Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Damian Koniecki
Nr albumu: 209524

Kompresja SMS-ów metodą kodowania arytmetycznego

Praca magisterska
na kierunku INFORMATYKA

Praca wykonana pod kierunkiem
dr Janiny Mincer-Daszkiewicz
Instytut Informatyki

Październik 2008

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data Podpis kierującego pracą

Oświadczenie autora pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data Podpis autora pracy

Streszczenie

Tematem pracy jest praktyczne zastosowanie algorytmów kompresji tekstów do przesyłania krótkich wiadomości tekstowych (SMS) na telefonach komórkowych obsługujących rozszerzenie Javy — Wireless Messaging API.

W pracy przedstawiam szczegółowo teoretyczne podstawy i realizację kolejnych etapów projektu: gromadzenie bazy tekstów, metody kompresji krótkich wiadomości, specyfikę implementacji aplikacji na telefonach komórkowych.

Słowa kluczowe

kodowanie arytmetyczne, kompresja, J2ME, urządzenia mobilne, Wireless Messaging API, SMS, korpus języka

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.3 Informatyka

Klasyfikacja tematyczna

E. Data
E.4 Coding and Information Theory

Tytuł pracy w języku angielskim

SMS compression using arithmetic coding method

Spis treści

Wstęp

Short Message System (SMS) jest protokołem komunikacyjnym umożliwiającym przesyłanie krótkich wiadomości tekstowych pomiędzy telefonami komórkowymi. Wysyłanie SMS-ów w latach dziewięćdziesiątych w krótkim czasie stało się podstawową, obok rozmów głosowych, funkcją telefonów. W zależności od przyjętego kodowania znaków SMS może mieć obecnie wielkość 70 (jeśli chcemy korzystać z polskich znaków diakrytycznych) lub 160 znaków (jeśli z tych znaków rezygnujemy).

Wraz ze wzrostem popularności SMS-ów okazało się, że ich dostępna wielkość nie jest wystarczająca. Użytkownicy zaczęli więc stosować różne metody pozwalające na przekazanie większej ilości informacji w tak krótkich tekstach. Spowodowało to nawet powstanie specyficznego wariantu języka stosowanego w takiej komunikacji — lakonicznego i z dużą liczbą skrótów. Część użytkowników zrezygnowała również ze spacji w SMS-ach, rozpoczynając dla zwiększenia czytelności każdy wyraz wielką literą (np. „ToJestZupelnieNieciekawySMS."). Powstały nawet aplikacje umożliwiające automatyczne skracanie SMS-ów w ten sposób. Za niewystarczającą pojemnością SMS-ów przemawia również fakt dość powszechnego w Polsce stosowania uproszczonego kodowania znaków (bez polskich znaków diakrytycznych).

Rozwój technologii pozwolił na poszerzanie funkcji telefonów o możliwość uruchamiania aplikacji w Javie, dla których dostępne są m.in. funkcje pozwalające na obsługę SMS-ów. Można więc stworzyć swoją własną aplikację wykorzystującą protokół SMS do komunikacji. Naturalnym wydaje się więc pomysł wykorzystania tych możliwości i stworzenia aplikacji kompresującej SMS-y tak, aby wysyłany tekst mieścił się w jak najmniejszej liczbie SMS-ów, co może znacząco obniżyć koszty użytkowania telefonu.

Autor spotkał się już z jedną realizacją tego pomysłu — aplikacją SMSZipper [SMSZipper], będącą wynikiem pracy [RGF06] oraz późniejszych udoskonaleń. Aplikacja ta oprócz kompresji SMS-ów umożliwia szyfrowanie, dodawanie tytułów oraz tworzenie wiadomości z określonym czasem życia. Nie obsługuje ona jednak języka polskiego, co ma zasadnicze znaczenie przy kompresji tekstu i dyskwalifikuje ją do użytku przez polskojęzycznych użytkowników. Nie posiada ona również interfejsu użytkownika w języku polskim, a w wersji darmowej wyświetlane są reklamy i zmniejszona funkcjonalność. Celem niniejszego projektu jest stworzenie darmowej aplikacji w języku polskim, obsługującej ten język.

W pracy przedstawiam więc własną realizację tego pomysłu. W rozdziale pierwszym opisuję wymagania dotyczące algorytmu kompresującego w naszym projekcie, przedstawiam teoretyczne podstawy kompresji oraz szczegółowe metody użyte w projekcie.

Drugi rozdział opisuje przygotowania do tworzenia modelu statystycznego języka, koniecznego przy zastosowanych algorytmach kompresji, a więc wybór i gromadzenie bazy tekstów (zwanej korpusem) służącej do tworzenia statystyk. Zostały w tym celu również opisane i użyte metody automatycznej klasyfikacji tekstów. Rozdział zawiera także analizę pewnych danych statystycznych na zebranym korpusie.

Kolejny rozdział opisuje główną część projektu — aplikację `WiseSMS`, będącą edytorem SMS-ów z funkcją kompresji. Zostaną tam przedstawione i uzasadnione główne decyzje pro-

jektowe, omówiona budowa oraz interfejs użytkownika, a następnie przedstawione wyniki testów.

Ostatni rozdział stanowi podsumowanie pracy i przedstawia potencjalne kierunki dalszego rozwoju.

W dodatkach opisałem procedurę instalacji aplikacji oraz zawartość płyty CD dołączonej do pracy.

Rozdział 1

Kompresja

W tym rozdziale przedstawione zostanie zagadnienie kompresji na potrzeby naszego projektu. W pierwszym podrozdziale przedstawię matematyczne podstawy kompresji wg teorii informacji. Pozwoli to na wprowadzenie precyzyjniejszego języka, poznanie teoretycznych ograniczeń kompresji oraz narzędzi mierzenia efektywności używanych metod kompresji. W drugim podrozdziale omówione zostaną wymagania dotyczące kompresji wynikające ze specyfiki postawionego zadania. W kolejnych podrozdziałach przedstawiona będzie klasyfikacja metod kompresji ze szczególnym uwzględnieniem kodowania arytmetycznego, które będzie podstawowym algorytmem używanym w projekcie.

1.1. Teorioinformacyjne podstawy kompresji bezstratnej

Teoria informacji zajmuje się matematyczną analizą zapisu, przesyłania i odtwarzania informacji. Za jej początek uważa się klasyczną pracę [Shan48]. Przedstawione dalej definicje, twierdzenia i dowody w podobnym sformułowaniu można znaleźć w dowolnym podręczniku teorii informacji. Naszym celem jest wprowadzenie ich jedynie w takim zakresie, w jakim są niezbędne do zrozumienia twierdzenia 1.1.4.

Dla naszego opisu przyjmujemy, że S będzie zbiorem (przeliczalnym) informacji możliwych do przesłania. Formalnie kompresja jest kodowaniem, tymczasowo będziemy używać tych pojęć zamiennie.

Definicja 1.1.1 *Kodowaniem będziemy nazywać dowolną funkcję:*

$$\phi : S \to \Sigma^*$$

gdzie Σ jest pewnym skończonym alfabetem.

Przykładowo jeśli S jest zbiorem liczb naturalnych, to ϕ może być funkcją przypisującą liczbie jej zapis w systemie dziesiętnym. Inny przykład to zapis tekstu w postaci kodów Morse'a.

Ogólnie wybór kodowania to pewien kompromis między podatnością kodowania na błędy, a średnią długością zakodowanej wiadomości. Podatność na błędy zmniejsza się, gdy wprowadzimy redundancję, tzn. małe zmiany w zakodowanej wiadomości powodują, że nie jest ona poprawna, co pozwala na wykrycie błędu. Jednak powoduje to wydłużenie kodów. Będziemy tutaj zainteresowani bezszumowymi kanałami informacji, tzn. założymy, że błędy w przesyle nie występują.

Dla rozpatrywania średniej długości wiadomości ze zbiorem S musimy związać też pewną dyskretną przestrzeń probabilistyczną, tzn. każdej wiadomości chcemy przypisać prawdopodobieństwo jej wystąpienia. Wtedy będziemy przy wyborze kodowania zainteresowani minimalizacją wartości średniej długości kodu:

Definicja 1.1.2 *Średnią długością kodu $\phi : S \to \Sigma^*$ będziemy nazywać wartość:*

$$E(|\phi(S)|) = \sum_{s \in S} p(s)|\phi(s)|$$

Ta wartość jest podstawową miarą skuteczności kompresji. Jej minimalizacja odpowiada założeniom intuicyjnym — chcemy, aby najczęściej występujące wiadomości miały jak najkrótsze kody.

1.1.1. Teoretyczne ograniczenie na skuteczność kompresji

Wprowadzimy teraz kilka twierdzeń potrzebnych do dowodu twierdzenia 1.1.4.

Twierdzenie 1.1.1 (Nierówność Krafta) *Niech $S, \Sigma, \phi : S \to \Sigma^*$ (S przeliczalny) będą jak wyżej oraz ϕ będzie kodem bezprefiksowym, $|\Sigma| = r$. Wtedy:*

$$\sum_{x \in S} \frac{1}{r^{|\phi(x)|}} \leqslant 1 \tag{1.1}$$

Dowód.

Ograniczymy się początkowo do S skończonych. Dowód przeprowadzimy przez indukcję h - po maksymalnej długości słowa w $\phi(S)$:

- $h = 1$

  Wszystkie słowa w $\phi(S)$ mają długość 1 i jest ich co najwyżej r:

  $$\sum_{x \in S} \frac{1}{r} \leqslant r\frac{1}{r} = 1$$

- Załóżmy, że twierdzenie jest prawdziwe dla ϕ takich, że $max\{|\phi(x)| : x \in S\} \leqslant h$. Pokażemy twierdzenie dla kodów długości nie przekraczającej $h+1$. Weźmy więc taki kod. Będziemy teraz zamieniać kody długości $h+1$ na kody długości h bez zwiększania sumy (1.1). Weźmy więc dowolny taki kod $a_0a_1a_2 \ldots a_h$. Zastąpimy teraz wszystkie kody długości $h+1$ o prefiksie $a_0a_1 \ldots a_{h-1}$ przez kod $a_0a_1 \ldots a_{h-1}$. Zamienianych kodów jest co najwyżej r (tyle wartości może przyjmować a_h), ich wkład do sumy wynosi więc co najwyżej:

  $$\sum_{a_0 \ldots a_{h-1} y \in \phi(S)} \frac{1}{r^{h+1}} \leqslant r\frac{1}{r^{h+1}} = \frac{1}{r^h}$$

  natomiast po zamianie wkład do sumy od $a_0a_1 \ldots a_{h-1}$ wynosi:

  $$\frac{1}{r^h}$$

  a więc nie jest mniejszy. Dodatkowo zauważmy, że przed zamianą ani $a_0a_1 \ldots a_{h-1}$, ani żaden jego prefiks nie należały do $\phi(S)$. Stąd otrzymujemy nowy kod bezprefiksowy o niemniejszej sumie (1.1). Wykonując taką redukcję dla wszystkich kodów długości $h+1$ dostajemy kod bezprefiksowy o długościach nie przekraczających h, do którego możemy zastosować założenie indukcyjne co kończy dowód przypadku skończonego.

W przypadku nieskończonym możemy ustawić wszystkie elementy $\phi(S)$ w ciąg $(x_1, x_2, \ldots)$. Mamy pokazać, że:

$$S = \sum_{i=1}^{\infty} \frac{1}{r^{|x_i|}} \leqslant 1$$

Mamy sumy częściowe:

$$S_n = \sum_{i=1}^{n} \frac{1}{r^{|x_i|}}$$

Każda taka suma odpowiada przypadkowi skończonemu twierdzenia (bezprefiksowość zachowuje się przy usuwaniu elementów z $\phi(S)$). Tak więc dla wszystkich $n \in \mathcal{N}$ zachodzi:

$$S_n \leqslant 1$$

Wszystkie wyrazy sumowanego ciągu są nieujemne, a więc zachodzi również:

$$S \leqslant 1$$

□

Podstawowym pojęciem przydatnym w ocenie źródła danych jest pojęcie entropii, wprowadzone przez Shannona (patrz np. [Shan48]), określające miarę "gęstości" informacji zawartej w źródle. W ogólnej wersji odnosi się ono do dowolnej dyskretnej zmiennej losowej:

Definicja 1.1.3 *Entropią dyskretnej zmiennej losowej X nazywamy:*

$$H(X) = -\sum_{x \in X} p(x) log\, p(x)$$

Z pojęciem entropii związana jest

Twierdzenie 1.1.2 (Nierówność Shannona-Gibbsa) *Niech $P = (p_1, p_2, \ldots, p_n)$ oraz $Q = (q_1, q_1, \ldots, q_n)$ będą rozkładami prawdopodobieństwa. Dodatkowo $p_i, q_i > 0$. Wtedy:*

$$-\sum_{i=1}^{n} p_i log_r p_i \leqslant -\sum_{i=1}^{n} p_i log_r q_i$$

przy czym równość zachodzi wtedy i tylko wtedy, gdy $p_i = q_i$ dla $i \in \{1, \ldots n\}$.

Dowód.

Mamy pokazać, że:

$$S = \sum_{i=1}^{n} p_i log_r p_i - \sum_{i=1}^{n} p_i log_r q_i \geqslant 0$$

$$S = \sum_{i=1}^{n} p_i log_r \frac{p_i}{q_i} = \sum_{i=1}^{n} q_i \frac{p_i}{q_i} log_r \frac{p_i}{q_i}$$

Zauważmy teraz, że $f(x) = x\, log_r x = x\, lnx\, \frac{1}{ln\, r}$ jest dla $x > 0$ funkcją ściśle wypukłą:

$$(x\, lnx)' = x\frac{1}{x} + lnx = 1 + lnx$$

$$(x\, lnx)'' = (1 + lnx)' = \frac{1}{x} > 0$$

$$(x\, lnx)''' = (\frac{1}{x})' = -\frac{1}{x^2} \neq 0$$

Możemy więc do niej zastosować ogólnie znaną nierówność:

Twierdzenie 1.1.3 (Skończona nierówność Jensena) *Dla układu wag* $(a_i)_{i=1}^n$, $a_i > 0$, $\sum_{i=1}^n a_i = 1$, *funkcji wypukłej* f *oraz* $x_i \in \mathcal{D}om(f)$ *zachodzi:*

$$f(\sum_{i=1}^{n} a_i x_i) \leqslant \sum_{i=1}^{n} a_i f(x_i)$$

przy czym dla f *ściśle wypukłej równość zachodzi wtedy i tylko wtedy, gdy* $x_i = const.$

Podstawiając $f(x) = xlnx$, $a_i = q_i$, $x_i = \frac{p_i}{q_i}$ dostajemy:

$$q_i \frac{p_i}{q_i} log_r \frac{p_i}{q_i} \geqslant (\sum_{i=1}^{n} p_i) log_r (\sum_{i=1}^{n} p_i) = 0$$

Przy czym równość zachodzi tylko wtedy, gdy $x_i = \frac{p_i}{q_i} = const.$, co jest możliwe tylko wtedy, gdy $p_i = q_i$.

