoval na několik částí, které řeší hlavní problémy tvorby appletu. Následuje popis těchto částí.

2.1 Hlavní panel appletu

Aby byla demonstrace činnosti maximálně přehledná, je velmi vhodné, aby bylo vše umístěno v jednom panelu a uživatel si vše mohl prohlížet z globálního pohledu bez nutnosti přepínání oken. Protože je třeba zobrazit velké množství prvků, zvolil jsem maximální možnou velikost panelu appletu takovou, aby bylo možné jej zobrazit v nejvíce využívaných prohlížečích s nejčastěji využívanými rozlišeními monitoru a stále byly viditelné všechny potřebné prvky panelu. Do zvolené velikosti panelu jsem umístil všechny potřebné prvky s ohledem na maximální využití prostoru při zachování přehlednosti.

Součástí uživatelského rozhraní bude i nápověda, která bude vzhledem ke své délce a k nevhodnosti otevírání dalších oken appletem, který bude obvykle zobrazován ve webové stránce, umístěna v textovém poli v dolní části appletu.

2.2 Algoritmus Boyer-Moore

BMA je ve své původní variantě rychlý a efektivní algoritmus pro vyhledávání prvního výskytu podřetězce v daném řetězci viz. [BMA]. Vzhledem k tomu, že vyhledání všech výskytů řetězce představuje pouze malou úpravu algoritmu (v cyklu je místo návratu z funkce uložení pozice a pokračování) a delší průběh cyklu, který by pro demonstraci činnosti algoritmu nebyl přínosem, využiji původní variantu algoritmu a vytvořený applet bude vyhledávat pouze první výskyt řetězce.

Aby byla činnost algoritmu (především indexace v polích) maximálně přehledná, zvolil jsem velmi přehlednou implementaci v jazyce Pascal, kterou jsem nalezl na stránkách ČVUT Praha [ABM] a pro větší přehlednost ji modifikoval tak, že delta1 neudává poslední výskyt prvku, ale posun a při vyhledávání se následně pouze porovnávají hodnoty posunu delta1 a delta2.

Vzhledem k tomu, že applet je implementován v jazyce Java, zvolil jsem tento jazyk nejen pro implementaci algoritmu, ale i pro jeho výpis, který bude zobrazen uživateli. Z tohoto důvodu nevyužiji dostupných efektivních implementací v jazyce Java, které zohledňují způsob indexace řetězců od 0, ale přepíši z Pascalu do Javy výše uvedenou implementaci. V dané implementaci řetězce začínají indexem 1 a při výpočtu tabulky delta2 je využit nulový index pole pro hodnotu mezivýsledku. Přepočet indexů by algoritmus silně znepřehlednil, protože by v různých polích byly různé indexace. Oba řetězce tedy před započetím činnosti algoritmu zkopíruji do polí od indexu 1 a následně využiji původní indexaci vycházející z Pascalu.

Protože je původní varianta algoritmu navržena pro anglické texty, využívá pole o velikosti celé abecedy indexované jednotlivými znaky. Při využití rozsáhlé znakové sady UTF-8 by toto

pole nabývalo velkých rozměrů, přičemž všechny položky kromě maximálně n (délka hledaného řetězce) položek by obsahovaly stejnou hodnotu. Protože n je obvykle výrazně menší, než velikost znakové sady UTF-8, a zobrazení takto velké tabulky by vizualizaci silně znepřehlednilo, modifikuji algoritmus tak, aby místo tohoto pole využíval asociativní kontejner, ve kterém budou uloženy hodnoty pro jednotlivé znaky z hledaného řetězce, a proměnnou, která bude uchovávat hodnotu, která je shodná pro všechny zbývající řetězce znakové sady.

2.3 Vizualizace

Aby uživatel mohl demonstrovaný algoritmus detailně prostudovat, pochopit a následně ihned využít, bude uživateli zobrazena kompletní implementace algoritmu, ve které bude zvýrazňován právě provedený krok. Pro úsporu času uživatele a vyhnutí-se dlouhému klikání, při kterém dochází např. ke vkládání stejné hodnoty na všechny indexy pole při jeho inicializaci, či pouze k posunu zvýraznění řádku přes deklarace proměnných, nebudou kroky vizualizace totožné s kroky algoritmu. Některé kroky vizualizace tedy budou představovat provedení většího množství operací, přičemž každá důležitá vizualizovatelná operace algoritmu bude umístěna v jiném kroku vizualizace.

