Pay With Friends

Teoretyczne podstawy działania

Spis treści

1. Oznaczenia

1.1. Definicje

V – zbiór wierzchołków grafu (wszyscy użytkownicy)

u_i – i-ty wierzchołek (i-tyużytkownik)

n – liczba użytkowników

b_i – bilans wierzchołku i (i-tego użytkownika), $b_i \in \mathbb{C}$

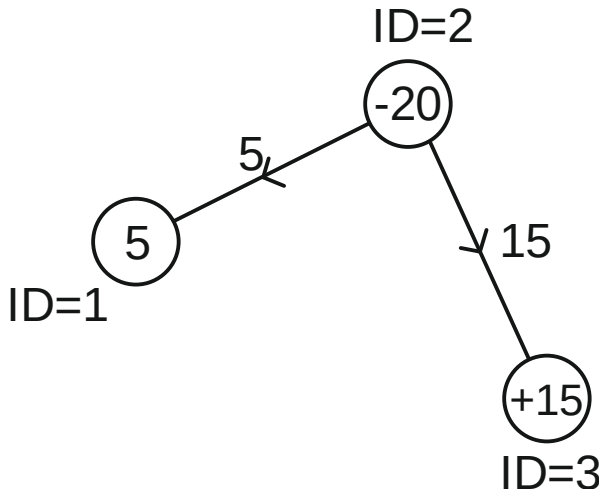
E – zbiór krawędzi grafu (wszystkie połączenia)

$e_{i,j}$ – krawędź pomiędzy wierzchołkami „i" i „j" (przyjaźń i-tego i j-tego użytkownika), kolejność i,j nie ma znaczenia, ponieważ graf jest symetryczny

$p_{i,j}$ – niewypełniona płatność od użytkownika „i" do użytkownika „j", $p_{i,j} \in \mathbb{C}$

1.2. Przykład transakcji

Dla sytuacji przedstawionej na obrazku:



- wierzchołki (użytkownicy): u_1, u_2, u_3
- krawędzie (przyjaźnie): $e_{1,2}, e_{2,3}$
- bilansy wierzchołków (użytkowników):
  - $b_1 = 5$
  - $b_2 = -20$
  - $b_3 = 15$
- (po zaakceptowaniu transakcji przez wszystkich użytkowników) niewypełnione płatności użytkowników:
  - $p_{1,2} = -p_{2,1} = -5$
  - $p_{2,3} = -p_{3,2} = 15$

Praktyczny sens sytuacji: „użytkownik 2" jest przyjacielem użytkowników „1" i „3". Użytkownik 2 jest winien „użytkownikowi 1" 5 jednostek, a „użytkownikowi 3" 15 jednostek.

2. Użycie oznaczeń w programie

Wierzchołek grafu – użytkownik. Każdy z użytkowników jest identyfikowany 15 cyfrowym numerem ID (który jest identyczny co Facebookowy numer ID użytkownika).

Krawędź grafu – przyjaźń użytkowników. Tylko użytkownicy będący przyjaciółmi (w Pay With Friends – nie w Facebooku, tak aby użytkownik mógł decydować z kim chce mieć powiązania finansowe).

Bilans wierzchołka – bilans użytkownika w transakcji. Po dodaniu nowej transakcji dla każdego użytkownika transakcji wyliczony jest jego bilans w tej transakcji, na tej podstawie będą potem wyliczone płatności pomiędzy użytkownikami.

3. Możliwe akcje użytkowników

3.1. Ogólnie

Stworzenie użytkownika – stworzenie wierzchołka reprezentującego daną osobę w aplikacji. Zarówno złożoność obliczeniowa jak i pamięciowa tej operacji wynosi $O(1)$.

Dodanie/potwierdzenia przyjaźni – stworzenie krawędzi pomiędzy dwoma użytkownikami w aplikacji. Zarówno złożoność obliczeniowa jak i pamięciowa tej operacji wynosi $O(1)$.

Dodanie nowej transakcji – zmiana bilansów pewnej grupu użytkowników (która będzie miała miejce dopiero po zaakceptiowaniu transakcji przez wszystkich użytkowników). Bilansy zostaną od razu przekształcone w płatności pomiędzy użytkownikami (jeśli nie jest to możliwe(brak krawędzi pomiędzy użytkownikami) transakcja nie będzie mogła być zaakceptowana).

Optymalizacja transakcji grupy użytkowników – zmiana płatności pomiędzy pewną grupą wierzchołków tak, aby zminimalizować pewien wskaźnik transakcji (np. ilość niezerowych płatności, suma wartości płatności).

Globalna optymalizacja transakcji – analogiczna optymalizacja przeprowadzona pomiędzy wszystkimi wierzchołkami.