□

Twierdzenie 1.1.4 *Dla skończonego kodu bezprefiksowego* $\phi : S \to \Sigma^*$ *zachodzi:*

$$E(|\phi(S)|) \geqslant H(S).$$

Dowód.

Przyjmujemy standardowo $|\Sigma| = r$. Będziemy też rozpatrywać tylko dodatnie prawdopodobieństwa. Mamy pokazać:

$$\sum_{i=1}^{n} p(s_i)|\phi(s)| \geqslant -\sum_{i=1}^{n} p(s_i) log_r p(s_i)$$

$$-\sum_{i=1}^{n} p(s_i) log_r \frac{1}{r^{|\phi(s)|}} \geqslant -\sum_{i=1}^{n} p(s_i) log_r p(s_i) \tag{1.2}$$

Niech $p_i = p(s_i)$ oraz $q_i = \frac{1}{r^{|\phi(s)|}}$ dla $i = 1, 2, \ldots n-1$ oraz $q_n = 1 - \sum_{i=1}^{n-1} q_i$. Z nierówności Krafta mamy, że

$$q_n = 1 - \sum_{i=1}^{n-1} q_i > 1 - \sum_{i=1}^{n} q_i \geqslant 1 - 1 = 0$$

Oczywiście $\sum_{i=1}^{n} q_i = 1$. Możemy więc do $(p_i)_{i=1}^n$ oraz $(q_i)_{i=1}^n$ zastosować nierówność Shannona-Gibsa (twierdzenie 1.1.2) i otrzymać (1.2).

□

Twierdzenie 1.1.4 daje nam podstawowe teoretyczne ograniczenie na skuteczność kompresji. Mianowicie średnia długość kodu nie może być mniejsza niż entropia źródła.

1.2. Metody kompresji — wymagania

Metod kompresji jest bardzo wiele. Ich wybór uwarunkowany jest specyfiką źródła - charakterem kompresowanych danych. Rozróżniamy metody stratne i bezstratne. W metodach stratnych nie wymagamy, aby po dekompresji otrzymać oryginalne dane, ale jedynie pewne ich przybliżenie. Jest to przydatne przykładowo w kompresji dźwięku i obrazu. W przypadku tekstów wymagamy jednak, aby po dekompresji otrzymać oryginał, a więc interesować nas będą wyłącznie metody bezstratne.

1.2.1. Specyfika kompresji tekstu

Jak wspomniano wcześniej wybór metody kompresji uwarunkowany jest specyfiką źródła danych. Tutaj mamy do czynienia z kompresją tekstu. Entropia języka naturalnego (będąca miarą zawartości informacji w tekście i równocześnie teoretycznym ograniczeniem na skuteczność kompresji) jest stosunkowo niska — w przypadku języka angielskiego szacowana jest na 0.6-1.3 bita/znak. Możliwości kompresji są więc duże, powstało również wiele skutecznych algorytmów.

1.2.2. Specyfika SMS-ów

W przypadku SMS-ów mamy do czynienia z krótkimi tekstami. Najbardziej interesuje nas skuteczna kompresja tekstów o długościach w przedziale 70-400 znaków. Dłuższe teksty są rzadko wpisywane przez użytkowników telefonów komórkowych, przy tekstach poniżej 70 znaków nie mamy już żadnego zysku w opłatach ponoszonych przez użytkownika. To ograniczenie długości będzie miało zasadniczy wpływ na użyte metody kompresji.

1.2.3. Specyfika telefonów komórkowych

Dodatkowym ograniczeniem na metody kompresji są możliwości telefonów komórkowych:

- prędkość procesora jest dużo mniejsza niż w przypadku komputerów — algorytm musi być efektywny obliczeniowo,
- ograniczona w zależności od modelu telefonu wielkość pliku .jar — przyjąłem ograniczenie wynoszące 500 kB (mniejsze niż dla większości nowoczesnych telefonów),
- dostępna pamięć (również zależna od telefonu) — przyjąłem tutaj domyślne ograniczenie wynoszące 2 MB,
- brak obsługi liczb zmiennopozycyjnych,
- ograniczenia języków programowania — brak dostępnych bibliotek do kompresji przydatnych w naszych zastosowaniach.

1.3. Metody kompresji - wybór

Przedstawię najpierw podstawowy podział metod kompresji tekstów, za książką [BCW90].

1.3.1. Metody słownikowe i statystyczne

Pierwszym kryterium klasyfikacji metod kompresji tekstów będzie podział na metody:

- słownikowe — metody kompresji polegające na zastępowaniu fragmentów tekstu ich kodami; przypisanie kodów fragmentom tekstu przechowywane jest w słowniku,
- statystyczne — metody kompresji, w których kod symbolu wejściowego jest zależny od szacowanego prawdopodobieństwa jego wystąpienia, kodowane są pojedyncze znaki, a nie jak poprzednio całe fragmenty tekstu.

Metody słownikowe są bardziej naturalne, mogą dawać dobry współczynnik kompresji i zwykle wymagają mniej zasobów od metod statystycznych. Metody statystyczne z kolei dają zwykle lepszy stopień kompresji. W szczególności dla każdej metody słownikowej można skonstruować metodę statystyczną dającą taki sam stopień kompresji (być może jednak wolniejszą). Podobnie jak autorzy pracy [RGF06] zdecydowałem się na metody statystyczne, ze względu na ich większą uniwersalność, skuteczność i prostotę implementacji.

1.3.2. Metody statyczne i adaptywne

Drugim kryterium klasyfikacji metod kompresji będzie użycie już skompresowanego fragmentu do konstrukcji modelu. Do skutecznej kompresji tekstu potrzebny jest statystyczny model języka — informacja o prawdopodobieństwie wystąpienia poszczególnych ciągów znaków. Ze względu na ten model mamy podział na metody:

- statyczne — model użyty do kompresji jest ustalony dla konkretnej instancji algorytmu i nie zmienia się podczas kompresji,
- adaptywne — model użyty do kompresji jest zależny od tekstu już skompresowanego.

Metody adaptywne dają zwykle lepsze rezultaty od metod statycznych — spowodowane jest to faktem, że informacje o specyfice tekstu najlepiej oddaje sam tekst. Jednak, aby ta przewaga była widoczna, tekst musi być odpowiednio długi. W przypadku krótkich tekstów znacznie lepiej radzą sobie metody statyczne. Konkretnie zdecydowałem się na wybór kodowania arytmetycznego wraz z prostym modelem statystycznym z dwuznakowym kontekstem.

1.4. Kodowanie arytmetyczne

1.4.1. Wprowadzenie

Kodowanie arytmetyczne jest metodą kompresji stworzoną w latach 60. (w liczbach rzeczywistych) i 70. (wykorzystanie liczb całkowitych). Jest ono bardzo skutecznym algorytmem kompresji, który będzie wykorzystywany w dalszej części pracy. Szczegółowo metoda ta opisana jest w książce [BCW90].

Do wprowadzenia kodowania arytmetycznego będziemy potrzebować pewnego modelu źródła wiadomości. Będziemy zakładać, że nasze źródło S jest zbiorem napisów nad pewnym alfabetem Σ, z wyróżnionym znakiem !, który będzie oznaczał koniec wiadomości (tzn. S zawiera tylko i wyłącznie wiadomości kończące się znakiem ! oraz ! występuje tylko jako ostatni znak wiadomości). Oznaczmy przez X_i zmienną losową określającą i-tą literę wiadomości. W algorytmie kompresji będziemy potrzebować wartości:

$$P(X_n = a_n | X_{n-1} = a_{n-1}, X_{n-2} = a_{n-2}, X_{n-3} = a_{n-3}, \ldots, X_1 = a_1)$$

gdzie $a_i \in \Sigma - \{!\}$ dla $i < n$ oraz $a_n \in \Sigma$.

Zauważmy, że wartości te dokładnie określają prawdopodobieństwo każdej wiadomości:

$$P(X_n = a_n, X_{n-1} = a_{n-1}, X_{n-2} = a_{n-2}, \ldots X_1 = a_1) \tag{1.3}$$

$$= P(X_1 = a_1) P(X_2 = a_2 | X_1 = a_1) \ldots P(X_n = a_n | X_{n-1} = a_{n-1}, X_{n-2} = a_{n-2}, \ldots X_1 = a_1)$$

Ten rozkład n-tej litery w zależności od poprzednich będzie w praktyce zależny od modelu źródła, którego będziemy używać. Dokładniej tematyka ta zostanie omówiona w rozdziale 1.5. Tutaj będziemy zakładać, że prawdopodobieństwa te są dane.

1.4.2. Kodowanie za pomocą liczby rzeczywistej

Podstawowym pomysłem w kodowaniu arytmetycznym jest reprezentacja wiadomości za pomocą przedziału liczb rzeczywistych pomiędzy 0 i 1 w ten sposób, aby przedziały odpowiadające różnym wiadomościom były rozłączne. Dokładniej podzielimy przedział $[0, 1)$ na części odpowiadające wiadomościom w zależności od pierwszej litery. W ten sposób wszystkie wiadomości rozpoczynające się na ustaloną literę będą reprezentowane w jednym mniejszym przedziale. Ten z kolei mniejszy przedział podzielimy w zależności od drugiej litery tekstu, ... itd. Długość przedziału przy takim podziale będzie proporcjonalna do prawdopodobieństwa wystąpienia danej litery w kontekście wszystkich wcześniejszych.

Aby to sformalizować wprowadźmy na alfabecie Σ liniowy porządek $\leqslant$. Wtedy:

$$\phi(a_0a_1a_2\ldots a_n!) = \tag{1.4}$$

$$(P(X_0 < a_0) + P(X_0 = a_0, X_1 < a_1) + \ldots + P(X_0 = a_0, X_1 = a_1, \ldots X_n = a_n, X_{n+1} <!),$$

$$P(X_0 < a_0) + P(X_0 = a_0, X_1 < a_1) + \ldots + P(X_0 = a_0, X_1 = a_1, \ldots X_n = a_n, X_{n+1} \leqslant !))$$

Podane we wzorze wartości nie są dane bezpośrednio, ale możemy obliczyć je ze wzoru (1.3).

Przy takim określeniu ϕ oczywistym jest następujący fakt:

Fakt 1.4.1 *Wszystkie wiadomości o prefiksie $a_0a_1a_2\ldots a_m$ mają kody zawarte w przedziale:*

$$(P(X_0 < a_0) + P(X_0 = a_0, X_1 < a_1) + \ldots + P(X_0 = a_0, X_1 = a_1, \ldots X_m < a_m),$$

$$P(X_0 < a_0) + P(X_0 = a_0, X_1 < a_1) + \ldots + P(X_0 = a_0, X_1 = a_1, \ldots X_m \leqslant a_m))$$

1.4.3. Przykład

Weźmy alfabet $\Sigma = \{a, b, c, !\}$ (z porządkiem $a < b < c <!$) oraz:

$$P(X_n = a|\ldots) = P(X_0 = a) \text{ dla } a \in \Sigma$$

$$P(X_0 = a) = 0.5, \quad P(X_0 = b) = 0.2, \quad P(X_0 = c) = 0.2, \quad P(X_0 =!) = 0.1$$

i wiadomość *bacab*!. Rysunek 1.1 przedstawia jak przebiega kodowanie. Dostajemy:

$$\phi(bacab!) = [0.5768, 0.577)$$

1.4.4. Własności

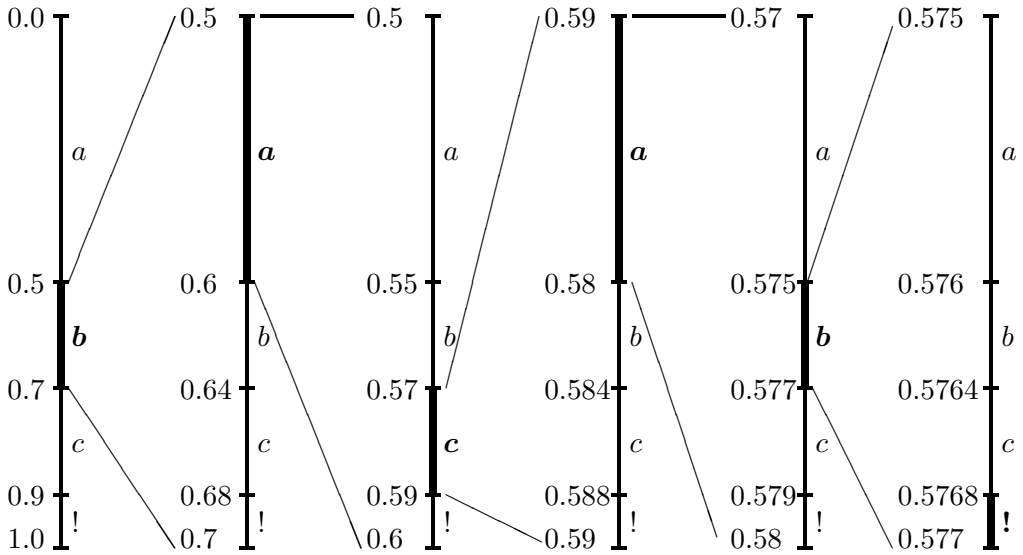
Zauważmy po pierwsze, że jeśli wprowadziliśmy znak kończący wiadomość, to nie musimy podawać całego przedziału reprezentującego wiadomość (tzn. obu jego krańców), ale wystarczy podać dowolną liczbę z tego przedziału. Wynika to z następującego faktu:

Fakt 1.4.2 *Dla różnych $x, y \in S$ zachodzi:*

$$\phi(x) \cap \phi(y) = \emptyset$$

Dowód.