Při vizualizaci, která bude mít často velké množství kroků, je vhodné, aby uživatel mohl dělat kroky vpřed i zpět. Zpětný chod algoritmu však není možný. Vizualizaci tedy není možné provádět přímo s během algoritmu, ale je třeba provést průchod algoritmem, při kterém se bude sledovat jeho činnost a datové struktury a uloží se informace pro následnou vizualizaci.

Vizualizace potom bude prováděna jako sekvence uložených změn, přičemž v každém kroku budou sekvenčně provedeny všechny změny, ke kterým v tomto kroku došlo. Při kroku zpět budou tyto změny v opačném pořadí vraceny.

Protože při kroku zpět jsou třeba informace o původních stavech měněných objektů, budou u informace o každé změně uloženy i informace pro její vrácení (původní obsah pole v tabulce, původní hodnota proměnné apod.). Pro získání těchto informací bude využit dopředný průběh vizualizace, přičemž se před provedením každé změny při kroku vpřed uloží informace o původním stavu měněného objektu.

Vzhledem k tomu, že algoritmus je proveden na začátku vizualizace, není možné, aby uživatel měnil zadané řetězce v průběhu vizualizace. Tato operace tedy uživateli nebude umožněna.

Vizualizován bude nejenom průchod algoritmem a vyobrazení tabulek, ale také vyobrazení aktuálního stavu proměnných. Protože se proměnné využívají především k indexaci tabulek a řetězců, bude právě indexované políčko tabulky zvýrazněno stejnou barvou, jakou má políčko s danou proměnnou. Pro přehlednost budou mezi zobrazované proměnné zahrnuty i některé hodnoty, které jsou vypočítané z hodnot proměnných a využity pro indexaci (např. i-j+1).

Protože v panelu s appletem není dostatek místa pro vyobrazení všech proměnných využitých v průběhu vykonávání algoritmu, budou některá políčka určena pro hodnoty dvou různých proměnných a jejich popisek se bude měnit dle kontextu (prováděné metody).

2.4 Tabulky

Vzhledem k tomu, že knihovna jazyka Java pro tvorbu uživatelských rozhraní neumožňuje vytvářet tabulky se záhlavími v řádcích a u tohoto charakteru tabulek by bylo zobrazení ve sloupcích nepřehledné, bude třeba implementovat vlastní prvek uživatelského rozhraní, který bude zobrazovat potřebný typ tabulky.

Vzhledem k různým potřebným šířkám (závisí především na velikosti číselných hodnot v tabulkách) a počtům sloupců je třeba za běhu měnit velikost tabulky a šířky sloupců dle zobrazených hodnot.

Pro přehlednost budou mít všechny sloupce tabulky kromě sloupce se záhlavím, který je obvykle širší, stejnou šířku. Protože tabulka bude využita i pro vizualizaci porovnávání řetězců, využití sloupce se záhlavím bude volitelné.

Protože v tabulce delta1 bude poslední sloupec určen pro znaky, které nejsou obsaženy v hledaném řetězci, bude obsahovat text „jiné znaky“, který je obvykle širší, než text ve zbylých buňkách tabulky. Z tohoto důvodu bude tabulka poskytovat možnost nastavení jiné šířky posledního sloupce.

Tabulku také musí být možné vytvořit bez ohraničení a musí umožnit změnu barvy buněk, která bude využívána při vizualizaci.

2.5 Panel s výpisem algoritmu

K přehlednému zobrazení zdrojového textu algoritmu je nutné jej zobrazit se zvýrazněním syntaxe. Žádný z dostupných prvků jazyka Java a jeho knihovny Swing však neumí provádět zvýraznění syntaxe a implementace lexikálního analyzátoru či využití knihoven třetích stran by byly pro krátký algoritmus dané délky zbytečné. Vytvořím tedy komponentu uživatelského rozhraní pro zobrazení panelu, do kterého bude možné vkládat úseky textu s různými barvami a zobrazovaný zdrojový text algoritmu rozdělím na podřetězce s různou barvou, které se následně dle potřeby budou vypisovat do panelu.

Panel umožní rovněž změnu barvy libovolného zobrazeného úseku textu, aby jej bylo možné zvýraznit při vizualizaci.