3.2. Dodawanie nowej transakcji

Akcje potrzebne do dodania nowej transakcji (n – liczba uczestników transakcji):

- obliczenie bilansu każdego użytkownika (proste działania artymetyczne) – złożoność $\boldsymbol{O(V)}$
- stworzenie SP (Spanning Tree) - drzewa spinającego wszystkich użytkowników po dostępnych krawędziach (przy zastosowaniu algorytmu BFS (Bread First Search – przeszukiwania w szerz) ma złożoność $\boldsymbol{O(E+V)}$) – jeśli nie jest możlwe stworzenie SP, oznacza to, że transakcja nie może zostać wprowadzona do systemu – potrzebne są dodatkowe krawędzie – przyjaźnie pomiędzy użytkownikami
- dla każdego wierzchołka mającego 1 krawędź stworzenie transakcji zmniejszającej bilans tego użytkownika do 0 i usunięcie krawędzi i wierzchołka z MSP (złożoność $\boldsymbol{O(V)}$)

Cały algorytm ma więc złożoność obliczeniową równą $\boldsymbol{O(E+V)}$.

3.3. Optymalizacja transakcji grupy użytkowników

W celu optymalizacja transakcji w grupie użytkowników korzystam z identycznego algorytmu co przy dodawaniu nowej transakcji, lecz jako bilans wierzchołka obliczam jego bilans względem grupy wierzchołków w obrębie której odbywa się transakcja. Złożoność algorytmu wynosi więc $\boldsymbol{O(E+V)}$.

3.4 Globalna optymalizacja transakcji

W celu optymalizacji globalnej można zastosować identyczny algorytm co przy lokalnej optymalizacji transakcji – wtedy algorytm ma złożoność $\boldsymbol{O(E+V)}$. Algorytm ma więc liniową złożoność Przy bardzo dużej liczbie wierzchołków. Nawet dla liczby wierzchołków rzędu 10^6, kiedy każdy z nich ma 100 krawędzi,a łączna liczba krawędzi to $5 * 10^7$ optymalizacja globalna nie będzie wielkim problemem dla aplikacji. Znacznie większy koszt obliczeniowy będzie w takim przypadku miało samo zczytanie mapy połączeń użytkowników z bazy danych. Pojawiające się pomysły optymalizacji algorytmu przez podział użytkowników na podgrupy i wykonywanie optymalizacji na tych grupach, przy łączeniu grup tylko za pomocą jednego użytkownika nie mają w tym przypadku większego sensu, ponieważ krawędzie w praktyce i tak będą występować głównie właśnie w takich podgrupach, a sam podział użytkowników zajałby wiele czasu procesora.

4. Bezpieczeństwo użytkowników

4.1. OAuth – autoryzacja użytkownika w aplikacji

W celu ułatwienia użytkownikom korzystania z aplikacji zdecydowałem się na wykorzystanie otwartego standardu autoryzacji Oauth korzystając z Facebooka jako z serwisu gwarantującego bezpieczeństwo autoryzacji. W celu zalogowania do serwisu wystarczy być zalogowanym na swoje konto na Facebook'u, a następnie wejść na stronę aplikacji („app.ezyd.pl" lub „facebook.com/ezyd_app").

4.2. Szyfrowanie danych

Aplikacja korzysta z szyfrownia danych za pomocą SSL przy wykorzystaniu 128-bitowego klucza. Niestety nie posiadam certyfikatu SSL, przez co gdy chce się korzystać z serwisu poprzez HTTPS wiele przeglądarek internetowych wyświetla komunikaty o zagrożeniu. Jeśli liczba użytkowników serwisu będzie się zwiększać zamierzam wykupić certyfikat SSL.

4.3. Bezpieczeństwo bazy danych

Baza danych MySQL, z której korzysta serwis jest postawiona na moim domowym komputerze tak samo jak reszta serwisu. Router zza którego łączę się z internetem jest zabezpieczony firewallem i połączenia z zewnątrz nie mogą połączyć się z bazą danych. Do tego dostęp do bazy danych jest na wszelki wypadek zabezpieczony hasłem.

5. Serwer HTTP/HTTPS

5.1. Dynamiczny adres IP serwera

Serwer na którym postawiona ma dynamicznie przydzielany adres IP. Aby połączenie się z serwerem było możliwe z zewnątrz po zmianie IP serwera wykorzystałem serwis „no-ip.org".

Po wejściu na stronę „app.ezyd.pl" lub „facebook.com/ezyd_app" zostajemy przekierowani na stronę „ezydapplication.no-ip.org", która z kolei przekierowuje na adres serwera, który automatycznie co każde 5 minut aktualizuje swój adres IP na serwisie „no-ip.org" (przez aplikację „No-IP DUC"). Taki czas aktualizacji adresu sprawia, że każda zmiana adresu IP serwera powoduje średnio 2.5 minutową przerwę w działaniu aplikacji, jednak nie wydaje się, żeby na obecnym etapie rozwoju serwisu był to jakiś problem.