Zauważmy, że przy wprowadzeniu znaku kończącego wiadomość wiadomości tworzą zbiór bezprefiksowy, tzn. dla każdej wiadomości $x \in S$ żaden z jej właściwych prefiksów nie należy do S.



Rysunek 1.1: Przykład kodowania arytmetycznego dla wiadomości bacab!

Weźmy dowolne wiadomości $a, b \in S$. Ponieważ żadna z nich nie jest prefiksem drugiej, więc istnieje taki najmniejszy indeks i, że $a_i \neq b_i$, a więc zgodnie z faktem 1.4.1 ich kody zawarte są w rozłącznych przedziałach. Jeśli wprowadzimy oznaczenie:

$$f(a_0 a_1 \dots a_i) = P(X_0 < a_0) + P(X_0 = a_0, X_1 < a_1) + \dots + P(X_0 = a_0, \dots X_i = a_i)$$

są to odpowiednio przedziały:

$$(f(a_0 a_1 \dots a_{i-1}) + P(X_0 = a_0, X_1 = a_1, \dots X_i < a_i),$$

$$f(a_0 a_1 \dots a_{i-1}) + P(X_0 = a_0, X_1 = a_1, \dots X_i \leqslant a_i))$$

oraz

$$(f(a_0 a_1 \dots a_{i-1}) + P(X_0 = a_0, X_1 = a_1, \dots X_i < b_i),$$

$$f(a_0 a_1 \dots a_{i-1}) + P(X_0 = a_0, X_1 = a_1, \dots X_i \leqslant b_i))$$

które są rozłączne.

□

Zauważmy również, że:

Fakt 1.4.3 *Dla $a \in S$ długość przedziału $\phi(a)$ jest równa $P(a)$.*

Dowód.

Ze wzoru (1.4) mamy, że ta długość jest równa:

$$P(X_0 = a_0, X_1 = a_1, \dots X_n = a_n, X_{n+1} \leqslant !) - P(X_0 = a_0, X_1 = a_1, \dots X_n = a_n, X_{n+1} < !) =$$

$$= P(X_0 = a_0, X_1 = a_1, \dots X_n = a_n, X_{n+1} = !) = P(a)$$

□

Ponieważ do reprezentacji $\phi(a)$ możemy wybrać dowolną liczbę z przedziału, więc wybierzemy taką, która ma możliwie najkrótsze binarne przedstawienie. Wykorzystamy w tym celu następujący fakt:

Fakt 1.4.4 *Mamy dany przedział $[a, b)$. Niech $k \in \mathcal{Z}$ będzie największe takie, że $2^k \leqslant b - a$. Wtedy istnieje $x \in [a, b)$ takie, że $2^k x \in \mathcal{N}$.*

Dowód.

Niech:

$$x = \frac{\lceil 2^k a \rceil}{2^k} \in \mathcal{N}$$

Mamy:

$$0 \leqslant \lceil 2^k a \rceil - 2^k a < 1$$

A wię

$$x - a = \frac{\lceil 2^k a \rceil}{2^k} - a \geqslant 0$$

oraz

$$x - a = \frac{\lceil 2^k a \rceil}{2^k} - a < 2^k \leqslant b - a$$

A więc $x \in [a, b)$.

□

Możemy więc do reprezentacji $\phi(a)$ użyć takiej właśnie liczby. Z ostatnich faktów wynika więc, że:

$$|\phi(a)| = -\lfloor logP(a) \rfloor \tag{1.5}$$

Twierdzenie 1.4.1 *Dla kodowania arytmetycznego dla skończonego zbioru S, $|S| = n$ zachodzi:*

$$H(S) \leqslant E(|\phi(S)|) \leqslant H(S) + 1$$

Dowód.

Pierwsza nierówność to wprost twierdzenie 1.2. Z własności $\lfloor \cdot \rfloor$ oraz z (1.5) mamy dla każdego $i = 1 \ldots n$:

$$|\phi(s_i)| = -\lfloor \log_2 p(s_i) \rfloor \leqslant -\log_2 p(s_i) + 1$$

$$p(s_i)|\phi(s_i)| \leqslant -p(s_i) \log_2 p(s_i) + p(s_i)$$

Po zsumowaniu tej nierówności dla $i = 1, \ldots n$ dostajemy:

$$\sum_{i=1}^{n} p(s_i)|\phi(s_i)| \leqslant \sum_{i=1}^{n} (-p(s_i) \log_2 p(s_i) + p(s_i)) = \sum_{i=1}^{n} (-p(s_i) \log_2 p(s_i)) + 1$$

□

Twierdzenie to orzeka więc, że kodowanie arytmetyczne jest optymalną metodą kompresji względem danego modelu prawdopodobieństwa z dokładnością do jednego znaku.

Transmisja przyrostowa

Zgodnie z dotychczasowymi rozważaniami naszymi kodami będą ciągi bitów będące binarnymi zapisami dowolnej (jak najkrótszej) liczby z przedziału reprezentującego wiadomość. Jednak pamiętanie informacji o całym takim ciągu i wykonywanie operacji arytmetycznych na nim podczas kompresji/dekompresji byłoby bardzo kosztowne. Ale już w trakcie kompresji można zapomnieć o części z tych bitów, jeśli są już one jednoznacznie wyznaczone. Przykładowo, gdy dostaliśmy już przedział $[0.5, 0.7)$, to wiemy, że pierwszym bitem kodu będzie 1. Możemy więc go zapamiętać i od tego momentu jako wejściowy przedział pamiętać $[0.5, 1)$, czyli przeskalować przedział $[0.5, 1)$ na $[0, 1)$. Również podczas dekompresji można działać przyrostowo, jak poprzednio identyfikując przedziały — jeśli pierwszym odczytanym bitem jest 1, to przedział $[0.5, 1)$ przeskalujemy na $[0, 1)$. Dodatkowo, gdy będziemy już pewni, że rozpoznana liczba z pewnością należy do przedziału odpowiadającego konkretnej literze, to będzie można ją już wyprowadzić na wyjście. Przykładowo, gdy dostaniemy ciąg bitów 100, to wiemy już, że kod zawiera się w przedziale $[0.5, 0.625)$, a więc również w przedziale $[0.5, 0.7)$, co pozwala wyprowadzić na wyjście literę b. Przy każdej takiej operacji długość aktualnego przedziału w którym zawarta jest wiadomość zwiększa się dwukrotnie, co pozwala na zwiększenie dokładności obliczeń.

1.4.5. Kodowanie za pomocą liczb całkowitych

Powstaje pytanie, jak reprezentować końce przedziałów. Z różnych względów najwygodniejsza jest reprezentacja za pomocą liczb całkowitych. I dlatego przy ustalonej liczbie MAX (równej przykładowo 2^{16}) przedział $[0, 1)$ będziemy reprezentować jako $[0, MAX - 1]$.

Wiąże się z tym kilka problemów:

- pożądana niezerowość prawdopodobieństw.

  Chcielibyśmy, aby każda wiadomość była możliwa do uzyskania, stąd każdy z przedziałów musi mieć dodatnią długość. Określając podprzedział w $[0, MAX - 1]$ na liczbach całkowitych musimy więc dbać o to, aby prawdopodobieństwo dla żadnej litery nie było mniejsze od $1/MAX$. Ze względu na dokładność przybliżeń pożądane jest, aby prawdopodobieństwa były nawet znacznie większe od tej wartości.

- zmniejszanie szerokości przedziału przy liczbach bliskich $0, 5$.

  Gdy docelowa liczba jest bliska $0, 5$, może to powodować zmniejszanie szerokości przedziału bez wyprowadzania na wyjście żadnych bitów, ponieważ lewy koniec przedziału może być ciągle w pierwszej połowie głównego przedziału, a prawy w drugiej. Może to spowodować szybką utratę dokładności, a nawet ze względu na obliczenia na liczbach całkowitych — zrównanie lewego i prawego końca przedziału. Dlatego oprócz skalowania głównego przedziału, gdy lewy i prawy koniec mieszczą się w tej samej połowie, będziemy tak również czynić, gdy znajdują się one w przedziale $[0.25, 0.75)$ i dodatkowo zapamiętywać ile takich operacji wykonaliśmy. Później jeśli ustali się już połowa aktualnego przedziału, w której znajduje się wiadomość, będzie można wypisać odpowiednią liczbę przeciwnych bitów. W przypadku zakończenia wiadomości przed wystąpieniem takiej sytuacji można wybrać dowolną część przedziału.

1.5. Modelowanie języka

1.5.1. Wprowadzenie

W poprzednim rozdziale rozważaliśmy zbiór S wszystkich wiadomości i entropię tego zbioru, do zdefiniowania której potrzebny był pewien rozkład prawdopodobieństwa na S. W tym rozdziale zajmiemy się problemem właściwego dla języka naturalnego określenia zbioru S i prawdopodobieństw poszczególnych wiadomości.

Zauważmy, że od określenia zbioru S i rozkładu prawdopodobieństwa zależy ograniczenie na skuteczność kompresji wynikające z twierdzenia 1.2.

Z postawionym zadaniem wiąże się kilka istotnych problemów:

- rozkład powinien mieć jak najmniejszą entropię, aby kompresja była efektywna,
- język naturalny jest skomplikowany i niejednoznaczny, chcemy modelować również niepoprawne wiadomości (np. zawierające literówki),
- większość wiadomości jest unikatowa, więc nowy tekst, który dostajemy do kompresji prawdopodobnie nigdy wcześniej w całości nie wystąpił,
- obliczanie rozkładu powinno być wydajne.

Ze względu na obszerność tematu zajmiemy się tylko wybranymi zagadnieniami i metodami przydatnymi w projekcie. W szczególności interesować nas będą głównie metody statystyczne. Alternatywny podejściem jest modelowanie języka od strony lingwistycznej, poprzez analizę gramatyczną tekstu — podział na części mowy i części zdania. Są one jednak zbyt skomplikowane i czasochłonne dla naszych zastosowań, ze względu na ograniczenia wymienione w podrozdziale 1.2.3.

1.5.2. Metody statystyczne

Zgodnie z wcześniejszymi rozważaniami dla kodowania arytmetycznego wygodnie nam będzie podzielić tekst na pewne jednostki tekstu (zwykle litery lub słowa) i rozpatrywać prawdopodobieństwa warunkowe wystąpienia danej jednostki w kontekście poprzednich. Przyjmiemy podobnie jak poprzednio następujące oznaczenia przy podziale wiadomości na jednostki tekstu:

$$s = x_0 x_1 \dots x_n \quad \text{gdzie } s \in S, x_i \in \Sigma$$

Wyznaczanie rozkładu prawdopodobieństwa musi opierać się na pewnych danych. Ponieważ rozpatrujemy metody statystyczne, dane te pochodzić będą z pewnego dużego zbioru tekstów S' (korpusu), reprezentatywnego dla języka (czyli zbioru S). Zbiór S' jest bardzo mały w porównaniu do zbioru S. Powoduje to, że nasz model nie może oddawać zbyt dokładnie zbioru S', musi rozszerzać swój zakres poza ten zbiór tak, aby bardziej oddać cechy języka, a nie konkretnych tekstów. W ekstremalnej sytuacji, gdy model akceptowałby z niezerowym prawdopodobieństwem tylko wiadomości ze zbioru S', żadna wiadomość spoza S' nie mogłaby zostać skompresowana. Jedną z metod na osiągnięcie tego celu jest ograniczenie kontekstu. W tym celu wprowadzamy pojęcie rzędu modelu:

Definicja 1.5.1 *Model nazywamy modelem k-tego rzędu, gdy:*

$$P(x_n = a_n | x_{n-1} = a_{n-1}, x_{n-2} = a_{n-2}, \dots, x_0 = a_0) = \tag{1.6}$$

$$= P(x_n = a_n | x_{n-1} = a_{n-1}, \dots, x_{n-k} = a_{n-k}) = P(x_{n-1} = a_n | x_{n-2} = a_{n-1}, \dots, x_{n-k-1} = a_{n-k})$$

a więc prawdopodobieństwo wystąpienia znaku będzie zależało tylko od poprzednich k liter, nie będzie zależało ani od pozostałych liter tekstu, ani od pozycji litery w tekście.

Przy tak określonych ograniczeniach możemy już wskazać sposób wyznaczenia prawdopodobieństw po prawej stronie wzoru 1.6 na podstawie zbioru S'. Mianowicie prawdopodobieństwa wystąpienia litery w konkretnym k-znakowym kontekście będzie proporcjonalne do liczby wystąpień tej litery w tym właśnie kontekście w zbiorze S'.

Przykład 1.5.1 *Dla modelu* 2*-go rzędu oraz korpusu:*

```
alabgalcalcalbala
```

mamy w kontekście al wystąpienia a, c, c, b oraz a, a więc:

$$P(a|al) = 0.4 \qquad P(c|al) = 0.4 \qquad P(b|al) = 0.2$$

Skuteczność modeli różnych rzędów możemy zobaczyć generując losowe wiadomości. Im bardziej przypominają one tekst języka naturalnego, tym lepszy jest model:

- 0-go rzędu (bez kontekstu - statystyki na pojedynczych literach):

  ```
  zomItjkoom,pyagi ócy .w zi.kje-  izicąsizd  gaalię  oj-iayaowtab
  ```

- 1-go rzędu:

  ```
  io cja c, st, nta LNa biacza?goosciele
  ```

- 2-go rzędu:

  ```
  omi fajcznetajedzie z A Szaciąże Vetracji, ali
  ```

Wygładzanie

Jednym z problemów przy modelach wysokiego rzędu jest spore prawdopodobieństwo nieadekwatności zbioru testowego. Długi kontekst może powodować, że tekst nie występujący w zbiorze testowym nie będzie miał odpowiedniego prawdopodobieństwa. Dodatkowo ze względów praktycznych wygodnie jest, aby każda wiadomość miała niezerowe prawdopodobieństwo (być może bardzo małe). Ale opierając statystyki na zbiorze testowym łatwo możemy dla dłuższego kontekstu dostać zerowe prawdopodobieństwa (przykładowo przy kontekście *al* możemy nie dostać *al*4 lub podobnej trójki liter). Dlatego wygodne jest wprowadzenie prawdopodobieństwa litery jako średniej ważonej. Na przykład dla modelu k-tego rzędu:

$$P(x_n = a_n|x_{n-1} = a_{n-1}, x_{n-2} = a_{n-2}, \ldots, x_0 = a_0) =$$

$$\alpha_k P(x_n = a_n|x_{n-1} = a_{n-1}, \ldots, x_{n-k} = a_{n-k}) + \alpha_{k-1} P(x_n = a_n|x_{n-1} = a_{n-1}, \ldots, x_{n-k+1} = a_{n-k+1}) +$$

$$\ldots + \alpha_1 P(x_n) + \alpha_0$$

gdzie $\alpha_i \geqslant 0$ oraz $\sum_{i=0}^{n} \alpha_i = 1$.
W szczególności wtedy jeśli $\alpha_0 > 0$, to wszystkie takie prawdopodobieństwa są dodatnie. Oczywiście dobór współczynników α_i jest rzeczą trudną i zwykle odbywa się to na drodze eksperymentalnej.

