VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY

DEPARTMENT OF INFORMATION SYSTEMS

Algoritmy pro segmentaci webových stránek

diplomová práce

Master´S THESIs

AUTOR PRÁCE Bc. Tomáš Laščák

AUTHOR

BRNO 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY

DEPARTMENT OF INFORMATION SYSTEMS

Algoritmy pro Segmentaci webových stránek

Web page segmentation algorithms

Diplomová práce

Master´S THESIs

AUTOR PRÁCE Bc. Tomáš Laščák

AUTHOR

VEDOUCÍ PRÁCE Ing. Radek Burget, Ph.D.

SUPERVISOR

BRNO 2016

Abstrakt

Segmentace webových stránek je jednou z disciplín extrakce informací. Umožňuje dělit stránky na různé sémantické bloky. Semestrální projekt se zabývá seznámením se segmentací jako takovou a také segmentační metodou. V práci jsou popsány různé příklady metod jako je VIPS, DOM PS atd. Je zde teoretický popis zvolené metody a taktéž Frameworku FitLayout, který bude o tuto metodu rozšířen. V závěru práce je uvedeno shrnutí a také popis dalších postupů při navázání této práce na diplomovou.

Abstract

Segmentation of web pages is one of the disciplines of information extraction. It allows to divide the page into different semantic blocks. Term project deals with learning about segmentation as well as a segmentation method. In this paper we describe various of examples method such as VIPS, DOM PS etc. There is a theoretical description of the chosen method and also FitLayout Framework, which will be expanded by this method. In the conclusion is a summary and description of the other procedures for the establishment of the work on the thesis.

Klíčová slova

Vizuální sémantika, Java, WWW, Segmentace, Document Object Model

Keywords

Visual semantic, Java, WWW, Segmentation, Document Object Model

Citace

Laščák Tomáš: Algoritmy pro segmentaci webových stránek, diplomová práce, Brno, FIT VUT v Brně, 2016

**Algoritmy pro segmentaci webových stránek**

Prohlášení

Prohlašuji, že jsem tuto práci vypracoval samostatně pod vedením pana Ing. Radka Burgeta, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……………………

Tomáš Laščák

15. 5. 2016

Poděkování

Rád bych tímto poděkoval vedoucímu této práce, Ing. Radku Burgetovi, Ph.D., za odborné rady a vedení při tvorbě této práce.

© Tomáš Laščák, 2016

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

[Obsah 1](#_Toc449455645)

[1 Úvod 2](#_Toc449455646)

[2 Segmentace webových stránek 4](#_Toc449455647)

[2.1 DOM-based Page Segmentation 4](#_Toc449455648)

[2.2 Vision-based Page Segmentation – VIPS 6](#_Toc449455649)

[2.3 Box Clustering Segmentation 8](#_Toc449455650)

[3 Segmentace webových stránek za pomocí vizuální sémantiky 10](#_Toc449455651)

[3.1 Příprava 10](#_Toc449455652)

[3.1.1 Vizuální bloky 11](#_Toc449455653)

[3.1.2 Předzpracování webových stránek 11](#_Toc449455654)

[3.2 Algoritmus segmentace 12](#_Toc449455655)

[3.2.1 Rozpoznání podobných vizuálních bloků 12](#_Toc449455656)

[3.2.2 Výpočet stupně sousednosti bloků 14](#_Toc449455657)

[3.2.3 Výpočet podobnosti obsahu bloků 15](#_Toc449455658)

[3.2.4 Segment webové stránky 18](#_Toc449455659)

[4 FitLayout 20](#_Toc449455660)

[4.1 FitLayout Moduly 20](#_Toc449455661)

[4.1.1 FitLayout API 20](#_Toc449455662)

[4.1.2 CSSBox 21](#_Toc449455663)

[4.1.3 Segmentace 22](#_Toc449455664)

[4.1.4 Klasifikace 23](#_Toc449455665)

[4.1.5 Služby 23](#_Toc449455666)

[4.1.6 Nástroje 23](#_Toc449455667)

[4.2 Práce s Frameworkem 24](#_Toc449455668)

[4.2.1 Operátory 25](#_Toc449455669)

[5 Implementace algoritmu 27](#_Toc449455670)

[5.1 Sousednost bloků 27](#_Toc449455671)

[5.1.1 Nesousedící bloky 27](#_Toc449455672)

[5.1.2 Sousedící bloky 31](#_Toc449455674)

[5.2 Vizuální obsah bloků 34](#_Toc449455675)

[5.2.1 Vektory obsahu 35](#_Toc449455676)

[5.2.2 Podobnost obsahu 36](#_Toc449455677)

[5.3 Řízení algoritmu 36](#_Toc449455678)

[5.4 Vstup a výstup algoritmu 37](#_Toc449455679)

[6 Dosažené výsledky 40](#_Toc449455680)

[6.1 Testování 40](#_Toc449455681)

[6.2 Změna vstupních parametrů 40](#_Toc449455682)

[7 Závěr 42](#_Toc449455683)

# Úvod

Mnoho uživatelů internetu o ní neví, že existuje, ale využívají ji prakticky každý den. Řeč je o segmentaci webových stránek. Na webových stránkách je mnoho irelevantního obsahu, který je z hlediska běžného uživatele nepoužitelný a je tak potřeba tento obsah filtrovat. Segmentaci lze tedy užitečně využít. Například by ji mohli využít webové vyhledávače, pokud by se snížila její výpočetní náročnost. Velkým přínosem segmentace je hlavně její filtrace nevyžádaného obsahu na webové stránce a umožňující nám tak získat podstatné a nejdůležitější informace z této stránky.

Dřívější techniky segmentace webových stránek jsou založeny především na algoritmech strojového učení a na algoritmech využívající různá heuristická pravidla. Avšak vzhledem k malému rozsahu testovacích množin dat, metody strojového učení mohou být aplikovány pouze v některých určitých oblastech webové stránky. Zatímco metody založené na heuristických pravidlech mohou fungovat na malých sadách stránek, nemohou být využitelné na větších sadách stránek.

V této práci implementujeme jednu z metod segmentace webových stránek. Předpokládáme, že se webová stránka skládá z konečného počtu bloků a může být segmentována pomocí vizuálních vlastností těchto bloků. Metoda tedy využívá těchto vizuálních vlastností. Metoda bude implementována v rámci Frameworku FitLayout a bude do něj integrována.

Co se týče obsahu práce, ve druhé kapitole si popíšeme příklady tří metod pro segmentaci. Nejprve se stručně seznámíme s metodou DOM-based Page Segmentation, která realizuje segmentaci webové stránky na základě jejího DOM stromu. Dále se pak seznámíme z jednou nejznámějších metod pro segmentaci a tou je Vision-based Page Segmentation. Ta pro segmentaci využívá vizuálních vlastností prvků na webové stránce. Na závěr této kapitoly si řekneme ještě o metodě Box Clustering Page Segmentation.

Ve třetí kapitole se zaměříme na metodu pro segmentaci webových stránek, kterou se zabývá tato práce. Jedná se o metodu využívající vizuálních vlastností stránek. Zde se soustředíme především na teoretickou část této metody. Ukážeme si, jaké se zde využívají vzorce pro výpočty různých vlastností, a také pseudokód samotného algoritmu.

Čtvrtá kapitola je soustředěna na seznámení se s Frameworkem FitLayout. Metoda, která bude implementovat segmentaci webových stránek, importujeme do tohoto Frameworku. Taktéž bude využívat jeho již hotové části a je tedy zapotřebí se s některými nejprve obeznámit. Je vhodné, abychom si také ukázali, ovládání grafického uživatelského rozhraní Frameworku.

V páté kapitole se dostaneme v podstatě k tomu nejdůležitějšímu, co se v této práci můžeme dočíst. Řekneme si všechny detaily implementace našeho segmentačního algoritmu. Jsou zde rozebrána různá úskalí, se kterými se můžeme při implementaci setkat a taktéž, jak je vyřešit. Většina takových informací je řečena prostřednictvím obrázků, protože se jedná o nejrychlejší způsob vysvětlení.

Dostáváme se na předposlední kapitolu, jež rozebírá dosažené výsledky v této práci. Jedná se o různé testování na konkrétní sadě webových stránek. Najdeme zde několik ukázek výsledků segmentace a srovnání s jinými implementacemi segmentačních metod.

Na závěr si shrneme dosažené výsledky a splnění cílů, které tato diplomová práce měla. Dále si rozebereme možnosti, jakými by se dala tato práce rozšiřovat, či pokud je to vůbec možné, tuto práci nějak vylepšit.

# Segmentace webových stránek

Segmentace webových stránek umožňuje dělení stránky podle určitých pravidel nebo vizuálních vlastností. Stránka je rozdělena na několik sémanticky rozdílných bloků a jejich obsah můžeme dále zkoumat nebo různě filtrovat podle potřeby.

Jak jsme již řekli, segmentace je slibná metoda, která by mohla být využívána u webových vyhledávačů. Slouží ke zpřesnění nalezených výsledků pro vyhledávaný řetězec a dokáže taktéž identifikovat jeho polohu v rámci webové stránky, na které byl nalezen. Můžeme tak rozpoznat a vynechat výskyty vyhledávaného řetězce. Jako příklad použití bychom mohli říct například v komentářích pod článkem nebo v odkazech sloužících k navigaci po stránce.

Existuje mnoho metod pro segmentaci webových stránek a v následujících podkapitolách si stručně charakterizujeme příklady tří z nich. Metoda, kterou se zabývá tato práce, bude popsána v samostatné kapitole.

## DOM-based Page Segmentation

Jednou z metod segmentace je metoda DOM-based Page Segmentation (dále DomPS) [12]. Ta realizuje segmentaci webové stránky na základě jejího DOM[[1]](#footnote-1) stromu se značkami jazyka HTML. Příklad DOM stromu je zobrazen na obrázku 2.2. Segmentovanými bloky jsou v případě této metody části uvozené v HTML značkách. Příkladem těchto značek je <TITLE>, <H1> až <H6>, <META>, <TABLE> atd. Tato metoda však má několik problému, se kterými se musí potýkat:

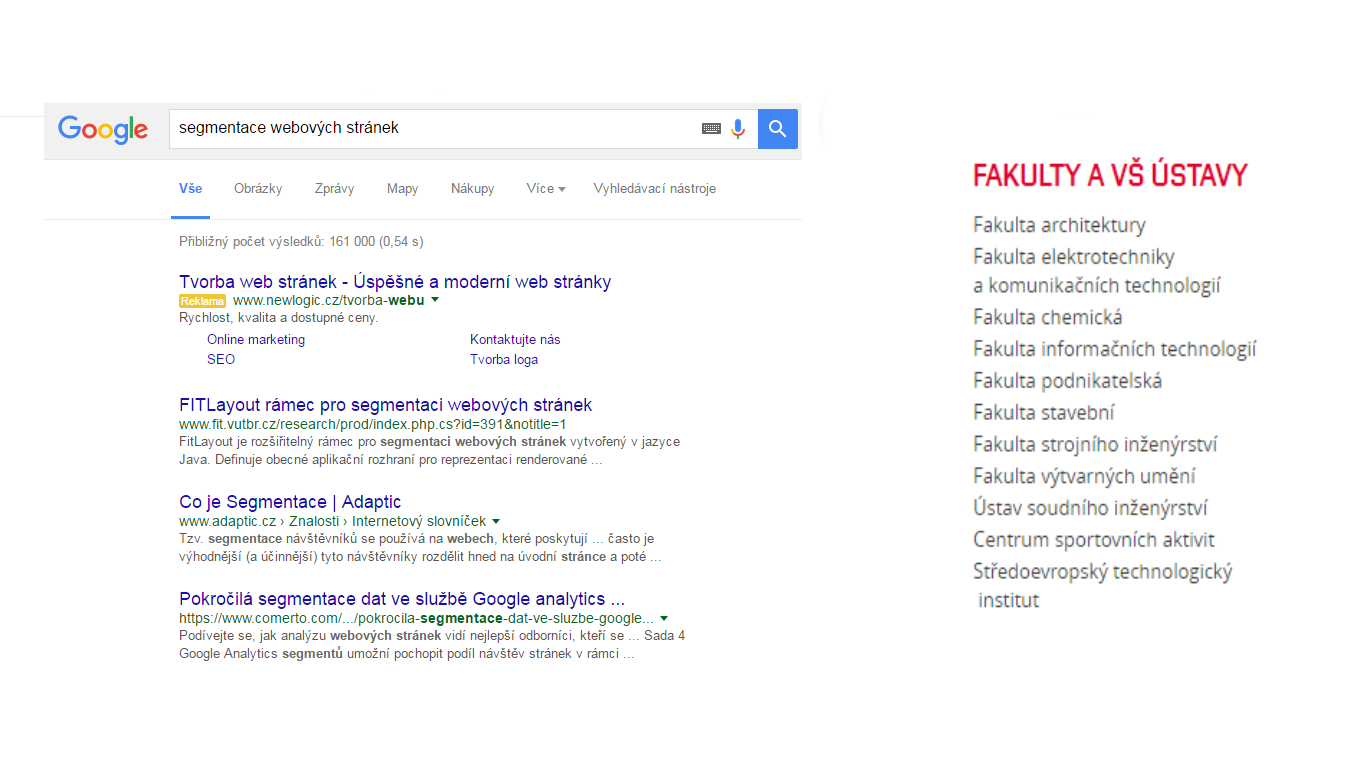
* Značky jako <TABLE> a <P> slouží nejen k prezentaci obsahu, ale také pro rozložení struktury. Je tedy obtížné zajistit vhodnou úroveň segmentace.
* Mnoho webových stránek nedodržuje specifikace HTML a ztěžují tak správnou segmentaci.

Příkladem algoritmu metody DomPS je například algoritmus WISH [14]. Algoritmus zpracuje DOM strom webové stránky a extrahuje požadovaný obsah. Jeho specifikem jsou speciální struktury, které se nazývají *datové záznamy*. Jedná se o části webové stránky, jež se opakují, avšak pokaždé s jiným obsahem. Datové záznamy jsou definovány jako elementy, které se nacházejí na stejné úrovni v DOM stromu. Dále také musí obsahovat sekvence opakujících se potomků a musí mít podobného předka. Předek se nazývá *datová oblast*. Na obrázku 2.1 můžeme vidět příklady datových záznamů. Jak vidíme, jsou to například výsledky po vyhledávání nebo menu s položkami.

WISH algoritmus nejdříve extrahuje uzly DOM stromu a hledá potenciální datové záznamy. Hledání vykonává pomocí postupného procházení DOM stromu po jednotlivých úrovních. Jestliže není na dané úrovni nalezen datový záznam, hledání se přesune o úroveň níže. Jakmile je hotová detekce všech potenciálních kandidátů na datové záznamy, zahájí se fáze filtrování. Ta využívá několik pravidel pro konečné rozhodnutí, zda se jedná o datový záznam či nikoliv. Pravidla jsou následující:

* Datové záznamy mají velké rozměry v porovnání s rozměry celé stránky.
* Datové záznamy se opakují více jak třikrát na celé stránce.
* Z popisu datového záznamu můžeme utvořit vzor, který můžeme aplikovat na ostatní záznamy.
* Záznamy se obvykle skládají z nízkého počtu HTML značek.

Po dokončení filtrace potenciálních datových oblastí je všem datovým oblastem přiřazena číselná hodnota. Tu vypočítáme na základě obsahu datových záznamů uvnitř těchto oblastí. Do výpočtu jsou taktéž zahrnuty znaky textu, obrázky atd. Datová oblast, jejíž vypočítaná hodnota je největší, označíme za hlavní obsah stránky.



Obrázek 2.1: Ukázka datových záznamů algoritmu WISH. Vlevo je seznam po vyhledání, vpravo seznam menu.