Aby uživatel nemusel při vizualizaci posouvat panelem, bude v každé fázi zobrazena pouze právě prováděná metoda. Když vizualizace nebude probíhat, zobrazí se celý zdrojový kód.

3. Implementace

Celý navržený applet jsem následně implementoval jako samostatný balíček v jazyce Java. Základ appletu tvoří třída představující applet, která zobrazuje hlavní panel appletu se všemi prvky uživatelského rozhraní a provádí vizualizaci.

Na třídě appletu je závislá třída pro provádění BMA a vytváření seznamu změn uživatelského rozhraní prováděných při vizualizaci.

Protože je využito velké množství konstant (texty, indexy operací, indexy proměnných, rozměry tabulek, apod.), vytvořil jsem také třídu obsahující pouze konstanty, jejíž instance je vytvořena třídou appletu a pro úsporu paměti je následně využívána i na něm závislou třídou s implementací BMA.

Pro zobrazování tabulek a panelu se zdrojovým textem BMA jsem vytvořil separátní třídy, které jsou popsány níže.

3.1 Hlavní panel appletu

Hlavní panel appletu je tvořen třídou appletu. V této třídě jsou implementovány také všechny metody pro změny uživatelského rozhraní a provádění vizualizace.

Při inicializaci appletu (metoda init volaná prohlížečem a jí volaná metoda jbInit) jsou nastaveny výchozí vlastnosti všech prvků uživatelského rozhraní a tyto prvky jsou vloženy do panelu appletu.

Metoda pro obsluhu kliknutí myší na tlačítko „Vpřed“ nejprve rozliší, zda probíhá nebo neprobíhá vizualizace dle aktuálního kroku vizualizace. Pokud vizualizace neprobíhá, zablokuje změny textových polí a volá metodu s BMA, která při svém provádění vytvoří seznam změn uživatelského rozhraní pro vizualizaci. Změní také obsah pole s nápovědou a zobrazí počítadlo kroků.

Pokud vizualizace probíhá a není na posledním kroku, volá metodu pro provedení kroku vizualizace, která je popsána níže.

Metoda pro obsluhu kliknutí na tlačítko „Zpět“ zkontroluje, zda lze krok zpět provést a volá níže popsanou metodu pro jeho provedení.

Metoda pro obsluhu kliknutí na tlačítko „Reset“ ukončí vizualizaci, vymaže obsah seznamu změn uživatelského rozhraní při vizualizaci a nastaví všechny komponenty

uživatelského rozhraní kromě polí se zadávanými řetězci do výchozího stavu. U polí pro zadávané řetězce povolí jejich modifikaci.

Aby byla třída pro zobrazení panelu se zdrojovým textem nezávislá, jsou zde implementovány také metody pro výpis bloku a celého algoritmu do tohoto panelu. Metoda pro výpis bloku algoritmu do panelu vypisuje zdrojový text algoritmu po jednotlivých různobarevných úsecích. Úseky algoritmu jsou stejně jako jejich barvy a informace o tloušťce písma uloženy v konstantních polích. Tato metoda nastavuje také proměnnou, která uchovává číslo bloku, který je právě zobrazen, a která umožňuje přepočet indexů tak, že zobrazené úseky textu v panelu jsou indexovány vždy od 0, ale pro vizualizaci jsou bloky algoritmu indexovány konstantními čísly, kde index 0 má začátek celého zdrojového textu BMA.

Metoda pro zvýraznění řádku algoritmu nejprve zruší zvýraznění právě zvýrazněného řádku (je-li některý řádek zvýrazněn) a následně zvýrazní řádek se zadaným číslem. Obě operace se skládají z výpočtu indexu počátečního a koncového úseku řádku v panelu a volání metody pro změnu barvy jednotlivých úseků. Indexy jsou vypočítávány z konstant pomocí indexu měněného řádku a indexu zobrazeného bloku, takže každý řádek algoritmu má své pevně přidělené číslo nezávisle na pozici v panelu.

Jsou zde také metody pro nastavení a získání obsahů polí s proměnnými dle indexů, které jsou polím přiřazeny ve třídě s konstantami a metody pro provedení dopředného a zpětného kroku vizualizace, které jsou popsány níže.