Zmienna długość kontekstu

W nowoczesnych metodach kompresji tekstu takich jak PPM (patrz [Ciw84]) stosowana jest zmienna długość kontekstu. Dla różnych miejsc w tekście dostępne są konteksty różnej długości (wybór tych kontekstów zależy od przyjętej strategii), a używany jest najdłuższy dostępny. Zwykle stosowane są one w metodach adaptywnych. Można jednak wyobrazić sobie ich użycie w metodach statycznych — dla częściej występujących znaków konteksty będą dłuższe, a dla rzadziej występujących — krótsze.

Rozdział 2

Korpus języka

W tym rozdziale opiszemy tworzenie modelu statystycznego języka potrzebnego przy zastosowaniu kodowania arytmetycznego. Opiszemy wybór i tworzenie korpusu oraz generowanie statystyk dla zebranych danych.

2.1. Personalizacja

Ze względu na wybór statycznych metod kompresji musimy dostarczyć zbiór tekstów na podstawie którego będzie dokonywana kompresja. Zbiór ten (zwany korpusem) powinien jak najlepiej odpowiadać językowi SMS-ów. Pojawiają się tutaj dwa przeciwstawne rozwiązania:

- korpus spersonalizowany — korpus zależny od treści wiadomości przesyłanych między dwoma konkretnymi użytkownikami lub w pewnej grupie użytkowników. Można sobie wyobrazić tworzenie takiego korpusu w etapie testowym działania aplikacji w pewnej umówionej grupie użytkowników, która później będzie już z takiego korpusu korzystać, lub z dynamicznym tworzeniem takiego korpusu i jego cyklicznymi zmianami po pewnej liczbie wysłanych SMS-ów (np. zmiana korpusu następowałaby co kilkadziesiąt wysłanych SMS-ów). Wydaje się, że takie rozwiązanie może dawać lepszy stopień kompresji. Nie jest to jednak wcale takie oczywiste, ponieważ taki korpus niekoniecznie musi być reprezentatywny (tematyka SMS-ów może się zmieniać w czasie) oraz zwykle będzie on stosunkowo niewielki (a większy korpus daje lepszy współczynnik kompresji). Dodatkowe problemy przy takim rozwiązaniu to ograniczenie komunikacji tylko do pewnych użytkowników (tzn. tych, którzy uczestniczą w generowaniu korpusu) oraz konieczność dodatkowej komunikacji innymi kanałami w celu synchronizacji korpusów.
- korpus globalny — korpus ustalony dla aplikacji, zależny tylko od języka (w naszym przypadku będzie to język polski). Taki korpus (a raczej statystyki z niego wygenerowane) może być zapamiętany sztywno w aplikacji lub ewentualnie dołączany jako dodatek do niej. Takie rozwiązanie jest znacznie bardziej ogólne, łatwiejsze w testowaniu i użytkowaniu, nie wymaga też od użytkowników żadnego dodatkowego działania.

Z podanych powodów zdecydowałem się na wybór korpusu globalnego.

2.2. Wybór korpusu

W związku z wyborem korpusu globalnego teksty użyte do modelowania języka należy zgromadzić podczas tworzenia aplikacji. Teksty te powinny możliwie dobrze odpowiadać językowi

używanemu podczas pisania SMS-ów oraz występować w odpowiedniej liczbie (rzędu co najmniej megabajtów). Godne rozważenia wydają się opisane dalej źródła.

Sieciowe korpusy języka polskiego

Lingwiści zajmują się zbieraniem i publikowaniem do badań nad językiem korpusów językowych. Istnieje kilka takich sieciowych korpusów, z których najpopularniejsze to korpus IPI PAN (www.korpus.pl) oraz Korpus Języka Polskiego Wydawnictwa Naukowego PWN (korpus.pwn.pl). Korpusy te są duże, profesjonalnie przygotowane i zrównoważone (czyli zawierają w odpowiednich proporcjach teksty z różnych dziedzin i źródeł). Ich podstawowym celem jest jednak reprezentacja współczesnej polszczyzny. Powstaje więc pytanie o adekwatność tych korpusów, ponieważ język SMS-ów różni się od typowej polszczyzny. Te różnice zauważalne są (i jest to już istotne z punktu widzenia użytych metod kompresji) przykładowo w częstości wykorzystania znaków innych niż alfanumeryczne. Dla celów porównawczych zdecydowałem się wykorzystać "korpus słownika frekwencyjnego polszczyzny współczesnej" z lat sześćdziesiątych dostępny na licencji GNU GPL na stronie `http://www.mimuw.edu.pl/polszczyzna/pl196x/index.htm`. Jest on co prawda stosunkowo skromny, ale w przeciwieństwie do większych korpusów jest darmowy i dostępny w formie surowej, którą niewielkim nakładem pracy można przekonwertować do czystego tekstu.

Baza SMS-ów

Alternatywą do wykorzystania gotowego korpusu jest samodzielne jego stworzenie. W takim przypadku najlepszym wydaje się sięgnięcie do źródła i stworzenie bazy SMS-ów służącej jako korpus. Jednak jest to zadanie skomplikowane. Aby zebrać odpowiednią ilość danych wystarczająco reprezentatywnych wymagana byłaby współpraca operatorów komórkowych oraz spory nakład pracy. Dodatkowo można napotkać na problemy prawne związane z ochroną informacji użytkowników. Wymagałoby to również oceny przeznaczenia przesyłanych SMS-ów, ponieważ część z nich jest generowana automatycznie (przez bramki internetowe, serwisy internetowe, banki), co może zaburzać statystyki. Dodatkowo byłby to korpus odzwierciedlający obecny język SMS-ów, dostosowany do ich obecnej długości — być może po wprowadzeniu kompresji ten język zmieniłby się. Ostatecznie więc w projekcie to źródło nie jest używane, ale nic nie stoi na przeszkodzie, aby zrobić to w przyszłości, jeśli powyższe trudności zostaną przełamane.

Internet

Ostatnim źródłem mogącym posłużyć do samodzielnego stworzenia korpusu jest Internet. Można wyobrazić sobie zbieranie tekstów ze stron internetowych, ich odpowiednią klasyfikację i użycie ich jako korpus języka. Jest to źródło wystarczająco obfite do naszych zastosowań. Przy odpowiedniej selekcji tekstów może być również adekwatne, tzn. z dobrym przybliżeniem oddawać język używany w SMS-ach. Takie więc rozwiązanie zostało przyjęte jako główne w tym projekcie. Pierwszym zadaniem przy zastosowanych przez nas metodach kompresji było zebranie bazy tekstów ze stron internetowych w celu użycia ich do stworzenia modelu statystycznego języka.

Przyjęte rozwiązanie wymaga ściągnięcia dużej ilości materiałów ze stron WWW. Przyjąłem, że będę korzystał wyłącznie z tekstów umieszczonych bezpośrednio na stronach w plikach `.html`, będących w całości w języku polskim (bez wyodrębniania polskich tekstów ze stron wielojęzycznych).

2.3. Rozpoznawanie języka

Ze względu na specyfikę Internetu do zbierania tekstów do analizy języka będziemy potrzebować metody ich klasyfikacji.

2.3.1. Opis problemu

Automatyczna kategoryzacja tekstów, czyli przypisywanie wejściowemu tekstowi pewnej kategorii, jest jednym z podstawowych zadań w elektronicznym przetwarzaniu dokumentów. Czasem możemy też chcieć podzielić tekst na fragmenty będące w różnych kategoriach. Skupimy się jednak na tym pierwszym zadaniu. W zależności od warunków używane są do kategoryzacji tekstów różne metody. Jesteśmy jednak zainteresowani metodami spełniającym następujące kryteria:

- Kategoryzacja powinna działać poprawnie pomimo błędów w tekście,
- Cały proces powinien być efektywny czasowo i pamięciowo, gdyż zwykle mamy do czynienia z kategoryzacją dużej liczby tekstów i przetwarzaniem strumieniowym,
- Kategoryzacja powinna umożliwiać wykrycie tekstu, który nie pasuje do żadnej kategorii lub należy do kilku z nich (np. tekst w kilku językach).

2.3.2. Kategoryzacja tekstów oparta na n-gramach

Przedstawię tutaj za pracą [Can94] podstawową i łatwą w implementacji metodę kategoryzacji tekstów opartą na n-gramach.

N-gramy

Definicja 2.3.1 *N-gramem nazywamy każdy n-znakowy ciągły fragment tekstu.*

Przykładowo dla tekstu:

`Ala_ma_kota.`

mamy następujące n-gramy:

- 1-gramy:

  `A, l, a, _, m, k, o, t, .`

- 2-gramy (bi-gramy):

  `Al, la, a_, _m, ma, _k, ko, ot, ta, a.`

- 3-gramy (tri-gramy):

  `Ala, la_, a_m, _ma, ma_, a_k, _ko, kot, ota, ota.`

N-gramy mają szerokie zastosowania w przetwarzaniu tekstów.

Przydatne tutaj będzie prawo Zipfa. Do jego sformułowania potrzebujemy pojęcia rangi jednostki tekstu (np. słowa). Jest to po prostu jej numer na liście uszeregowanej malejąco względem częstości występowania. Prawo Zipfa (jest to prawo empiryczne) stanowi, że

Twierdzenie 2.3.1 (Prawo Zipfa) *Iloczyn rangi jednostki tekstu i jego częstości jest stały:*

$$p(r) = \frac{\mu}{r}, \quad r = 1, 2, 3, \ldots$$

gdzie $p(r)$ jest częstością r-tej jednostki tekstu na liście, a μ jest pewną stałą.

Konsekwencją tego prawa jest to, że konkretne wartości częstości nie są takie istotne (będą prawdopodobnie spełniać prawo Zipfa), ale istotna jest kolejność występowania tych jednostek na liście. Okazuje się, że najczęściej występujące n-gramy w tekście (te o bardzo małym r w prawie Zipfa) bardzo dobrze charakteryzują tekst (m.in. pod względem języka). Ze względów praktycznych zgodnie z [Kran05] najlepszym wyborem okazały się n-gramy dla $n = 2, 3$.

Możemy więc wprowadzić dla każdego n wektor pewnej liczby n-gramów najczęściej występujących w tekście uszeregowanych względem częstości:

```
ie, ni, nie, ra, pr, rz, ze, ...
```

Taki ciąg n-gramów będziemy nazywać profilem. Dla każdej z interesujących nas kategorii (np. języka polskiego) możemy wyznaczyć taki profil na odpowiednio dobranym zbiorze treningowym. Generalnie dla wejściowego tekstu z danej kategorii profil nie będzie identyczny z wyznaczonym na zbiorze treningowym. Potrzebujemy więc pewnej miary podobieństwa profili do oceny jak bardzo tekst pasuje do danej kategorii oraz wyboru, do której kategorii pasuje najlepiej.

Odległość profili

Do porównywania profili można stosować wiele różnych miar. Najprostszą z nich i jednocześnie za [Hab05] wystarczająco skuteczną jest następująca miara określająca jak bardzo poszczególne elementy są "nie na miejscu" na liście:

$$s(v, w) = 1 - \frac{\sum_{i=1}^{l} d_{v,w}(v_i) + \sum_{i=1}^{l} d_{w,v}(v_2)}{2l^2}$$

gdzie

$$d_{v,w}(v_i) = \begin{cases} |i - j| & \text{gdy } w_j = v_i \\ l & \text{gdy } v_i \text{ nie ma na liście } w \end{cases}$$

natomiast v i w są profilami (wektorami indeksowanymi rangą).

Miara ta jest to po prostu suma po wszystkich elementach obu wektorów ich odległości na obu listach przeskalowana do przedziału $(0, 1)$.

Możliwe są również inne miary jak na przykład iloczyn skalarny między wektorami.

2.3.3. Kategoryzacja

Posiadając wzorcowe profile dla każdej z kategorii oraz miarę odległości profili można dla każdego tekstu wyznaczyć najbliższy wzorcowy profil i jego kategorię przypisać tekstowi. W naszym projekcie mamy tylko jeden wzorcowy profil — dla języka polskiego. Chcemy dla każdego tekstu ocenić, czy jest on odpowiedni dla tego profilu. Można więc użyć miary odległości profili (wzorcowego i dla danego tekstu) do tej oceny i ustawić pewien próg odległości, powyżej którego będziemy uznawać tekst za zgodny z tą kategorią.

Metoda ta spełnia przedstawione w rozdziale 2.3.1 wymagania:

- Jest odporna na drobne błędy w tekście — nie wpływają one w znaczący sposób na profil.

- Jest efektywna czasowo i pamięciowo — gromadzenie profilu może być wykonywane strumieniowo (wystarczy pamiętać kilka ostatnich znaków), a samo obliczanie odległości również jest szybkie. Sposób realizacji zostanie opisany w rozdziale 2.6.4.

- Umożliwia wykrywanie tekstów nie odpowiadających żadnej kategorii — jest tak w sytuacji, gdy profil tekstu jest odległy od wszystkich profili wzorcowych.