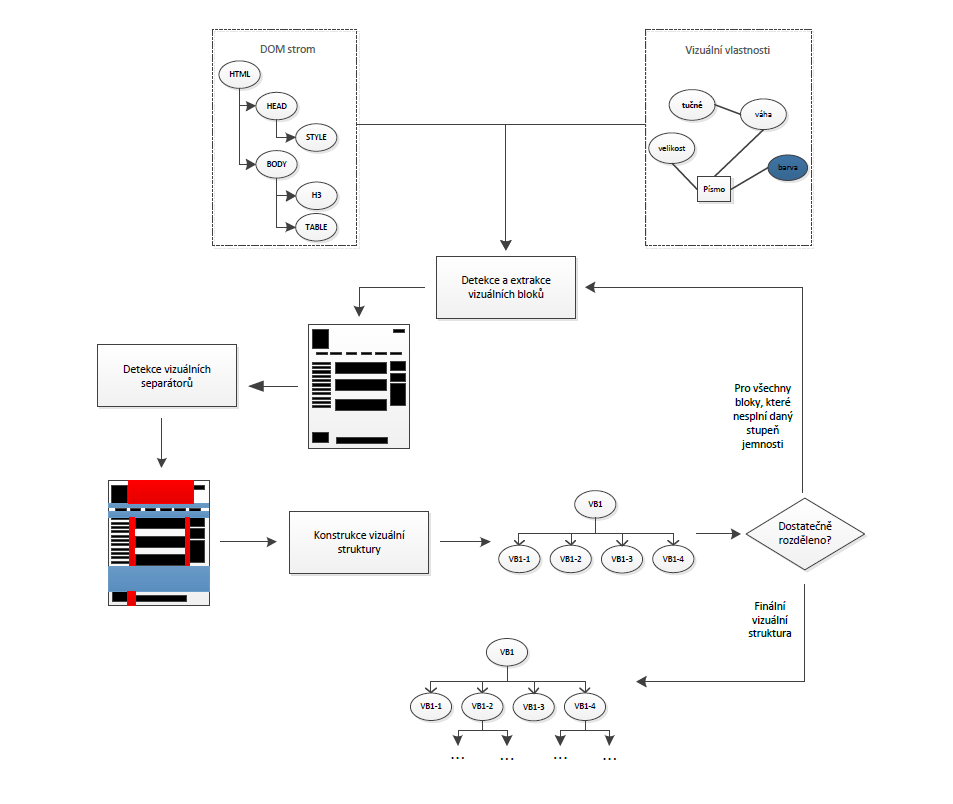
Obrázek 2.1: Příklad DOM stromu

## Vision-based Page Segmentation – VIPS

Webová stránka obsahuje mnoho vizuálních podnětů, které pomáhají člověku rozeznávat, pomocí webového prohlížeče, jednotlivé stránky. Stránka obsahuje několik dílčích vizuálních prvků, jako jsou obrázky, barvy, čáry a jiné značky. Metoda využívající vlastnosti vizuálních prvků webové stránky je Vision-based Page Segmentation neboli VIPS. Podrobnější popis algoritmu najdeme zde [5].

Jak tedy VIPS stručně funguje. Nejprve je získán DOM strom z webového prohlížeče. Jakmile je strom získán, začíná se od kořenového uzlu. Probíhá proces extrakce vizuálního uzlu na vizuální uzly dané úrovně z DOM stromu, podle vizuálních podnětů. Každý uzel DOM stromu je zkontrolován a hodnocen, zda tvoří jednoduchý blok nebo ne. Pokud není jednoduchým blokem, synovský uzel je zkontrolován a hodnocen stejně jako rodič. Jakmile jsou extrahovány všechny bloky dané úrovně, vloží se do dané ukládací struktury. Dále pak těmto blokům identifikujeme vizuální separátory. Poté, je vytvořena struktura rozložení dané úrovně. Každý nově vytvářený vizuální blok je kontrolován, zda vyhovuje požadavku granularity. Pokud ne, je dále dělen. Jakmile jsou zkontrolovány všechny bloky, výsledná datová struktura tohoto algoritmu je dána na výstup. Algoritmus VIPS lze vidět na obrázku 2.3 a skládá se z tří částí:

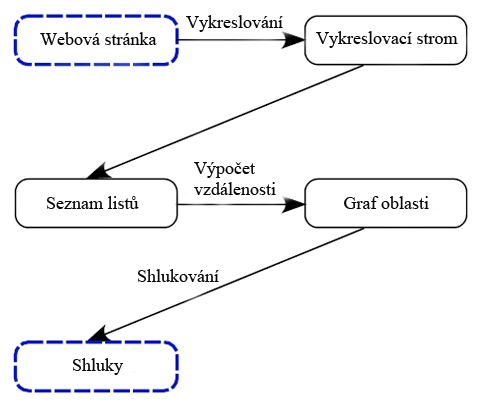
* Extrakce vizuálních bloků – v této části jsou především hledány všechny vizuální bloky nacházející se podstromu. Dalo by se říct, že každý uzel v DOM stromu může reprezentovat vizuální blok.
* Detekce vizuálních separátorů – zde se detekují vizuální separátory. Jsou buďto horizontální nebo vertikální. Pomocí nich dokážeme rozpoznat rozdílné vizuální vlastnosti v rámci webové stránky.
* Konstrukce struktury obsahu – v poslední fázi se konstruuje výsledná struktura, která je vrácena na výstup. Konstrukce začíná od separátoru s nejnižší váhou a opakuje se až po separátory s nejvyšší váhou. Na závěr tak máme naplněnou strukturu s vizuálními bloky.



Obrázek 2.2: Postup segmentace metody VIPS. Obrázek převzat z [6].

## Box Clustering Segmentation

Poslední příkladem algoritmu, který zde uvedeme, je Box Clustering Segmentation [7]. Tato metoda je založená čistě na vzhledu stránky. Když ji porovnáme s ostatními metodami, pak výsledek není v žádné hierarchické struktuře, ale spíše jsou uspořádány ve struktuře podobné dlaždicím.



Obrázek 2.3: Architektura algoritmu segmentace Box Clustering. Obrázek převzat z [7].

Celou architekturu algoritmu můžeme vidět na obrázku 2.3. Každý box na obrázku reprezentuje stav zpracovávaných dat. Každý přechod mezi dvěma stavy reprezentuje akci, která je zrovna prováděna. První box označený jako *Webová stránka* reprezentuje vstup celého algoritmu. Vykreslování je prováděno mimo algoritmus a můžet být pro to použit libovolný způsob vykreslování. Výsledkem vykreslovacího procesu je *Vykreslovací strom.*

Samotný algoritmus je uskutečněn pomocí tří kroků:

* Extrakce boxu – v tomto kroku se vezme vykreslený strom a odfiltrují se všechny jeho část, které nejsou užitečné pro následující shlukování.
* Výpočet vzdálenosti – vzdálenost mezi zbývajícími boxy je spočítána v druhém kroku a je založena na dalších kritériích popsaných v [7]. V tomto kroku je vytvořen neuspořádaný vážený graf.
* Shlukování – v posledním kroku je zpracováván graf získaný v předešlém kroku a identifikují se segmenty webové stránky. Shlukují se tak boxy patřící do stejného segmentu.

# Segmentace webových stránek za pomocí vizuální sémantiky

Tato práce je zaměřená na segmentaci webových za pomocí algoritmu používajícího vizuálních vlastností jednotlivých elementů na webu [8]. Je tedy důležité se s tímto algoritmem podrobně seznámit. V této kapitole se zaměříme na podrobnější popis již zmíněné metody.

Budeme předpokládat, že webová stránka se skládá z konečného počtu bloků a je možné ji za pomocí vizuálních vlastností, těchto bloků, rozdělit. Vizuální vlastnosti můžeme rozdělit do tří částí:

* Podobné vizuální bloky mají podobné vizuální vlastnosti (např. na stránce internetového obchodu záznamy produktů jsou uspořádány podobným rozvržením)
* Příslušné bloky jsou vždy přehledně uspořádány a to vizuálně blízko sebe
* Bloky s rozdílnou funkcionalitou obsahují jiné typy obsahu (např. na zpravodajském webu, dlouhý text může obsahovat hlavním obsah, velký obrázek může být inzerát atd.)

I když jsou vizuální vlastnosti rozdílné, tak je člověk schopen identifikovat každý segment bez jakéhokoli popisu. Vizuální vlastnosti zde budeme nazývat vizuální sémantikou. Sémantika je tedy intuitivní a pro člověka přátelská. Pro stroj ale není. Počítač obtížně rozpoznává vizuální sémantiku a je zapotřebí ji formulovat tak, aby ji chápal správně. Musíme tak formulovat opatření pro reprezentaci vizuální sémantiky:

* Layout tree (strom rozvržení) – je používán k rozpoznání podobných vizuálních bloků.
* Seam degree (stupeň sousednosti) – používá se k popisu, jak přehledně jsou bloky uspořádány.
* Content similarity (podobnost obsahu) – je používán k popisu obsahu stupně soudržnosti mezi bloky.

## Příprava

Než se pustíme do samotného algoritmu je potřeba se seznámit s různými vlastnostmi a připravit si webové stránky pro aplikaci algoritmu.

### Vizuální bloky

Webová stránka se skládá z konečného počtu bloků. Můžeme zde tyto bloky nazývat vizuální bloky nazývat nebo pouze bloky pro zkrácení. Uvažujme, že vizuální bloky jsou viditelné obdélníkové oblasti na webové stránce. Potom můžeme definovat vizuální blok následovně:

**Definice 3-1:** Vizuální blok , kde je DOM objekt a reprezentuje viditelnou obdélníkovou oblast, kde je zobrazen na webové stránce.

Podle W3C[[2]](#footnote-2) standardu, webová stránka může být transformována do DOM stromu a každý DOM objekt má odpovídající elementy na této stránce. Pokud je element viditelný, je zobrazen uvnitř obdélníkové oblasti na webové stránce. Proto můžeme říct, že DOM objekt a jeho obdélníková oblast reprezentují blok na webové stránce. Nicméně je třeba definovat synovský blok a koncový blok následovně:

**Definice 3-2:** Mějme dva vizuální bloky a , pokud je synovský uzel , tak je synovský blok bloku .

**Definice 3-3:** Pokud vizuální blok nemá žádný synovský blok, pak se jedná o blok koncový.

### Předzpracování webových stránek

Podle Definice 3-1, každý vizuální blok má odpovídající DOM objekt. Tudíž prvním krokem našeho algoritmu je získání DOM stromu z webové stránky.

V DOM strom je každý uzel DOM objekt. DOM objekt může být rozdělen na pět typů: element, text, atribut, komentář a dokument. Elementy můžeme dále rozdělit na dvě kategorie a to na viditelné a neviditelné elementy. Viditelné elementy mají dvě vlastnosti, výšku a šířku. Pokud je šířka a výška nenulová, lze tak element zobrazit pomocí prohlížeče. Neviditelné elementy obsahují objekty, jejichž tagy jsou např. <head>, <script>, <meta> atd., které nemají vizuální vlastnosti. Nicméně nás zajímají především ty viditelné, které si můžeme rozdělit do dvou kategorií. Těmi jsou řádkové objekty a více řádkové objekty. Řádkové objekty ovlivňují vzhled textu a mohou být použity na řetězce znaků, které nejsou odřádkovány a zahrnují objekty, jejichž tagy jsou <b>, <big>, <font>, atd. DOM strom však obsahuje spoustu objektů, které jsou v rámci segmentace zanedbatelné. Je tak vhodné jej o tyto objekty oříznout. K tomu nám slouží následujících 5 pravidel:

* Atributy uzlů, komentáře k uzlům, uzly dokumentu by měly být vypuštěn.
* Neviditelné uzly, jejichž tagy jsou <head>, <script>, <meta>, atd. by měly být vypuštěny.
* Viditelné uzly, jejichž šířka a výška je nulová a nejsou tak zobrazeny v prohlížeči, by měly být vypuštěny.
* Pokud uzel obsahuje pouze jeden uzel, jehož uzlové jméno je <#text>, potom <#text> by měl být vypuštěn.
* Pokud jediný obsah uzlu jsou <#text> uzly, řádkové uzly a každý řádkový uzel má pouze jeden <#text> uzel, potom všechny <#text> uzly a řádkové uzly jsou zanedbatelné.

Jakmile je DOM strom ořezán, jeho zbývajícím obsahem jsou viditelné a textové uzly. Poznamenejme, že oba elementární uzly a textové uzly mají odpovídající objekty v DOM stromu. Podle Definice 3-1 má každý vizuální blok odpovídající DOM elementární objekt a obdélníkovou oblast. Tudíž je důležité znát obdélníkovou oblast každého elementárního objektu. Elementární objekt neobsahuje absolutní souřadnice odpovídající HTML elementu, ale pouze relativní souřadnice rodičovského HTML elementu. Naštěstí většina prohlížečů nabízí API k získání absolutních souřadnic a lze tak souřadnice získat jednoduše. Zaměříme-li se na textové uzly, tak jejich šířka a výška může být vypočítána analýzou výšky a šířky jejich rodičovských a příbuzných uzlů. Jakmile jsou určeny obdélníkové oblasti, jsou určeny i odpovídající vizuální bloky.

## Algoritmus segmentace

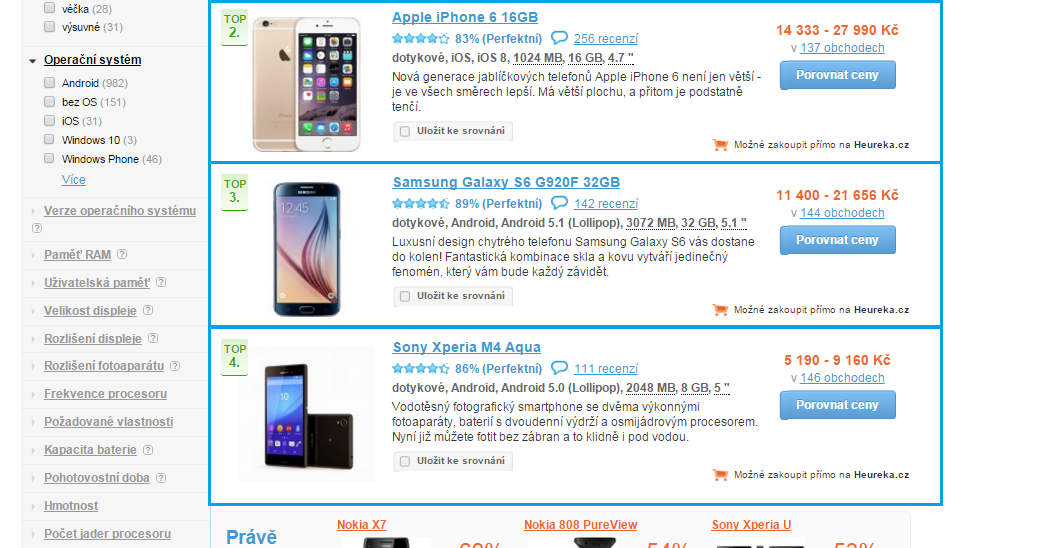
Po pečlivé přípravě se dostáváme k samotnému algoritmu segmentace s využitím vizuálních bloků. Jakmile webovou stránku v ořezaném DOM stromu předáme jej na vstup algoritmu. Ten se skládá z několika části konkrétně ze čtyř, se kterýma se dále lépe seznámíme.