3.2 Algoritmus Boyer-Moore

BMA jsem implementoval tak, že jsem přepsal z Pascalu do Javy zdrojový kód uvedený na stránce [ABM] a modifikoval jej tak, že v tabulce delta1 jsou místo posledních výskytů hodnoty posunů. Také jsem nahradil pole delta1 asociativním stromovým kontejnerem a proměnnou pro hodnotu posuvu pro znaky, které se nevyskytují v hledaném řetězci. Proměnné jsem pojmenoval dle původní implementace se zohledněním uvedených modifikací, ale tabulky delta1 a delta2 jsem pojmenoval podle [BMA] stejně jako pole s hledaným řetězcem.

Oproti implementaci v Pascalu se moje implementace liší i tím, že vrací index nalezeného řetězce v prohledávaném textu, přičemž před vrácením hodnoty se provádí korekce vraceného indexu o 1, aby odpovídal indexaci vstupního řetězce v Javě.

Ve třídě pro provádění algoritmu BMA je zdrojový text algoritmu ve stejné podobě jako v panelu pro uživatele (v panelu pro uživatele jsou pro úsporu místa ubrány vypustitelné závorky) a je dále doplněn o kód pro vytváření seznamu změn uživatelského rozhraní při vizualizaci. Za kódem každého kroku algoritmu, který lze vizualizovat, je tedy umístěn kód pro vytvoření položek seznamu, které tuto vizualizaci realizují. Protože seznam změn pro vizualizaci, instance třídy s konstantami a proměnná pro počet kroků vizualizace jsou společné pro třídu appletu i pro třídu s BMA (umístěny ve třídě appletu), tyto třídy jsou na sobě závislé.

3.3 Vizualizace

Vizualizace je prováděna metodami pro dopředný a zpětný krok vizualizace, které jsou umístěny ve třídě appletu. Tyto metody pomocí iterátoru (globální, vytvořen při zahájení vizualizace) procházejí seznamem změn uživatelského rozhraní a provádějí změny, které do tohoto seznamu uložily metody třídy BMA.

Každá změna uživatelského rozhraní je v tomto seznamu reprezentována objektem třídy, která slouží k uchování daných informací. Tato třída obsahuje veřejné proměnné s popisem změny a různé konstruktory, přičemž jednotlivé konstruktory se liší v tom, které proměnné nastavují a tedy i v tom, které změny takto vytvořené objekty představují.

Metoda pro dopředný krok vizualizace inkrementuje číslo aktuálního kroku a následně pomocí iterátoru postupně prochází seznam změn uživatelského rozhraní směrem vpřed, dokud

se číslo kroku v objektu změny shoduje s číslem aktuálního kroku, a provádí jednotlivé změny. Postupně tak provede všechny změny, které k danému kroku patří. Provedení každé změny se skládá z určení operace, určení měněného (ovlivněného) objektu a provedení změny. Před provedením změny se také uloží do objektu popisujícího danou změnu informace o tom, jaký byl původní stav měněného objektu.

Metoda pro zpětný krok vizualizace pomocí iterátoru postupně prochází seznam změn uživatelského rozhraní směrem vzad, dokud se číslo kroku shoduje s číslem aktuálního kroku, a pomocí informací v objektech změn (doplněných o informace získané v dopředných krocích) vrací jednotlivé změny. Vrátí tak všechny změny provedené aktuálním krokem. Nakonec dekrementuje číslo kroku.

Třída s BMA vytváří jednotlivé kroky vizualizace tak, že nejprve inkrementuje počet kroků ve třídě appletu a následně přidává do seznamu změn jednotlivé změny s tímto číslem. Tímto je zajištěn konzistentní a korektní stav počtu kroků a seznamu změn při libovolném průchodu řídícími strukturami BMA a při opuštění metody libovolným bodem. Jednotlivé změny realizují nejenom nastavení obsahů políček s proměnnými a buněk tabulek, ale také změny rozměrů tabulek, změny popisků, záměny bloků kódu v panelu s výpisem algoritmu, vizualizaci porovnávání řetězců ve zvláštní tabulce určené k tomuto účelu (řetězce jsou zde posouvány aby byly porovnávané znaky pod sebou) apod.

3.4 Tabulky

Protože knihovny pro tvorbu uživatelských rozhraní, kterými jazyk Java disponuje neumožňují vytvořit tabulku se záhlavími v řádcích, implementoval jsem vlastní třídu pro zobrazení tabulky.