2.4. Roboty internetowe

Do automatycznego przeszukiwania i archiwizacji danych z sieci służą programy zwane robotami internetowymi (ang. *crawlers*). Robot taki przeszukuje sieć rozpoczynając od pewnego ustalonego zbioru adresów, podążając za odnośnikami znalezionymi w już przeszukanych zasobach. Standardowo robot przechowuje w pamięci kolejkę adresów, z której wybiera następny adres do odwiedzenia, a po ściągnięciu strony archiwizuje ją, a otrzymane z niej odnośniki dołącza do kolejki.

Do naszych zastosowań typowy przypadek użycia robota jest właśnie taki: rozpoczynając od pewnej puli adresów robot przeszukuje sieć. Z każdego ściągniętego dokumentu `.html` wydobywa odnośniki do innych dokumentów, które wykorzystywane są do dalszego przeszukiwania, oraz tekst, który jest oceniany pod kątem użyteczności i w razie potrzeby archiwizowany (po pozytywnej weryfikacji).

Podczas tworzenia robota występuje wiele problemów, takich jak:

- ekstrakcja odnośników — potrzebny jest parser kodu, który pozwoli ze ściągniętej strony wydobyć adresy, na które dana strona wskazuje,

- obsługa błędów — przy przeszukiwaniu Internetu występuje wiele błędów, które muszą zostać obsłużone bez konieczności ponownego uruchamiania całego procesu,

- tworzenie logów — przydatne dla operatora obsługującego działanie robota,

- obsługa `robots.txt` — jest to specjalny plik umieszczany na większości serwerów, który określa politykę serwera względem robotów,

- wielowątkowość — ze względów wydajnościowych wygodnie jest mieć wiele wątków obsługujących różne adresy,

- równoważenie obciążenia — ściąganie z wielu serwerów jednocześnie tak, aby zbytnio nie obciążać tylko wybranych,

- obsługa przerywania i wznawiania zadań.

Tworzenie od początku własnego dobrego robota jest zadaniem skomplikowanym. Dlatego zdecydowałem się na skorzystanie z gotowego rozwiązania i dostosowanie go do własnych potrzeb.

2.5. Heritrix

Heritrix jest darmowym, modyfikowalnym, skalowalnym, wielowątkowym robotem internetowym. Więcej informacji można znaleźć na stronie [Heritirix]. Podstawowe cechy projektu, które zadecydowały o jego wyborze, to:

- wielowątkowość,
- strategia przeszukiwania wgłąb napotkanych adresów,
- akceptuje `robots.txt`,
- możliwość przeglądania poszczególnych domen, węzłów oraz stosowania wyrażeń regularnych do selekcji adresów,
- modularna budowa, dostępność kodu na licencji GPL, łatwe modyfikacje,
- konfigurowalność:
  - położenia archiwów i logów,
  - ograniczeń pamięciowych, liczby wątków, wielkości ściąganych zasobów, ograniczeń przepustowości,
  - rozbudowany system filtrowania zasobów,
  - ułatwione dodawanie opcji konfiguracyjnych do własnych modułów,
- tworzenie raportów z wykonanych zadań,
- darmowość i łatwa dostępność,
- przenośność — napisany w Javie,
- interfejs operatora przez WWW (rysunek 2.1).

2.5.1. Schemat działania

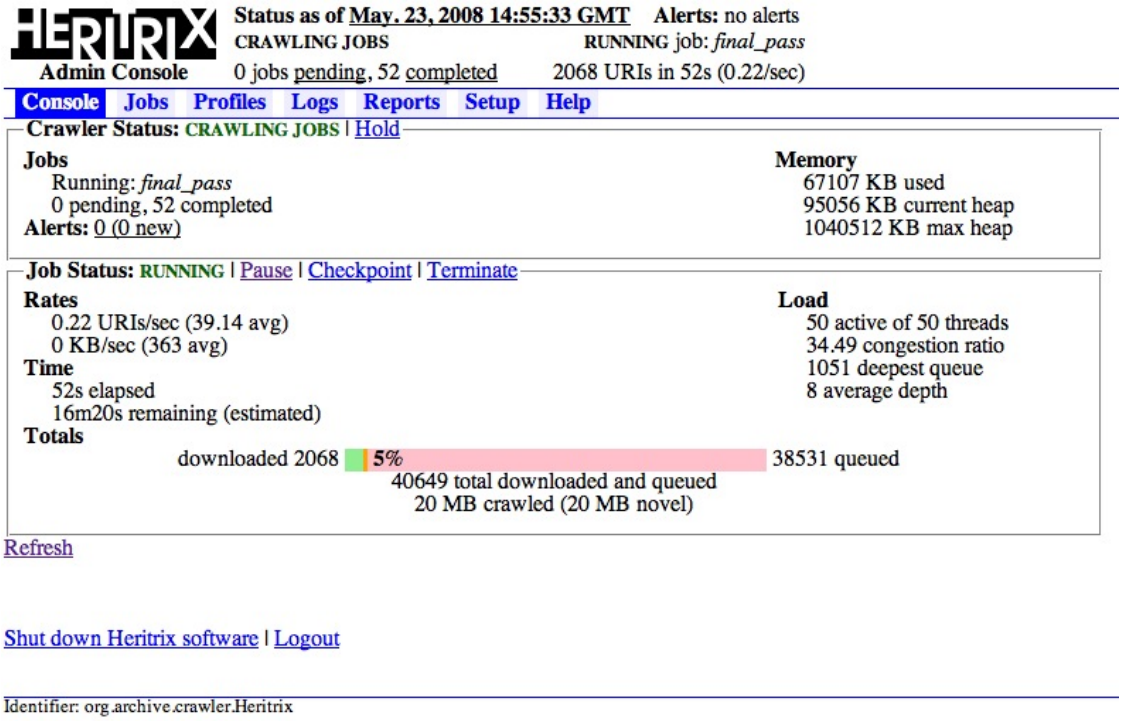
Podstawowy schemat działania Heritriksa jest typowy dla wszystkich robotów:

- wybierz adres z kolejki,
- ściągnij dane z tego adresu,
- archiwizuj dane,
- wydobądź ze ściągniętych danych adresy do odwiedzenia i dopisz je do kolejki,
- odznacz adres jako odwiedzony i wróć do punktu 1.

Podstawowe komponenty Heritriksa odpowiedzialne za te działania to:

- `Scope` — odpowiedzialny za selekcję adresów do odwiedzenia i dodanie ich do kolejki,
- `Frontier` — odpowiedzialny za wybór następnego adresu do odwiedzenia oraz eliminację z kolejki już odwiedzonych adresów,
- `Processor Chains` — łańcuch komponentów, które dla konkretnego adresu są kolejno uruchamiane, obejmują między innymi ściąganie danych, ich analizę i zwracanie otrzymanych odnośników,

Dokładniej budowę robota obrazuje rysunek 2.2.



Rysunek 2.1: Interfejs Heritriksa

2.6. Wymagane zmiany

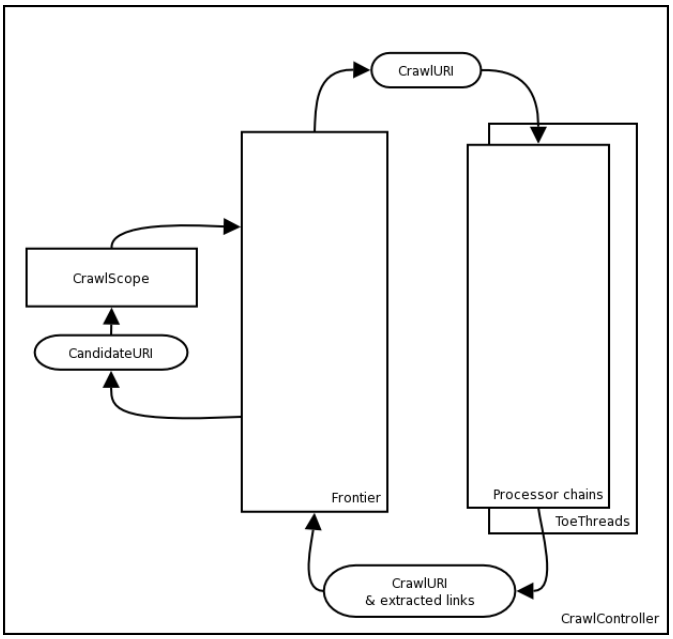
Heritrix jest rozbudowaną aplikacją i posiada wiele opcji konfiguracyjnych, wymagał jednak sporej liczby zmian, aby dostosować jego działanie do naszych potrzeb. W dalszej części tego podrozdziału przedstawię wykonane zmiany.

2.6.1. Ekstrakcja tekstu z dokumentów HTML

Ściągnięte dane zawierają kod HTML-owy, z którego należy wydobyć tekst, który wyświetlany jest na stronie. Do realizacji tego zadania wykorzystałem bibliotekę `Jericho` — parser HTML-a w Javie, używaną przez Heritriksa, która zawiera klasę `au.id.jericho.lib.html.Source` pozwalającą m.in. na ekstrakcję tekstu.

2.6.2. Zapisywanie tekstu

Nie będziemy archiwizować wszystkich ściągniętych danych, ale sam tekst. Standardowy sposób archiwizacji tworzy skomplikowaną strukturę katalogów (o strukturze podobnej do adresu witryny), która nie jest konieczna przy naszych zastosowaniach, a zmniejsza wydajność. Z drugiej strony niepożądane jest przechowywanie wszystkich tekstów w jednym pliku. Mogłoby to spowodować problemy z wielowątkowością. Konieczne jest więc stworzenie własnej podklasy klasy `Writer`, która będzie odpowiedzialna za tworzenie plików docelowych i zapisywanie do nich otrzymanego tekstu. Przyjąłem, że poszczególne fragmenty tekstu zapisywane będą w osobnych plikach o nazwach będących kodami `MD5` z ich treści, w katalogu wskazanym w opcjach konfiguracyjnych, co pozwoli dodatkowo eliminować duplikaty.



Rysunek 2.2: Budowa robota, schemat ze strony http://crawler.archive.org

2.6.3. Obsługa kodowań polskich znaków

W polskim Internecie szerzej stosowane są 3 kodowania polskich znaków — `iso-8859-2`, `windows-1250` oraz `UTF`. Konieczne więc było wybranie jednego z nich jako kodowania głównego. `iso-8859-2` jest kodowaniem 8-bitowym.W telefonach komórkowych używanych w Polsce cały zestaw znaków zawarty jest w kodowaniu `iso-8859-2`. Nie ma więc potrzeby używania 16-bitowego kodowania `UTF`. Stąd jako główne przyjąłem kodowanie `iso`. Do konwersji wykorzystałem standardowe metody Javovej klasy `String`.

2.6.4. Rozpoznawanie języka i odrzucanie niechcianych tekstów

Ściągnięte teksty wymagają jeszcze klasyfikacji. Stoją za tym dwa ważne powody:

- trudności w rozpoznawaniu języka na podstawie znaczników `HTML` oraz kodowania znaków.

  Informację ze znaczników `HTML` oraz o kodowaniu znaków uznałem za zbyt mało wiarygodną dla określenia języka witryny. Przykładowo kodowanie `UTF` nie określa praktycznie w ogóle języka strony, a standard `iso-8859-2` obejmuje znaki diakrytyczne kilku języków. Dodatkowo nawet witryna oznaczona jako polska może zawierać np. tekst w języku angielskim lub w kilku różnych językach.

- niska jakość materiałów sieciowych.

  Nawet jeśli na stronie używany jest język polski, to część stron zawiera materiały nieprzydatne dla naszych zastosowań: nazwy elementów menu, słowa kluczowe, teksty powtarzające się na wielu stronach. Dlatego ważna jest ocena jakości ściągniętego tekstu.

Być może problem z rozpoznawaniem języka udałoby się rozwiązać analizując dla każdej strony czy zawiera ona polskiego znaki diakrytyczne i w jakim procencie. Jednak rozwiąza-

nie problemu niskiej jakości tekstów wymaga już bardziej wyrafinowanych metod. Za pracą [Can94] stosuję tutaj metody opisane w rozdziale 2.3.2. Przypomnijmy, że metoda ta opiera się na założeniu, że profile złożone z najczęściej występujących n-gramów dobrze charakteryzują język i są stosunkowo niezależne od tematyki tekstu. Należy więc najpierw stworzyć profil wzorcowy dla języka polskiego, a następnie dla każdego ocenianego tekstu wyznaczyć odległość jego profilu od profilu wzorcowego. W praktyce użyłem profilu długości 100 złożonego z n-gramów długości 2 i 3. Profil wzorcowy stworzyłem na podstawie dostępnej w internecie książki w języku polskim (`http://mateusz.pl/ksiazki/dszp/`). Nie jest ona oczywiście zbyt reprezentatywna dla języka polskiego jako całości, ale do naszych zastosowań, ze względu na niezależność krótkich profili od tematyki tekstu, jest wystarczająca. Następnie dla każdego ściągniętego przez robota tekstu jego profil był porównywany z profilem wzorcowym i jeśli odległość była mniejsza od ustalonej stałej wyznaczonej doświadczalnie na podstawie kilku przykładów (m.in. sprawdzałem odległości profili dla tekstów w innych językach), to tekst był uznawany za wystarczająco wysokiej jakości. Ponieważ ściąganie tekstu było najdłużej trwającą operacją w całym procesie, samo wyznaczanie profili nie wymagało optymalizacji — wystarczające okazało się zliczanie wszystkich n-gramów długości 2 i 3, a następnie ich posortowanie oraz porównanie z profilem wzorcowym.

2.7. Implementacja zmian

W wyniku dostosowania Heritriksa do potrzeb projektu przygotowałem następujące klasy:

- `GetHTMLText` — odpowiedzialna za ekstrakcję tekstu z dokumentu HTML,
- `TextWriterProcessor` dziedzicząca po `Processor` — klasa odpowiedzialna za podstawową obsługę ściągniętych danych, tworząca plik wyjściowy i zapisująca do niego tekst,
- `NgramRecorder` — klasa odpowiedzialna za ekstrakcję tekstu, przyznanie mu wagi i decyzję o zapisie.

