Politechnika Poznańska
Wydział Informatyki i Zarządzania
Instytut Informatyki

Praca dyplomowa magisterska

RÓWNOLEGŁE ODKRYWANIE REGUŁ ASOCJACYJNYH ZAIMPLEMENTOWANE NA PROCESORY GRAFICZNE

inż. Tomasz Kujawa

Promotor
dr inż. Witold Andrzejwski

Poznań, 2011

Tutaj przychodzi karta pracy dyplomowej;
oryginał wstawiamy do wersji dla archiwum PP, w pozostałych kopiach wstawiamy ksero.

Spis treści

Rozdział 1

Wstęp

Proces informatyzacji przedsiębiorstw, rozpoczęty kilka dekad temu, wprowadził światową gospodarkę na nowe, dotąd nieznane tory rozwoju. Skrócenie procesu produkcyjnego, wprowadzenie kontroli komputerowych, czy też skomputeryzowanych maszyn skróciło i ułatwiło produkcję, a także zarządzanie procesami w firmach i przedsiębiorstwach. Przed ludźmi stanęły możliwości, ale także wyzwania, z którymi nigdy wcześniej nikt nie musiał sobie radzić. Zmiany, jakie nastąpiły przez ostatnie trzy dekady są nieodwracalne i zmuszają informatyków do tworzenia nowych aplikacji, które będą w stanie sprostać wymaganiom im stawianym.

Informatyzacja firm, instytucji oraz innych jednostek organizacyjnych powinna realizować dwa podstawowe cele. Z jednej strony powinna ona usprawniać pracę pojedynczego pracownika poprzez automatyzację realizowanych przez niego rutynowych zadań. Dzięki wykorzystaniu możliwości komputerów działania te powinny być wykonywane szybciej i w sposób bardziej niezawodny. Z drugiej strony celem informatyzacji jest wpływanie na działanie całych firm w wyniku wspomagania decyzji kadry zarządzającej przedsiębiorstwami. Szybka analiza bazująca na pełnej i aktualnej informacji o stanie firmy może ułatwić kadrze zarządzającej podejmowanie trafnych i szybkich decyzje o strategicznym znaczeniu dla rozwoju danego przedsiębiorstwa.

Wprowadzenie komputerów do właściwie każdej przestrzeni ludzkiego życia wpłynęło na wyprodukowanie olbrzymich ilości danych. Reprezentowane są one w sposób umożliwiający ich składowanie i przetwarzanie komputerowe przez aplikacje analityczne. W chwili obecnej ludzkość jest świadkiem eksplozji ilości danych produkowanych przez różnego rodzaju systemy komputerowe. Analiza tych danych przynieść może wymierne korzyści nie tylko w kwestiach finansowych, ale również poznawczych. Dzięki analizie danych zebranych w przeszłości możliwe jest lepsze dopasowanie planów w przyszłości - na tej podstawie planowane mogą być np. akcje marketingowe, czy też promocje w supermarketach spożywczych. Wykorzystanie wiedzy uzyskanej w ten sposób jest niezwykle szerokie i może być użyte w każdym rejonie działalności firmy.

Odkrycie zależności pomiędzy zgromadzonymi danymi bez zastosowania narzędzi informatycznych jest procesem bardzo skomplikowanym i wymagającym do realizacji dużo czasu. Przy obecnej złożoności większości systemów oraz rozmiarom danych produkowanych przez te systemy, kosz czasowy jest na tyle duży, że ręczna analiza tych danych stała się niemożliwa. Dlatego też tworzone są narzędzia umożliwiające odkrywanie prawidłowości w dużych zbiorach danych, by człowiek na tej podstawie mógł podejmować decyzje i wyciągać wnioski.

Dział informatyki, który zajmuje się odkrywaniem ukrytych dla człowieka prawidłowości i reguł w danych nazywa się eksploracją danych (ang. *Data Mining*), który jest jednym z etapów procesu *odkrywania wiedzy z baz danych* (*KDD*, ang. *Knowledge Discovery in Databases*). Proces odkrywania wiedzy w bazach danych obejmują zwykle działania bardziej złożone niż tylko eksploracja

danych. Eksploracja danych to proces odkrywania wiedzy w postaci nowych, użytecznych, poprawnych i zrozumiałych wzorców w bardzo dużych wolumenach danych [FPSSU96]. Możliwości stosowania technik eksploracji danych w praktyce, wymagają efektywnych metod przeszukiwania ogromnych plików lub baz danych. Warto przy tym wspomnieć, że tego typu technologie nie są w chwili obecnej dobrze zintegrowane z systemami zarządzania bazami danych.

