SYSTEMY INFORMACYJNE I WYSZUKIWANIE INFORMACJI 3
Projekt

System wyszukiwania dokumentów tekstowych

Katarzyna Rzerzicha, Wojciech Skórski, Paweł Szołtysek

SPPI 3

6 czerwca 2008

Streszczenie

W dobie informatyzacji społeczeństwa oraz błyskawicznego postępu technologicznego coraz większego znaczenia nabiera możliwość trafnego i sprawnego przeszukiwania zasobów elektronicznych. Niniejsza praca przedstawia projekt systemu wyszukiwania dokumentów tekstowych.
Duży nacisk położono na szybkość wyszukiwania, wskutek czego zaproponowano autorski sposób indeksowania dokumentów opierający się na strukturach grafowych. Ponadto dokonano analizy i modyfikacji istniejących sposobów reprezentacji tego typu struktur, dzięki czemu zoptymalizowano zapis cyfrowy indeksu. Innowacyjność systemu polega także na zindywidualizowaniu wymagań osób korzystających z wyszukiwarki, dzięki stworzeniu profilu użytkownika. Zarówno personalizacja, jak i prowadzenie rankingu publicznego dokumentów pozwala na bardziej relewantne wyszukiwanie oraz pozycjonowanie wyników. Poza tym, spore znaczenie ma wprowadzenie nowatorskiego procesu rekomendacji - podczas korzystania z systemu proponowane jest użytkownikowi przeglądnięcie dodatkowych, potencjalnie interesujących go dokumentów. Wysoka jakość tego procesu jest osiągana dzięki wykorzystaniu struktur grafowych. Zaprezentowano także propozycje implementacji systemu, pomocne w fazie fizycznej.
Zastosowanie prekursorskich rozwiązań pozwala na znaczne zwiększenie efektywności działania systemu wyszukiwania.

Streszczenie

Facing society computerization and immediate technical developement, availability of relevant and fast searching in electronic resources means more and more. This work shows a project of text documents searching system.
Thanks to the fact, that main emphasis has been putted on the speed of the searching process, the new way of the document indexation based on the graph structures was introduced. Moreover, analysis and modification of existing methods of these type of structure reprezentation has been done, what leads to optimization them for this specific example. Innovation of the system is as well about individual needs of users, thanks to profiles for each of them. Personalization, as well as public rank, allows more relevant searching and better positioning of results. Furthermore, quite big importance has original recommendation process - while using, the system itself suggests other, potentally noteworthy documents in this topic. High level of quality of this action is reached thanks to the usage of the graph structure. Part of the work shows as well proposition of implementation, which can be useful in physical stage.
Usage of precursor solutions allows to increase the efficiency of searching system significantly.

Spis treści

Rozdział 1

Wstęp

Zadanie wyszukiwania tekstu wydaje się być nieskomplikowanym procesem. Jednak zaprojektowanie wydajnego i sprawnego systemu okazuje się w praktyce bardziej złożonym problemem. Dzieje się tak z kilku względów.

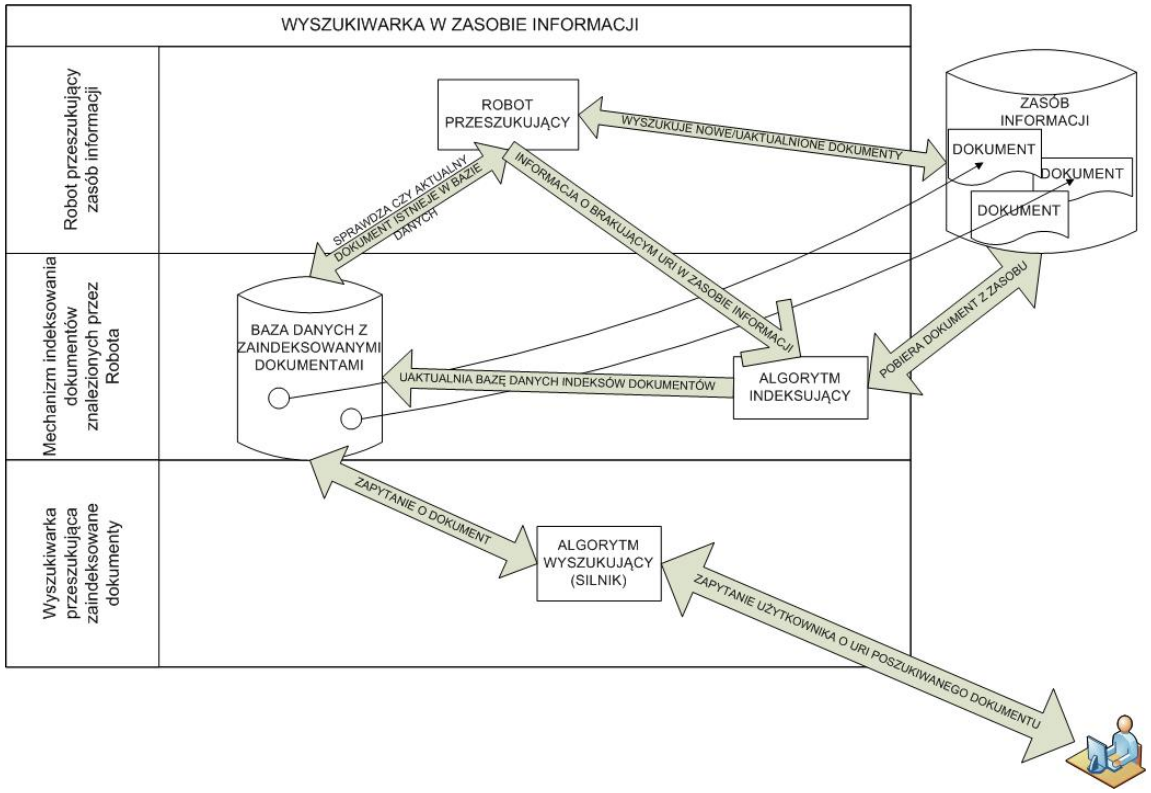
1. Istnieją słowa, które, chociaż mają taką samą budowę składniową, różnią się semantycznie. Między innymi z tego powodu komputer nie jest w stanie zrozumieć, co jest napisane w dokumencie.

2. Ludzka wiedza oraz tworzone dokumenty oparte są na doświadczeniach i inteligencji. Dzięki temu człowiek jest w stanie zrozumieć i opisać dokumenty, które czyta.

3. Ważnym jest zauważenie pewnej paralelności (wielowątkowości) wyszukiwarki: z jednej strony, musi ona nieustannie indeksować nowe dokumenty, a z drugiej bez przerwy odpowiadać na zapytania do niej kierowane.

4. Dokumenty wcześniej zaindeksowane mogą podlegać zmianom.

Z perspektywy wyszukiwarki, dokument jest przetwarzany w trzech głównych etapach. Najpierw specjalny robot przeszukuje pewien zasób dokumentów, w celu znalezienia nowych (takich, które nie istnieją w bazie danych) lub uaktualnionych (takich, które już istnieją w bazie, ale jest dostępna ich nowa wersja). Po odnalezieniu takowego, wysyła odpowiednią informację do algorytmu indeksującego, który ponownie pobiera dokument z zasobu. Następnie dokonuje operacji złączenia (*merging*) istniejącej bazy danych z nowo znalezionym dokumentem. Po wykonaniu tego, algorytm wyszukujący będzie uwzględniał ten dokument. Proces ten jest zobrazowany na rysunku 1.1. Widać więc, że każdy etap ma znaczny wpływ na jakość wyszukiwania. Niska jakość któregoś z modułów będzie się bezpośrednio przekładała na słabsze i mniej dokładne wyszukiwanie w indeksie. Wobec tego ważnym etapem



Rysunek 1.1: Zobrazowanie schematu wyszukiwania.

przy projektowaniu systemów do tego służących jest poprawne zdefiniowanie struktury bazy danych. Ponadto należy wnikliwie zastanowić się nad sposobem ich indeksowania, umożliwiającym sprawne i trafne przeszukiwanie.

W tej pracy skupiono się nad każdym kolejnym etapem tworzenia takiej wyszukiwarki.
W rozdziale 2. zostanie przedstawiony temat organizacji bazy dokumentów. Kolejno będą najpierw rozbijane zdania semantycznie na obiekty URI, do których się ono odnosi, po czym takie rozbicia będą upraszczane. Zostaną też przedstawione dwa typowe, proste podejścia do indeksowania dokumentów tekstowych, na podstawie których będzie zaprezentowany autorski algorytm indeksowania.
Rozdział 3. przedstawia przykładowe zapytania, jakie mogą być kierowane do silnika wyszukiwarki. W postaci sformalizowanej oraz UML przedstawione zostaną takie wyszukiwania dla zaproponowanego wcześniej indeksu. W rozdziale tym można znaleźć też diagram przypadków użycia wyszukiwarki.
Procesy rekomendacji zostaną omówione w rozdziale 4. Umieszczono tam m.

in. strukturę zaprojektowanego profilu użytkownika, sposób tworzenia rankingów oraz cztery sposoby rekomendacji, w tym jeden autorski.
Rozdział 5. zawiera propozycje implementacji zarówno funkcji (w formie pseudokodu) potrzebnych przy indeksowaniu i wyszukiwaniu w zaproponowanym systemie. Dodatkowo ma miejsce analiza dostępnych form reprezentacji grafu i na ich podstawie zaproponowano własny sposób takiej reprezentacji, przystosowany do przedstawionej struktury indeksu.

1.1 Uwagi ogólne

W dalszych rozważaniach przyjmujemy, że:

- średnia wielkość dokumentu wynosi 50 000 liter (tj. 50kB);
- średniej wielkości dokument zawiera 7 000 słów, czyli ok. 5 000 różnych termów;
- średni term liczy ok 7 liter (tj. 7B).

1.2 Użyte oznaczenia

System wyszukiwania informacji składa się z:

- zbioru dokumentów D,
- zbioru profilów użytkownika P,
- zbioru zapytań Z,
- zbioru termów T,
- słownika S,
- zbioru kategorii K.

Definiujemy graf $G = (V, E)$ jako:
V - zbiór wierzchołków taki, że $V \subseteq T$,
E - zbiór krawędzi etykietowanych taki, że $\forall e \in E : e = (v_i, v_j, w)$, gdzie $v_i, v_j \in V$ oraz $w \in \mathbb{R}$

Profil użytkownika $p \in P$ jest wyznaczany przez:

- p_k - zbiór kategorii wskazanych bezpośrednio przez użytkownika jako te, którymi się interesuje,

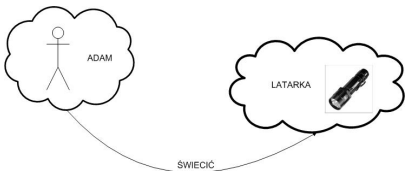
- p_l - zbiór wszystkich kategorii z przypisanymi wagami definiowanymi jako stopień zainteresowania użytkownika daną kategorią. Wagę określa się za pomocą liczby pozytywnie ocenionych dokumentów w kontekście kategorii.