Dodatkowe opcje w konfiguracji (w zakładce odpowiadającej `TextWriterProcessor`):

- `write-full-content`: czy zapisywać ściągnięty tekst,
- `ignore-multiple-spaces`: czy traktować wielokrotne spacje jako jedną — polecane do włączenia, ponieważ w ściągniętych tekstach wielokrotne spacje występują stosunkowo często i mogą zaburzać statystyki,
- `write-n-gram`: czy zapisywać statystyki n-gramów,
- `test-only`: czy uruchomienie testowe (służące do tworzenia profilu testowego) — profil jest jedynie tworzony (bez obliczania odległości), a wynik jest zapisywany w pliku ze statystykami,
- `overall-stats-location`: lokalizacja pliku ze statystykami,
- `number-of-shingles`: liczba n-gramów najwyższego rzędu branych pod uwagę przy ocenianiu przydatności tekstu — ostatecznie przyjąłem 100 jako wartość domyślną,
- `language-treshold`: poziom odcięcia dla przydatności tekstu,
- `path`: względna ścieżka do katalogu z plikami wynikowymi.

Opcje te służą do właściwego ustawienia w trakcie przebiegu testowego (służącego do stworzenia profilu wzorcowego języka) oraz przebiegu głównego.

2.8. Tworzenie statystyk

Wynikiem działania zmodyfikowanego robota jest katalog plików zawierających teksty ze stron internetowych. Na tak przygotowanym materiale można już obliczać statystyki. W tym celu zostało przygotowane kilka skryptów w BASH-u, C i C++, które obliczały i zapisywały do plików liczby wystąpień poszczególnych liter, par liter oraz trójek liter w zebranych tekstach. W przypadku trójek liter informacje te okazały się zbyt obszerne, zostały więc zapamiętane jedynie statystyki dla 49 (z 256) najczęściej występujących znaków, co pozwoliło ograniczyć rozmiar aplikacji do założonych wcześniej 500 KB. Później te pliki zostały odpowiednio dołączone do aplikacji jako zasoby.

2.9. Rezultaty

2.9.1. Tworzenie korpusu

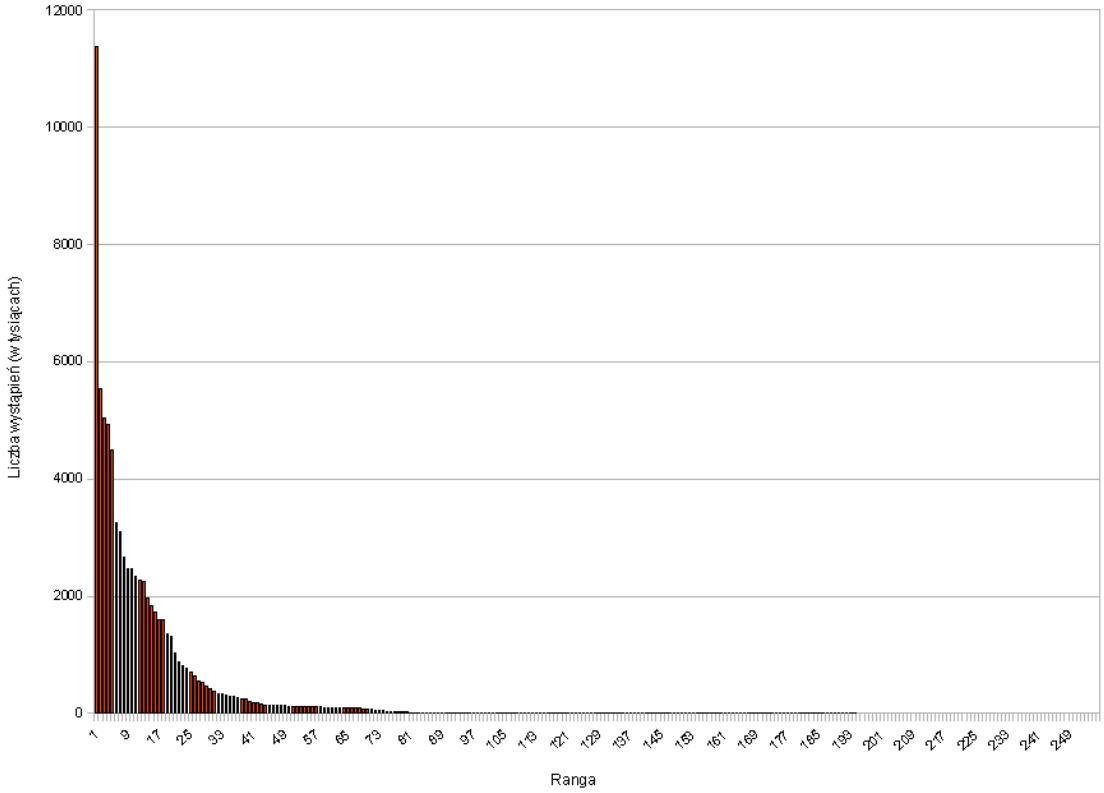
Na łączu w akademiku Uniwersytetu robot osiągał stosunkowo wysoką prędkość ok. 1,5 MB/s. Po selekcji danych dokonanej przez mój moduł, około 1/10 danych okazywało się przydatne. Po kilku testowych uruchomieniach robota zostało zebrane 86 MB tekstów w 6352 plikach. Na tych danych zostały policzone statystyki.

2.9.2. Statystyki

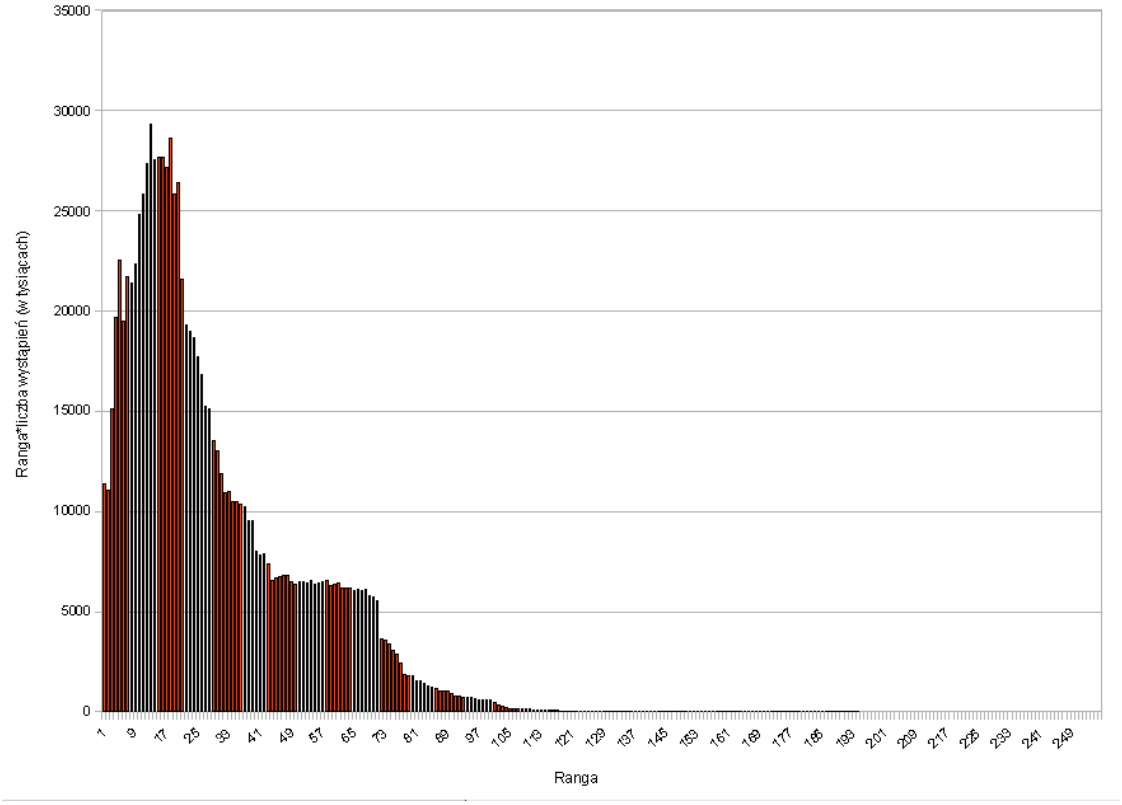
Najlepszą miarą jakości zebranych danych przy naszych zastosowaniach jest współczynnik kompresji osiągnięty przy ich użyciu (który zostanie podany w dalszej części pracy), ale już na tym etapie można wykonać pewne analizy ich jakości.

Przykładowo wykres 2.3, przedstawia rozkład występowania pojedynczych znaków, uszeregowany względem częstości. Na osi poziomej podano rangę znaku, w kolejności od najczęściej do najrzadziej występujących w korpusie, na osi pionowej natomiast — liczbę wystąpień. Wykres ten pokazuje jak duże są różnice w częstości występowania znaków — z wyróżnioną spacją, jako znakiem najczęściej występującym, następnie znakami alfanumerycznymi i przestankowymi oraz pozostałymi znakami występującymi w śladowych ilościach.

Możemy również sprawdzić w jakim stopniu dane te odpowiadają prawu Zipfa (twierdzenie 2.3.1). Ilustruje to wykres 2.4, w którym na osi poziomej ponownie oznaczono rangę znaku, a na osi poziomej iloczyn tej rangi oraz liczby wystąpień. Prawo Zipfa orzeka, że wartości tej funkcji powinny być stałe. Rzeczywiście, w przedziale znaków alfanumerycznych przynajmniej rząd wielkości jest zachowany.



Rysunek 2.3: Częstość wystąpień liter



Rysunek 2.4: Prawo Zipfa na statystykach liter

Rozdział 3

Aplikacja kompresująca

Po wyborze metody kompresji oraz zebraniu bazy tekstów pozostaje nam jeszcze część implementacyjna. W rozdziale tym opiszę aplikację `WiseSMS` będącą edytorem SMS-ów z funkcją kompresji; uzasadnię podjęte decyzje projektowe, użyte technologie, interfejs i budowę oraz przedstawię wyniki testów.

3.1. Metody przesyłania tekstu za pomocą urządzeń mobilnych

Najbardziej znaną metodą przesyłania tekstów za pomocą telefonów komórkowych jest bezpośrednie wykorzystanie protokołu SMS. Możliwe są jednak inne rozwiązania. Wykorzystując protokół GPRS można m.in. połączyć się z darmowymi bramkami SMS i w ten sposób taniej wysyłać SMS-y. Można również wykorzystując protokół MMS wysyłać tekst zakodowany w wysyłanym obrazku.

Zaletą tych rozwiązań jest możliwość przesyłania stosunkowo tanio długich tekstów. W typowej sytuacji jednak przesyłana przez użytkownika pojedyncza wiadomość jest krótka ze względu m.in. na stosunkowo mało wygodny sposób wpisywania tekstu w telefonie komórkowym. Dodatkowo rozwiązania te wymagają od użytkownika pewnej dodatkowej wiedzy i mogą być dla niego kłopotliwe.

Wiadomość może jednak być na tyle długa, że do jej wysłania potrzeba kilku zwykłych SMS-ów. Wystarczy więc skompresować taką wiadomość do jednego SMS-a i w ten sposób znacząco obniżyć koszty komunikacji. Użycie protokołu SMS jest więc wystarczające.

3.2. Platforma programistyczna

W środowiskach mobilnych wybór platformy programistycznej i dostępność API wpływają w bardzo znaczący sposób na możliwości realizacji projektu. Rozpoczniemy więc od krótkiego przeglądu możliwych rozwiązań.

3.2.1. API w Javie

Najłatwiej dostępną możliwością programowania aplikacji dla telefonów komórkowych jest J2ME, czyli Java dla urządzeń mobilnych. Zawiera ona rozszerzenia przydatne w realizacji tego projektu.

Wireless Messaging API

Wireless Messaging API (specyfikacja — [WMA]) jest opcjonalnym pakietem dla J2ME, który umożliwia ustandaryzowany dostęp do zasobów komunikacji bezprzewodowej telefonu komórkowego z poziomu MIDletu Javy. Korzystając z WMA instancje aplikacji działające na różnych telefonach mogą komunikować się między sobą za pomocą specjalnych SMS-ów. Są one podobne do zwykłych SMS-ów, z tą różnicą, że przeznaczenie wiadomości do konkretnej aplikacji identyfikowane jest za pomocą numeru portu (wybranego przez twórcę aplikacji z pewnej dostępnej puli), który jest dołączany do adresu. Zwykłe SMS-y takiego numeru portu nie posiadają.

Wady

WMA posiada jednak zabezpieczenia użytkowników przed nieuczciwymi programistami aplikacji, które z punktu widzenia realizacji tego projektu mogą zostać uznane za wady:

- brak możliwości odbierania wiadomości wysłanych bez numeru portu, przeznaczonych do standardowej skrzynki odbiorczej — nie można więc ich przechwytywać i dekompresować jeśli są skompresowane,
- brak dostępu do skrzynki odbiorczej — nie można również dokonać dekompresji już po umieszczeniu skompresowanej wiadomości w skrzynce odbiorczej,
- brak reakcji telefonu w przypadku wysłania wiadomości specjalnej do telefonu nie nasłuchującego na danym porcie (pojawienie się potencjalnie nieczytelnej wiadomości w standardowej skrzynce odbiorczej byłoby bardziej pożądane z punktu widzenia użytkownika),
- prośba o potwierdzenie przy każdej próbie wysłania SMS-a - decydują tutaj słuszne względy bezpieczeństwa,

Konsekwencje

W konsekwencji przedstawionych powyżej możliwości i ograniczeń pakietu WMA jedynym rozwiązaniem jest stworzenie i oprogramowanie niezależnego (z wyjątkiem listy kontaktów) od standardowego kanału komunikacji tekstowej wykorzystującego SMS-y specjalne, a więc w szczególności własnego edytora wiadomości i stworzenie zupełnie niezależnego mechanizmu ich przechowywania w stosunku do oryginalnych rozwiązań zastosowanych w telefonie.

Push Registry

Push Registry (opis w artykule [PushRegistry]) jest mechanizmem, który pozwala MIDletom na asynchroniczne, automatyczne uruchamianie przez oprogramowanie telefonu w przypadku wystąpienia różnego typu zdarzeń. W naszym przypadku takim zdarzeniem jest odebranie przez telefon wiadomości tekstowej o określonym numerze portu. W takim przypadku następuje uruchomienie aplikacji i przekazanie jej tego zdarzenia.