5.2. Server www

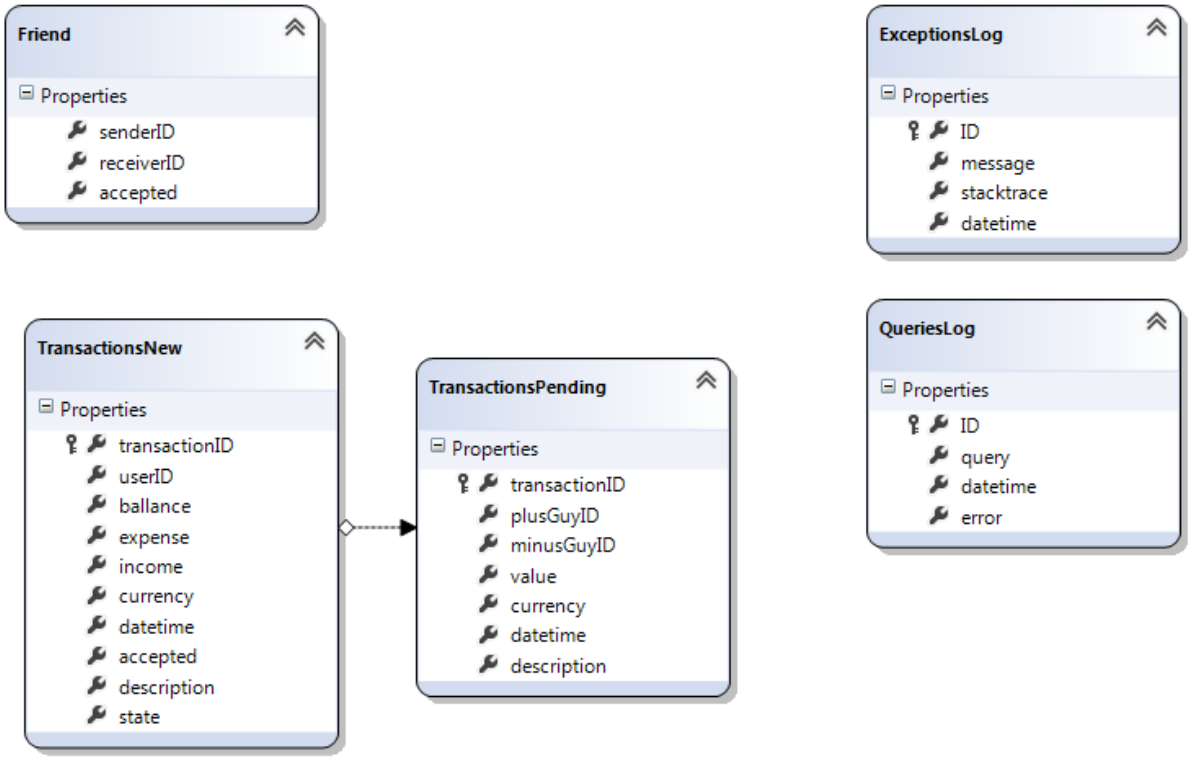
Serwer HTTP/HTTPS postawiony jest na moim domowym komputerze na „Internet Information Services 7" („IIS7"). Jest to oprogramowanie dostępne wraz z Windowsem 7 i pozwala stworzyć bardzo niezawodny serwer www.

5.3. Serwer bazy danych

Jako serwera bazy danych użyłem WAMP'a (Windows-Apache-MySQL-PHP). Zdecydowałem się na niego, ponieważ jest to darmowe i bardzo stabilne środowisko. Bałem się korzystania z komercyjnych baz danych takich jak MS SQL, czy baz Oracle, ponieważ są to bardzo drogie środowiska, a w przyszłości zamierzam rozwijać „Pay With Friends" także komercyjnie. Niestety przy korzystaniu z MySQL nie mogłem korzystać z LINQ, które bardzo uprościłoby pisanie aplikacji (słyszałem o możliwości skorzystania z „ADO.NET Entity Framework" w celu mapowania obiektów bazy danych w kodzie, lecz podobno sprawia to wiele problemów i nie zdecydowałem się na użycie tego frameworka).

6. Baza danych

6.1. Tabele



6.2. Opis tabel

Opis poszczególnych tabel w bazie danych:

- Friend – informacje na temat krawędzi grafu
- TransactionsNew – informacje na temat niezaakceptowanych transakcji
- TransactionsPending – informację na temat oczekujących transakcji
- [opcjonalnie] ExceptionsLog – debugowa tabela przechowywująca nieosbłużone wyjątki, które wyrzucił program
- [opcjonalnie] QueriesLog – debugowa tabela przechowująca wszystkie zapytania wysłane do bazy danych przez serwis (poza tymi które dotyczą debugowych tabel)

6.3. Użycie bazy danych w aplikacji

.NET nie daje wsparcia dla LINQ dla baz MySQL, więc zastosowałem bibliotekę MySql.Data.MySqlClient. Niestety łączenie z bazą danych, jak i wszystkie zapytania musiały zostać przeze mnie napisane odręcznie (oczywiście zostały one owrappowane funkcjami).