Rozdział 2

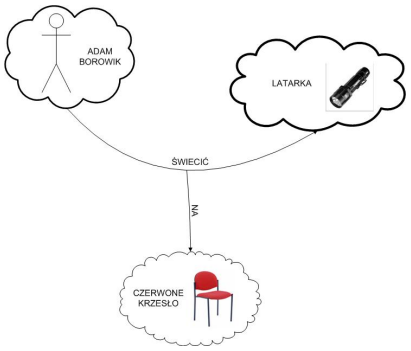
Organizacja bazy dokumentów

2.1 Dokument w pojęciu semantycznym

Zwykle, dokument opisuje pewną część świata w którym się poruszamy. Konkretniej, składa się z pewnych prostych zdań (np. *Adam świeci latarką.*) opisujących związek między pewnymi podmiotami realnie istniejącymi w świecie (lub lepiej - związek między pewnymi URI). Ich ewentualne rozwinięcia (np. *Adam Borowik świeci latarką na czerwone krzesło.*) sprowadzają się tylko do precyzowania tego związku. Obrazowo zostało to przedstawione odpowiednio na rysunkach 2.1 oraz 2.2.

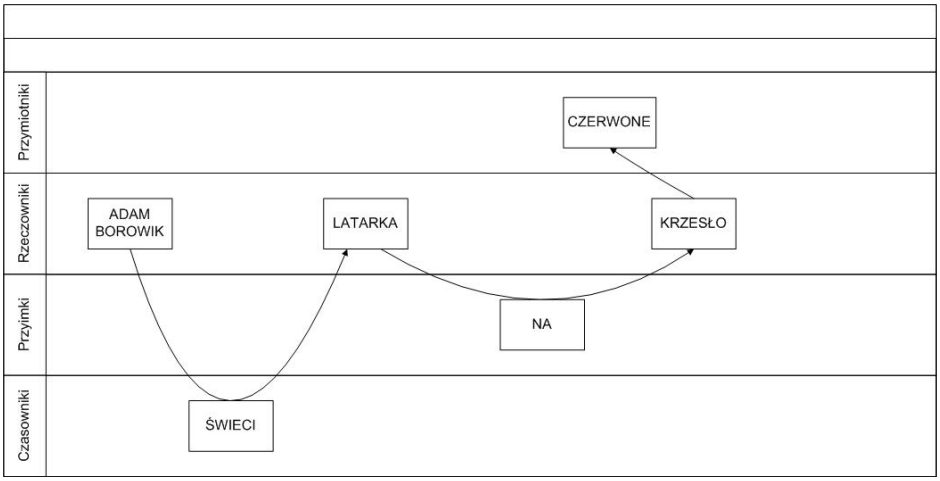


Rysunek 2.1: Semantycznie rozumiane zdanie "Adam świeci latarką".



Rysunek 2.2: Semantycznie rozumiane zdanie "Adam Borowik świeci latarką na czerwone krzesło".

Rysunek 2.3 przedstawia natomiast rozkład gramatyczny zdania rozwiniętego. Widać zasadnicze różnice między obydwoma podejściami. Generalnie można je jednak sprowadzić do następującego stwierdzenia: *Podczas, gdy*



Rysunek 2.3: Gramatyka zdania "Adam Borowik świeci latarką na czerwone krzesło".

rozbicie semantyczne zdania dzieli je na podmioty i akcje, które zachodzą między podmiotami, rozbicie gramatyczne dzieli je na wyrazy odpowiednio ze sobą połączone. Generuje to nam szereg różnych wniosków. Większość z nich nie będzie rozpatrywana w tej pracy. Jednym z takich wniosków jest fakt, iż obie wersje mają strukturę zbliżoną do grafowej, w której wierzchołkami są URI lub wyrazy.
Taka reprezentacja w istocie bardzo trafnie odzwierciedla powiązania między wyrazami, jednakże analiza gramatyczna języka jest trudnym i czasochłonnym zadaniem. Znacznie prostszym sposobem przedstawienia struktury zdań w formie grafowej jest wykorzystanie bliskości wyrazów. W przeważającej liczbie przypadków grafy uzyskane obiema metodami są do siebie zbliżone.

2.2 Indeksowanie

Istnieje wiele czynników, które wpływają na zadanie indeksowania. Niektóre z nich wymienione są poniżej.

Złączenie określa sposób dodawania i uaktualniania danych, które istnieją w bazie indeksów.

Zapisywanie indeksu, który następnie podlega filtrowaniu przez algorytm wyszukujący.

Wielkość indeksu, czyli ilość powierzchni dyskowej do wykorzystania na bazę danych indeksów.

Szybkość przeglądania indeksu, czyli czas potrzebny na znalezienie szukanego wyrażenia.

Przechowywanie indeksu w czasie.

Tolerancja na błędy indeksu w obu sensach - semantycznym oraz bazodanowym.

W zależności od tego, który czynnik jest ważniejszy w konkretnym przypadku, można indeksować na różnorodne sposoby. Poniżej przedstawione zostaną dwa elementarne sposoby indeksowania, na których będzie opierała się autorska koncepcja. Analizowane przykłady będą opierać się na następujących trzech dokumentach:

Dokument 1: *Dzisiaj pięknie świeci słońce.*
Dokument 2: *Portfel Zenka świeci pustkami.*
Dokument 3: *Lampa w pokoju pięknie świeci.*

2.2.1 Przykłady sposobów indeksowania

Forward indexing

Forward indexing w skrócie polega na przypisywaniu do dokumentu listy słów w nim występujących.

Dokument 1	dzisiaj, pięknie, świeci, słońce
Dokument 2	portfel, zenka, świeci, pustkami
Dokument 3	lampa, w, pokoju, pięknie, świeci

Składają się na niego więc pary dokument-słowo, gdzie słowo należy do dokumentu.
Charakteryzuje on się prostotą oraz szybkością sprawdzania, czy dany wyraz występuje w konkretnym dokumencie. Często spotykana w takiej reprezentacji indeksu jest ilość wystąpień wyrazów.
Prosta do wykonania jest transformacja na Inverted indexing, który jest opisany poniżej.

Inverted indexing

Inverted indexing polega na przypisywaniu do słów listy dokumentów, w których występują.

pięknie	Dokument 1, Dokument 3
świeci	Dokument 1, Dokument 2, Dokument 3
lampa	Dokument 3

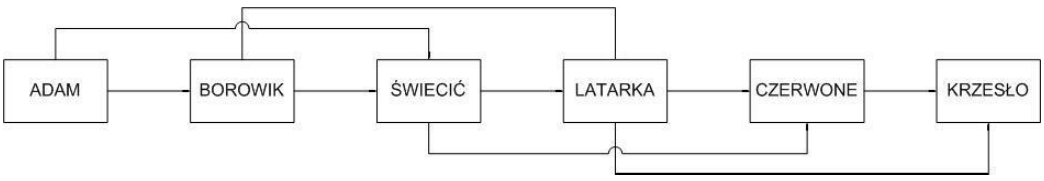
Charakteryzuje on się prostotą oraz szybkością sprawdzania, który dokument posiada dany wyraz.
Bardzo często wykorzystywana jest taka forma indeksu w silnikach wyszukiwarek, w których sprawdzamy po prostu czy pewne wyrazy należą do podanych dokumentów, i na tym opieramy nasze wyniki wyszukiwań. Jednocześnie wyszukiwarka może automatycznie sprawdzać inne dokumenty w których występuje dany wyraz.
Podobnie jak w Forward Indexing opisanym wcześniej, istnieją pewne rozszerzenia tej metody. Do najczęściej spotykanych należą częstość występowania słowa w dokumencie oraz jego pozycja (zwykle względem otagowania HTML). Oba pozwalają na lepsze sortowanie wyników.

2.2.2 Własna koncepcja indeksowania

Przedstawione przez nas rozwiązanie będzie hybrydą dwóch powyżej opisanych form indeksowania. Dzięki temu podstawowe cechy tych form zostaną zachowane, zwiększy się natomiast funkcjonalność indeksowania.

1. Dla każdego dokumentu tworzona jest lista słów, jak w metodzie forward indexing.
2. Dla każdego słowa z listy znajdowana jest jego podstawowa forma gramatyczna i zwiększana jest o 1 liczba wystąpień danego wyrazu w dokumencie.
3. Dla każdego wyrazu uaktualniana jest lista dokumentów, w których on występuje (inverted indexing), przy uwzględnieniu częstości wystąpienia.
4. Dokonywane jest wyodrębnienie zdań w każdym dokumencie.
5. Do dokumentu przyporządkowuje się kategorię na podstawie częstości występowanych wyrazów.
6. Dla każdego zdania dokonywana jest analiza jego struktury i tworzone są relacje między wyrazami zgodnie z kolejnością występowania wyrazów. Relacje te pozwolą na bardziej trafne wyszukiwanie.

W ten sposób otrzymuje się graf nieskierowany, którego wierzchołkami są słowa, a krawędzie określają sąsiedztwo tych słów w dokumencie.



Rysunek 2.4: Postać grafowa naszego indeksowania zdania "Adam Borowik świeci latarką na czerwone krzesło".

2.2.3 Formalizacja procesu indeksowania

Budowanie indeksu będzie opierać się na trzech następujących funkcjach:

- $\delta : D \rightarrow S'$, gdzie $S' = \{s_1, s_2, ..., s_n\}$, $s_i \in S$, $i = 1..n$,
  która zwraca listę słów S' ze słownika S występujących w dokumencie d;

- $\varphi : S' \rightarrow T'$, gdzie $\forall s_i \in S', s_i \in S$ i $\forall t_i \in T', t_i \in T, T' = \{t^{(1)}, t^{(2)}, ..., t^{(T_l)}\}$, gdzie T_l - liczba termów na liście. Funkcja ta zamienia wyraz ze słownika S, znajdujący się na liście S' na term ze zbioru T;

- $\gamma : T' \rightarrow G$, funkcja, która zamienia listę termów na graf.
  Dla kolejnego termu $t^{(i)} \in T'$, sprawdza się czy istnieje wierzchołek grafu G o etykiecie $t^{(i)}$. Jeśli nie, jest dodawany do grafu jako v_k, tworzona jest pętla $(v_k, v_k, 1)$ oraz dla $m \in \{1, 2, .., 5\}$ szukany jest wierzchołek v_p o etykiecie $t^{(i-m)}$ i - jeśli taki wierzchołek istnieje - tworzona jest krawędź $(v_k, v_p, \frac{1}{m})$.
  Jeśli natomiast istnieje wierzchołek v_k o etykiecie $t^{(i)}$, to znajdowana jest krawędź (v_k, v_k) i zwiększana jest jej etykieta o 1. Dla każdego $m \in \{1, 2, .., 5\}$ jeśli istnieje krawędź między wierzchołkami v_k i v_p o etykiecie $t^{(i-m)}$, to zwiększana jest wartość etykiety krawędzi o $\frac{1}{m}$; w przeciwnym przypadku tworzona jest krawędź $(v_k, v_p, \frac{1}{m})$. Wartości krawędzi wszystkich etykiet grafu G zostają znormalizowane, przy czym krawędzi o etykiecie z największą wartością zostaje przypisana nowa wartość równa 15, a krawędzi o etykiecie z najmniejszą liczbą zostaje przypisane 1.