Eksploracja danych (w literaturze spotkać można również określenie drążenie danych, ekstrakcja danych, pozyskiwanie wiedzy, czy też wydobywanie danych [EN05]) odbywa się najczęściej w środowisku baz lub hurtowni danych, które stanowią doskonałe źródła danych do analizy - głównie ze względu na łatwość dostępu oraz usystematyzowaną strukturę przechowywanych informacji. Ponieważ liczba odkrytych wzorców w wielu przypadkach może być bardzo duża, odkryte wzorce bardzo często zapisuje się w osobnych relacjach bazy lub hurtowni danych. Pozwala to na ich dalsze przetwarzanie w trybie off-line przez użytkowników końcowych. Pojęcie eksploracji zyskuje coraz większą popularność (również w wymiarze marketingowym) i jest wykorzystywane w wielu dziedzinach ludzkiego życia.

Jednym z najczęściej wykorzystywanych modeli wiedzy w eksploracji danych są reguły asocjacyjne. Reguła asocjacyjna ma postać $X \Rightarrow Y$, gdzie X oraz Y są wzajemnie rozłącznymi zbiorami elementów. Przykładem reguły, która mogła zostać odkryta w badzie danych sklepu komputerowego, może być reguła postaci $komputer \wedge myszka \Rightarrow monitor$. Reguła ta prezentuje fakt, że klienci kupujący komputer oraz myszke z dużym prawdopodobieństwem kupią również monitor. W [Agr94] po raz pierwszy sformułowany został problem odkrywania reguł asocjacyjnych. Podstawą wielu algorytmów odkrywania reguł asocjacyjnych jest algorytm Apriori, zaprezentowany w [AIS93].

W ostatnich latach pojawiły się nowe możliwości wykorzystania współczesnych komputerów. W roku 2007 firma NVIDIA udostępniła programistom uniwersalną architekturę obliczeniową CUDA (ang. *Compute Unified Device Architecture*), który umożliwia wykorzystanie mocy obliczeniowej *procesorów graficznych* (*GPU*, ang. *Graphics Processing Unit*), bądź innych procesorów wielordzeniowych, do rozwiązywania ogólnych problemów numerycznych w sposób znacząco wydajniejszy niż w przypadku tradycyjnych, sekwencyjnych procesorów [Cor07a]. Choć w grach komputerowych moc obliczeniową jednostek graficznych można wykorzystać do obliczeń fizyki, to CUDA idzie jeszcze dalej, umożliwiając przyspieszenie obliczeń w takich dziedzinach, jak biologia, fizyka, kryptografia, bioinformatyka oraz inne. Dla potrzeb tego segmentu NVidia opracowała specjalny procesor graficzny *Tesla* [Cor07b].

Do tej pory bardzo małe jest zainteresowanie wykorzystaniem tej technologii w procesie odkrywania wiedzy, a w szczególności znajdowania reguł asocjacyjnych. Wyniki przeprowadzonych eksperymentów pozwalają przypuszczać, że algorytm wykorzystujący możliwości procesorów wielordzeniowych będzie wyraźnie szybszy od klasycznych algorytmów eksploracji danych zaimplementowany na tradycyjnych procesrach.

1.1 Cel i zakres pracy

Celem pracy jest zaprojektowanie i zaimplementowanie algorytmu odkrywającego reguły asocjacyjne, który będzie wykorzystywał możliwośći współczesnych kart graficznych dzięki wykorzystaniu technologii CUDA oraz porównanie zaprojektowanego i zaimplementowanego algorytmu do innych, podstawowych algorytmów odkrywania reguł asocjacyjnych. W ramach pracy dokonane zostanie również zebranie wiedzy dotyczącej algorytmów eksploracji reguł asocjacyjnych.