RMS

MIDP udostępnia mechanizm Record Management System (RMS — opis w artykule [RMS]), czyli mechanizm stałego składowania informacji, na który można patrzeć jako na prostą bazę danych. W projekcie może on być wykorzystany do składowania informacji o wiadomościach i opcjach aplikacji.

PIM

PIM API (opis w artykule [PIM]) jest opcjonalnym pakietem J2ME umożliwiającym MIDletom dostęp do baz danych składowanych w telefonie takich jak lista kontaktów, lista zdarzeń oraz lista To-Do. W przypadku edytora SMS-ów będzie potrzebny dostęp do listy kontaktów.

3.2.2. Aplikacje pod Symbiana lub Windows Mobile

W przypadku urządzeń z systemem operacyjnym Symbian lub Windows Mobile nie występuje przynajmniej część ograniczeń J2ME. Przykładowo programując w C++ pod Symbianem można monitorować foldery zawierające odebrane przez telefon standardowe SMS-y. Poważną wadą tego typu rozwiązań w stosunku do J2ME jest jednak mała popularność telefonów udostępniających taką funkcjonalność oraz ich mała przenośność. Zgodnie z badaniami ABI Research globalny udział Smartphone'ów (czyli wielofunkcyjnych urządzeń mobilnych z funkcjami telefonu komórkowego) w rynku telefonów komórkowych wynosi około 10%, a dalej następuje jeszcze podział na poszczególne systemy operacyjne. Korzystając natomiast z J2ME mamy w przypadku stosunkowo nowych telefonów pewność, że aplikacja będzie działać. Specyfikacja rozszerzenia WMA używanego przez nas w wersji 1.0 powstała już w roku 2002, a średni czas życia telefonu komórkowego wynosi poniżej dwóch lat. W związku z tym zdecydowałem się na wykorzystanie Javy podczas realizacji projektu.

3.3. Interfejs użytkownika

Interfejs użytkownika przypomina typowy interfejs edytora SMS-ów. Składa się z kilku ekranów, których krótki opis przedstawiam w dalszej części tego podrozdziału. Diagram 3.1 przedstawia przejścia pomiędzy ekranami, które może wykonać użytkownik.

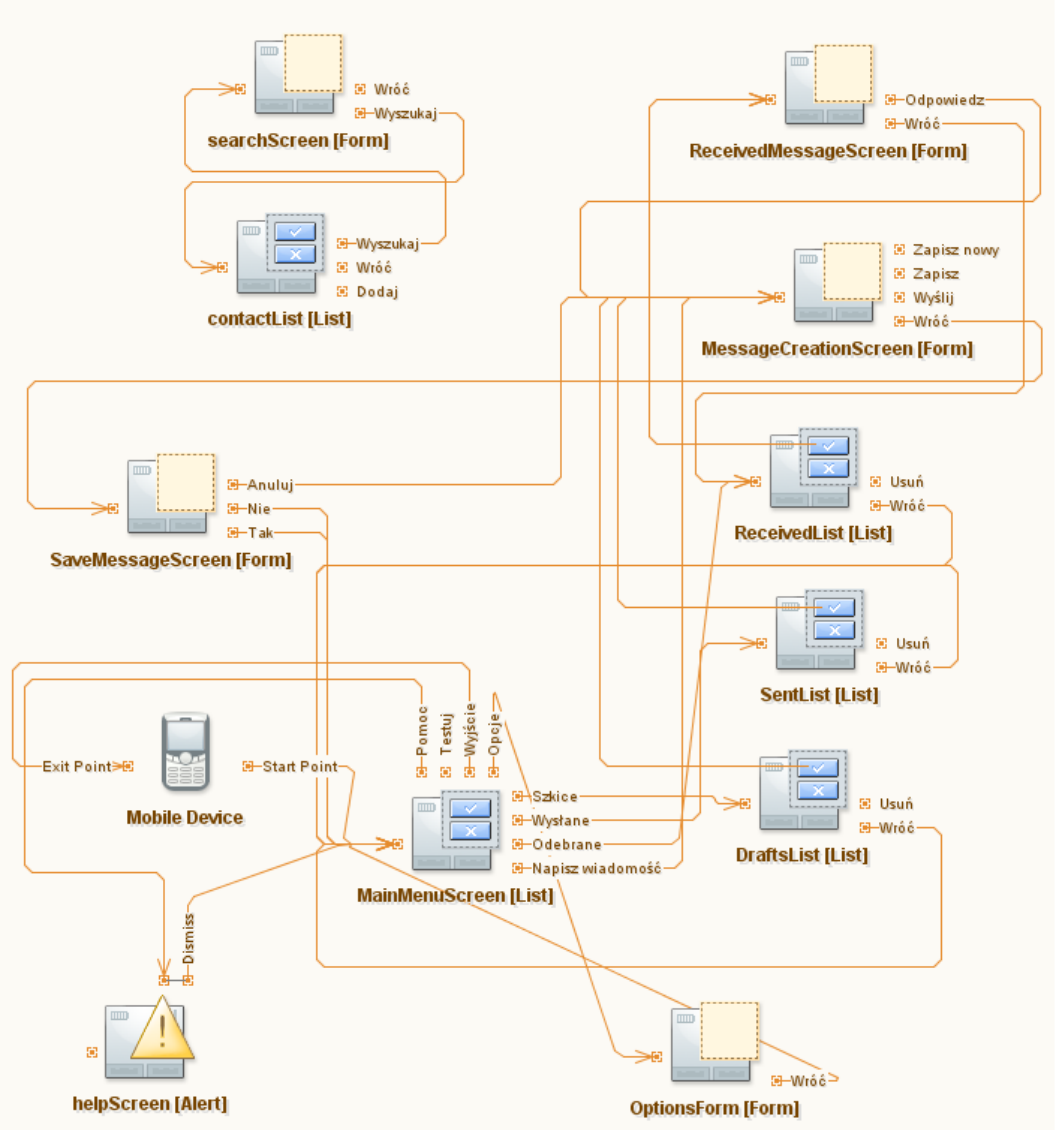
MainMenuScreen

Ekran startowy aplikacji (rysunek 3.2). Umożliwia wybór kolejnej akcji, którą chce wykonać użytkownik:

- napisanie nowej wiadomości,
- wyświetlenie listy wiadomości odebranych,
- wyświetlenie listy szkiców,
- wyświetlenie listy wiadomości wysłanych,
- edycję opcji,
- zamknięcie aplikacji,
- uruchomienie testu skuteczności kompresji (tylko w wersji deweloperskiej),
- wyświetlenie pomocy.

ReceivedList

Lista wiadomości odebranych posortowana względem kolejności ich zapisania w pamięci telefonu. Po wybraniu wiadomości można zobaczyć jej treść w ekranie `ReceivedMessageScreen`.



Rysunek 3.1: Diagram przejść między ekranami

SentList

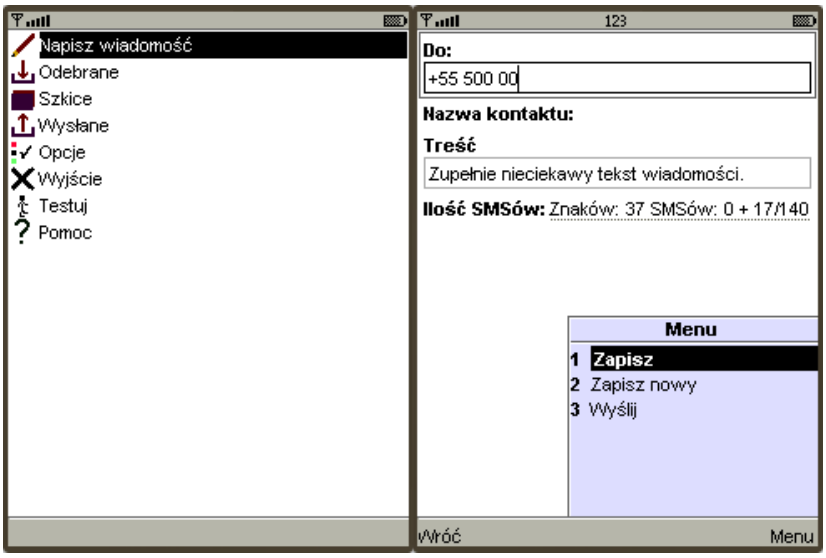
Lista wiadomości wysłanych (patrz rysunek 3.3), posortowana względem kolejności wysłania. Możliwość edycji w ekranie `MessageCreationScreen`.

DraftsList

Lista szkiców wiadomości, zapisywane są tutaj wszystkie nie wysłane wiadomości. Ułożenie wiadomości jest podobne jak dla pozostałych list wiadomości.

OptionsForm

Ekran opcji (rysunek 3.4). Dostępne opcje:



Rysunek 3.2: MainMenuScreen, MessageCreationScreen

- możliwość wybrania, czy obliczenie długości skompresowanej wiadomości jest dokonywane na bieżąco, czy na żądanie użytkownika.

W przyszłości planowane jest dodanie kolejnych opcji.

MessageCreationScreen

Ekran edycji wiadomości do wysłania (rysunek 3.2). Jego kolejne elementy to:

- pole tekstowe do wpisania numeru telefonu odbiorcy, w czasie edycji następuje wyszukiwanie numeru w książce adresowej, w przypadku sukcesu wyszukiwania jest on wyświetlany,
- nazwa kontaktu skojarzonego z numerem telefonu w polu Do: o ile taki zostanie odnaleziony,
- treść wiadomości,
- liczba SMS-ów, które zostaną wykorzystane po kompresji do wysłania wiadomości.

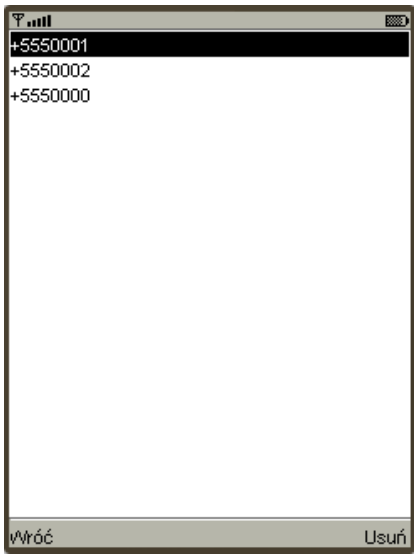
W przypadku próby wyjścia z edycji wiadomości jeśli nie została ona zapisana, następuje przejście do ekranu `SaveMessageScreen`.

ReceivedMessageScreen

Ekran edycji wiadomości otrzymanej, istotną różnicą w porównaniu do ekranu `MessageCreationScreen` jest dostępna komenda Odpowiedz.

SaveMessageScreen

Ekran zapytania o zapisanie edytowanej wiadomości do folderu Szkice.



Rysunek 3.3: SentList

contactList

Ekran opcjonalny, umożliwia wybór pozycji z książki adresowej (niepotrzebny w większości telefonów, które robią to automatycznie przy odpowiednich ustawieniach pola tekstowego).

searchScreen

Ekran opcjonalny umożliwia wyszukiwanie po prefiksie nazwy kontaktu na liście kontaktów z książki adresowej.

helpScreen

Ekran pomocy programu (rysunek 3.4).

3.4. Budowa

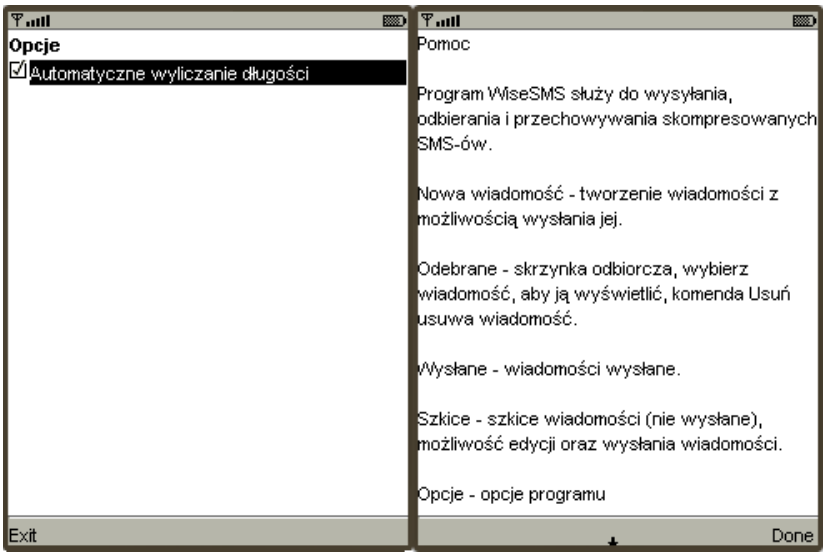
Rysunek 3.5 przedstawia podstawowe klasy występujące w projekcie wraz z podstawowymi zależnościami między nimi. Zawartość pakietów `storage` oraz `compression` zostanie przedstawiona w dalszej części rozdziału.

3.4.1. Komunikacja

W pakiecie `communication` zebrano klasy odpowiedzialne bezpośrednio za komunikację. Ze względu na fakt, że interfejs `MessageListener` (a więc odbiór wiadomości) implementuje klasa `WiseSMS` w pakiecie `frontend`, jest to tylko klasa `Sender`.

Sender

Klasa odpowiedzialna za obsługę kompresji (wywołuje metody z pakietu `compression`) i wysyłanie SMS-ów, które realizowane są w osobnym wątku.



Rysunek 3.4: OptionsForm, HelpScreen

3.4.2. Kompresja

W pakiecie `compression` zebrano wszystkie klasy i interfejsy odpowiedzialne za kompresję. Zależności między klasami przedstawia rysunek 3.6.

CompressorInterface

Interfejs dla metod kompresji — udostępnia metody, które każda klasa odpowiedzialna za kompresję powinna implementować, interfejs ten używany jest na zewnątrz pakietu.