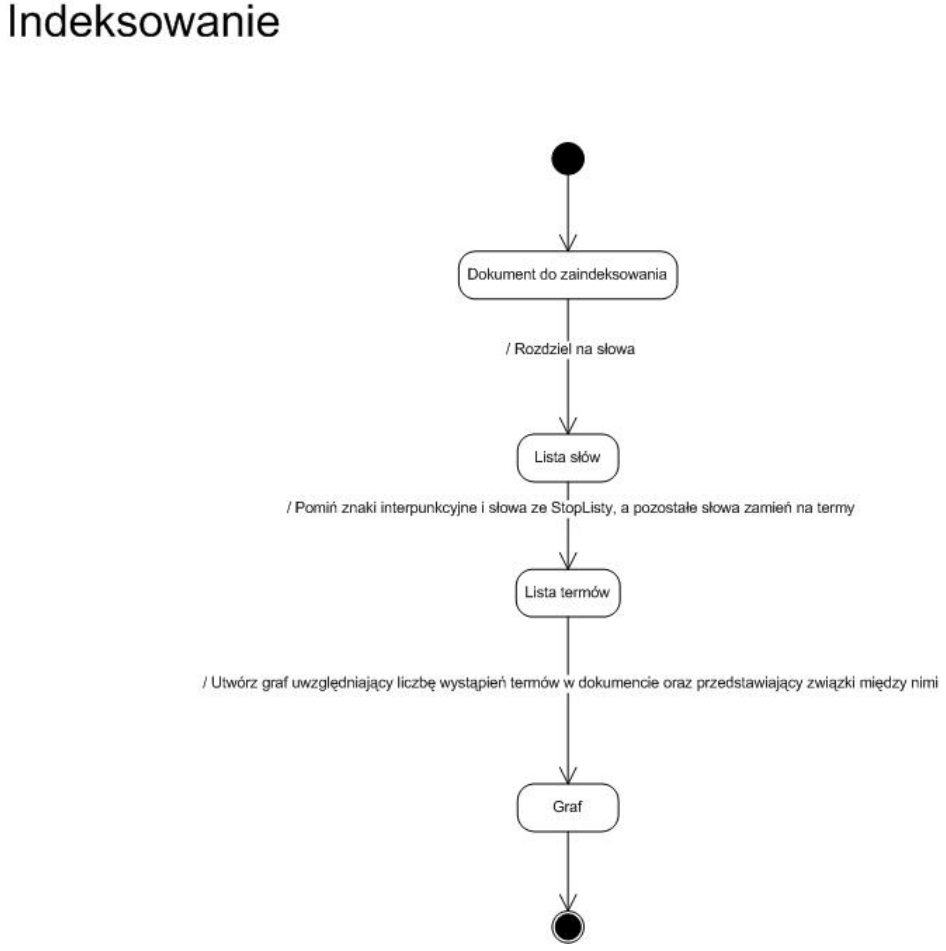
- $\mu : G \rightarrow K$ - funkcja przypisująca dokument do danej kategorii.
  Tworzony jest wektor tymczasowy x o rozmiarze R, gdzie R - ilość kategorii. Dla kazdego $v \in V$ za pomocą funkcji λ sprawdzane są kategorie, do których może należeć term i dodawana jest wartość etykiety krawędzi (v, v) do pól w wektorze x, oznaczających te właśnie kategorie.

Kategoria, która uzyska największą wartość, oznacza kategorię, do której należy graf.

Na początku procesu otrzymuje się na wejściu nowy, niezaindeksowany dokument $d_k \in D$.
Za pomocą funkcji δ tworzy się listę słów występujących w dokumencie.
Następnie z wykorzystaniem funkcji φ dokonuje się przekształcenia każdego słowa z listy na odpowiadający mu term (podstawowa forma gramatyczna). Ponadto w wyniku działania tej funkcji zostają pominięte znaki interpunkcyjne oraz wyrazy z tzw. STOP LISTY.
W kolejnym kroku, za pomocą funkcji γ tworzony jest właściwy graf, będący reprezentacją dokumentu, zawierający liczbę wystąpień danych termów w dokumencie, oraz związki między termami z przypisanymi wagami.

2.2.4 Schemat UML indeksowania

Proces indeksowania w formie UML przedstawia rysunek 2.5.



Rysunek 2.5: Indeksowanie.

2.3 Struktura bazy

Dane w bazie systemu będą przechowywane w następujących tabelach:

- Wyrazy - tabela zawierająca słowa w podstawowej formie gramatycznej.

ID-WYR	Wyraz
1	słońce
2	...

- Odmiany - pomocnicza tabela wskazująca podstawową formę wyrazu.

ID-ODM	ID-WYR	Wyraz
1	1	słońcem
2	1	słońcu
3	...	...

- Kategorie - tabela zawierająca kategorie występujące w profilu użytkownika.

ID-KAT	Kategoria
1	Sport
2	...

- Dokumenty - tabela zawierająca odnośniki URL do dokumentów.

ID-DOK	URL	ID-KAT
1	www.example.com/ex.txt	6
2	...	...

- Historia - tabela informująca o zapytaniach stawianych przez danych użytkowników.

ID-HIST	ID-UZYT	Zapytanie
1	1	Dom
2	1	Czarny pies
3	...	...

- Ranking - tabela relacyjna z ocenami dokumentów w powiązaniu z poszczególnymi zapytaniami.

ID-RANK	ID-ZAP	ID-DOK	Ocena
1	5	6	6787
2	9	2	-13
3	...	...	...

Rozdział 3

Mechanizm wyszukiwania

3.1 Meta-język wyszukiwania

W wyszukiwarce dostępne będą następujące metody wyszukiwania:

- *słowo* - dokumenty, w których występuje *słowo*,
- *słowo1 słowo2* == (*słowo1 słowo2*) - dokumenty, w których występują oba słowa połączone ze sobą,
- (*słowo1*) (*słowo2*) - dokumenty, w których występują oba słowa, niekoniecznie połączone ze sobą,
- (*słowo1*) NOT(*słowo2*) - dokumenty, w których występuje *słowo1*, ale nie *słowo2*,
- (*słowo1*) OR (*słowo2*) - dokumenty, w których występuje *słowo1* lub *słowo2*.

3.2 Silnik wyszukiwania

Wprowadźmy kolejną funkcję, która przyporządkowuje danemu termowi kategorie, do których może należeć (korzysta przy tym ze słownika WordNet); oznaczmy ją jako $\lambda : T \rightarrow K^r$, gdzie r - liczba kategorii.

Wyszukiwanie: słowo
Dane jest zapytanie z ze zbioru Z, gdzie $z = s \in S$ (jest pojedynczym wyrazem).
Z wykorzystaniem funkcji δ, zapytanie sprowadzane jest do termu t.

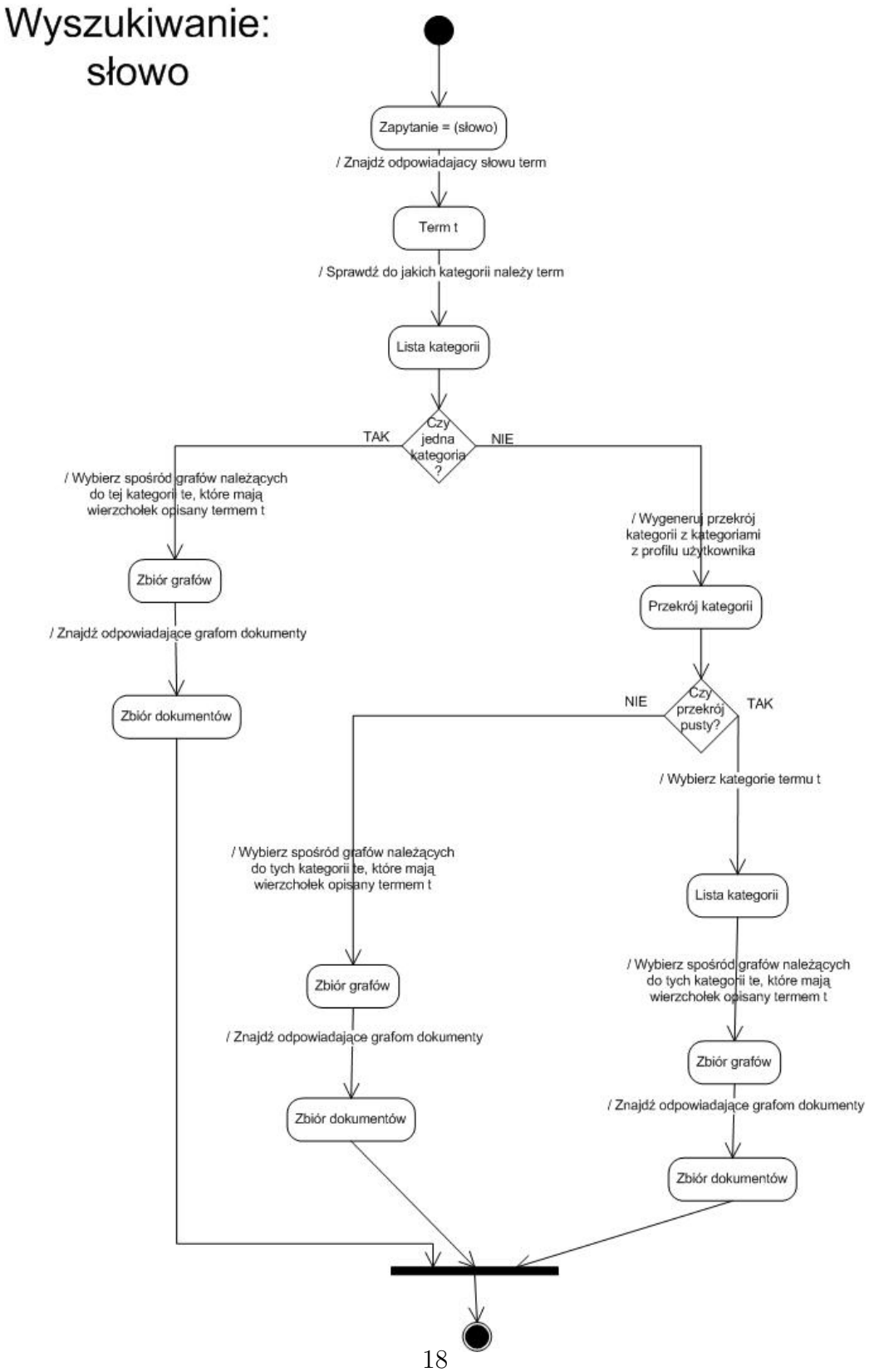
Za pomocą funkcji λ sprawdza się, do jakich kategorii należy term. Jeśli należy do jednej kategorii k, to wyszukiwanie odbywa się wśród dokumentów z tej właśnie kategorii. Jeśli natomiast należy do więcej niż jednej, sprawdzając w profilu użytkownika p listy p_k i p_l określa się, która z tych kategorii jest najbardziej interesująca dla użytkownika. W zależności od rezultatu szukanie odbywa się we wszystkich kategoriach, do których należy term, bądź w jakiejś ich części.
Oznaczmy $D' \subseteq D$ jako podzbiór dokumentów, w których ostatecznie nastąpi wyszukiwanie. Proces wyszukiwania opiera się na grafach $g \in G$. Zwracany jest zbiór dokumentów, w których grafach występuje wierzchołek zaetykietowany przez szukany term.
Schemat ten w formie UML przedstawia rysunek 3.1.