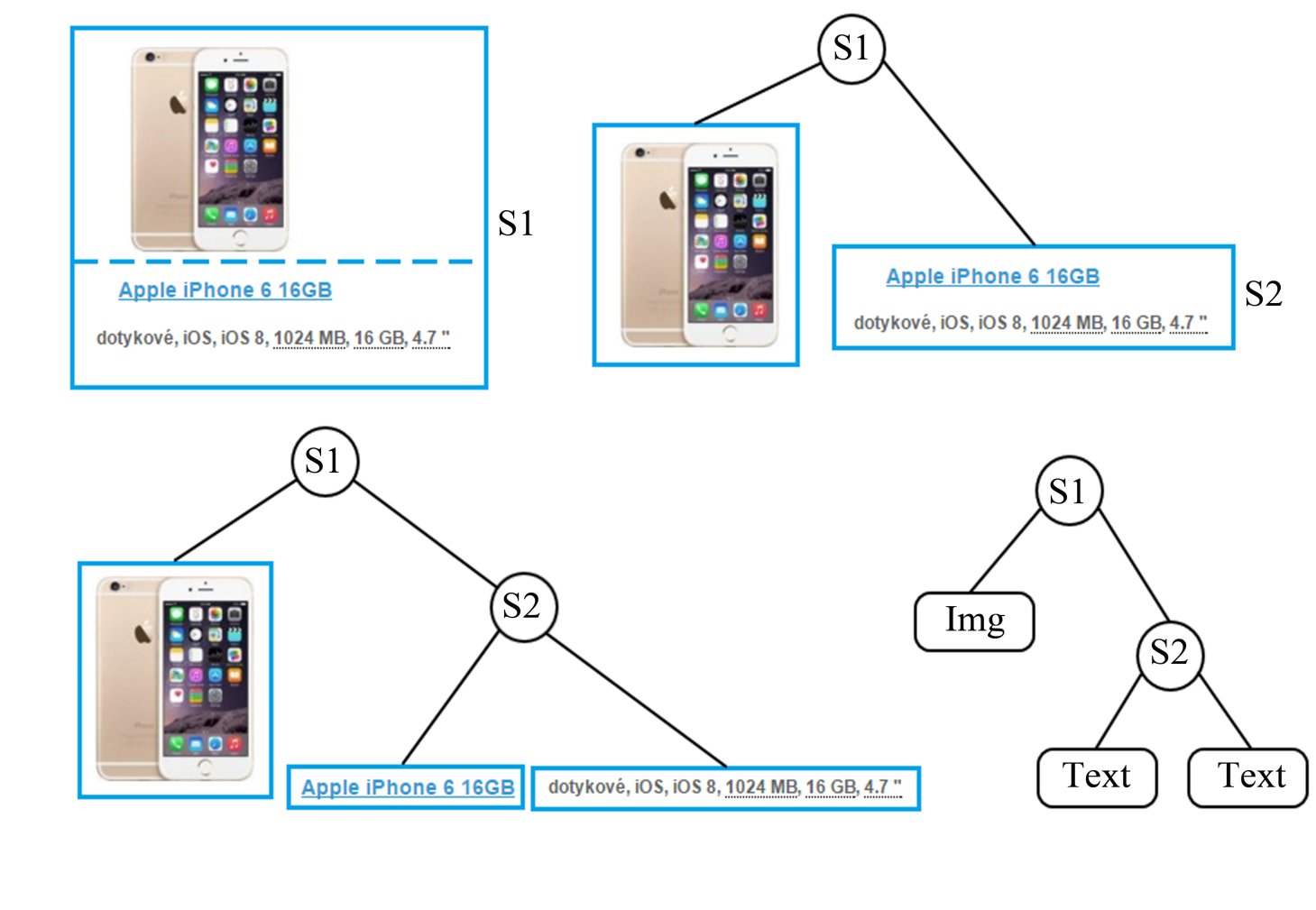
### Rozpoznání podobných vizuálních bloků

Některé webové stránky obsahují podobné vizuální bloky. Např. na obrázku 3.1, se nachází webová stránka e-shopu a modré obdélníky naznačují bloky s vizuálními vlastnostmi. Každý blok je záznam produktu s jeho krátkým popisem a rozmezím udávající jeho cenu. Můžeme tedy říct, že vyznačené bloky mají podobné vizuální vlastnosti a neměli bychom je dále dělit. Proto bychom se měli zaměřit na hledání takovýchto bloků.

Metoda k identifikování podobných vizuálních bloků a získání tak stromu rozložení se nachází v článku [9]. Každý blok, který není koncovým blokem, můžeme transformovat do stromu rozložení. Příklad takové transformace je na obrázku 3.2.

Na obrázku se nacházejí dva separátory a . Každý separátor může rozdělit blok na dvě menší části. Separátory mohou považovány za kořeny strom a dvě menší části mohou být považovány jako pravý a levý podstrom. Obvykle, pokud je separátor horizontální, pak horní část je levý podstrom a nižší část je pravý strom. Pokud je separátor vertikální, pak je levá část levý podstrom a pravá část pravý podstrom. Tudíž můžeme daný blok transformovat na strom.



Obrázek 3.1: Příklad podobných vizuálních bloků.

Obrázek 3.2: Ukázka vytváření stromu rozložení.

Pro dva dané bloky, které nejprve transformujeme na stromy rozložení, můžeme vypočítat jejich podobnost. K tomu slouží algoritmu TED (Tree Edit Distance), který se nachází v článku [10]. Pokud je podobnost menší než nastavený práh, potom jsou dva bloky označeny jako vizuálně podobné. Podle článku [8] je optimální práh . Za pomocí této metody mohou být rozpoznány všechny podobné vizuální bloky.

### Výpočet stupně sousednosti bloků

Bloky jsou viditelné obdélníky nacházející se na webové stránky, a proto jsou vždy uspořádány podle nějakých pravidel. Relevantní bloky jsou vždy přehledně uspořádány vizuálně blízko sebe. Pro každé dva bloky může být jejich uspořádání klasifikováno do tří typů:

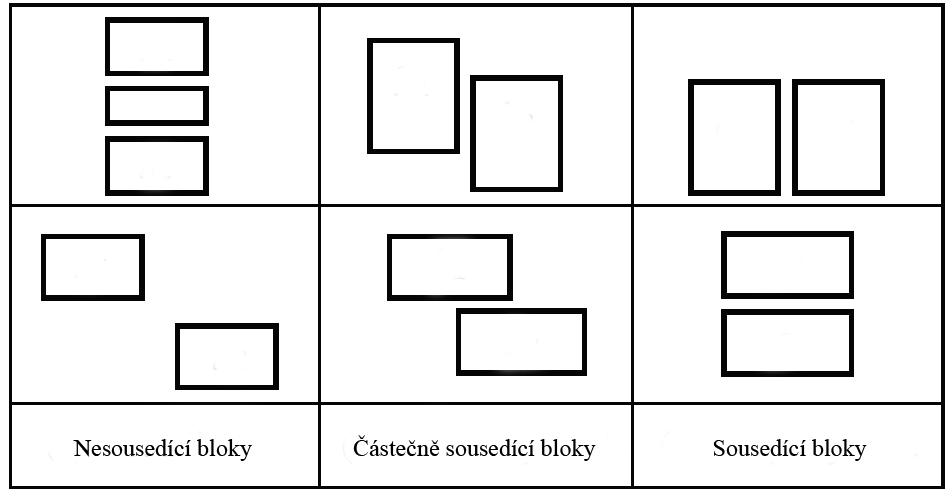
* Nesousedící – tyto bloky považujeme za vizuálně nerelevantní
* Sousedící - sousedností tvoří obdélník
* Sousedící – jejich sousednost tvoří minimální obdélník

Příklad těchto vlastností lze vidět na obrázku 3.3. Zaměříme-li se na vlastnosti jednotlivých sousedností, tak sousedící dva bloky mohou tvořit segment. Nejpravděpodobnější volbou jsou však především bloky, které spolu tvoří minimální obdélník. Jak blízko jsou dva bloky vedle sebe sousedící lze vypočítat a tuto hodnotu budeme nazývat stupeň sousednosti (seam degree). Pro dva sousední bloky a , pokud vedle sebe sousedí nahoře nebo dole, tak můžeme vypočítat jejich stupeň sousednosti následovně:

|  |  |  |
| --- | --- | --- |
|  |  | (1.) |

Kde reprezentuje délku sousednosti a , a udává šířku . Podobně pokud jsou dva bloky a sousední zleva nebo zprava, stupeň sousednosti může být vypočítán:

|  |  |  |
| --- | --- | --- |
|  |  | (2.) |



Obrázek 3.3: Ukázka sousednosti bloků. Obrázek převzat z [8].

Kde je výška . je mezi 0 a 1. Jelikož je stupeň sousednosti založen na vizuální informaci bloků, může indikovat vizuální koherentní stupeň odsazených bloků.

Pokud blok má také synovský blok, průměrný stupeň sousednosti navzájem sousedních synovských bloků může indikovat vizuální koherentní stupeň synovských bloků daného bloku. Pro daný vizuální blok množina synovských bloků bloku je . Pokud jsou dva synovské bloky a sousedy, tak se počítají jako jeden pár. Řekněme, že máme párů navzájem odsazených synovských bloků. Průměrný stupeň sousednosti se vypočítá:

|  |  |  |
| --- | --- | --- |
|  |  | (3.) |

Kde a jsou dva navzájem sousední bloky. je taktéž mezi 0 a 1. Pokud je výsledek blíže 0, potom vizuální koherentní stupeň synovských bloků je nižší. V opačném případě, pokud je výsledek blíže 1, je vizuální koherentní stupeň synovských bloků vyšší.

### Výpočet podobnosti obsahu bloků

Bloky s různou sémantikou vždy mají rozdílné typy obsahu. Tyto rozdílné obsahy mají také rozdílné vizuální vlastnosti. Pokud je obsah dvou bloků podobný, potom tyto dva bloky mají vysoký obsahový koherentní stupeň. Teď uvedeme podobnost obsahu k popisu tohoto obsahového koherentního stupně. Stručně charakterizujme obsahy do čtyř skupin:

* Textový obsah (zkráceně TO) – do této kategorie spadá všechen text, kromě textu, který obsahuje hypertextový odkaz.
* Odkazový textový obsah (zkráceně OTO) – tato kategorie obsahuje text, který obsahuje hypertextový text.
* Obrázkový obsah (zkráceně OO) – do této kategorie patří obrázky, fotky, ikony atd.
* Vstupní obsah (zkráceně VO) – tato kategorie zahrnuje elementy, které mohou přijímat vstup, jako jsou: textové boxy, rádiové tlačítko, vysouvací menu atd.

Pro daný blok , je obsah množina . Zaprvé, obsahy klasifikujeme do čtyř kategorií, jenž, byly popsány výše. Poté můžeme získat obsahy těchto čtyř množin, budeme je psát zkratkami TO, OTO, OO a VO. Jak je zjevně tyto množiny jsou podmnožiny množiny . Pokud je jedna z podmnožin , znamená to, že neobsahuje odpovídající typ obsahu. Obsahy jsou taktéž elementy na webové stránce a každý z něj má tím pádem odpovídající blok. Použijeme pro reprezentaci oblasti odpovídajícího bloku obsahu . Pokud je textový nebo odkazový textový obsah, aproximací vypočítáme oblast pomocí:

|  |  |  |
| --- | --- | --- |
|  |  | (4.) |

Kde reprezentuje byte textu, reprezentuje velikost fontu daného textu.

Pro danou obsahovou podmnožinu, v závislosti na oblasti obsahů, obsah dané podmnožiny může být seřazen od velké po menší oblast. Využitím seřazeného obsahu podmnožin, čtyři obsahové oblastní vektory mohou být získány a značí se , , , . Hodnoty elementu v těchto čtyřech vektorech jsou oblasti odpovídající obsahům. Poté co jsou vektory určeny, podobnost obsahu dvou bloků může být vypočítán.

Pokud jsou obsahové oblastní vektory dvou daných bloků určeny, podobnost každého obsahového oblastního vektoru může být vypočítán. Existuje mnoho algoritmů, pro vypočítání podobnosti dvou vektorů. Zde využijeme jednoduchého algoritmu a to kosínové podobnosti [11]. Vezmeme vektor textového obsahu jako příklad pro výpočet kosínové podobnosti. Mějme dva dané bloky a , jejich textové obsahové oblastní vektory jsou a . Řekněme, že , a . Protože kosínová podobnost vyžaduje, že oba vektory musí mít stejný počet elementů, proto potřebujeme dát do elementů, jejichž hodnota bude 0. Takto nově vytvořený vektor popisujeme . Kosínová podobnost a může být vypočítána pomocí:

|  |  |  |
| --- | --- | --- |
|  |  | (5.) |

Pokud oba a , jsou , je nepoužitelný. V tomto případě, definujeme jako nulový. Podobně vypočítáme podobnost obsahu pro ostatní vektory.

Dodatečně, čtyři typy obsahů mohou mít rozdílné váhy a . Zde také můžeme vzít jako příklad textový obsah a vysvětlit na něm výpočet váhy. Pro dané dva bloky a , jejichž textové obsahové oblastní vektory jsou a . Váha textového obsahu může být vypočítána vzorcem:

|  |  |  |
| --- | --- | --- |
|  |  | (6.) |

Kde reprezentuje totální oblast všech obsahů v . Znamená to, že čím větší oblast odpovídajícího obsahu tím větší váhu blok bude mít.

Poté co máme vypočítané kosínovou podobnost a váhu všech obsahových vektorů oblastí, můžeme vypočítat podobnost obsahu pro a následovně:

|  |  |  |
| --- | --- | --- |
|  |  | (7.) |

Kde reprezentuje váhu čtyř typů obsahu a reprezentuje kosínovou podobnost odpovídající obsahovému vektoru oblasti. je mezi 0 a 1. Protože podobnost obsahu je založena na informaci obsažené v bloku, může indikovat obsahový koherentní stupeň bloku.

Pokud blok má synovský blok, průměrná podobnost obsahu sousedních synovských bloků může indikovat obsahový koherentní stupeň synovských bloků v bloku. Mělo být poznamenáno, že pouze uvažujeme podobnost obsahu sousedních synovských bloků. Pro daný vizuální blok je množina jeho synovských bloků . Pokud dva synovské bloky a jsou soused, počítáme je jako 1 pár. Předpokládejme, že máme párů sousedních synovských bloků. Průměrná podobnost obsahu může být vypočítána pomocí:

|  |  |  |
| --- | --- | --- |
|  |  | (8.) |

Kde a jsou sousední synovské bloky. Jako předchozí výpočty i je výsledek mezi 0 a 1. Pokud je blíže 0, je koherentní stupeň obsahu synovského bloku nižší. V opačném případě pokud je blíže 1, je vyšší.

### Segment webové stránky

Poté co získáme ořezaný DOM strom, můžeme rozpoznat podobné vizuální bloky a stupeň sousednosti, podobnost obsahu může být dále stanoveny. Zde uvedeme prahovou hodnotu a prahovou hodnotu . Empiricky stanovíme, že je 0.9 a 0.8. Naše metoda pro segmentaci webové stránky je shora dolů. Začíná tedy kořenovým uzlem DOM strom, který je nastaven na aktuální právě zpracovávaný uzel. Odpovídající blok aktuálního uzlu bude hodnocen podle kroků uvedených v tabulce 3.1. Pokud by se měl aktuální uzel dělit, potom jeho synovské bloky budou hodnoceny stejným způsobem. Pokud se aktuální uzel dělit nemá, je vložen do seznamu segmentů a jeho synovské bloky se již dále neuvažují. Pseudokód můžeme vidět níže na algoritmu 3.1.

|  |  |
| --- | --- |
| Krok 1 | Pokud je daný blok blokem koncovým tak jej nadále neděl. Jinak pokračuj krokem 2. |
| Krok 2 | Pokud daný blok obsahuje opakující se bloky, potom jej rozděl. Jinak pokračuj krokem 3. |
| Krok 3 | Pokud je daný blok jedním z opakujících se bloků, potom jej dále neděl. Jinak pokračuj krokem 4. |
| Krok 4 | Pokud daný blok obsahuje pouze jediný synovský blok, potom jej rozděl. Jinak pokračuj krokem 5. |
| Krok 5 | Pokud daného bloku je menší než , potom jej rozděl. Jinak pokračuj krokem 6. |
| Krok 6 | Pokud daného bloku je menší než , potom jej rozděl. Jinak pokračuj krokem 7. |
| Krok 7 | Pokud je oblast daného bloku větší než polovina klientské oblasti, potom jej rozděl. Jinak pokračuj krokem 8. |
| Krok 8 | Pokud daný blok nepatří do žádné z předešlých podmínek, potom ho neděl. |

Tabulka 3.: Kroky pro vyhodnocování bloků.

Algoritmus 3.1: Pseudokód webové segmentace pomocí vizuálních vlastností.

# FitLayout

Základním stavebním kamenem této práce je dozajista Framework FitLayout[[3]](#footnote-3). Jedná se o jednoduše rozšiřitelný Framework pro analýzu webových stránek. Je psán v jazyce Java a definuje obecné Java API pro reprezentaci renderované webové stránky a její rozdělení na vizuální oblasti. Dále pak poskytuje základ pro implementaci segmentačních algoritmů se společným aplikačním rozhraní.

Jako vzorová segmentační metoda je zde metoda založená na rekurzivním slučování vizuálních oblastí a detekování separátoru. Framework také zahrnuje nástroje pro zpracování výsledků segmentace různými texty nebo vizuálními klasifikačními metodami. V neposlední řadě také poskytuje nástroje pro řízení segmentace a zkoumání jejích výsledků prostřednictvím grafického uživatelského rozhraní.