Vzhledem k různým potřebným šířkám sloupců je třeba za běhu měnit šířky sloupců tabulky dle zobrazených hodnot. Tyto změny mohou být navíc volitelně prováděny zvlášť pro první, poslední a všechny ostatní sloupce. Mnou implementovaná třída tyto změny provádí automaticky tak, že pokud je buňka příliš malá pro vložení daného řetězce, je daný sloupec (skupina sloupců) automaticky rozšířen. Zúžení není prováděno automaticky dle aktuálně nejdelšího řetězce, ale lze jej volat explicitně pomocí k tomu určené metody.

Konstruktor třídy nejprve nastaví proměnné s vlastnostmi tabulky dle svých parametrů a následně vytvoří a inicializuje pole pro obsahy, barvy a řezy písma buněk. Nakonec vypočítá a aktualizuje rozměry kreslícího plátna, na které se tabulka vykreslí, a volá metodu pro překreslení tabulky.

Metoda pro nastavení obsahu buňky tabulky nejprve zjistí, zda buňka není v prvním či posledním sloupci a zda tento první či poslední sloupec nemá jinou šířku než zbylé sloupce tabulky (vlastnost nastavená konstruktorem). Následně nastaví obsah buňky, zjistí šířku nového obsahu v px (pomocí metriky fontu) a případně upraví šířku daného sloupce či skupiny sloupců. Na závěr přepočítá velikost plátna a volá metodu pro překreslení tabulky. Metoda pro nastavení barev buňky pouze nastaví barvy a volá metodu pro překreslení.

Metoda pro minimalizaci šířky tabulky upraví rozměry sloupců tabulky dle nejširšího obsahu buňky v daném sloupci (skupině sloupců). Tuto operaci provede tak, že projde všechny buňky každého sloupce (skupiny sloupců), nalezne nejširší obsah, upraví rozměr, přepočítá velikost plátna a zavolá metodu pro překreslení tabulky.

Metoda pro změnu rozměrů tabulky pracuje ve dvou fázích. Nejprve mění počet řádků a potom počet sloupců. Při každé z těchto operací nejprve vyhodnotí, zda se počet zmenšuje, nebo zvětšuje. Při zmenšení počtu řádků (sloupců) pouze nastaví nový počet. Pokud se počet řádků (sloupců) zvětšuje, vytvoří nová pole požadované velikosti, zkopíruje do nich informace z původních polí (cykly přes všechny buňky), nastaví výchozí vlastnosti nových buněk a provede záměnu polí. Po záměně polí aktualizuje daný rozměr tabulky. V obou případech nakonec přepočítá velikost plátna a volá metodu pro překreslení tabulky.

Metoda pro vyprázdnění tabulky projde všechny buňky, vymaže jejich obsah a nastaví jim výchozí barvy. Nastaví také výchozí šířky buněk, přepočítá rozměry plátna a volá metodu pro překreslení.

Metoda pro posun obsahu buněk v řádku tabulky projde dvěma cykly (posun a doplnění prázdných řetězců) daný řádek tabulky a posune obsah jeho buněk o požadovaný počet požadovaným směrem. Na straně tabulky, od které se posouvá, tedy vkládá do tabulky prázdné řetězce a druhé straně obsah buněk, které se posunou mimo tabulku, zahazuje.
Při provedení posuvu tedy dochází ke ztrátě dat a nelze jej využít při vizualizaci vyhledávání řetězce v textu (při posuvu zpět by nebyl obnoven původní obsah buněk). Tato metoda se využívá při vizualizaci hledání podřetězců při výpočtu tabulky delta2, kde jsou rozměry tabulky a pozice řetězců zvoleny tak, že řetězec není nikdy vysunut mimo tabulku a zahazují se tedy pouze prázdná políčka. Umožňuje zde výraznou úsporu počtu objektů popisujících změny, kterou při vyhledávání řetězce v textu nelze využít a musí se tedy zaznamenávat změna každé buňky zvlášť.

Metoda pro překreslení tabulky nastaví rozměry oblasti ve skrolovacím panelu dle rozměrů plátna a volá metody pro přepočítání rozměrů a překreslení tabulky (zde žádost o překreslení, nikoliv přímo vykreslovací funkci).