Wyszukiwanie: słowo1 słowo2 == (słowo1 słowo2)

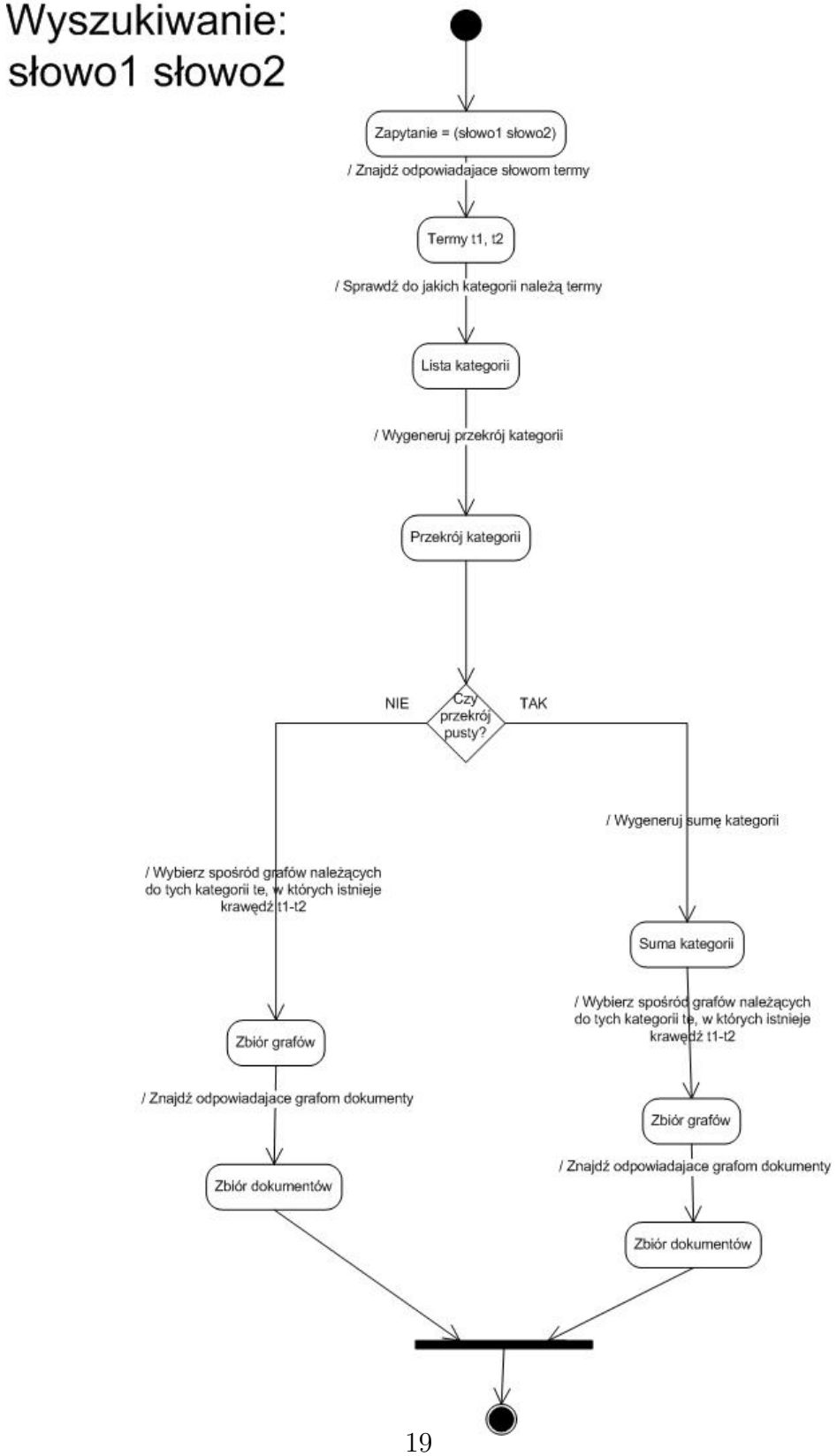
Dane jest zapytanie z ze zbioru Z, gdzie $z = (s_1 \cup s_2) \in S \times S$.
Z wykorzystaniem funkcji δ, zapytanie sprowadzane jest do dwóch termów t_1 i t_2.
Za pomocą funkcji λ sprawdza się, do jakich kategorii należą termy. Tworzony jest przekrój wyznaczonych kategorii. Jeśli jest niepusty, to po nim odbywa się wyszukiwanie, a jeśli nie to wyszukuje się po sumie kategorii. Oznaczmy $D' \subseteq D$ jako podzbiór dokumentów, w których ostatecznie nastąpi wyszukiwanie. Proces wyszukiwania opiera się na grafach $g \in G$. Zwracany jest zbiór dokumentów, w których grafach występuje krawędź łącząca dwa termy z zapytania.
Schemat ten w formie UML przedstawia rysunek 3.2.

Wyszukiwanie: słowo1 słowo2 słowo3 ... słowo$n$

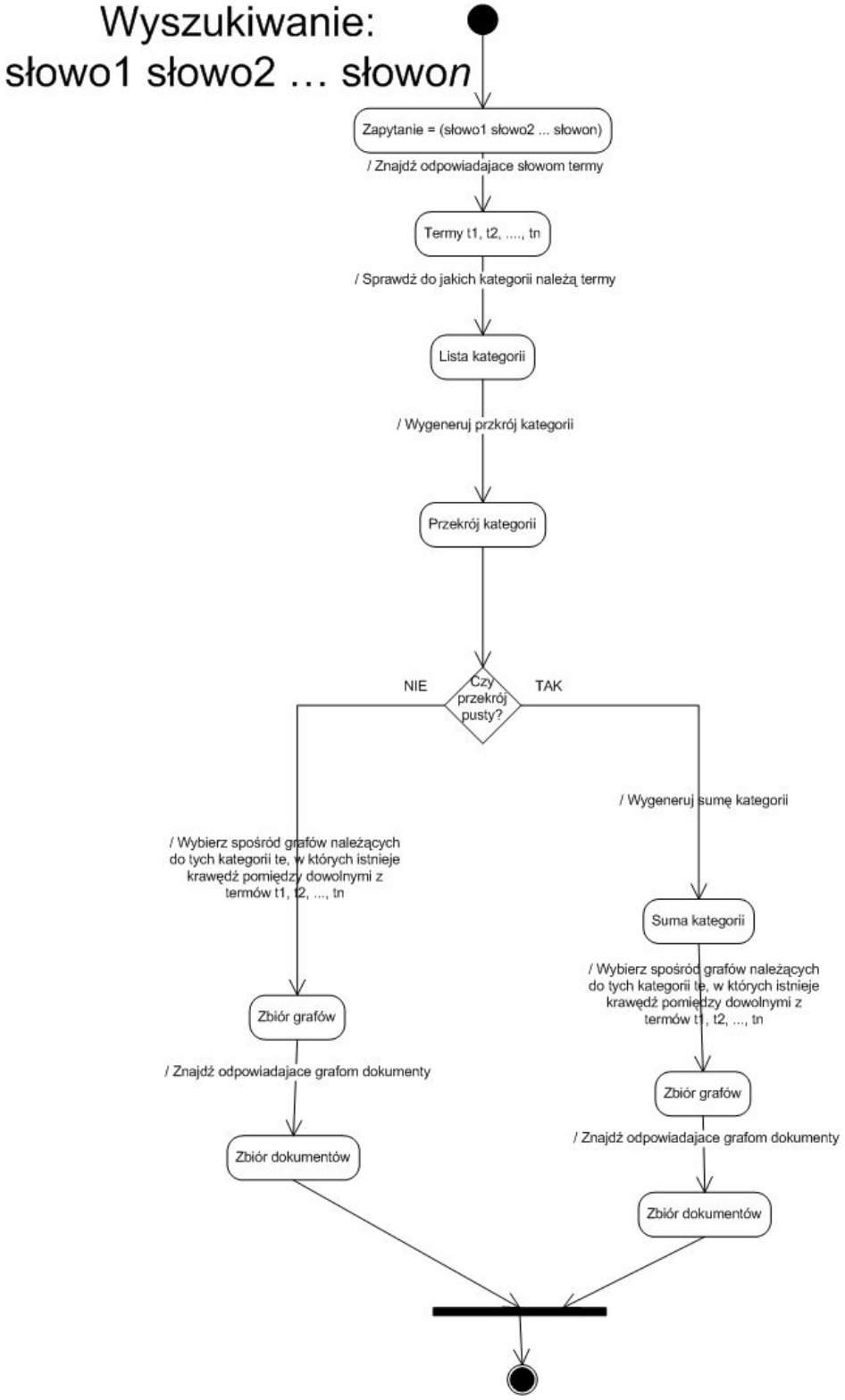
Dane jest zapytanie z ze zbioru Z, gdzie $z = (s_1 \cup s_2 \cup ... \cup s_n) \in S \times S \times ... \times S$.
Z wykorzystaniem funkcji δ, zapytanie sprowadzane jest do dwóch termów $t_1, t_2, ..., t_n$.
Za pomocą funkcji λ sprawdza się, do jakich kategorii należą termy. Tworzony jest przekrój wyznaczonych kategorii. Jeśli jest niepusty, to po nim odbywa się wyszukiwanie, a jeśli nie to wyszukuje się po sumie kategorii. Oznaczmy $D' \subseteq D$ jako podzbiór dokumentów, w których ostatecznie nastąpi wyszukiwanie. Proces wyszukiwania opiera się na grafach $g \in G$. Zwracany jest zbiór dokumentów, w których grafach występuje krawędź pomiędzy dowolnymi termami z zapytania.
Schemat ten w formie UML przedstawia rysunek 3.3.