Rozdział 1 - wstęp.. Tutaj dalszy opis struktury pracy - zrobiony na koniec, gdy wszystko dalej będzie już znane.

Rozdział 2

Podstawy teoretyczne

W rozdziale tym przedstawiony zostanie przegląd literatury, który stanowi podstawy wiedzy na temat eksploracji danych, a w szczególności problemu odkrywania reguł asocjacyjnych w dużych zbiorach danych. Zebrana wiedza posłużyła autorowi do opracowania algorytmu wykorzystującego możliwości współczesnych kart graficznych.

2.1 Definicje

2.1.1 Model teoretyczny

Niech $I = \{i_1, i_2, ..., i_m\}$ będzie *zbiorem elementów* o liczności $|I| = m$. *Transakcją* nazwano dowolny, niepusty podzbiór $X \subseteq I$ zbioru elementów. Bazą danych DB nazwano dowolny zbiór par (id, X), gdzie X jest transakcją, a id jest dowolną wartością unikalną w ramach bazy danych nazywaną *identyfikatorem transakcji*. Bez utraty ogólności założono iż $id \in \mathbb{N}$.

Wsparciem (ang. *support*) $sup(X)$ transakcji X, w bazie danych nazwano częstość wystąpień transakcji w bazie danych. Formalnie przedstawia to wzór 2.1.

$$sup(X) = \frac{|\{id : (id, Y) \in DB \wedge X \subseteq Y\}|}{|DB|} \tag{2.1}$$

Łatwo zauważyć, że jeśli poziom ten jest niski, to oznacza to, że nie ma jednoznacznych dowodów na łączne występowanie elementów zbioru $Z = X \cup Y$, ponieważ zbiór Z występuje w niewielkiej liczbie transakcji.

Definicja 1. Niech będą dane dwie transakcje X i Y takie, że $X \cap Y = \emptyset$. Implikację postaci $X \Rightarrow Y$ nazwano *regułą asocjacyjną*.

Poziom ufności (ang. *confidence*) jest miarą zdefiniowaną dla implikacji reprezentowanej przez regułę asocjacyjną [EN05].

Definicja 2. Pioziom ufności ($conf$) reguły asocjacyjnej $X \Rightarrow Y$ jest równy

$$conf(X \Rightarrow Y) = \frac{sup(X \cup Y)}{sup(X)} \tag{2.2}$$

Po analizie definicji 2 łatwo zauważyć, że poziom ufności może być interpretowany, jako estymacja prawdopodobieństwa $P(Y|X)$.

Reguły asocjacyjne zazwyczaj powinny spełniać pewne wymagania zdefiniowane przez użytkownika - minimalne wsparcie oraz minimalny poziom ufności, oznaczane odpowiednio $minsup$ oraz $minconf$. Wyznaczają one dla aplikacji progi, jakie powinny spełniać zbiory oraz reguły, aby były brane pod uwagę w trakcie analizy.

Definicja 3. Zbiorem częstym $X \subseteq I$ nazywamy taki zbiór, który spełnia zależność $sup(X) \geq minsup$.

Generowanie reguł asocjacyjnych zazwyczaj sprowadza się do dwóch, niezależnych kroków:

1. Minimalne wsparcie jest używane do odnalezienia wszystkich zbiorów częstych w bazie danych DB.
2. Znalezione zbiory często oraz minimalny poziom ufności są używane do wygenerownia reguł asocjacyjnych.

Znalezienie wszystkich zbiorów częstych w bazie danych DB jest zadaniem wymagającym przeszukania wszystkich możliwych kombinacji bez powtórzeń ze zbioru I. Zbiór możliwych zbiorów elementów ma liczność równą $2^n - 1$ (wszystkie zbiory, poza zbiorem pustym, który nie jest w tym wypadku zbiorem sensownym w znaczeniu poddania go analizie).

Warto zauważyć, że dla każdego zbioru częstego Y, każdy jego podzbiór X jest również zbiorem częstym [AIS93]. Korzystając z tej właściwości wsparcia możliwe jest w sposób efektywny znalezienie wszystkich zbiorów częstych w zadanej bazie danych - z tej zależności korzysta algorytm apriori opisany w rozdziale 2.2.1. Dodatkowo, wszystkie reguły zbudowane na podstawie zbioru częstego Y muszą spełniać warunek minimalnego wsparcia, ponieważ spełnia ten warunek zbiór Y, a suma zbiorów reguły jest zbiorem wyjściowym Y.