## FitLayout Moduly

Framework se skládá z několika modulů, které jsou navzájem propojeny za pomocí API. Dále pak také poskytuje přidání dalších modulů či rozšíření stávajících. Za další, může být defaultní modul CSSBox, což je renderovací nástroj, nahrazen jinou hlavní implementací.

### FitLayout API

API je založeno na ontologickém popisu zpracovávané stránky jako publikované v [1]. Související ontologie jsou popsány na obrázku 4.1. Pro každou třídu ontologie API definuje rozhraní v jazyce Java s podobnými vlastnostmi. Tyto rozhraní jsou dostupné v balíku org.fit.layout.model.

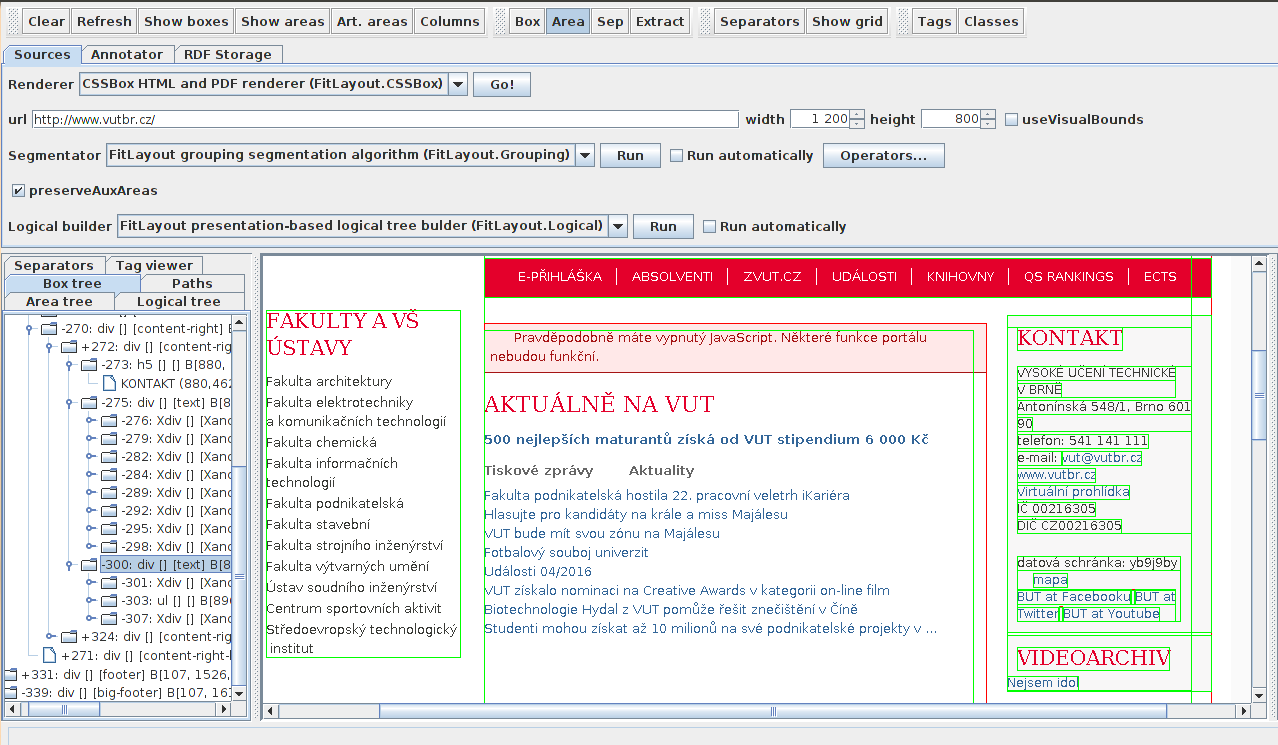
### CSSBox

Obrázek 4.1: API ontologie A) box ontologie B) ontologie vizuálních oblastí C) tagování. Obrázek převzat z [13].

CSSBox [15] je použit jako výchozí renderovací nástroj. Jedná se o (X)HTML/CSS renderovací nástroj napsaný v jazyce Java. Hlavním účelem tohoto nástroje je poskytnout kompletní a další zpracovatelné informace o poskytovaném obsahu rozložení webové stránky. Nicméně CSSBox se dá taktéž využít, pro prohlížení vyrenderovaných dokumentů v Java Swing aplikacích[[4]](#footnote-4).

Vstupem tohoto nástroje je DOM strom webové stránky a sada stylů odkazovaných z této stránky. Výstupem je objektově orientovaný model rozložení stránky. Tento model může být přímo zobrazen, avšak hlavně je vhodný k dalšímu zpracování a k různým analýzám, jako jsou segmentační algoritmy. V našem případě se výstup nazývá box strom (dále *BoxTree*). Ukázka takto vyrenderované stránky i s vyznačenými prvky *BoxTree* můžeme vidět na obrázku č. 4.2.

Jádro knihovny CSSBoxu může být taktéž použito pro získání bitmapy nebo vektoru obrázku renderované stránky. Při použití v kombinaci s Java Swing aplikací, může být CSSBox použit jako interaktivní součást webového prohlížeče.



Obrázek 4.2: Ukázka vyrenderované stránky ve FitLayoutu z označenými boxy z BoxTree.

### Segmentace

Modul segmentace poskytuje obecný framework pro implementaci algoritmů segmentace webových stránek. Představuje stránku, na které je strom obsahující zjištěné vizuální oblasti. Další důležitou částí, kterou segmentaci poskytuje je definování rozhraní pro implementaci metod zpracování libovolných stromů. Jedná se o tzv. operátory, které implementují skutečnou segmentaci.

Výchozí algoritmus segmentace je založen na [2]. Používá přístup zdola nahoru a slučuje jednotlivé boxy do větších vizuálních oblastí. Avšak Framework FitLayout je navržen a implementován tak, že je snadno rozšiřitelný o další algoritmy pro segmentaci. Výsledkem segmentace stránky je strom vizuálních oblastí.

Při provádění segmentace budeme využívat několik struktur pro správnou funkčnost algoritmu. Tyto struktury jsou již implementovány ve Frameworku. První využívanou strukturou je *Box.* Struktura *Box* reprezentuje box z vyrenderované webové stránky. Konkrétně se jedná o prvek z *BoxTree*. Box může obsahovat například text nebo objekt s obsahem (což mohou být například obrázky). Zmínili jsme zde *BoxTree*, který reprezentuje celou vyrenderovanou vstupní webovou stránku.

Další důležitou strukturou je vizuální oblast (*Area*). Protože náš algoritmus se zabývá vizuálními prvky na webové stránce, je pro nás důležitá *Area.* Ta obsahuje několik vizuálních boxů. Může však obsahovat pouze jeden vizuální box. Tato struktura je důležitá především proto, protože na vstup našeho algoritmu vstupuje strom oblastí (dále *AreaTree*). A tento strom se skládá právě z vizuálních oblastí. *AreaTree*  získáme z *BoxTree* a budeme jej používat, jak již bylo řečeno, jako vstupní prvek našeho algoritmu. (možná ještě lehce rozepsat víc)

### Klasifikace

Jednotlivé oblasti zjištěné při segmentaci stránky mohou být označeny libovolnými značkami. Modul klasifikace implementuje značkování oblastí podle klasifikace jejich vizuálních vlastností, jak je navrženo v [3] nebo dle textových vlastností jako v [4].

### Služby

Architektura Frameworku FitLayout je jednoduše rozšiřitelná vytvářením nových pluginů jako jsou nový kód pro *BoxTree* (render dokumentů), segmentační algoritmy, přidání nových operátorů pro zpracování *AreaTree* nebo rozšíření grafického rozhraní. Můžeme se říci o typech služeb, které Framework obsahuje:

* BoxTreeProvider – Služba vytvářející *BoxTree*. Založen na vstupních parametrech, jako je webová stránka atd. Renderuje stránku na *BoxTree*.
* AreaTreeProvider – Služba vytvářející *AreaTree*. Veme *BoxTree* a transformuje jej na vizuální *AreaTree*, který reprezentuje vysegmentovanou stránku.
* AreaTreeOperator – Operace aplikované na vizuální *AreaTree*. Mohou být přidávány další.
* LogicalTreeOperator – Analyzátor, který přijme výsledný *AreaTree* a přiřadí vybraným oblastem sémantiku.

Každá služba je identifikována za pomocí unikátního identifikátoru poskytovaného metodou getId(). Všechny služby mohou přijímat několik vstupních parametrů. Implementují rozhraní ParametrizedOperation, jenž dovoluje získat informaci o vyžadovaných vstupních parametrech (jejich jméno a typ) a přiřazuje jim hodnoty.

Pro přístup k službám, FitLayout poskytuje jednoduchý ServiceManager, který obsahuje statické metody pro lokalizování služeb daného typu.

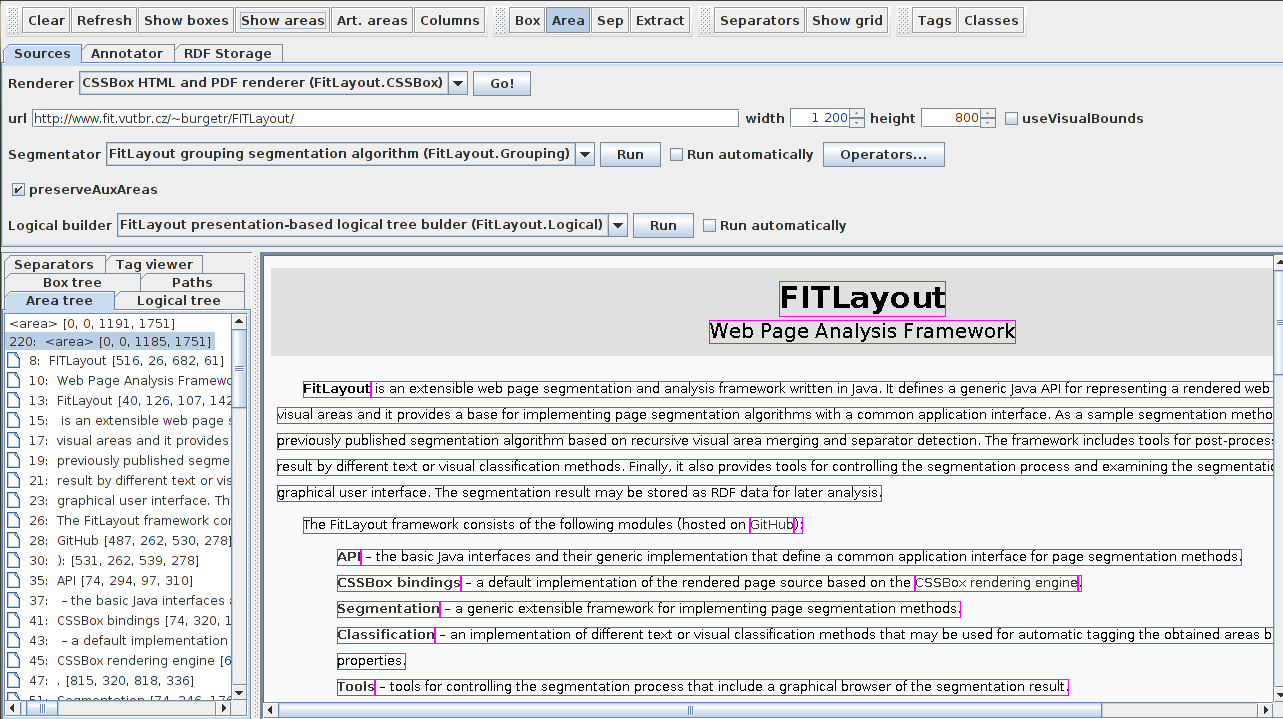
### Nástroje

Nástroje FitLayoutu umožňují spustit segmentační algoritmy nad danou URL adresou a zkoumat pak výsledky segmentace za pomocí jeho GUI. Jedná se tedy o dvě části procesor a grafické uživatelské rozhraní.

Procesor je třída zodpovědná za provádění všeho, co se kolem segmentace odehrává. Jedná se především o vytváření stromu základních vizuálních oblastí a aplikování segmentačních operátorů na tento strom. Základní funkčnost je definována jako abstraktní třída BaseProcessor. Ve FitLayoutu existují dvě dostupné implementace:

* ScriptableProcessor používá Javascript pro nastavení oblastních operátorů, které by měly být aplikovány.
* GUIProcessor zde se nachází nastavení operátorů, které mohou být modifikovány zvenčí, typicky za pomocí grafického uživatelského rozhraní.

O spuštění grafického rozhraní se stará třída BlockBrowser, kde je implementován výchozí prohlížeč za pomocí Swing prvků. Uživatel si zde může nastavit různá nastavení *BoxTree*, *AreaTree* nebo výběr a nastavení aplikovaných operátorů. Příklad GUI je na obrázku 4.3.



Obrázek 4.3: Ukázka GUI ve FitLayoutu. Na obrázku jsou vyznačené vizuální bloky nalezené segmentačním algoritmem.

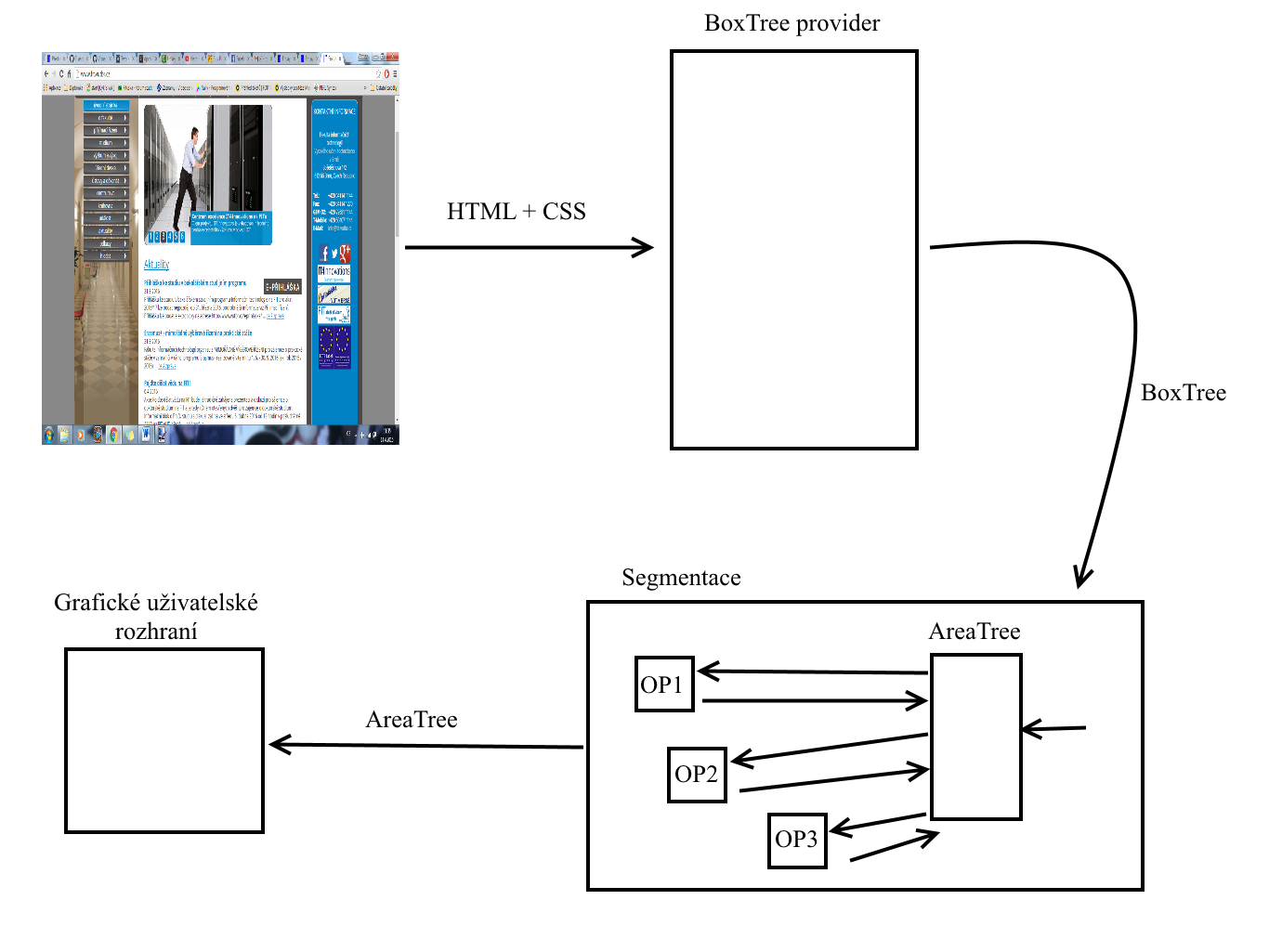
## Práce s Frameworkem

Než se pustíme do samotné implementace algoritmu, je vhodné seznámit se také s ovládáním Frameworku FitLayout. Příklad jak vypadá grafické uživatelské rozhraní, jsme si ukázali na obrázku 4.3. Avšak je vhodné si taktéž sdělit, jak Framework funguje při segmentaci. Důležitými dvěma prvky při segmentaci za použití Frameworku jsou *BoxTree* a *AreaTree*.