Rysunek 3.1: Wyszukiwanie: słowo.



Rysunek 3.2: Wyszukiwanie: słowo1 słowo2.



Rysunek 3.3: Wyszukiwanie: słowo1 słowo2 słowo3 ... słowo$n$.

Wyszukiwanie: (słowo1) (słowo2)

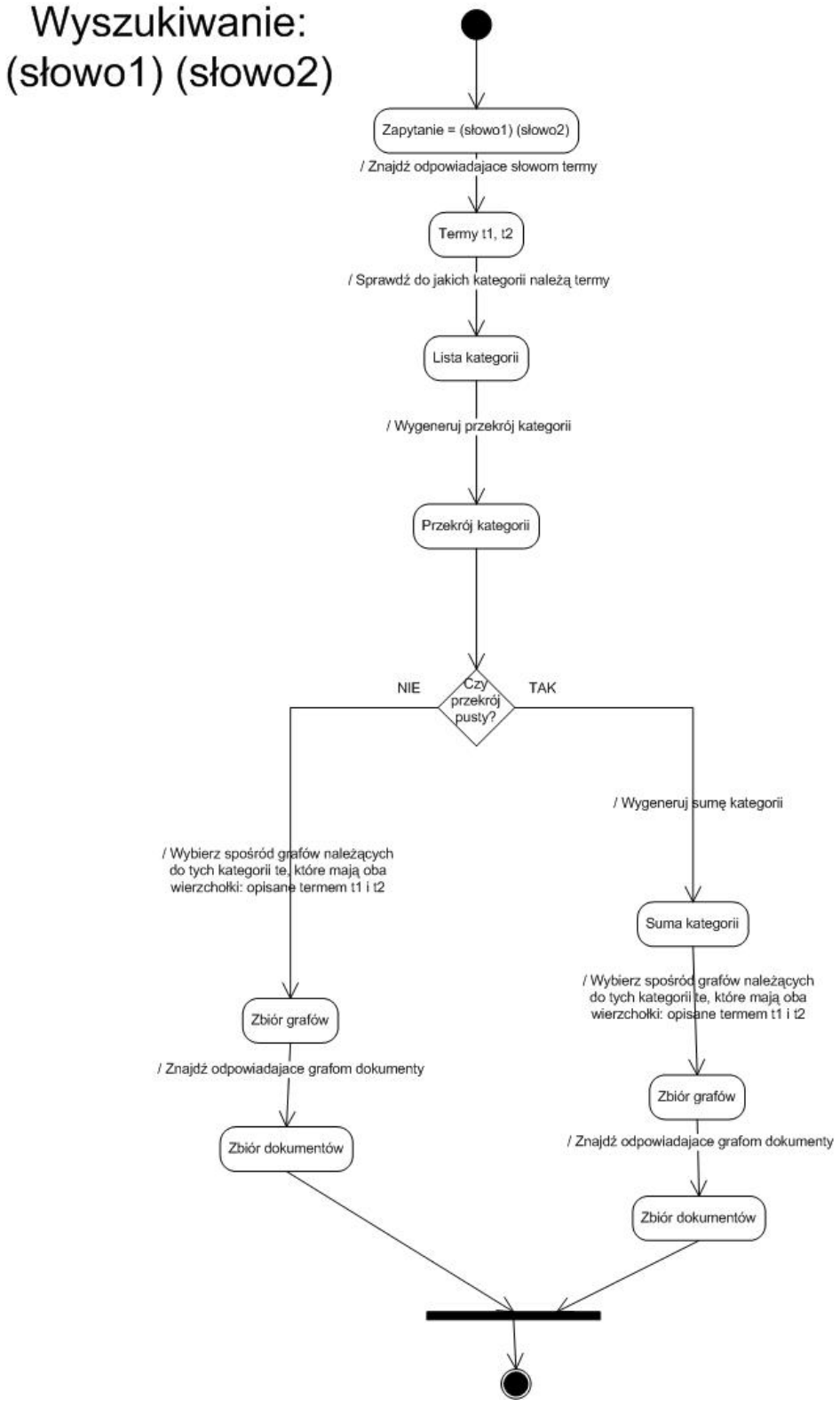
Dane jest zapytanie z ze zbioru Z, gdzie $z = (s_1 \cup s_2) \in S \times S$.
Z wykorzystaniem funkcji δ, zapytanie sprowadzane jest do dwóch termów t_1 i t_2.
Za pomocą funkcji λ sprawdza się, do jakich kategorii należą termy. Tworzony jest przekrój wyznaczonych kategorii. Jeśli jest niepusty, to po nim odbywa się wyszukiwanie, a jeśli nie to wyszukuje się po sumie kategorii. Oznaczmy $D' \subseteq D$ jako podzbiór dokumentów, w których ostatecznie nastąpi wyszukiwanie. Proces wyszukiwania opiera się na grafach $g \in G$. Zwracany jest zbiór dokumentów, w których grafach występują szukane słowa, niezaleznie od tego czy są w związku ze sobą czy też nie.
Schemat ten w formie UML przedstawia rysunek 3.4.

Wyszukiwanie: (słowo1) OR (słowo2)

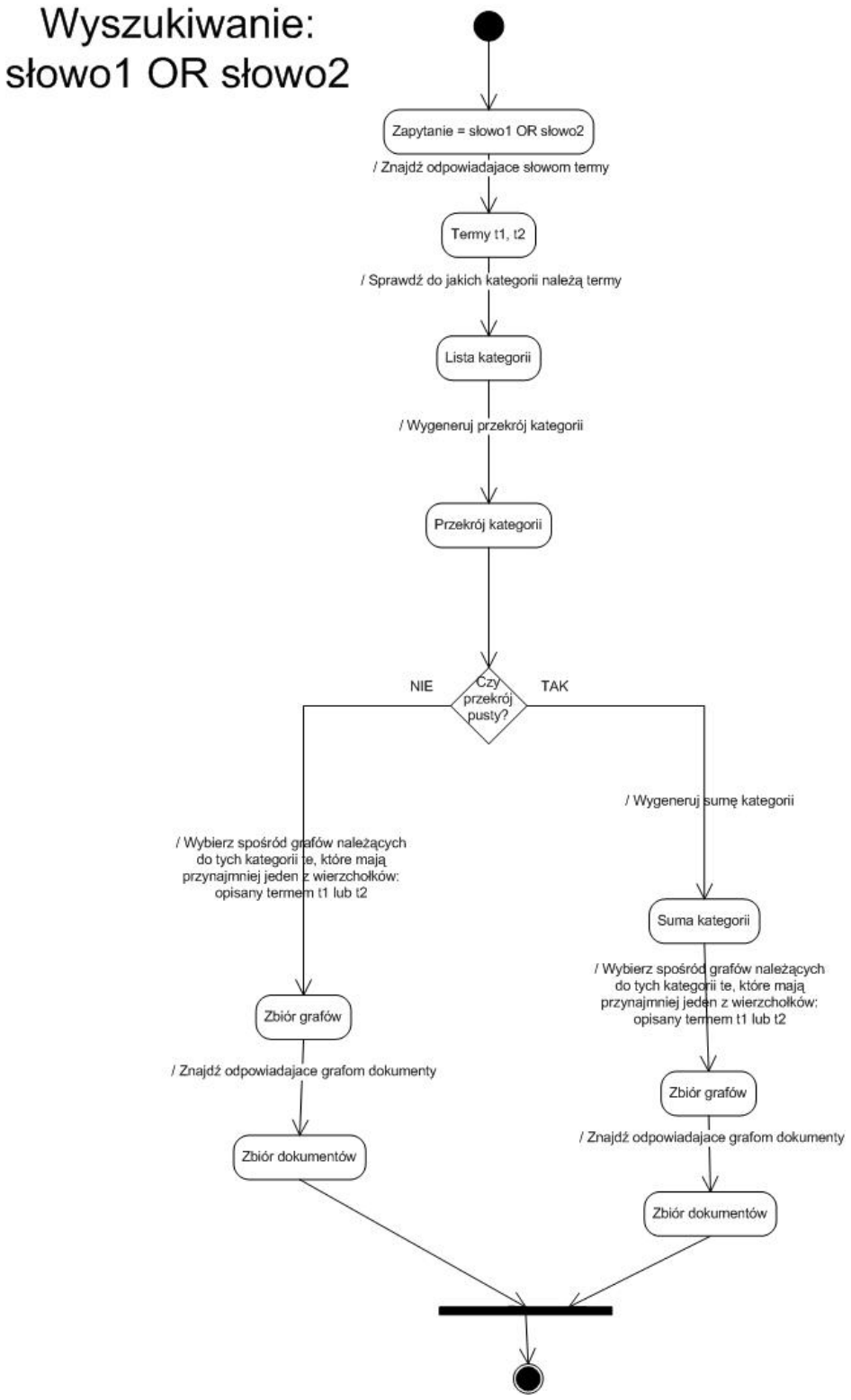
Dane jest zapytanie z ze zbioru Z, gdzie $z = (s_1 \cup s_2) \in S \times S$.
Z wykorzystaniem funkcji δ, zapytanie sprowadzane jest do dwóch termów t_1 i t_2.
Za pomocą funkcji λ sprawdza się, do jakich kategorii należą termy. Tworzony jest przekrój wyznaczonych kategorii. Jeśli jest niepusty, to po nim odbywa się wyszukiwanie, a jeśli nie to wyszukuje się po sumie kategorii. Oznaczmy $D' \subseteq D$ jako podzbiór dokumentów, w których ostatecznie nastąpi wyszukiwanie. Proces wyszukiwania opiera się na grafach $g \in G$. Zwracany jest zbiór dokumentów, w których grafach występuje dowolne ze słów z zapytania.
Schemat ten w formie UML przedstawia rysunek 3.5.