Metoda pro vykreslení tabulky je předefinovanou metodou vykreslení komponentu z rodičovské třídy. Nejprve volá metodu z rodičovské třídy, která vykreslí kreslící plochu. Následně vypočítá invarianty cyklů a v cyklech vykreslí pozadí a obsahy jednotlivých buněk. Pokud jsou první či poslední sloupec jiné šířky, jejich vykreslení je provedeno po průběhu cyklů separátně. Následně se určí, zda má tabulka ohraničení a případně se toto vykreslí v cyklech po jednotlivých čárách. Pokud je první sloupec jiné šířky, je jeho ohraničení vykreslováno separátně. Separátně je také vykreslováno obdélníkové ohraničení celé tabulky, které má dvojnásobnou tloušťku čáry.

Tabulka má také metody pro zjištění aktuálního obsahu a barev buněk a rozměrů tabulky, které se využívají při ukládání informací pro zpětné kroky vizualizace.

Aby se tabulka správně zobrazovala a posouvala ve skrolovacím panelu, implementuje rozhraní Scrollable. Má tedy metody pro zjištění rozměrů plátna a kroků posuvu při skrolování, přičemž krok posuvu, který vrací, odpovídá šířce běžného sloupce tabulky.

3.5 Panel s výpisem algoritmu

Panel s výpisem algoritmu umožňuje vkládání barevných úseků textu, které mohou být běžným nebo tučným písmem. Jednotlivé úseky čísluje dle pořadí vložení a následně umožňuje měnit barvu jednotlivých úseků podle jejich čísel.

Mnou vytvořená třída rozšiřuje funkcionalitu textového panelu z knihovny Swing. Protože jednotlivé úseky textu jsou do panelu vkládány přidáním do stylovaného dokumentu, který panel zobrazuje, je nutné uchovávat informace o pozicích a délkách jednotlivých úseků v panelu, aby s nimi bylo později možné manipulovat. Pro uchování informací o úseku textu má tato třída vnořenou podtřídu, která má veřejné proměnné pro začátek a délku úseku a barvy a řez písma, se kterými byl úsek vložen do panelu. Má také konstruktor, který tyto proměnné nastavuje.

Informace o jednotlivých úsecích textu jsou ukládány do stromového asociativního kontejneru v objektech výše uvedené vnořené třídy.

Konstruktor třídy volá konstruktor rodičovské třídy, který vytvoří panel. Následně získá referenci na stylovaný dokument ve vytvořeném panelu, vytvoří kontejner pro informace o úsecích textu a vynuluje počet úseků.

Metoda pro přidání úseku textu do panelu nejprve vytvoří stylový objekt, kterým nastaví styl (barvu a vlastnosti písma) vkládaného úseku, zjistí délku přidávaného řetězce a délku dokumentu (budoucí pozici přidávaného úseku v dokumentu) a uloží informace o vkládaném úseku do kontejneru. Následně vloží úsek textu do dokumentu a inkrementuje počet úseků v dokumentu.

Metoda pro nastavení barvy úseku textu nejprve zjistí informace o daném úseku, vytvoří nový stylový objekt pro daný úsek a odebere z dokumentu starý stylový objekt úseku. Následně z dokumentu zjistí textový obsah úseku, úsek z dokumentu odebere a znovu jej vloží s novým stylem.

Metoda pro nastavení výchozí barvy úseku textu pracuje obdobným způsobem jako metoda pro nastavení barvy úseku, ale pro tvorbu nového stylového objektu využívá informace o výchozích barvách (barvách, se kterými byl úsek vložen do dokumentu) uložené v kontejneru.

Metoda pro vyprázdnění panelu odebere z dokumentu celý jeho textový obsah a vytvoří nový kontejner pro informace o úsecích.

4. Závěr

Implementovaný applet provádí přehlednou demonstraci Boyerova-Mooreova algoritmu na zadaných řetězcích. Umožňuje procházení průběhu algoritmu oběma směry a restart činnosti. V průběhu vizualizace jsou přehledně zobrazovány všechny datové struktury i zdrojový kód algoritmu v jazyce Java.

5. Literatura

- [ABM] Mládek, P.: *Algoritmus Boyer-Moore* [online], ČVUT Praha, únor 1999 [cit. 18.4.2008], dostupný z WWW:
  < http://moon.felk.cvut.cz/~pjv/Jak/_info/i449/algoritmy.html >
- [BMA] Boyer, R. S., Moore, J. S.: *A Fast String Searching Algorithm* [online], ACM New York, říjen 1977 [cit. 18.4.2008], ISSN 0001-0782, dostupný z WWW:
  < http://www.cs.utexas.edu/~moore/publications/fstrpos.pdf >