Prvně jmenovaný prvek vzniká přímo ze vstupní stránky. Odpovídá tak DOM stromu stránky. *BoxTree* získáme ze vstupní webové stránky za pomocí renderovacího nástroje. Ten je v případě FrameWorku CSSBox, který byl popsán výše v kapitole 4.1.1. Když to tedy shrneme, můžeme říct, že *BoxTree* je strom boxů, které odpovídají prvkům v DOM stromu vstupní webové stránky.

Naopak *AreaTree* je strom vizuálních oblastí, které získáme převodem z *BoxTree*. *AreaTree* se používá jako vstupní parametr do všech algoritmu pro segmentaci webových stránek ve Frameworku FitLayout. Je taktéž výstupem těchto algoritmů. Na obrázku 4.4 můžeme vidět obecně, jak Framework funguje při segmentaci webových stránek.

V tuto chvíli se dostáváme ke zmíněnému ovládání Frameworku. Na obrázku 4.3 vidíme grafické uživatelské rozhraní a je patrné, že jeho ovládání bude velmi jednoduché. Uživatel zde může vložit libovolnou webovou stránku. Poté spustí tlačítkem *Renderer*  renderovací nástroj a získá tak *BoxTree* této stránky. Poté si může celý strom vykreslit přímo na zobrazené stránce. Tlačítka mají jednoznačné popisky, tudíž je pro uživatele ovládání jednoduše zvládnutelné. Více informací, jak rozšířit či použít FitLayout se dozvíte zde [16].



Obrázek 4.4: Obecné znázornění Frameworku FitLayout při provádění segmentace.

### Operátory

Poněvadž rozšiřujeme Framework FitLayout o segmentační metodu, je vhodné zmínit, kam ji přesně v rámci Frameworku umístíme, a jakým způsobem bude volána při spuštění. Náš algoritmus implementujeme jako jeden z operátorů, které slouží pro specifikaci segmentace. To znamená, že při provádění segmentace nad vloženou webovou stránkou si můžeme vybrat, které operátory budeme na ni aplikovat. Operátory můžeme vzájemně spolu kombinovat a sledovat tak zajímavé výsledky segmentace.

Vstupem operátorů je opět zmíněný *AreaTree* a taktéž je jejich výstupem. Pokud je vybráno více operátorů, vzájemně si předávají *AreaTree* a pokaždé na tento vstupní strom aplikují vlastní algoritmus. Poslední operátor tak vrací výstupní strom, který můžeme prozkoumat v grafickém uživatelském rozhraní. Na obrázku VLOZIT můžeme vidět, jak vypadá výběr operátorů a zároveň příklad úpravy jejich vstupních parametrů.

Základní třídou sloužící pro implementaci nového operátoru je BaseOperator. Ta zde slouží jako rozhraní, podle nějž se tvoří operátory a dědí základní konstrukce metod, které jsou shodné pro každý operátor. Při tvorbě nového operátoru tedy zdědíme toto rozhraní a pouze reimplementujeme již předdefinované metody a můžeme jej tak použít v rámci segmentování webových stránek ve Frameworku FitLayout.

# Implementace algoritmu

Doposud jsme se v této práci soustředili pouze na teoretickou stránku použitého segmentačního algoritmu [8]. Je tedy vhodné, abychom přešli i k jeho implementaci. K tomu slouží tato kapitola, kde se zaměříme na popis implementace jednotlivých částí naší segmentační metody.

Implementace byla vytvořena jako samostatný projekt, který spolupracuje s Frameworkem FitLayout. Skládá se z hlavní třídy, která řídí celý algoritmus, a provádí jej podle uvedených postupů v kapitole 3. Algoritmus byl ve FitLayoutu implementován jako jeden z operátorů. Rozšiřuje tedy rozhraní BaseOperator a je zahrnut v konfiguračních souborech. To nám umožňuje v grafickém uživatelském rozhraní Frameworku selektovat námi implementovaný algoritmus mezi ostatními operátory.

Rozebereme si jednotlivé části algoritmu z hlediska implementační části. Zaměříme se především na záludnosti, s kterými jsme se museli vypořádat, a zároveň jak jsme tyto problémy vyřešili.

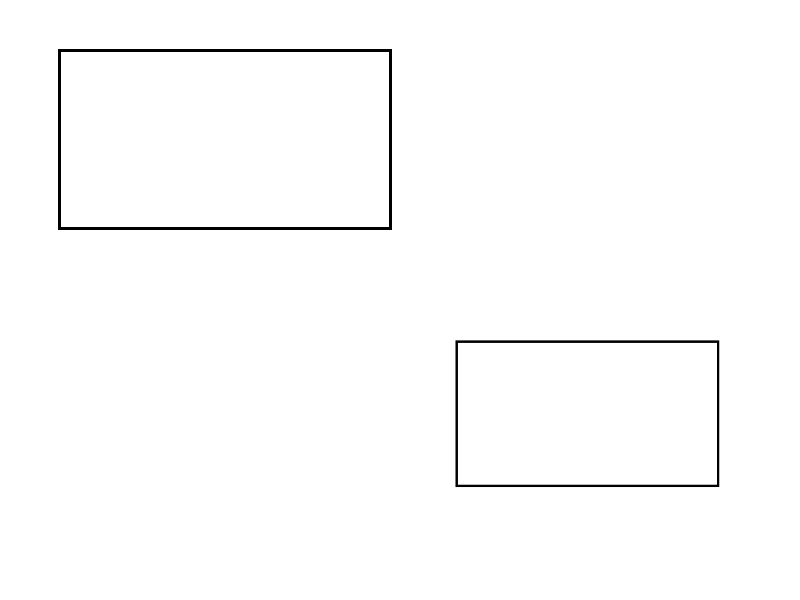
## Sousednost bloků

Důležitou částí implementace segmentačního algoritmu podle vizuálních vlastností stránky je výpočet průměrného Seam Degree. Sousedností bloků rozumíme jejich vzájemné uspořádání. V kapitole 3.2.2 jsme si definovali sousedící a nesousedící bloky a taktéž seam degree. Při programování této části vyvstaly určité problémy, se kterými jsme se museli potýkat a dále si je přiblížíme i s jejich řešením.

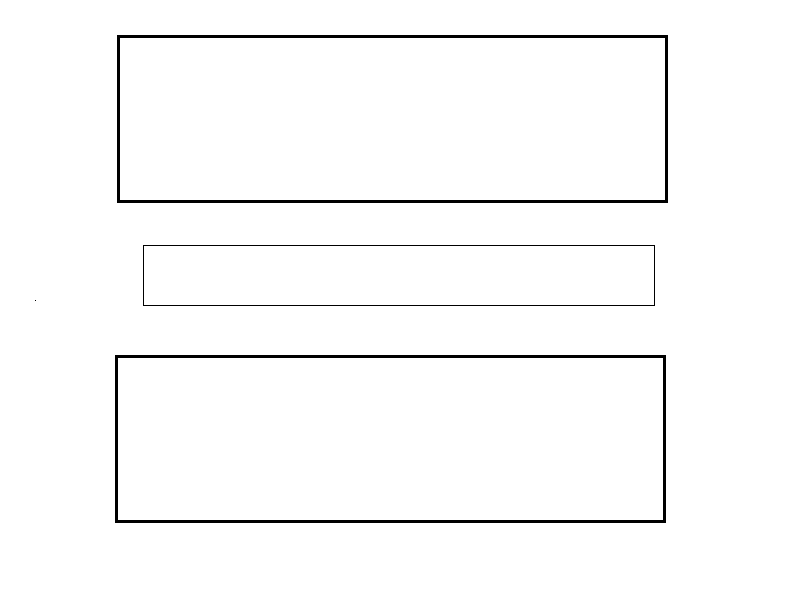
### Nesousedící bloky

Dříve než začneme počítat seam degree pro dva dané bloky, potřebujeme určit, zda jsou bloky sousedící či nikoliv. To můžeme určit několika způsoby, avšak zde si ukážeme pouze ta řešení, na které jsme se zaměřili. Taktéž musíme říci, že mohou být dva druhy navzájem nesousedících bloků:

* Dva bloky vzájemně se nepřekrývající žádnou částí vertikálně, ani horizontálně. Ukázku takových bloků lze vidět na obrázku č. 5.1.
* Dva bloky, mezi kterými se nachází další blok/bloky. Ukázku tohoto typu nesousedících bloků můžeme vidět na obrázku č. 5.2. Uvažované nesousedící bloky jsou na obrázku zvýrazněny tučně.



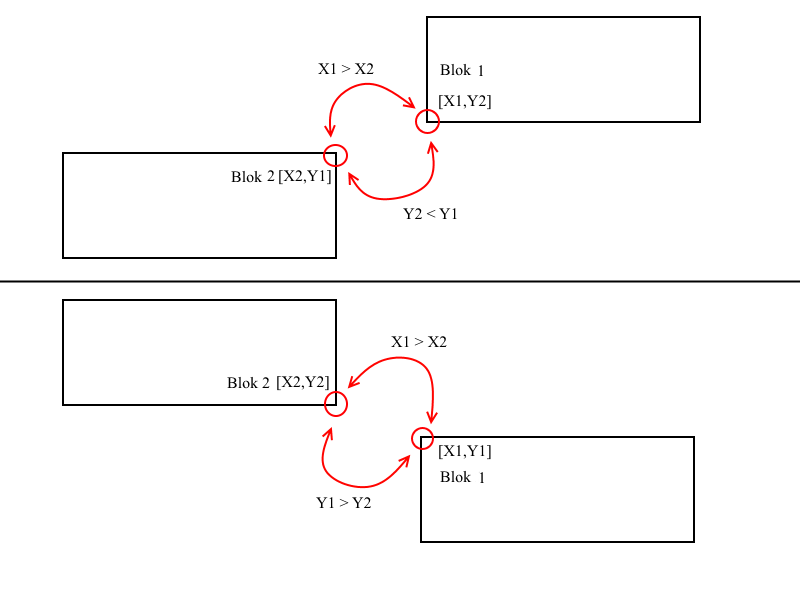
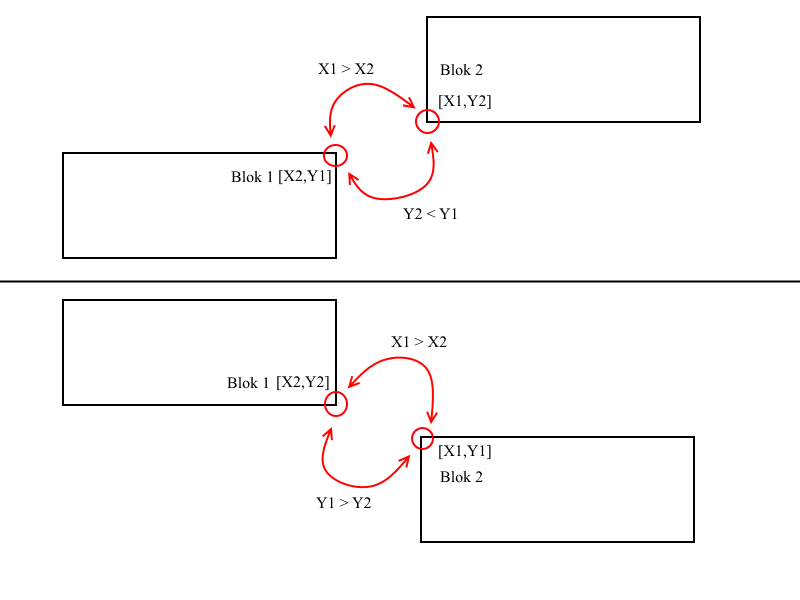
Obrázek 5.1: Ukázka nesousedících bloků.



Obrázek 5.2: Ukázka nesousedících bloků, mezi nimiž je jiný blok.

První typ nesousedících bloků jsme vyřešili za pomocí souřadnic krajních bodů obou bloků. Musíme porovnat jak souřadnice pro vertikální sousednost, tak i pro horizontální. Na obrázcích 5.3. a 5.4 vidíme, za pomocí šipek, které souřadnice daných bloků se porovnávají. A taktéž, která z nich musí být větší nebo menší, aby byly dva bloky nesousedící.

Pokud se mezi dvěma porovnávanými bloky nachází jiný blok, jsou tyto bloky nesousedící. Implementačně jsme to vyřešili za pomocí obdélníku. Ten je vytvořen jako oblast mezi porovnávanými bloky. Pokud se nachází v tomto obdélníku jiný blok, znamená to tedy již zmíněnou situaci a bloky jsou určeny jako nesousedící. Celou tuto situaci můžeme vidět na obrázku č. 5.5. Vyšrafovaný obdélník je nákres pomocného obdélníku a červeně vyšrafovaný obdélník je nalezený blok mezi dvěma danými bloky.



Obrázek 5.3: Ukázka nesousedících bloků. Blok 2 se nachází vlevo a blok 1 vpravo.

Obrázek 5.4: Ukázka nesousedících bloků. Blok 2 se nachází vpravo a blok vlevo.

### 

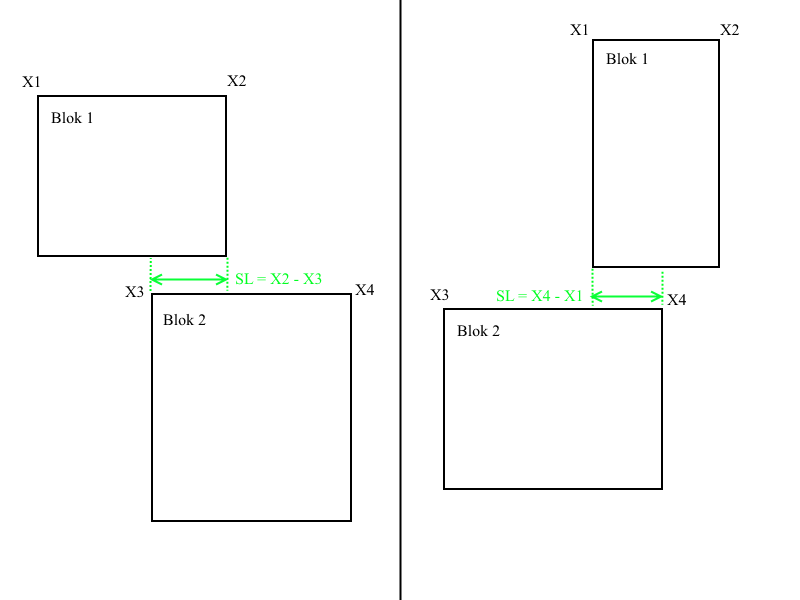
Obrázek 5.5: Ukázka hledání bloku mezi danými dvěma bloky.

### Sousedící bloky

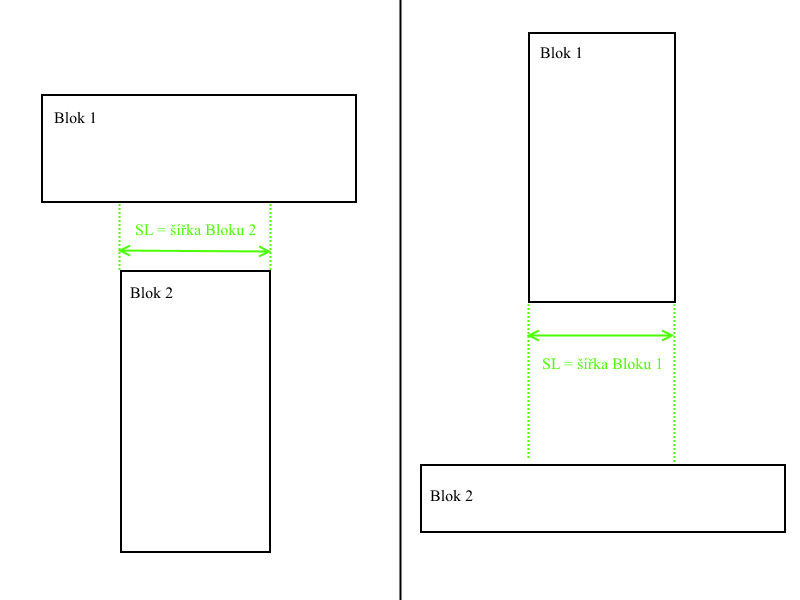
Pokud jsou dva bloky sousedící, což můžeme určit z toho, že nejsou nesousedící, tak musíme zajistit informaci o tom, jak spolu navzájem sousedí. Z toho vyplývá, že mohou sousedit buďto vertikálně nebo horizontálně.