Wyszukiwanie: (słowo1) NOT(słowo2)

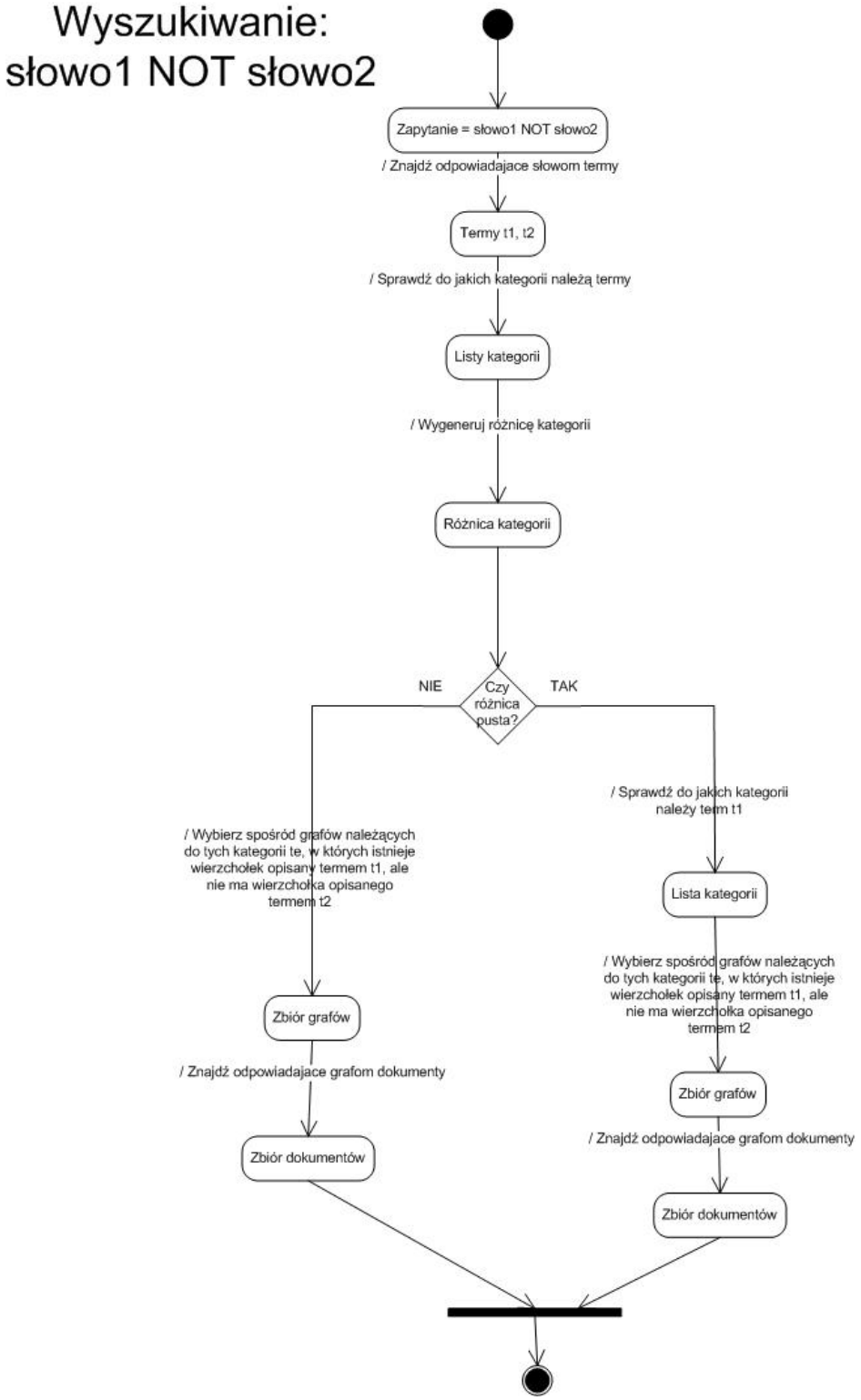
Dane jest zapytanie z ze zbioru Z, gdzie $z = (s_1 \cup s_2) \in S \times S$.
Z wykorzystaniem funkcji δ, zapytanie sprowadzane jest do dwóch termów t_1 i t_2.
Za pomocą funkcji λ sprawdza się, do jakich kategorii należą termy. Tworzona jest różnica wyznaczonych kategorii. Jeśli jest niepusta, to po niej odbywa się wyszukiwanie, a jeśli nie to wyszukuje się po kategoriach, do których należy term t_1. Oznaczmy $D' \subseteq D$ jako podzbiór dokumentów, w których ostatecznie nastąpi wyszukiwanie. Proces wyszukiwania opiera się na grafach $g \in G$. Zwracany jest zbiór dokumentów, w których grafach występuje pierwszy wyraz, ale nie ma drugiego z podanych.
Schemat ten w formie UML przedstawia rysunek 3.6.



Rysunek 3.4: Wyszukiwanie: (słowo1) (słowo2).



Rysunek 3.5: Wyszukiwanie: (słowo1) OR (słowo2).



Rysunek 3.6: Wyszukiwanie: (słowo1) NOT (słowo2).

3.3 Pozycjonowanie

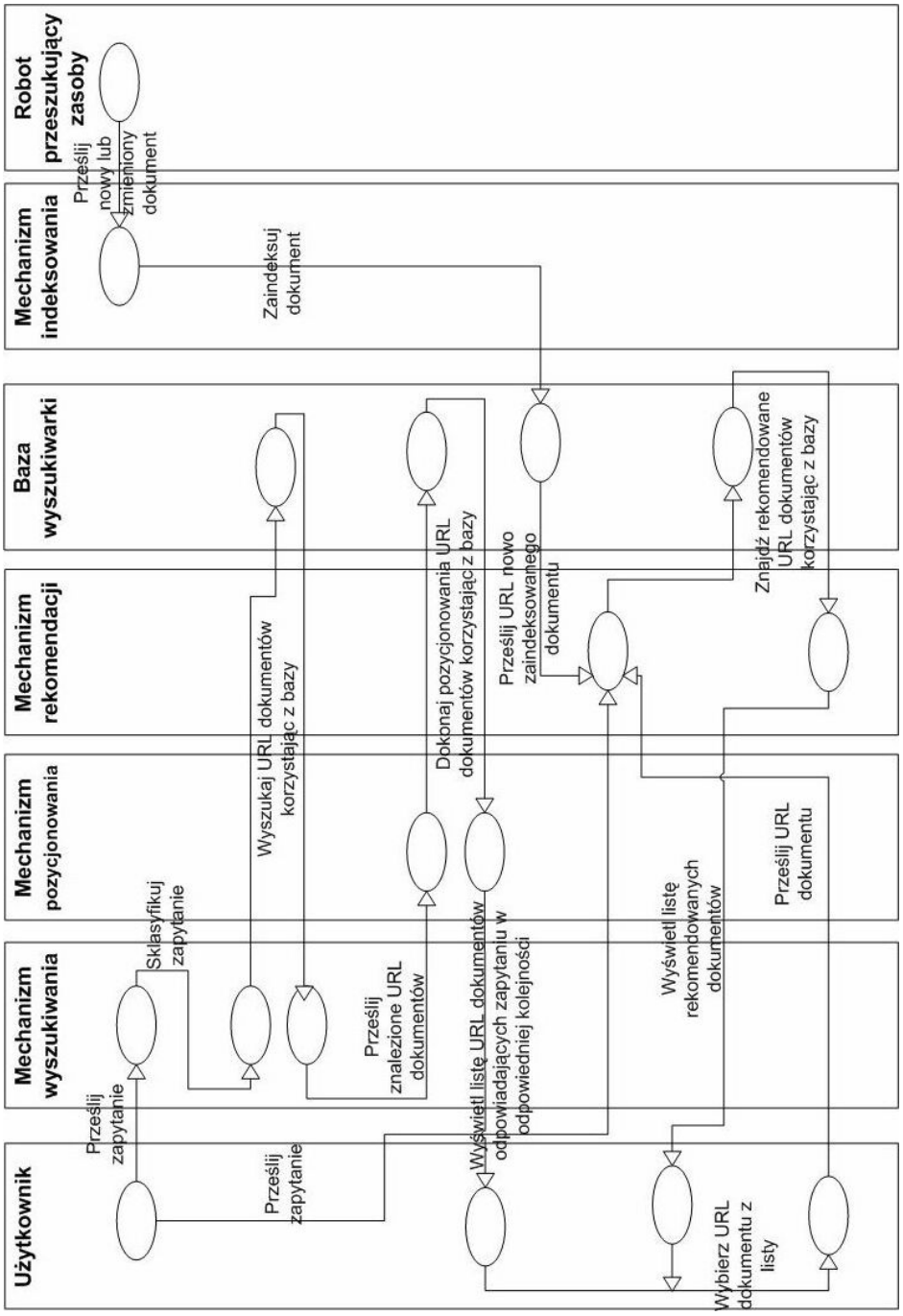
Przy wyświetlaniu wyników wyszukiwań będą brane pod uwagę takie parametry jak:

- znormalizowana wartość wystąpień szukanego termu/związku termów w dokumencie (waga w grafie) - w,
- ranking publiczny dokumentu (r)

Do wyznaczenia pozycji dokumentu w wyświetlonych wynikach, zastosowana będzie odpowiednio zdefiniowana funkcja $\psi_i(w_i^{(1)}, w_i^{(2)}, ..., w_i^{(I)}, r_i)$, gdzie I - liczba krawędzi w grafie g między krawędziami oznaczającymi różne termy z zapytania.

3.4 Przebieg procesu wyszukiwania

W formie UML na rysunku 3.7 można zobaczyć przebieg procesu wyszukiwania oraz indeksowania.



Rysunek 3.7: Diagram przypadków użycia.

Rozdział 4

System rekomendacji

4.1 Profil użytkownika

Po wyselekcjonowaniu dokumentów spełniających zapytanie użytkownika będą one porządkowane z uwzględnieniem danych zawarte w jego profilu.

4.1.1 Struktura profilu użytkownika

Profil użytkownika zawiera:

- dane identyfikujące użytkownika (login, hasło, e-mail, itp.)
- zainteresowania, które wybiera z podanej mu listy:
  - biznes
  - dom i rodzina
  - fotografia
  - komputery i internet
  - książki
  - kuchnia
  - kultura, sztuka, film
  - media masowe
  - moda
  - motoryzacja
  - muzyka
  - nauka i technika

  - polityka
  - przyroda
  - sport
  - turystyka
  - uroda
  - zdrowie

- ranking osobisty (lista kategorii z przypisanymi wartościami liczbowymi odzwierciedlającymi zainteresowanie użytkownika daną kategorią).

4.2 Rankingi

Użytkownik ma możliwość oceny każdego przeglądanego dokumentu. W ten sposób modyfikowane będą dane zarówno w profilu użytkownika, jak i w rankingu samego dokumentu.