2.2 Aktualna wiedza

W rozdziale tym zebrana została oraz opracowana dotychczasowa wiedza (ang. *state-of-the-art*) na temat odkrywania reguł asocjacyjnych. Przedstawione zostaną dwa podstawowe algorytmy wykorzystywane w tym procesie: Apriori oraz FP-growth. W chwili obecnej te dwa algorytmy stanowią podstawę, na której budowane są nowe algorytmy, wykorzystujące możliwości współczesnych algorytmów (*Czy wymieniać tutaj przykłady algorytmów, które bazują na nich + refy do kilku artykułów?*).

2.2.1 Algorytm Apriori

Pierwszy algorytm odkrywający reguły asocjacyjne został przedstawiony w roku 1994 w pracy [AS94]. Niżej przedstawione zostaną szczegóły działania tego algorytmu nazwanego algorytmem *Apriori*.

Zagadnienie odkrywania reguł asocjacyjnych można podzielić na dwa etapy [AIS93]:

1. Odkrywanie zbiorów częstych, których wartość wsparcia jest wyższa od wartości $minsup$.
2. Generowanie reguł asocjacyjnych na podstawie znalezionych zbiorów częstych. Reguła $X \Rightarrow Y$ jest wynikiem działania algorytmu dla zbioru $Z = X \cup Y$, jeżeli spełnia ona nierówność $conf(X \Rightarrow Y) \geq minconf$. Ponieważ zbiór $Z = X \cup Y$ jest zbiorem częstym, to reguła spełnia również warunek przekraczania minimalnego wsparcia.

   Na tym etapie możliwe jest tworzenie reguł, w których w zbiorze *poprzedników* (X z oznaczeń z definicji 1) jest wiele elementów oraz jeden w *następniku* (zbiór Y z definicji 1) [AIS93] lub dopuszczana jest możliwość wielu elementów również w następniku [AS94]. W niniejszej pracy analizowany jest sposób generowania reguł, w którym oba zbiory mogą być zbiorami wieloelementowymi.

W kolejnych podrozdziałach przedstawione zostaną etapy tworzące razem algorytmy Apriori.

k-zbiór	Zbiór zawierający k elementów.
L_k	Zbiór zawierający k-zbiory. Każdy zbiór zawarty w L_k zawiera dwa pola: i) zbiór oraz ii) wartość *support*.
C_k	Zbiór k-zbiorów kandydatów (potencjalnych zbiorów częstych). Każdy zbiór zawarty w C_k zawiera dwa pola: i) zbiór oraz ii) wartość *support*.

Tablica 2.1: Oznaczenie skrótów w opisie algorytmu

Generowanie zbiorów częstych

W celu wyznaczenia zbiorów częstych algorytm dokonuje analizy bazy danych DB, by w kolejnych iteracjach generować rodziny coraz to liczniejszych zbiorów, będących zbiorami częstymi dla zadanej wartości *minsup*. Algorytm zaczyna od znalezienia wszystkich zbiorów jednoelementowych, które są zbiorami częstymi. W każdym kolejnym kroku generowane są zbiory częste na podstawie zbiorów wygenerowanych w kroku poprzednim. Proces ten jest kontynuowany do momentu aż nie zostaną znalezione żadne zbiory częste.

Algorytm generuje zbiory kandydatów jedynie na podstawie zbiorów częstych odkrytych w kroku poprzednim - co ważne generowanie ich odbywa się bez wielokrotnego przeglądania bazy danych transakcji. Intuicja podpowiada, że każdy podzbiór zbioru częstego jest zbiorem częstym. Zatem, każdy zbiór częsty zawierający k elementów może być wygenerowany na podstawie połączenia dwóch zbiorów posiadających $k-1$ elementów, a na koniec kasując te zbiory, których jakikolwiek podzbiór nie jest częsty [AS94].

Tabela 2.2.1 zawiera spis oznaczeń używanych w opisie algorytmu.

