**Autorzy:**

Wojciech Urbański

Marcin Waligórski

Daniel Wegner

Łukasz Wróblewski

Inteligentne wyszukiwanie informacji

Wizualizacja Wikipedii

# Opis zadania

Naszym zadaniem projektowym było stworzenie wizualizacji angielskiej Wikipedii w postaci Simple. Zawiera ona 56565 artykułów oraz 2248532 połączeń pomiędzy nimi. Pomimo i tak pomniejszonej objętości w stosunku do pełnej Wikipedii, stanowi ona dość trudne zadanie do zrealizowania.

# Zapoznanie z tematem

Pracę nad projektem rozpoczęliśmy od zapoznania się z dostępnymi gotowymi rozwiązaniami wizualizującymi Wikipedię oraz przeglądem istniejących bibliotek wspomagających rysowanie grafów i drzew. Już podczas korzystania z istniejących programów pojawił się pierwszy z problemów, z którymi musieliśmy się zmierzyć. Otóż żaden z programów nie radził sobie z takim rozmiarem wejścia. Do każdego z nich staraliśmy się dostosować dane do jednego z obsługiwanych formatów – najczęściej miały one format listy wierzchołków w postaci: indeks i nazwa oraz listy krawędzi, opisanych jako 2 indeksy wierzchołków, które łączą. Niezależnie od technologii w jakiej programy był wykonany; Java, C, C++, HTML z CSS i Javascript, żaden z nich nie potrafił jednocześnie załadować takiej ilości danych. Kończyło się to całkowitym wyczerpaniem całej dostępnej pamięci (nawet do 4GB) lub po prostu brakiem reakcji ze strony aplikacji. W ramach testów próbowaliśmy wizualizować tylko część danych np. 5-10 tys. wierzchołków. Jeśli nawet, już któryś z programów sobie poradził to wykres był tak gęsty lub skomplikowany, że nie dało się z nic z niego odczytać lub w ogóle korzystać z programu.

Po pierwszym etapie, podjęliśmy decyzję , że spróbujemy zaimplementować samodzielnie jeden z algorytmów służących do rysowania grafów i dodatkowo wykorzystamy istniejącą bibliotekę do wspomagania samej wizualizacji. Wybór padł na dość często wykorzystywany algorytm force-based layout (sam algorytm zostanł opisany w dalszej części sprawozdania). Kolejnym krokiem był wybór technologii w jakiej zostanie wykonany projekt. Podczas przeglądu bibliotek zdecydowaliśmy się na skorzystanie z biblioteki d3.js, będącej napisanej w Javascript i sam interfejs stworzyć w oparciu o HTML 5 i CSS3 oraz część kodową na framework’u Fuel. Dodatkowo d3.js posiada możliwość wizualizacji z wykorzystaniem wybranego przez nas algorytmu. Podczas tworzenia naszej wersji wzorowaliśmy się na ich implementacji, dostosowując ją do naszego projektu.

To wszystko jednak nie wystarczyło do poradzenia sobie w pełni z ilością danych do wyrenderowania. Po ostatecznych konsultacjach z prowadzącym, uzgodniliśmy, że będziemy wizualizować mniejsze podgrafy wszystkich danych tzn. użytkownik będzie mógł wybrać, jaki artykuł go interesuje i następnie zostanie narysowany graf połączeń tego artykułu z innymi. Nasza aplikacji umożliwi wtedy przejście do kolejnego artykułu i wyświetlenie jego sąsiadów. Umożliwi to klarowniejszą prezentację dostępnych danych (mniej danych do przetworzenia dla użytkownika naraz, czytelniejsza reprezentacja).

# Algorytm

W projekcie wykorzystaliśmy algorytm Force-Based Layout. Służy on do rysowania grafów w dwu i trzy wymiarowych przestrzeniach w taki sposób, aby każda krawędź, była mniej więcej równej długości i było jak najmniej przecięć. Osiąga on to poprzez traktowanie krawędzi jak sprężyn, a wierzchołków jak naładowane elektrycznie cząsteczki. Cały graf jest następnie symulowany jak system fizyczny. Siły na siebie oddziaływają przyciągając lub oddalając wierzchołki od siebie. Te akcje są powtarzane iteracyjnie, aż układ osiągnie stan spoczynku, tzn. zmiany w każdej iteracji będą coraz mniejsze. Dodatkowo algorytm implementuje siłę pseudo-grawitacyjną, która utrzymuję graf w widocznej strefie i uniemożliwia jego ucieknięcie poza ekran.

Zaletami wybranego przez nas algorytmu są głównie dobre rezultaty dla małych i średnich grafów z jakimi mamy do czynienia w naszym projekcie. Dodatkowo jest on dość elastyczny i umożliwią rozbudowę o dodatkowe parametry. Wadami za to jest, że niektóre większe grafy potrzebują dłuższej chwili na wygenerowanie ale jest to akceptowalny czas oraz w naszym przypadku nie takich grafów aż tak dużo.

sdfsd