Ranking prywatny definiuje, w jakim stopniu użytkownik jest zainteresowany daną tematyką. Początkowo jest to lista kategorii z przypisanymi domyślnymi wartościami "0".
Rankingiem dokumentu jest liczba r_i, która określa stopień przydatności dokumentu w przypisanej do niego kategorii.
Użytkownik, po przeglądnięciu dokumentu może wybrać jedną z poniższych opcji (domyślnie opcja trzecia):

- dokument jest interesujący - zostaje dopisany punkt do dziedziny, z której pochodzi dokument w p_l w profilu użytkownika oraz punkt do rankingu dokumentu;

- interesująca jest dziedzina, natomiast dokument jest nieprzydatny - zostaje dopisany punkt w p_l w profilu użytkownika do dziedziny, z której pochodzi dokument, natomiast w rankingu dokumentu zostaje odjęty punkt;

- dziedzina nie jest dla mnie interesująca - brak zmian w rankingach.

4.3 Typy rekomendacji

4.3.1 Rekomendacja typu A - nowo zaindeksowane dokumenty

Na stronie głównej wyszukiwarki osobno ukazywana jest tabela z pięcioma ostatnio zaindeksowanymi dokumentami d_1, d_2, d_3, d_4, d_5 przypisanymi do kategorii k_j, która w profilu użytkownika posiada największą wagę.

4.3.2 Rekomendacja typu B - na podstawie historii zapytań innych użytkowników

System otrzymuje zapytanie z. Znajdowani są użytkownicy, u których w historii zapytań istnieje już zapytanie z. Następnie spośród znalezionych użytkowników, wybierani są ci, którzy mają zainteresowania podobne do zainteresowań aktualnie przeszukującego. Będą to tacy użytkownicy, w których profilach wagi kategorii są najbardziej zbliżone do wag w profilu użytkownika aktualnie wyszukującego dokumenty.
W następnym kroku sprawdza się, jakie zapytania występowały w historii wyselekcjonowanych użytkowników zaraz po zapytaniu z. Rekomendowanych jest 5 najczęściej występujących wyników - zapytania: z_1, z_2, z_3, z_4, z_5.

4.3.3 Rekomendacja typu C - wykorzystanie synonimów

System otrzymuje zapytanie z, które zgodnie z opisanym wcześniej algorytmem (za pomocą funkcji φ) zamienia na listę termów: $t_1, t_2, ..., t_n$, gdzie n-liczba termów występujących w zapytaniu.
Wyszukiwarka sugeruje n wersji zmiany zapytania podmieniając kolejne termy na ich synonimy w ten sposób, że w każdym rekomendowanym zapytaniu podmieniony jest inny term. W tym kroku algorytm opiera się na słowniku WordNet.

4.3.4 Rekomendacja typu D - na podstawie podobieństwa dokumentów

Jak wiadomo, dokumenty przedstawiane są w indeksie w postaci grafów G. Podczas przeglądania dokumentu d_i przez użytkownika, proponuje się przeglądnięcie podobnych dokumentów, tj. takich, których grafy są najbardziej zbliżone do grafu odpowiadającego aktualnie przeglądanemu dokumentowi.

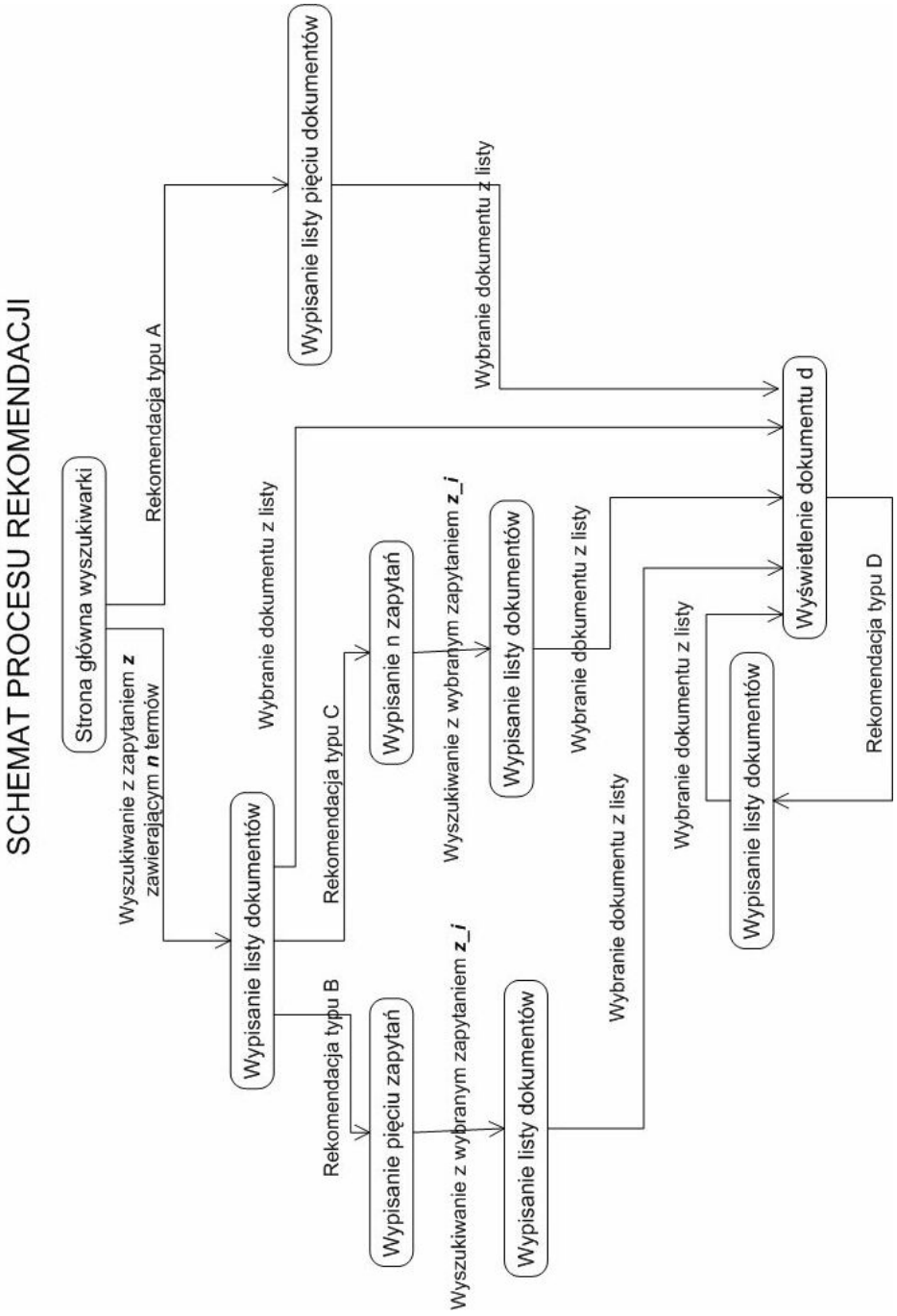
System rekomenduje 5 dokumentów d_1, d_2, d_3, d_4, d_5, których grafy G_1, G_2, G_3, G_4, G_5 są najbliższe G_i odpowiadającemu dokumentowi d_i.
Do zrealizowania tego podejścia należy wprowadzić miarę podobieństwa grafów. Oznaczmy funkcję
$\theta : G \times G \rightarrow N$, która zwraca stopień podobieństwa, zliczając sumę wag krawędzi występujących w obu strukturach.

$$\theta(G_i, G_j) = \sum_{(v_a, v_b, w_{ab}) \in G_i \cap G_j} w_{ab}$$

Zatem algorytm rekomendacji typu D, znajdzie 5 grafów o najwyższej wartości funkcji θ w porównaniu z tym, który odpowiada przeglądanemu aktualnie dokumentowi. Dokumenty odpowiadające wyznaczonym grafom, zostaną zarekomendowane użytkownikowi.

4.4 Diagram UML rekomendacji

Wszystkie cztery typy rekomendacji w formie UML przedstawia rysunek 4.1.



Rysunek 4.1: Proces rekomendacji typu A, B, C i D.

Rozdział 5

Wskazówki do implementacji

Zaproponowany przez nas system wyszukiwania dokumentów tekstowych można stosunkowo prosto zaimplementować używając w zasadzie dowolnego języka programowania. W tym rozdziale zostaną zaproponowane typy danych, które mogą być użyte do implementacji systemu, wraz z uzasadnieniem. Ogólnie przedstawia się też pseudokod głównych funkcji, które zostały opisane we wcześniejszych rozdziałach.

5.1 Funkcje

Główne funkcje które należy zaimplementować są przedstawione poniżej.

5.1.1 GetWords

W formalizacji funkcja opisana jako δ będzie zwracała listę słów, a jej parametrem będzie dokument.

Listing 5.1: Funkcja GetWords

```
function GetWords(string D)
        new queue lista
        new int i, j, lastspace
        new string word
        for i:=0 to length(D)
                if D[i] == " "
                        if i != lastspace+1
                                for j:=lastspace to i
                                        word[j] = D[
                                            lastspace+j]
```

```
10                                      lista.push(word)
11                                      lastspace=i
12                              else
13                                      lastspace=i
14          return list
```

5.1.2 GetTerms

Funkcja opisana w formalizacji jako φ zwraca dla każdego słowa znajdującego się na liście wejściowej oznaczenie (term) mu odpowiadające.

Listing 5.2: Funkcja GetTerms

```
1  function Term(string word)
2          return query("SELECT `ID_WYR` FROM `odmiany` WHERE
               `wyraz`='"+word+"'")
3
4  function GetTerms(list lista)
5          new queue listat
6          new string word
7          repeat
8                  lista.pop(word)
9                  if Term(word) != ""
10                         listat.push(Term(word))
11         until word == ""
12         return listat
```

5.1.3 CreateGraph

Funkcja γ z formalizacji przekształca listę termów będących na liście dokumentów na reprezentację grafu.