Procedura Apriori Frequent Set Generaion przedstawia pseudokod realizujący opisywany w tym rozdziale algorytm generowania zbiorów częstych.

Apriori Frequent Set Generaion

```
1  L_1 ← {1-zbiory częste }
2  for (k = 2; L_k-1 ≠ ∅; k++)
3       do C_k ← aprioriGen(L_k-1)
4          for each trasakcja t ∈ DB
5               do C_t ← subset(C_k, t)
6                  for each kandydat c ∈ C_t
7                       do c.count++
8          L_k ← {c ∈ C_k | c.count ≥ minsup}
9  Answer ← ⋃_k L_k
```

Procedura aprioriGen Procedura *aprioriGen* reprezentuje proces twórzenia zbiorów k-elementowych kandydatów na podstawie zbiorów wejściowych $k-1$-elementowych. Procedura ta jest podzielona na dwa etapy: łączenia oraz przycinania.

Jak łatwo zauważyć wynikiem działania Join Step są zbiory k-elementowe, które powstały na podstawie zbiorów wejściowych L_{k-1}, a ich zawartość różni się tylko jednym elementem - ostatnim. Ważnym faktem jest to, iż elementy w zbiorach są uporządkowane leksykograficzne, co wykorzystywane jest w tej procedurze.

JOIN STEP

1 **insert into** C_k
2 **select** p.item$_1$, p.item$_2$, ..., p.item$_{k-1}$, q.item$_{k-1}$
3 **from** L_{k-1} p, L_{k-1} q
4 **where** p.item$_1$ = q.item$_1$, ..., p.item$_{k-2}$ = q.item$_{k-2}$, p.item$_{k-1}$ < q.item$_{k-1}$

Warto zauważyć, że JOIN STEP jest ekwiwalentem rozszerania zbioru L_{k-1} każdym elementem zbioru elementów I, a następnie kasowania tych $(k-1)$-zbiorów otrzymanych przez usuwanie $(k-1)$ elementu, które nie są w L_{k-1}.

Warunek p.item$_{k-1}$ < q.item$_{k-1}$ zapewnia, że nie będą generowane duplikaty. Dlatego też po etapie łączenia zachodzi zależność $C_k \supseteq L_k$.

Następnym krokiem jest PRUNE STEP, w którym usuwane są wszystkie elementy $c \in C_k$, którego jakikolwiek podzbiór $(k-1)$-elementowy zbioru c nie należy do L_{k-1}.

PRUNE STEP

1 **for each** zbiór $c \in C_k$
2 **do**
3 **for each** $(k-1)$-podzbiór s zbioru c
4 **do if** $s \notin L_{k-1}$
5 **then delete** c z C_k

Celem operacji przycinania (ang. *prune*) jest ograniczenie rozmiaru zbioru C_k przed sprawdzeniem wsparcia dla kandydatów w bazie danych DB. W tym celu wykorzystywana jest właściwość, z której wynika, że jeśli jakiś $(k-1)$-podzbiór danego kandydata $(c \in C_k)$ nie występuje w L_{k-1}, to kandydatk c nie jest zbiorem częstym i powinien być usunięty z C_k.

Generowanie reguł asocjacyjnych

Po zakończeniu pierwszego etapu algorytm przystępuje do drugiego, czyli do budowania reguł asocjacyjnych na podstawie odkrytych zbiorów. Podobnie, jak w [AS94] algorytm będący przedmiotem analizy niniejszej pracy, generuje wszystkie możliwe reguły asocjacyjne dla zadanego zbioru. Mniej ogólny sposób generowania reguł został przedstawiony w pracy [AIS93], jednakże podjęto decyzję, że jest to sposób zbyt mało użyteczny w środowisku produkcyjnym.

Aby wygenerować reguły, dla każdego zbioru częstego l znajdowane są niepuste podzbiory - podzbiór taki oznaczony jest jako a. Dla takich oznaczeń wygenerowna zostanie reguła $a \Rightarrow (l-a)$, jeżeli spełniona jest nierówność $\frac{support(l)}{support(a)} \geq minconf$. Warto zauważyć, że dla każdego zbioru częstego generowane są wszystkie możliwe niepuste podzbiory - zapewnia to, że odkryte zostaną wszystkie możliwe reguły.