#### Horizontální sousednost

Horizontálně sousedící bloky jsou uspořádány vedle sebe. Při horizontálně sousedících blocích používáme vzorec č. 1. Problém, který musíme vyřešit, se nachází při výpočtu hodnoty *SeamLength,* což značí délku překrytí. Při výpočtu této hodnoty, která udává délku překrytí bloků, musíme vzít v potaz vzájemnou polohu daných dvou bloků. Všechny možné způsoby, jak mohou být dva bloky vzájemně horizontálně uspořádány, můžeme vidět na následujících obrázcích č. 5.6 a č. 5.7. U každého obrázku taktéž můžeme vidět, jak se vypočítá délka překrytí, pro dané vzájemné umístění daných dvou bloků. Délka překrytí je na obrázcích označená jako *SL.*



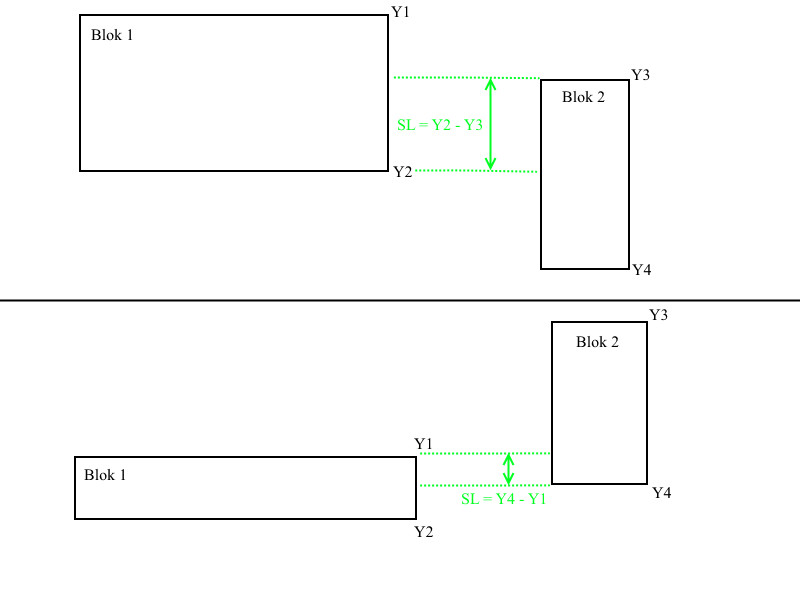
Obrázek 5.6: Ukázka výpočtu horizontální délky překrytí, při částečném překrytí.



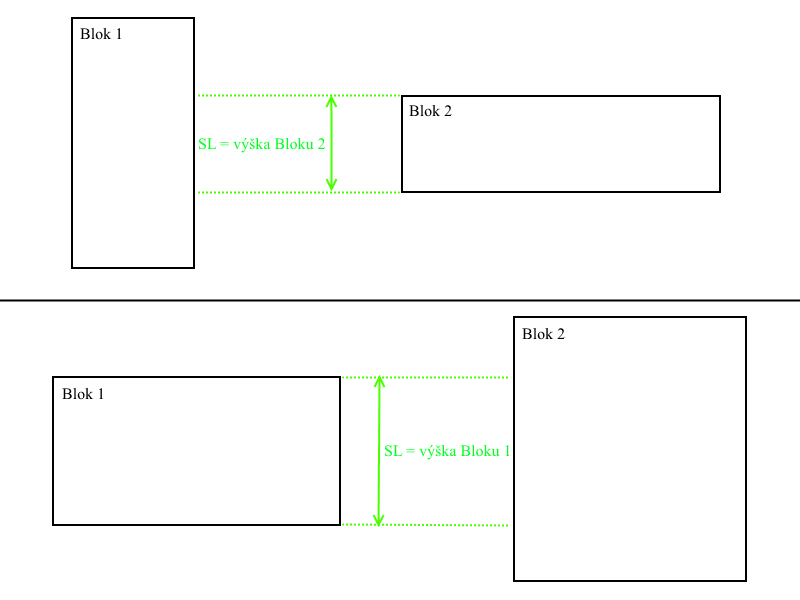
Obrázek 5.7: Ukázka výpočtu horizontální délky překrytí, při úplném překrytí.

#### Vertikální sousednost

Vertikálně sousedící bloky jsou uspořádány pod sebou. Při vertikálně sousedících blocích používáme vzorec č. 2. Problém, který musíme vyřešit, se nachází při opět výpočtu hodnoty *SeamLength*. Při výpočtu této hodnoty, která udává délku překrytí bloků, musíme vzít v potaz vzájemnou polohu daných dvou bloků. Všechny možné způsoby, jak mohou být dva bloky vzájemně vertikálně uspořádány, můžeme vidět na následujících obrázcích č. 5.8 a č. 5.9. U každého obrázku taktéž můžeme vidět, jak se vypočítá délka překrytí, pro dané vzájemné umístění daných dvou bloků.



Obrázek 5.8: Ukázka výpočtu vertikální délky překrytí, při částečném překrytí.



Obrázek 5.9: Ukázka výpočtu vertikální délky překrytí, při úplném překrytí.

Jakmile zjistíme, zda jsou bloky sousedící či nikoliv. Můžeme při kladné odpovědi vypočítat délku překrytí daných dvou bloků. Avšak, než ji vypočítáme, musíme zjistit, jaká je jejich vzájemná poloha. Jakmile máme vypočítanou hodnotu délky překrytí, můžeme vypočítat hodnotu Seam Degree pro aktuálně dané dva bloky. Dle algoritmu popsaného v kapitole 3. můžeme dále spočítat průměrnou hodnotu Seam Degree pro všechny bloky uvnitř právě zpracovávaného bloku. V tabulce 3.1 lze vidět, kterého kroku se tento výpočet týká. Pokud vypočítaná hodnota nesplňuje zadanou podmínku, přechází se na následující krok.

## Vizuální obsah bloků

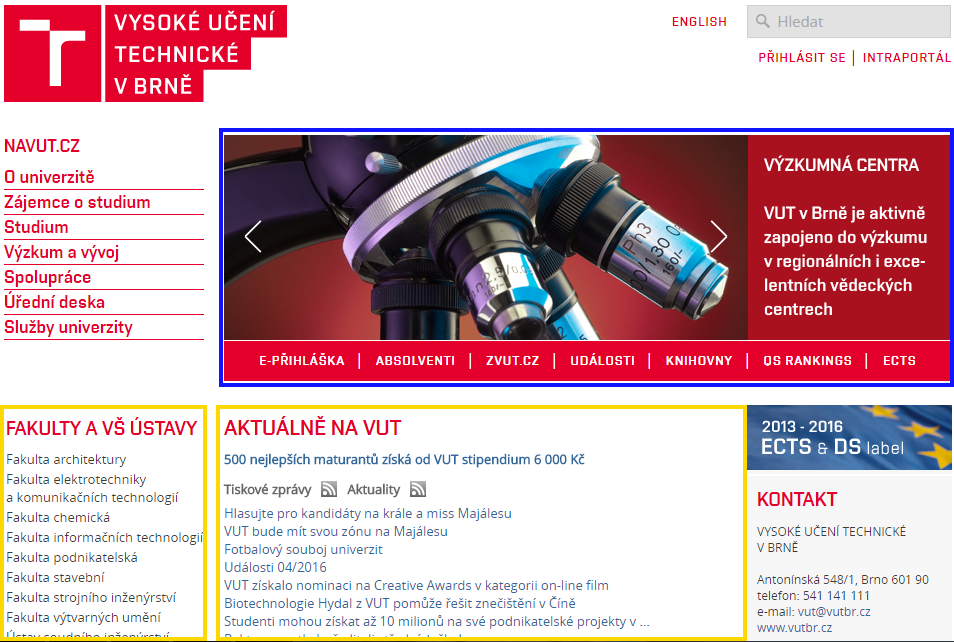
Další nedílnou částí námi popisovaného algoritmu je výpočet podobnosti obsahu bloků. Obsah vizuálního bloku může být různý a při rozhodování zda daný blok dále dělit či nikoliv, potřebujeme vědět, jaký obsah se v něm nachází. Podle toho, pak můžeme hodnotu, která je pro náš algoritmus důležitá vypočítat. Dále si tedy přiblížíme, jakým způsobem jsme tento výpočet implementovali.

### Vektory obsahu

Jak jsme si již sdělili, obsah vizuálního bloku může být různý. Jedná se například o obrázkový obsah, textový obsah atd. Máme-li daný blok, potřebujeme pro výpočet jednotlivých vzorců, uvedených kapitole 3. znát tzv. vektory obsahu. Vektorem se v tomto případě myslí seznam.

Tento seznam obsahuje ty vizuální oblasti, které jsou pro něj určené. Tudíž to může být vektor textového obsahu nebo obrázkového obsahu atd. Potřebujeme tedy vzít všechny části, které daný blok obsahuje a zjistit, jaký obsah má každá dílčí část a naskládat je do jednotlivých vektorů. Na obrázku č. 5.10 můžeme vidět příklady, jak může takový obsah daného bloku vypadat.

Pokud chceme zjistit obsah bloku přímo ve Frameworku FitLayout, musíme přistoupit k Boxu daného bloku. To znamená, že si vezmeme DOM uzel toho bloku za pomocí Box Tree, který byl popsán v kapitole 4. a můžeme pak pomocí metod přistoupit přímo k hodnotě určující typ obsahu toho bloku.



Obrázek 5.10: Ukázka vizuálních prvků na webové stránce. Žlutě jsou označený bloky obsahující převážně textový obsah a modře je označen blok, jak s textovým obsahem, tak i s obrázkovým.

### Podobnost obsahu

Jakmile získáme vektory obsahu pro dané dva bloky, můžeme přejít k dalším částem výpočtu podobnosti obsahu. Ten se skládá z několika výpočtů, které jsme si již definovali v kapitole 3. K výpočtu podobnosti obsahu tedy potřebujeme další dvě pomocné hodnoty. Jedná se o kosínovou podobnost a váhu.

V první řadě chceme zjistit podobnost jednotlivých typů vektorů pro dva dané vstupní bloky. Podobnost dvou vektorů můžeme spočíst několika způsoby, avšak my jsme kvůli jednoduchosti zvolili kosínovou podobnost. Ta předpokládá stejnou délku počítaných vektorů. Při neshodě se kratší vektor doplní nulami, aby se velikosti obou vektorů rovnaly. Jakmile splníme tento požadavek, můžeme přejít na výpočet. Ten se provádí, jak je uvedeno v rovnici č. 5. Můžeme doplnit, nastane-li situace, kdy jsou oba vektory prázdné, výsledná hodnota kosinové podobnosti je nulová.

Další hodnotu, kterou potřebujeme, je váha daného typu vektoru. Je to dáno tím, že každý blok, může mít například víc textového obsahu než obrázkového než ten druhý atd. Váha se tedy počítá jako součet sum daného typu vektoru, pro který počítáme váhu, a ty jsou poděleny součtem celkových oblastí zabíraných danými dvěma bloky.

Máme-li hodnotu kosínové podobnosti a váhy pro daný typ vektoru, můžeme vypočíst podobnost obsahu. Získáme-li průměrnou podobnost obsahu daného bloku, porovnáme jej s definovanou hodnotou a tím rozhodneme, zda budeme opět blok dělit či nikoliv.

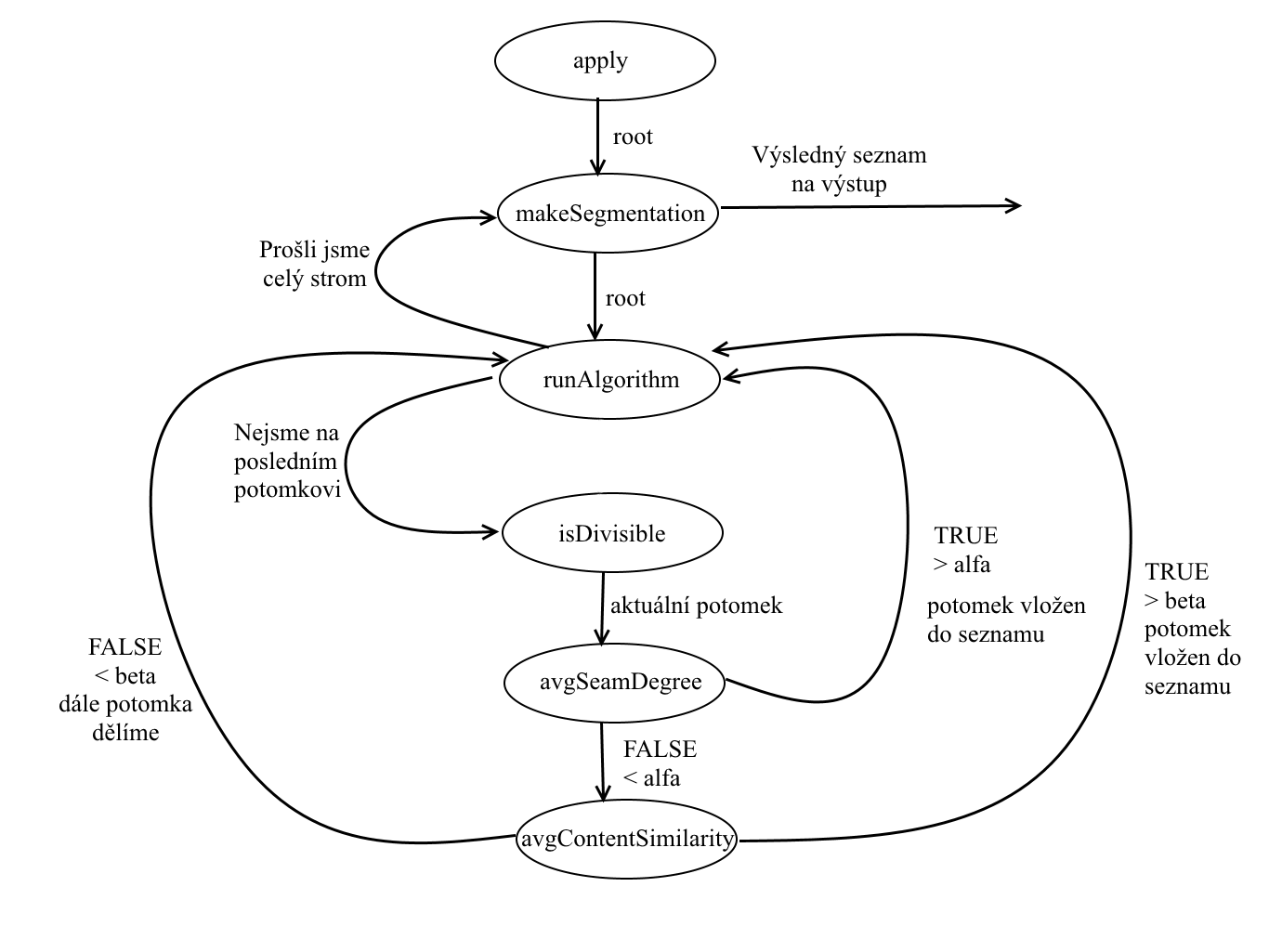
## Řízení algoritmu

Je pravdou, že k tomu, abychom mohli provádět segmentaci webových stránek, nestačí nám si říct jak vypočítat několik vzorců. Musíme si říct, jak je celý použitý algoritmus řízen, a jakým způsobem je postupně tvořen výstup.