Listing 5.3: Funkcja CreateGraph

```
1  function CreateGraph(list listat)
2          new int i, j, last5place
3          new list listag
4          new array last5[5] of string, M[sizeof(listat),
               sizeof(listat)] of float
5          new string term
6          repeat // Tworze tymczasowa macierz.
```

```
7                   listat.pop(term)
8                   if !(listag.exists(term)) // dodanie termu
9                           listag.push(term)
10                  M[listag.place(term),listag.place(term)]+=1
                        // dodanie petli
11                  for i:=0 to 4 // dodanie krawedzi do innych
                        termow
12                          j:=last5place-1-i
13                          if j<0
14                                  j+=5
15                          if last5[i] != ""
16                                  M[listag.place(last5[i]),
                                        listag.place(term)]+=1/(
                                        last5place+i+1)
17                  last5[last5place] = term
18                  last5place+=1   // przesuniecie licznika
19          until term == ""
20          // normalizacja
21          new int temp
22          for i:=0 to sizeof(listat)
23                  for j:=0 to sizeof(listat)
24                          if M[i,j]>temp
25                                  temp:=M[i,j]
26          for i:=0 to sizeof(listat)
27                  for j:=0 to sizeof(listat)
28                          if (floor(M[i,j]*15/temp)==0) and (
                                M[i,j]>0)
29                                  M[i,j]:=floor(M[i,j]*15/
                                        temp)+1
30                          else
31                                  M[i,j]=floor(M[i,j]*15/temp
                                        )
32          // zapisanie w zwiezlej postaci
33          // referencje w sekcji "Indeks"
34          new string code
35          for i:=0 to sizeof(listag)
36                  for j:=i to sizeof(listag)
37                          if j=i
38                                  M[i,j]=listag.tag(listag.
                                        element(i))
39                          else
40                                  code.append(dectobin(M[i,j
                                        ]))
```

```
41          // przerobienie na ASCII
42          bintoasc(code)
43          return listag, code
```

5.1.4 GetCatForTerm

Funkcja λ z formalizacji przekształca term na wektor, który określa do jakiej kategorii należy dany term.

Listing 5.4: Funkcja GetCatForTerm

```
1  function GetCatForTerm(string term)
2          new array Cat[NumberOfCategories]
3          new string categories
4          new int i, j, lastspace
5          categories=query("SELECT * FROM `kategorie` WHERE `
               term`='"+term+"'");
6          for i:=0 to length(categories)
7                  if categories[i] == " "
8                          if lastspace != 0
9                                  Cat[j]=strtoint(categories[
                                       i-1])
10                                 j+=1
11                                 lastspace=i
12                         else
13                                 lastspace=i
14         return Cat
```

5.1.5 GetCatForDoc

Przedstawiona w formalizacji funkcja μ dokonuje konkretnego określenia kategorii dla dokumentu, który jest parametrem funkcji.

Listing 5.5: Funkcja GetCatForDoc

```
1  function GetCatForDoc(list listag)
2          new array Cat[NumberOfCategories]
3          new string categories, term
4          new int i, j, k, lastspace
5          for k:=0 to sizeof(listag)
6                  listag.push(term)
```

```
7                   categories=query("SELECT * FROM `kategorie`
                        WHERE `term`="+term+"");
8                   for i:=0 to length(categories)
9                           if categories[i] == " "
10                                  if lastspace != 0
11                                          Cat[j]+=strtoint(
                                                categories[i-1])
12                                          j+=1
13                                          lastspace=i
14                                  else
15                                          lastspace=i
16          j:=0
17          for i:=0
18                  if Cat[i]>j
19                          j=Cat[i]
20          return i
```

5.2 Indeks

W naszym sposobie indeksowania główną strukturą, która musi zostać zaimplementowana jest graf. Będzie to najczęściej występująca struktura w całym systemie, pozostanie bowiem w relacji 1:1 z każdym dokumentem, jej implementacja więc musi być możliwie jak najbliższa optymalnej.

5.2.1 Sposoby implementacji struktur grafowych

W ogólności, istnieje wiele sposobów reprezentacji grafów.

Macierz połączeń przedstawia graf jako macierz M o wielkości $N \times N$, gdzie N to ilość wierzchołków, a każda klatka $M[m, n]$ to lista krawędzi między wierzchołkami m i n.
W rozpatrywanym przypadku taka macierz będzie miała wielkość $5\,000 \times 5\,000 = 25\,000\,000$ klatek, w których będą przechowywane listy krawędzi wraz z ich etykietami.

Macierz incydencji jest reprezentacją zapisującą graf jako macierz M o wielkości $N \times O$, gdzie O to ilość połączeń między termami.
Wielkość takiej macierzy w tym przypadku to około $5\,000 \times 25\,000 = 125\,000\,000$ klatek, z odpowiednimi etykietami.

Lista sąsiadów zapisuje graf jako tablicę list, gdzie każdy element tablicy odpowiada wierzchołkowi i listowane są wszystkie wierzchołki do niego incydentne.
Dla niniejszego zastosowania, struktura ta będzie miała wielkość $5\,000 \times 20 = 100\,000$ elementów, z odpowiednimi etykietami.

Liczby te wydają się być duże, a przede wszystkim znacznie przekraczają wielkość dokumentu. Reprezentacje te mogą zostać jednak odpowiednio zoptymalizowane.

5.2.2 Optymalizacja reprezentacji grafu

Ograniczenia, które posłużą do przystosowywania najbardziej popularnych reprezentacji do tworzonego systemu są następujące.

- Graf odpowiadający dokumentowi jest zawsze nieskierowany.
- Graf zwykle jest niespójny. Istnieje bardzo wiele wierzchołków które nie mają incydentnej krawędzi.

Macierz połączeń

Dla macierzy połączeń duże znaczenie ma fakt nieskierowanych krawędzi. Dzięki temu wielkość macierzy znacznie się zmniejszy. W rezultacie macierz ta będzie górnotrójkątna bez przekątnej (bardzo rzadkie wyszukiwania dwóch takich samych termów). Każda klatka będzie przedstawiała wagę połączenia dwóch termów. Jej etykietą będzie wartość z zakresu 0 oraz 15 (takimi progami ograniczyliśmy wagi połączeń).
Aby macierz nie przedstawiała wszystkich termów, a tylko te, które były użyte w dokumencie, będzie wymagana krotka z zapisaną kolejnością występowania termów.

Macierz incydencji

Macierz ta będzie macierzą rzadką, z racji faktu, że z reguły istnieje większa liczba wierzchołków niż dwa. Oczywista jest optymalizacja do zbioru trójek (dwa wierzchołki oraz waga), implementowanych jako tabela trójargumentowa, gdzie dwa argumenty odpowiadają za termy, które występują w związku, a trzeci przypisuje mu wagę.
Dalsza optymalizacja polega na wyeliminowaniu powtarzania jednego z argumentów, wskutek czego otrzymuje się reprezentację listową sąsiadów.

Lista sąsiadów

W tej reprezentacji wspomniane ograniczenia nie dają efektu zmniejszenia wielkości. Teoretycznie można wykorzystać fakt nieskierowania w taki sposób, że krawędź istniejąca między wierzchołkami będzie zapisywana tylko raz (np. zapisując wierzchołek o numerze mniejszym jako wierzchołek w tabeli, a o numerze większym na liście). Sprawi to, że czas przeszukiwania indeksu znacznie wzrośnie, nawet jeśli w listach wierzchołki będą uporządkowane.

5.2.3 Ostateczny sposób reprezentacji

Z powyższych powodów, nie bierze się pod uwagę macierzy incydencji.

Macierz połączeń

Zostanie więc wybrany sposób reprezentacji poprzez odpowiednio zmodyfikowane do naszej koncepcji macierze połączeń. Dla dokumentu w takiej reprezentacji na ilość miejsca zajętego przez indeks będzie wpływać wielkość listy, zawierającej ułożone w odpowiedniej kolejności dane na temat wystąpień termów, a także wielkość odpowiedniej reprezentacji macierzy górnotrójkątnej. Pseudokod został przedstawiony na listingu w sekcji 5.1.3.

Wielkość zaproponowanego sposobu reprezentacji

Po analizie pseudokodu wybranego sposobu reprezentacji, można spróbować określić jej wielkość.
Reprezentacja składa się z dwóch części - listy termów występujących w dokumencie oraz ciągu bitów determinujących fakt występowania krawędzi i określających etykiety na krawędziach między poszczególnymi termami.
I tak, dla n termów wielkość:

- listy termów będzie równa sumie wielkości numerów ID termów z tabeli WYRAZY, a także każdy term będzie miał określoną częstość wystąpień. Przeznaczono na to około $4n$ bajtów;
- ciągu bitów będzie równa liczbie klatek niezerowych w pełnej macierzy górnotrójkątnej pomnożonej przez cztery bity, czyli $\frac{n*(n-1)}{4}$.

Dla średniej wielkości dokumentu reprezentacja zajmie około 6267500 bajtów.
Taka reprezentacja zostanie poddana kompresji. Z różnorodności języka naturalnego wynika fakt, że spośród możliwych połączeń między termami, w

rzeczywistości istnieje ich tylko ok. 1%. Można więc założyć, że w ciągu binarnym będzie występowało ok. 99% zer.
Zatem przy odpowiedniej kompresji, np. metodą runów zerowych, a później Huffmana, otrzymany indeks będzie o wiele mniejszy (nawet o kilka rzędów).

Lista sąsiadów

Na wielkość indeksu zapisanego jako lista sąsiadów będzie wpływać ilość termów oraz ilość jego wystąpień w dokumencie.

Wielkość zaproponowanego sposobu reprezentacji

Reprezentacja taka, dla n termów i p połączeń będzie miała wielkość $n * p * 4,5$. Dla średniej wielkości dokumentu wartość ta wyniesie około 450000 bajtów.
Należy jednak pamiętać o fakcie, iż zapisana w ten sposób macierz będzie charakteryzowała się znacznie wolniejszym wyszukiwaniem danych.

Rozdział 6

Podsumowanie

W pracy w sposób jednolity i spójny przedstawiono projekt oryginalnego systemu wyszukiwania dokumentów. Zostały przeanalizowane kolejno wszystkie etapy zarówno konceptualnego, jak i analitycznego konstruowania wyszukiwarki. Przedstawiono formalizację poszczególnych faz tworzenia systemu oraz zaproponowano szereg abstrakcyjnych struktur danych, co w przyszłości pozwoli na sprawną implementację. Na szczególną uwagę zasługuje innowacyjność indeksu, która stwarza możliwość wyszukiwania z uwzględnieniem nowych aspektów. Jednym z nich jest zaproponowany przez nas proces rekomendacji.
Stworzony system można w prosty sposób rozwijać, nadając mu wachlarz nowych możliwości. Przykładowo wyszukiwarka mogłaby posłużyć do odnajdywania plagiatów różnego rodzaju dokumentów. Także zaproponowany proces indeksowania można dowolnie rozszerzać, np. o analizę gramatyczną tekstu.

Bibliografia

[1] Czesław Daniłowicz, Huy Cuong Nguyen, Ngoc Thanh Nguyen: *Model of user profiles and personalization for web-based information retrieval system.*

[2] Mieczysław Alojzy Kłopotek: *Inteligentne wyszukiwarki internetowe*, Akademicka Oficyna Wydawnicza Exit, Warszawa 2001.

[3] Sung-Shun Weng, Yu-Jen Lin: *A study on searching for similar documents based on multiple concepts and distribution of concepts*, Expert Systems with Applications 25 (2003) 355–368.