Procedura GENERATE FREQUENT ITEMSETS prezentuje generowanie reguł asocjacyjnych na podstawie odkrytych zbiorów częstych.

GENERATE FREQUENT ITEMSETS

1 **for each** zbiór częsty l_k, $k \geq 2$
2 **do call** genrules(l_k, l_k)

W powyższym algorytmie wykorzystana została funkcja GENRULES, która na podstawie dwóch zbiorów generuje reguły asocjacyjne. Zapis pseudokodu tej funkcji przedstawiony jest poniżej.

GENRULES(l_k: k-zbiór częsty, a_m: m-zbiór częsty)

```
1  A ← {(m − 1)-zbiór a_{m−1} | a_{m−1} ⊂ a_m}
2  for a_{m−1} ∈ A
3       do conf ← support(l_k) / support(a_{m−1})
4          if conf ≥ minconf
5             then output reguła a_{m−1} ⇒ (l_k − a_{m−1})
   ufność = conf oraz wsparcie= support(l_k)
6                    if m − 1 > 1
7                       then call genrules(l_k, a_{m−1})
   generowanie reguł podzbiorów zbioru a_{m−1}
```

2.2.2 Algorytm FP-growth

Podstawową wadą algorytmu Apriori jest wysoki koszt przetważania dużych zbiorów danych. Przykładowo, dla 10^4 1-zbiorów częstych, algorytm Apriori wygeneruje około 10^7 2-zbiorów kandydatów, które następnie poddane zostaną weryfikacji, czy są zbiorami częstymi. Poza tym algorytm ten wymaga wielokrotnego odczytywania zawartości bazy danych - w każdym kroku algorytmu należy odczytać całą bazę danych w celu obliczenia wsparcia zbiorów kandydujących.

Wymienione wyżej wady algorytmu Apriori nie występują w algorytmie *FP-growth* przedstawionym w pełni w pracy [HPYM04]. Algorytm ten pozwala wyeliminować konieczność generowania tak dużej liczby kandydujących zbiorów elementów oraz ogranicza liczbę dostępu do bazy danych do absolutnego minimum. Co więcej algorytm ten harakteryzuje się kompletnością, co oznacza, że znajdowane są wszystkie wzorce o określonej częstości.

Algorytm FP-growth można podzielić na trzy podstawowe kroki.

1. W kroku pierwszym generowana jest skompresowana wersja bazy danych DB, mająca postać drzewa częstych wzorców.

2. Drugim krokiem jest transformacja tak skonstruowanego drzewa do postaci *FP-drzewa* (patrz definicja 4).

3. Trzeci krok polega na analizie FP-drzewa celem odnalezienia reguł asocjacyjnych. W kroku tym stosowana jest metoda dziel i zwyciężaj (ang. *divide-and-conquer*) zamiast podejścia Apriori, czyli generowania na każdym poziomie zbioru kandydatów na zbiory częste, a następnie odcinaniu kandydatów nie spełniających kryteriów akceptacji. Takie podejście przekształca problem znajdowania długich reguł w problem szukania krótszych, a następnie konkatenacji wyników.

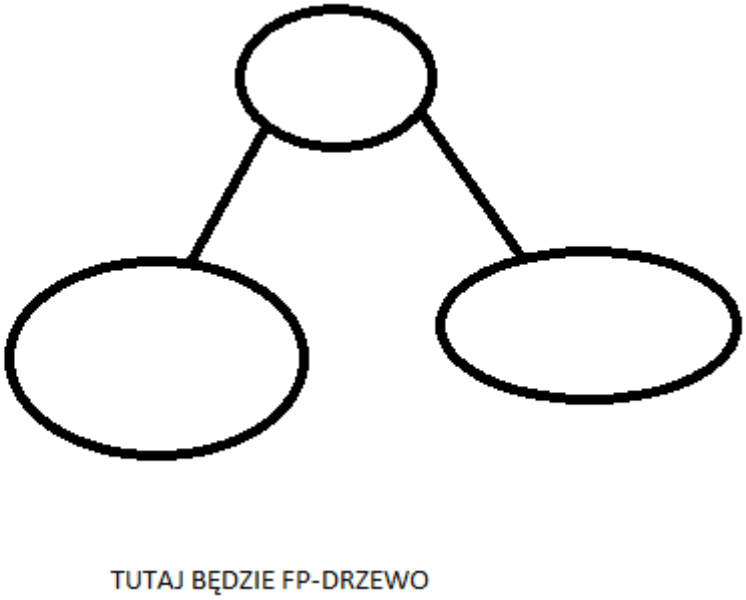
Definicja 4. FP-drzewo (ang. *frequent-pattern tree*) jest to ukorzeniony, etykietowany w wierzchołkach graf acykliczny spełniający poniższe cechy.