Celý algoritmus je řízen především z jedné třídy, ve které jsou implementovány všechny metody a atributy důležité pro vykonání metody. Třída se nazývá VisualSemanticsOperator a rozšiřuje rozhraní BaseOperator. To znamená, že musí obsahovat několik pevně daných metod, které musí být implementovány. Ty slouží především pro správnou funkčnost z hlediska grafického rozhraní a taktéž spuštění námi implementovaného algoritmu.

Po spuštění algoritmu se jako první volá metoda Apply().Této metodě je předán jako vstupní parametr *AreaTree*, který je velmi blízký DOM stromu, jenž je brán jako vstupní prvek našeho algoritmu. Poté můžeme začít s algoritmem. Ten řídí metoda makeSegmentation(). Ta obdrží kořenový uzel stromu a nad ním spustí rekurzivní metodu runAlgorithm(), která prochází celý strom a podle pravidel v tabulce 3.1 řeší dělení jednotlivých prvků vstupního stromu. Což jsou i naše bloky.

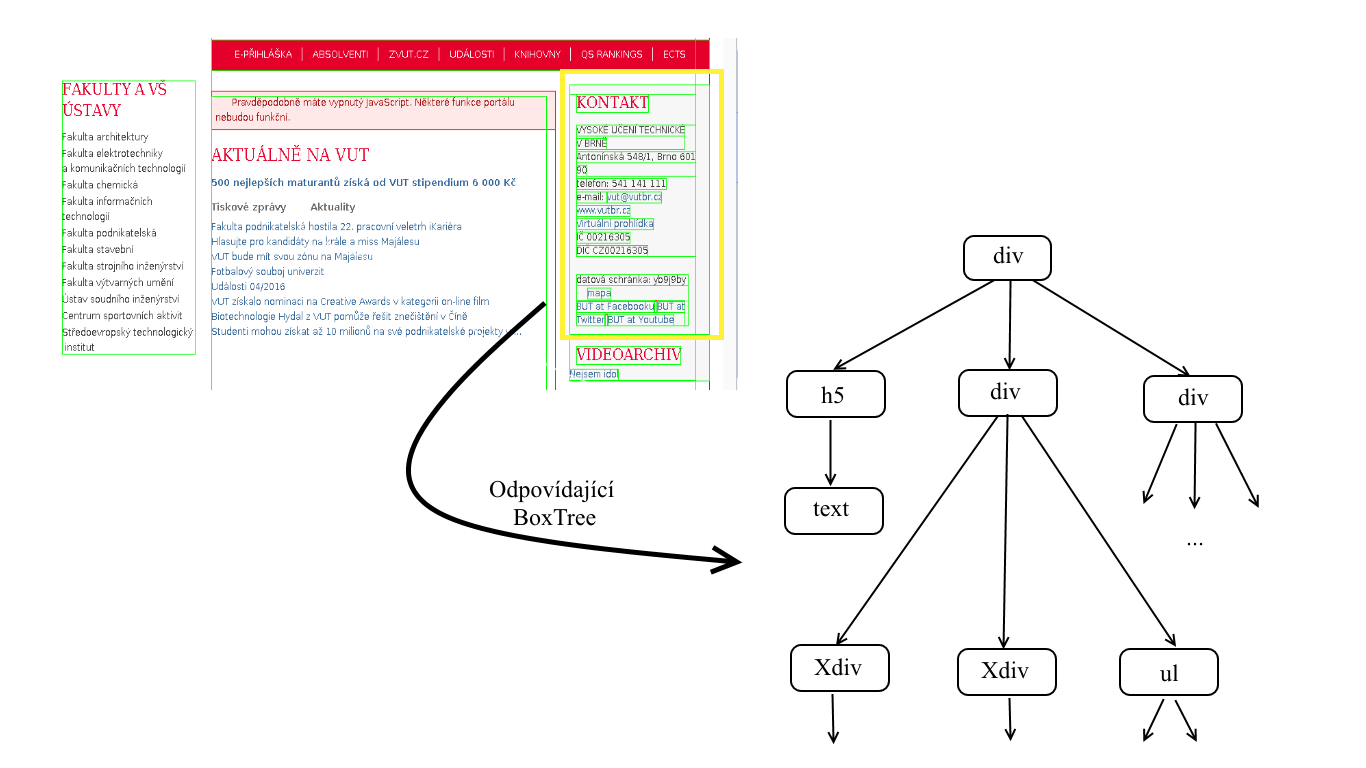
Pokaždé, když se blok nedá dělat je vložen do seznamu. Jakmile je seznam hotový, končí také rekurzivní funkce. Může tak být tvořen výstupní strom, který má stejný formát jako vstupní. Opět se jedná o *AreaTree*. Bereme postupně uzly z našeho seznamu a vytváříme výsledný strom. U každého uzlu zachováváme jeho potomky, aby byl strom opět hiearchicky uspořádán. Na obrázku č. 5.11 můžeme vidět přehledný graf postupně volaných metod celého algoritmu.



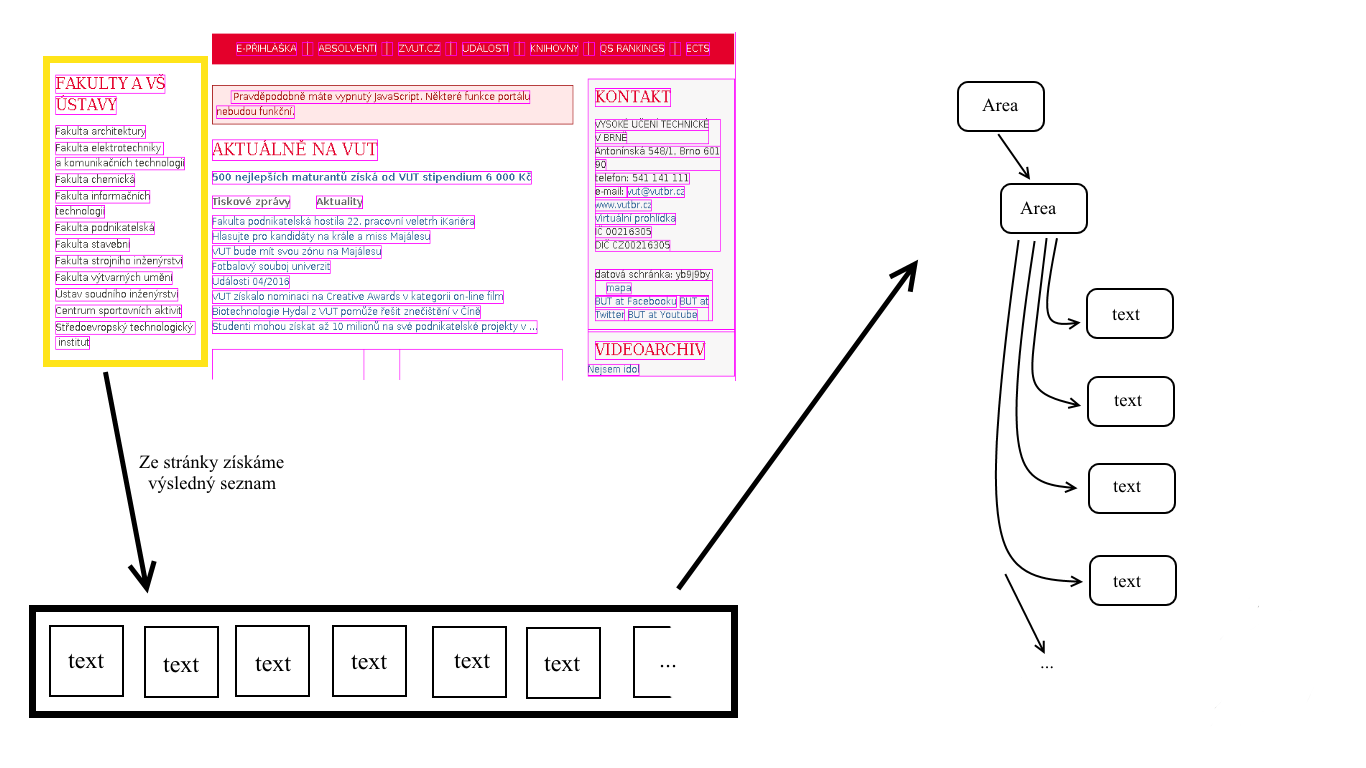
Obrázek 5.11: Zjednodušený obrázek grafu volaných metod při provádění algoritmu.

## Vstup a výstup algoritmu

Nyní už víme, jak celý algoritmus funguje a je vhodné si předvést, jak vypadá vstupní a výstupní stránka. Na vstupu máme webovou stránku prohnanou CSS boxem, což je renderovací nástroj. Vstupní stránka se převede na Box Tree, který je shodný s DOM stromem, a ta se poté převede na totožný Area Tree, který přijímá na vstup náš algoritmus. Na následujících obrázcích můžeme vidět, jak vypadají vstupní a výstupní prvky a taktéž jejich odpovídající stromy. U *BoxTree* se jedná o webovou stránku jako vstupní prvek a výstupním je strom. To můžeme vidět na obrázku 5.12, kde je část webové stránky a vyznačené zeleně nalezené prvky v BoxTree. Dále pak je zde část BoxTree, který odpovídá žlutě vyznačené části. Kdežto u *AreaTree* se jedná o seznam jednotlivých prvků webové stránky, což jsou naše již nedělitelné vizuální bloky, a na výstupu opět vzniká strom. To opět vidíme na obrázku a to konkrétně 5.13. Zde lze vidět část webové stránky, na které jsou vyznačené růžově nalezené vizuální oblasti a dále seznam, z kterého byly oblasti skládány do výsledného stromu. Ten je zde taktéž zaznačen. Můžeme si všimnout i z tak krátké ukázky, že výsledný strom našeho segmentačního algoritmu je velmi plytký. Toto upravíme za pomocí vstupních parametrů, které změníme v uživatelském rozhraní a ovlivníme tak výstupný seznam.



Obrázek 5.12: Ukázka *BoxTree*, který odpovídá bloku na webové stránce.



Obrázek 5.13: Ukázka *AreaTree*, který odpovídá seznamu nalezeného ze stránky.

# Dosažené výsledky

Po každé implementaci bychom měli ověřit její validitu a to za pomocí kvalitních testů. V této kapitole si ukážeme, jak jsme testování prováděli, a taktéž jakých jsme dosáhli výsledků při segmentaci vstupní webové stránky. Jelikož se jedná především o experimentální odvětví, je potřeba implementaci správně otestovat na vhodně zvolené sadě webových stránek.

Seznámíme se zde tedy se zvolenou testovací sadou. Taktéž je vhodné, abychom provedli porovnání s nějakou jinou segmentační metodou. Testování je důležité i z toho hlediska, že nám může pomoci odhalit některé chyby v implementaci. Pokud se nějaké naleznou, budou zde taktéž uvedeny i s jejich řešením.

Dalšími důležitými faktory, které velmi ovlivňují výsledek segmentace, jsou vstupní parametry. Již jsme se o nich zmínili v předešlých kapitolách a jedná se o parametry *alfa* a *beta*. Jaký vliv mají na výsledek segmentace, se dozvíme níže v podkapitole 6.2.

## Testování

Správné a pečlivé testování je klíč k úspěchu každé implementace. Pomáhá nám ověřit si jednotlivé části, a poté i kompletní část implementace. Taktéž je nápomocná při odhalení mnoha chyb, kterých jsme si nemuseli při implementaci všimnout.

Vezmeme to z hlediska našeho algoritmu. Protože se jedná o experimentální tvorbu, musí tomu odpovídat i testy. Ty musíme volit správně a účelně, abychom dosáhli správných výsledků a odhalili chyby specifické pro náš algoritmus. Především musíme vybrat správnou sadu vybraných webových stránek. Jakmile vybereme správnou sadu, můžeme přejít k samotnému testování. To spočívá ve spouštění našeho segmentačního algoritmu na vstupní sadě při různé kombinaci se vstupními parametry. Jak tyto parametry ovlivňují naši segmentační metodu, se dočteme níže. Ukázku sady webových stránek můžeme vidět na obrázku VLOZIT.

Po otestování na vybrané sadě můžeme prozkoumat výsledky a zhodnotit tak, zda se nám podařilo správně implementovat námi popisovanou segmentační metodu. Jakmile tedy otestujeme na výše vybrané sadě, můžeme odhalit chyby v naší implementaci.

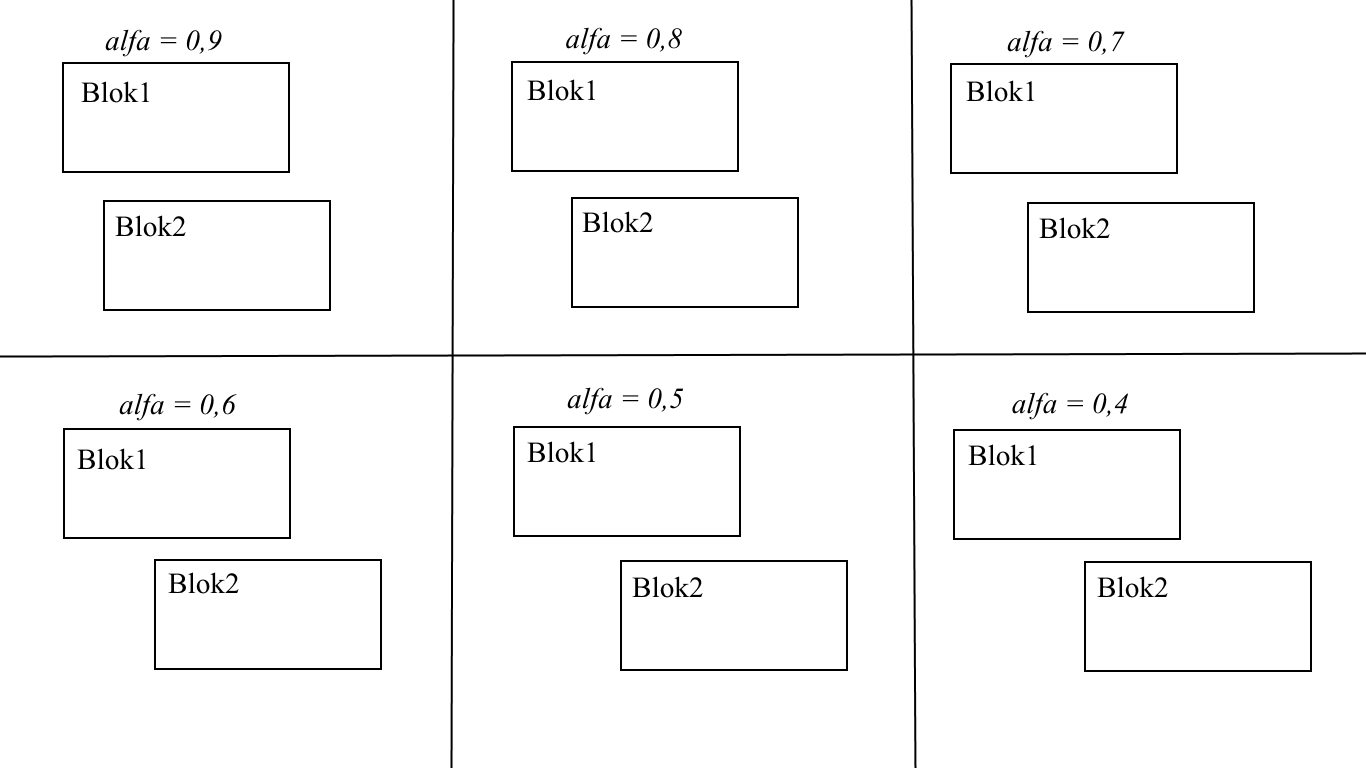
Ještě asi lehce rozepsat..

## Změna vstupních parametrů

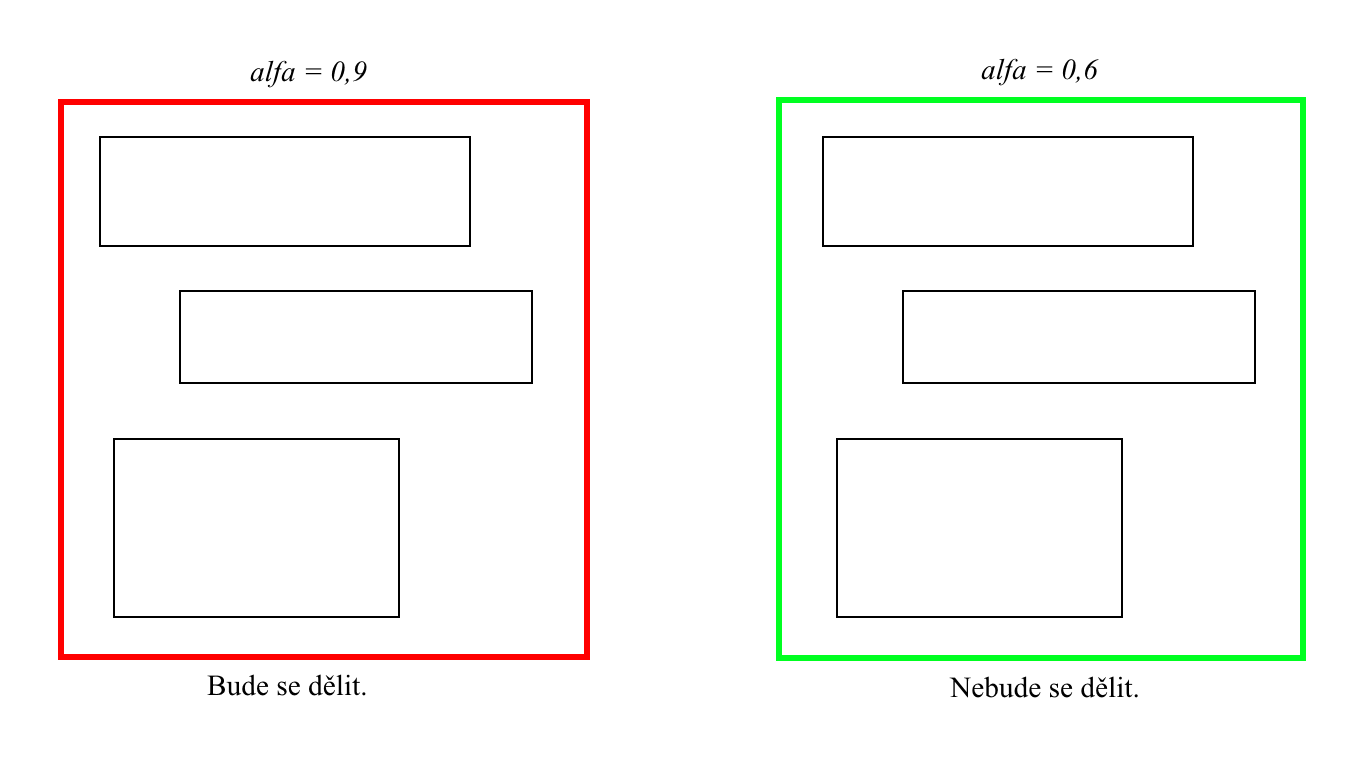
V našem algoritmu jsou dvě důležité hodnoty, taktéž je můžeme nazývat vstupními parametry, ovlivňující výsledek segmentace. Jedná se o *alfu* ()a *betu* (). Jsou to tzv. prahové hodnoty, s kterými jsou porovnávány výpočty uvedené v kapitole 3. Konkrétně pro parametr se jedná o výpočet č. 3 a pro parametr se jedná o výpočet č. 8. Při výběru naší implementace jako algoritmu, který bude ve Frameworku FitLayout provádět aktuální segmentaci, můžeme tyto parametr různě nastavit. V počátečním nastavení je a = 0.8. Tyto hodnoty jsme empiricky zvolili podle [8].

Přejděme tedy k přímo k tomu, jak vstupní parametry ovlivňují výsledek segmentace vstupní webové stránky. Konkrétně parametr je porovnávány s průměrným vypočítaným SeamDegree. Z toho vyplývá, že ovlivňuje porovnávání sousednosti jednotlivých bloků. Necháme-li *alfu* nastavenou na počáteční hodnotu, sousední bloky se budou muset překrývat těsněji, aby se od sebe nedělily. Avšak jakmile hodnotu parametru *alfa* snížíme, tak se zvětší tolerance překrytí sousedních bloků. Toto především ovlivní strukturu výsledného stromu. Bude-li se *alfa* blížit nebo rovna počáteční hodnotě výsledný *AreaTree* bude plytký. Kdežto bude-li se *alfa* snižovat, tento výsledný strom bude více strukturovaný. Ostatně se můžeme podívat na obrázky č. 6.1 a č. 6.2, kde jsou příklady, co vše ovlivňuje parametr *alfa* ukázány. Na prvním z nich můžeme vidět, jakou polohu mohou mít vzájemně dva sousedící bloky, dle zadané *alfy*. Na druhém z nich vidíme, že červeně ohraničený blok se bude muset dělit na menší bloky, protože *alfa* je větší, kdežto zelený blok se dělit nebude, poněvadž *alfa* je nižší a je tak povolena větší tolerance odsazení při vzájemně sousedících dvou blocích.

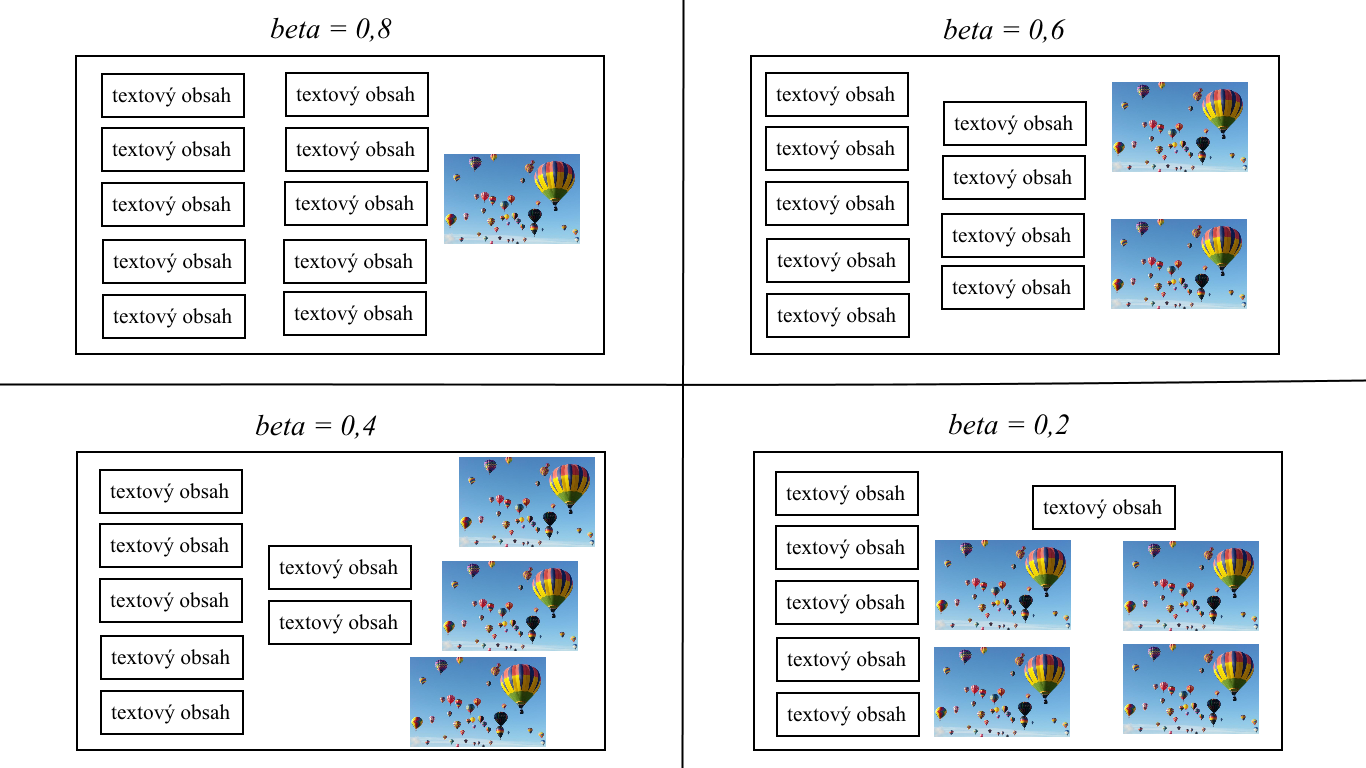
Poněvadž máme dva vstupní parametry, je vhodné si ukázat i druhý parametr, jak může ovlivnit výsledek našeho segmentačního algoritmu. Hodnota parametru *beta* je porovnávána s průměrnou podobností obsahu. Na rozdíl od *alfy*, která ovlivňuje vzájemnou polohu při porovnávání dvou bloků, *beta* ovlivňuje porovnávání obsahu v daném bloku. Co to tedy pro nás znamená. Spouštíme-li naši segmentační metodu s nastavením *bety*, tak jak jsme si zmínili výše (tedy prahovou hodnotou), blok se bude dělit na dílčí, již při menší různorodosti obsahu. Pokud budeme parametr *beta* snižovat, zvětší se tolerance, a blok bude moci obsahovat různé prvky, například z 80% text a zbytek obrázky. Nejvýstižněji to můžeme vysvětlit opět za pomocí obrázků a konkrétně tato situace je na obrázku č. 6.3. Tady lze vidět situaci, kdy porovnáváme podobnost obsahu daného bloku, a jakou toleranci v této chvíli hraje vstupní parametr *beta*.



Obrázek 6.1: Ilustrace, jak *alfa* ovlivňuje toleranci, při porovnávání sousednosti dvou bloků. Všechny ilustrované bloky se nebudou dále dělit, podle zadané *alfy* nad nimi.



Obrázek 6.2: Ukázka bloků, které se při dané *alfě* budou dělit či nikoliv. Červeně je označen blok, který se bude dělit na nižší a zeleně blok, který se dělit bude.



Obrázek 6.3: Ilustrace, jak *beta* ovlivňuje toleranci při porovnání obsahu v daném bloku. Všechny ilustrované bloky se nebudou dále dělit, podle zadané *bety* nad nimi.

# Závěr

V této práci jsme se seznámili se segmentací webových stránek. Uvedli jsme stručně již existující algoritmy pro segmentaci webových stránek. Pro připomenutí to byly metody DOM-based Segmentation, VIPS a Box Clustering Segmentation. Hlavním cílem práce bylo rozšířit Framework FitLayout o segmentační metodu.

V našem případě se jednalo o metodu využívající vizuálních vlastností prvků na webových stránkách. Nejprve jsme se seznámili s algoritmem pouze teoreticky. Taktéž bylo za potřebí, seznámit se samotným Frameworkem. Podrobněji se o tom můžeme dočíst v kapitole 3. kde je teoretický popis námi vybraného algoritmu a ve 4. kapitole nalezneme seznámení s FitLayoutem.

Jakmile jsme se seznámili s potřebnou teorií a Frameworkem mohli jsme přejít na implementaci. Ta byla tvořena jako samostatný projekt, komunikující s FitLayoutem. Implementace byla dokončena v předstihu, abychom se mohli dále zaměřit na kvalitní testování a sepsání této práce.

Protože se jednalo o experimentální vývoj, museli jsme k tomu přizpůsobit i samotné testování. Vybrali jsme pro to specifickou sadu webových stránek, na kterých jsme testování provedli. Testování prokázalo správnost implementace i se spouštěním s různými parametry. Závažné chyby jsme neodhalili a to i díky včasnému odhalení některých již při tvorbě implementace.

Jako nedostatek bychom mohli uvést, že testování parametru *beta* nemohlo být dokonalé z důvodu nedostatečného renderovacího nástroje. CSSBox totiž nevykresluje obrázkový obsah, a tak nešlo dokonale odhalit nějakou závažnější chybu. Avšak můžeme konstatovat, správnou funkčnost našeho algoritmu, poněvadž jsme při implementaci dodrželi všechny teoreticky definované postupy. Tudíž můžeme říct, že zadání diplomové práce bylo splněno.

Protože segmentačních algoritmů existuje celá řada a my jsme se seznámili pouze s omezeným počtem, navázáním na tuto práci bychom mohli rozšířit Framework FitLayout o další implementace segmentačních algoritmů. Dále bychom mohli poznamenat, že Framework je kvalitní a lehce rozšiřitelný nástroj. Tím pádem může být rozšiřován o celou řadu implementací a může být dále zdokonalován. Například přidáním vylepšeného renderovacího nástroje by mohl lépe fungovat i námi zvolený algoritmus.

Literatura

[1] Milička, M.; Burget, R. Information Extraction from Web Sources based on Multi-aspect Content Analysis. In: Semantic Web Evaluation Challenges, SemWebEval 2015 at ESWC 2015. Portorož: Springer International Publishing, 2015, str. 81-92. ISBN 978-3-319-25517-0. ISSN 1865-0929.

[2] Milička, M.; Burget, R. Web Document Description Based on Ontologies. In: Proceedings of the 2nd annual conference ICIA 2013. Łódź: The Society of Digital Information and Wireless Communications, 2013, str. 288-293. ISBN 978-1-4673-5255-0.

[3] Burget, R. Layout Based Information Extraction from HTML Documents. In: 9th International Conference on Document Analysis and Recognition ICDAR 2007. Curitiba: IEEE Computer Society, 2007, str. 624-629. ISBN 0-7695-2822-8.

[4] Burget, R.; Rudolfová, I. Web Page Element Classification Based on Visual Features. In: 1st Asian Conference on Intelligent Information and Database Systems ACIIDS 2009. Dong Hoi: IEEE Computer Society, 2009, str. 67-72. ISBN 978-0-7695-3580-7.

[5] Deng, C.; Shipeng, Y.; Ji-Rong, W. VIPS: a Vision-based Page Segmentation Algorithm. 2003, [online]

[6] Popela, T. Implementace algoritmu pro vizuální segmentaci www stránek, diplomová práce, Brno, FIT VUT v Brně, 2012

[7] Zelený, J.; Burget R. [Cluster-based Page Segmentation - a fast and precise method for web page pre-processing](http://www.fit.vutbr.cz/~burgetr/pubs.php?id=10252). In: *The Third International Conference on Web Intelligence, Mining and Semantics*. Madrid: Association for Computing Machinery, 2013, str. 1-12. ISBN 978-1-4503-1850-1.

[8] Zeng, J.; Flanagan, B.; Hirokawa, S. A Web Page Segmentation Approach Using Visual Semantics. IEICE TRANS. INF. & SYST, Vol.E97-D, NO.2 February 2014

[9] Zeng, J.; Flanagan, B.; Hirokawa, S. Layout tree-based approach for identyfing visual similar blocks from web pages. Proc. Int´l Symposium Web Engeneering and related problems, June 2013, str. 66-70

[10] Zhang, K.; Shasha, D. Simple fast algorithms for the editing distance between trees and related problems. SIAM J. Comput, vol. 18, no. 6, str. 1245-1262, 1989

[11] Amit, S. Modern information retrieval: a brief overwiev. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, vol. 24, no. 4, str. 35-43, 2001.

[12] Deng, C.; Shipeng, Y.; Ji-Rong, W. Block-based Web Search. 2004, [online], <http://www.zjucadcg.cn/dengcai/VIPS/VIPS_July-2004.pdf>

[13] Zelený, J.; Burget R. FITlayout Web Page Analysis Framework, 2014-2015, [online], <http://www.fit.vutbr.cz/~burgetr/FITLayout/>

[14] Zelený, J: Web page segmentation and classification. 2011, [online], <http://www.feec.vutbr.cz/EEICT/2011/sbornik/03-Doktorske%20projekty/08-Informacni%20systemy/10-xzelen11.pdf>

[15] Burget, R: CSSBox. 2007-2016, [online], <http://cssbox.sourceforge.net/>

[16] Burget, R: FITLayout Web Page Analysis Framework, 2014-2015, [online], <http://www.fit.vutbr.cz/~burgetr/FITLayout/manual/>

**Přílohy**

1. Document Object Model - http://www.w3.org/DOM/ [↑](#footnote-ref-1)
2. W3C - World Wide Web Consortium je mezinárodní konsorcium, jehož členové společně s veřejností vyvíjejí webové standardy pro World Wide Web [↑](#footnote-ref-2)
3. Ke stažení na https://github.com/FitLayout/FitLayout.github.io [↑](#footnote-ref-3)
4. Více o těchto aplikacích zde https://docs.oracle.com/javase/tutorial/uiswing/ [↑](#footnote-ref-4)