1. Korzeniem drzewa jest jeden element *null*, zbiór poddrzew prefiksowanych elementami (jako dzieci elementu *null*) oraz tablicy nagłówkowej zawierającej wpisy *element* → *wskaźnik na element drzewa*.

2. Każdy wierzchołek poddrzewa składa się z trzech elementów: nazwy elementu (ang. *item name*), licznika (ang. *count*) oraz wskaźnika na inny wierzchołek. Nazwa elementu (*itemName*) w sposób jednoznaczny identyfikuje element ze zbioru elementów I, licznik przechowuje liczbę transakcji reprezentowanych przez ścieżkę od *null* do tego elementu, natomiast wskaźnik wskazuje na kolejny wierzchołek w FP-drzewie, którego nazwa jest identyczna do danego.

3. Każdy wpis w tablicy nagłówkowej (ang. *frequent-item-header table*) skałada się z dwóch elementów: nazwy elementu oraz wskaźnika na pierwszy element w drzewie posiadający identyczną nazwę.

Rysunek 2.1 przedstawia przykładowe FP-drzewo wraz z tablicą nagłówkową.



Rysunek 2.1: Ilustracja przykładowego FP-drzewa

Kompresja bazy danych

Pierwszy etap polega na znalezieniu wszystkich 1-zbiorów częstych występujących w bazie danych DB. Po ich odnalezieniu (F_1 - zbiór znalezionych 1-zbiorów częstych) z każdej transakcji T usuwany jest ten element, który nie należy do F_1.

W wyniku usunięcia elementów nie tworzących jednoelementowych zbiorów częstych, baza ma zazwyczaj znacznie mniejszy rozmiar niż wyjściowa baza danych. Dodatkowo w tym kroku elementy w każdej transakcji zostają posortowane według maljącej wartości ich wsparcia.

Konstrukcja FP-drzewa

Algorytm 1. *Konstrukcja FP-drzewa.*
Input: Baza danych transakcji DB oraz minimalne wsparcie (*minsup*).
Output: FP-drzewo utworzone na podstawie zawartości DB
Metoda: Poniżej zostanie opisany proces konstrukcji FP-drzewa.

1. Przeskanowanie bazy danych transakcji DB odbywa się jednokrotnie. Utworzony na tej podstawie zostanie zbiór F, zawierający 1-zbiory częste. Posortowany malejąco zbiór F na podstawie wartości *support* dla każdego elementu tworzy listę $FList$, czyli listę wsystkich elementów tworzących jednoelementowe zbiory częste.

2. Tworzony jest pierwszy element drzewa - korzeniem zostaje (zgodnie z definicją 4) element z etykietą *null*. Dla każdej transakcji w bazie danych DB wykonywane jest, co następuje.

Wybierane są elementy częste z transakcji, a następnie sortowane zgodnie z kolejnością w $FList$. Niech taka posortowana lista elementów częstych ma postać $[p|P]$, gdzie p jest pierwszym elementem, a P jest pozostałą częścią listy. Następnie wywoływana jest funkcja $insertTree([p|P], T)$, gdzie T jest FP-drzewem.

Funkcja insertTree Jeśli drzewo T ma dziecko N takie, że $N.itemName = p.itemName$, zwiększana jest wartość $N.count$ o wartość 1; w przeciwnym wypadku tworzony jest nowe element N z wartością $count = 1$, a wskaźnik rodzica ustawiany jest na T oraz wskaźnik sąsiedzctwa elementu ustawiany jest na element z takim samym $itemName$. Jeśli lista P była niepusta, to wywoływana jest funkcja $insertTree(P, T)$ rekurencyjnie, w przeciwnym wypadku kończone jest działanie funkcji.

Eksploracja FP-drzewa

Po utworzeniu FP-drzewa przeprowadzana jest jego analiza w celu znalezienia wszystkich zbiorów częstych. Eksploracja bazuje na obserwacji, że dla każdego 1-zbioru częstego α wszystkie częste nadzbiory tego zbioru są reprezentowane w FP-drzewie przez ścieżki zawierające wierzchołek (bądź wierzchołki) α.

Analiza rozpoczyna się od znalezienia dla każdego 1-zbioru częstego α wszystkich ścieżek w FP-drzewie, których końcowym wierzchołkiem jest wierzchołek odpowiadający zbiorowi α. Pojedyncza ścieżka, na której końcu znajduje się wierzchołek α w dalszej analizie będzie nazywana *ścieżką prefiksową wzorca* α.

Poniżej zaprezentowany zostanie pseudokod algorytmu przeszukiwania FP-drzewa celem odnalezienia reguł asocjacyjnych.

Algorytm 2. *FP-growth: Przeszukiwanie FP-drzewa celem odnalezienia reguł asocjacyjnych.*

Input: Baza danych transakcji DB reprezentowana przez FP-drzewo zwrócone przez algorytm 1 oraz minimalne wsparcie (*minsup*).

Output: Kompletny zbiór reguł asocjacyjnych.

Metoda: Wywołanie FP-GROWTH($FP - drzewo, null$).

FP-GROWTH(*Tree*, α)

```
1  if Tree zawiera jedną ścieżkę prefiksową
2     then P ← ścieżka prefiksowa drzewa Tree
3          Q ← wieloczęściowa ścieżka z najwyższym elementem zastąpionym przez korzeń null
4          for each kombinacja (β) elementów z P
5               do generuj regułę β ∪ α z support = minimalna wartość support elementów w β
6                  niech freqPatternSet(P) będzie zbiorem wygenerowanych do tej pory reguł
7     else Q ← Tree
8          for each element a_i ∈ Q
9               do generuj regułę β ← a_i ∪ α z support = a_i.support
10                 stwórz drzewo warunkowe z β oraz FP-drzewo (Tree_β) dla β
11                 if Tree_β ≠ ∅
12                    then call FP-GROWTH(Tree_β, β)
13                 niech freqPatternSet(Q) będzie zbiorem wygenerowanych do tej pory reguł
14 return freqPatternSet(P) ∪ freqPatternSet(Q) ∪ (freqPatternSet(P) × freqPatternSet(Q))
```

Literatura

[Agr94] Rakesh Agrawal. Quest: a project on database mining. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1994.

[AIS93] Rakesh Agrawal, Tomasz Imielinski, Arun Swami. Mining association rules between sets of items in large databases. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, strony 207–216, 1993.

[AS94] Rakesh Agrawal, Ramakrishnan Srikant. Fast algorithms for mining association rules. *VLDB*, 1994.

[Cor07a] NVIDIA Corporation. Cuda - zone. [on-line] `http://www.nvidia.pl/object/cuda_home_new_pl.html`, 2007.

[Cor07b] NVIDIA Corporation. Nvidia tesla 20. [on-line] `http://www.nvidia.pl/object/cuda_home_new_pl.html`, 2007.

[EN05] Ramez Elmasri, Shamkand B. NAvathe. *Wprowadzenie do systemów baz danych.* Addison-Wesley, Reading, MA, USA, 2005.

[FPSSU96] Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, Ramasamy Uthurusamy. *Advances in Knowledge Discovery and Data Mining.* AAAI/MIT Press, 1996.

[HPYM04] Jiawei Han, Jian Pei, Yiwen Yin, Runying Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery*, 2004.

Instytut Informatyki, Wydział Informatyki i Zarządzania
Politechnika Poznańska

Skład przy użyciu systemu LaTeX.

BibTeX:

```
@mastersthesis{ key,
    author = "Tomasz Kujawa",
    title = "{Równoległe odkrywanie reguł asocjacyjnyh zaimplementowane na procesory graficzne}",
    school = "Poznan University of Technology",
    address = "Pozna{\'n}, Poland",
    year = "2011",
}
```