CompressorTest

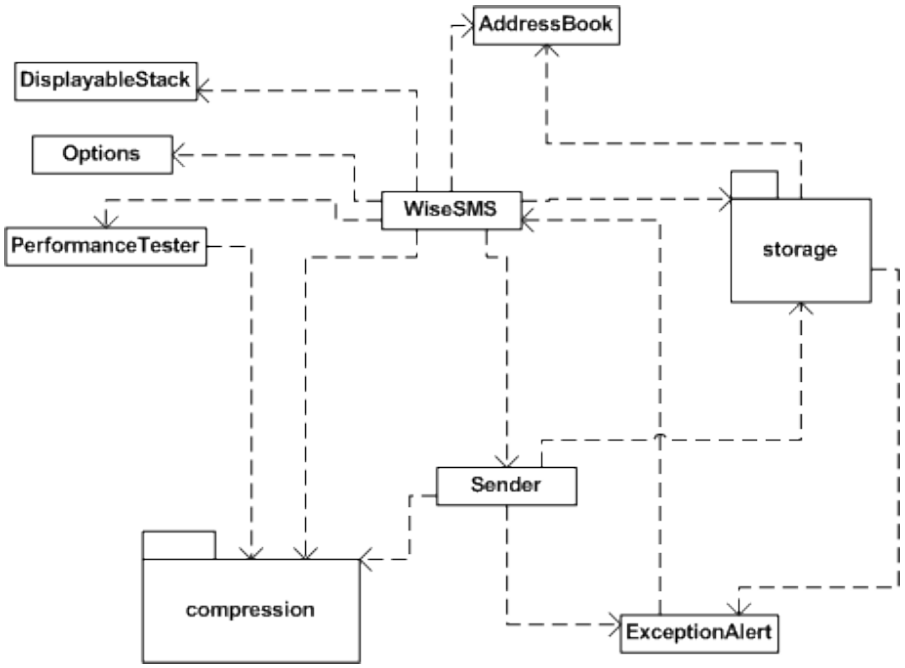
Testowa implementacja `CompressorInterface`.

ArithmeticCoder

Klasa implementująca interfejs `CompressorInterface`, odpowiedzialna za kompresję i dekompresję metodą kodowania arytmetycznego. Inicjowanie odbywa się w osobnym wątku, gdyż zawiera wczytywanie statystyk dla modelu prawdopodobieństwa i jest objęta sekcją krytyczną, aby przed wczytaniem tych statystyk nie została podjęta próba kompresji (ponieważ statystyki są konieczne do jej działania).

Probs

Interfejs używany przez `ArithmeticCoder` reprezentujący dowolny model rozkładu prawdopodobieństw wiadomości. Zawiera funkcje, które dla danego kontekstu pokazują liczbę wystąpień znaku w tym kontekście oraz licznik wystąpień wszystkich znaków w tym kontekście, zgodnie z metodami z rozdziału 1.5.2.



Rysunek 3.5: Schemat zależności klas

ProbsStandard

Abstrakcyjna klasa reprezentująca standardowe modele prawdopodobieństw — ze stałym kontekstem.

ProbsLevel2

Implementacja modelu z jednoznakowym kontekstem. Jeśli nie podano kontekstu, to nie jest on uwzględniany (używany jest rozkład wg modelu 0-go rzędu).

ProbsLevel3

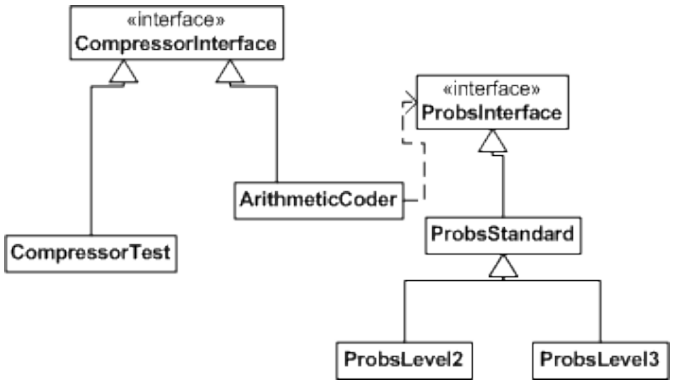
Implementacja modelu z dwuznakowym kontekstem (dla 49 najczęściej występujących znaków) oraz jednoznakowym dla pozostałych znaków. Jeśli nie podano kontekstu, to używany jest model 0-go rzędu.

3.4.3. Interfejs użytkownika

Pakiet `frontend` zawiera klasy odpowiedzialne za komunikację z użytkownikiem oraz wyświetlanie informacji.

DisplayableStack

Klasa reprezentująca stos obiektów `Dislayable`, czyli możliwych do wyświetlenia obiektów zajmujących cały wyświetlacz telefonu. Służy ona do obsługi komendy powrotu do poprzedniego wyświetlonego ekranu.



Rysunek 3.6: Klasy pakietu *compression*

WiseSMS

Najbardziej zewnętrzna klasa implementująca pozostałą część interfejsu użytkownika oraz cykl życia aplikacji (włącznie z odbieraniem wiadomości).

3.4.4. Przechowywanie danych

Pakiet `storage` zbiera klasy odpowiedzialne za przechowywanie wiadomości SMS.

SMS

Właściwie typ danych reprezentujący pojedynczą wiadomość. Odpowiedzialna za tworzenie rekordów, odtwarzanie wiadomości z rekordów bazy danych oraz zapis i odczyt.

SMSBook

Reprezentuje zbiór SMS-ów, umożliwia podstawowe operacje na nim, takie jak dodanie i pobieranie oraz wylistowanie.

SMSType

Reprezentuje typ wiadomości — wysłana, otrzymana lub szkic.

3.4.5. Testowanie

Pakiet `testing` zawiera klasy odpowiedzialne za testy.

PerformanceTester

Przygotowanie i przeprowadzenie testu wydajności kompresji na dwóch przykładowych zbiorach kilkuset wiadomości — mojej własnej skrzynce nadawczej i odbiorczej oraz gotowych SMS-ach ze stron internetowych.

3.4.6. Dodatki

Pakiet `utilities` zawiera klasy pomocnicze.

Rząd metody	Przed kompresją	Po kompresji	%	bitów/znak	Długość SMS-a
0	24382	15397	63,15%	5,05	221,7
1	24382	12379	50,77%	4,06	275,75
2*	24382	12116	49,69%	3,98	281,73

Tabela 3.1: Wyniki testów — zbiór prywatnych SMS-ów, korpus stron internetowych

AddressBook

Książka adresowa — odczyt, sortowanie rekordów z książki, listowanie.

ExceptionAlert

Umożliwia wyświetlenie Alertu w przypadku wystąpienia wyjątku wraz z wszystkimi o nim informacjami.

Options

Obsługa opcji aplikacji — zapisywanie i odczytywanie poprzedniego stanu z pamięci, obsługa wartości domyślnych, odczyt, zapis i zmiana opcji.

SimpleContact

Typ danych reprezentujący pojedynczy wpis w książce adresowej.

3.5. Testy

W celu przeprowadzenia testów przygotowane zostały dwa zbiory SMS-ów:

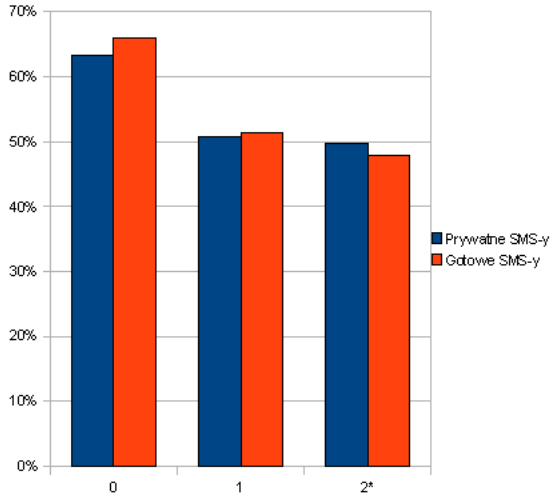
- zbiór SMS-ów z mojej prywatnej skrzynki odbiorczej i nadawczej — 422 SMS-y o średniej długości 57,7 znaków,
- zbiór gotowych SMS-ów z Internetu (życzenia świąteczne, cytaty) — 413 SMS-ów o średniej długości 99,5 znaków.

3.5.1. Korpus stron internetowych

Tabele 3.1 i 3.2 przedstawiają wyniki testów odpowiednio dla zbioru prywatnych SMS-ów oraz dla zbioru gotowych SMS-ów z Internetu. Kolejne kolumny przedstawiają odpowiednio: rząd modelu prawdopodobieństwa (z zastrzeżeniem, że w metodzie 2-go rzędu stosowano kontekst dwuznakowy tylko dla 49 najczęściej występujących znaków), sumaryczną liczbę znaków (w kodowaniu `iso`, a więc 8-bitowym) w zbiorze testowym, liczbę 8-bitowych znaków zajmowaną przez skompresowany tekst, współczynnik kompresji, liczbę bitów na znak przy stosowanej metodzie oraz średnią maksymalną długość SMS-a jaką można osiągnąć przy takim stopniu kompresji. Stosunek wielkości skompresowanych danych do danych surowych przedstawia również wykres 3.7.

Rząd metody	Przed kompresją	Po kompresji	%	bitów/znak	Długość SMS-a
0	40917	26999	65,98%	5,28	212,17
1	40917	21001	51,33%	4,11	272,77
2*	40917	19560	47,80%	3,82	292,86

Tabela 3.2: Wyniki testów — zbiór gotowych SMS-ów z Internetu, korpus stron internetowych



Rysunek 3.7: Współczynnik kompresji dla różnych rzędów metod i zbiorów testowych, korpus stron internetowych

3.5.2. Korpus słownika frekwencyjnego polszczyzny współczesnej

Wykonano również dla porównania testy, w których za korpus posłużył korpus słownika frekwencyjnego polszczyzny współczesnej z lat sześćdziesiątych. Tabele 3.3 oraz 3.4 i wykres 3.8 przedstawiają uzyskane rezultaty.

Wynika z nich, że osiągnięto o kilka-kilkanaście procent mniejszy stopień kompresji (w zależności od testu). Stąd przy zastosowanych metodach kompresji i testach korpus stron internetowych okazał się być bardziej adekwatnym od korpusu słownika frekwencyjnego.

3.5.3. Dyskusja

Ostatecznie więc maksymalny osiągnięty stopień kompresji to $3.8-3.9$ bitów/znak co pozwala na tworzenie SMS-ów o długości dochodzącej do $280-290$ znaków.

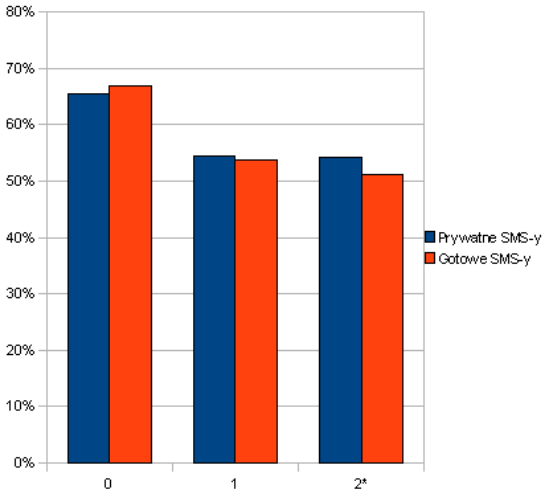
Osiągnięty stopień kompresji wydaje się być zadowalający. W porównaniu do standardowych 70 znaków na SMS-a, które dostępne są przy użyciu polskich znaków diakrytycznych

Rząd metody	Przed kompresją	Po kompresji	%	bitów/znak	Długość SMS-a
0	24382	15949	65,41%	5,23	214,02
1	24382	13292	54,52%	4,36	256,81
2*	24382	13189	54,09%	4,33	258,81

Tabela 3.3: Wyniki testów — zbiór prywatnych SMS-ów, korpus słownika frekwencyjnego polszczyzny współczesnej

Rząd metody	Przed kompresją	Po kompresji	%	bitów/znak	Długość SMS-a
0	40917	27321	66,77%	5,34	209,67
1	40917	22017	53,81%	4,3	260,18
2*	40917	20902	51,08%	4,09	274,06

Tabela 3.4: Wyniki testów — zbiór gotowych SMS-ów z Internetu, korpus słownika frekwencyjnego polszczyzny współczesnej



Rysunek 3.8: Współczynnik kompresji dla różnych rzędów metod i zbiorów testowych, korpus słownika frekwencyjnego polszczyzny współczesnej

osiągnięto ponad 4-krotną poprawę. W pracy [RGF06] autorzy podają, że dla tekstów w języku angielskim (przy bardziej zaawansowanych metodach kompresji) osiągali kompresję rzędu od 4 do nawet 3.2 bitów/znak. Nasz wynik jest więc porównywalny.

Dla porównania warto podać również, że całego zbioru SMS-ów umieszczonych w jednym pliku korzystając z programu `zip` przy jego standardowych ustawieniach otrzymujemy kompresję rzędu 3.3 bitów/znak, a więc również porównywalną do uzyskanej. Pokazuje to jednak, że możliwe są jeszcze dalsze ulepszenia.

Rozdział 4

Podsumowanie

Zaprezentowana w niniejszej pracy aplikacja jest kompletnym rozwiązaniem, które może być ciekawą propozycją dla użytkowników telefonów komórkowych. Spełnia ona wszystkie założone wcześniej wymagania. Został osiągnięty zadowalający stopień kompresji, przy wykorzystaniu mocno ograniczonych zasobów mocy obliczeniowej oraz pamięci. Interfejs aplikacji podobny jest do standardowych interfejsów edytorów SMS-ów w typowych telefonach komórkowych, nie sprawi więc użytkownikom zbytnich trudności w obsłudze.

Możliwych jest wiele dalszych kierunków rozwoju projektu, właściwie na wszystkich jego istotnych etapach. Niektóre pomysły znalazły już zastosowanie w aplikacji SMSZipper (więcej informacji na stronie projektu — [SMSZipper]). Między innymi są to szyfrowanie oraz wprowadzenie tytułów do SMS-ów (co ułatwia wybór interesującej nas wiadomości z listy).

Nasza aplikacja umożliwia bardzo łatwą zamianę korpusu używanego do generowania modelu statystycznego. Używając korpusu internetowego osiągnięto zadowalające rezultaty. Natomiast wykorzystanie korpusu SMS-ów powinno umożliwić uzyskanie jeszcze lepszego współczynnika kompresji. Ponieważ powstała ta aplikacja, łatwiej będzie zachęcić użytkowników lub operatorów sieci komórkowych do udostępnienia takich danych.

Możliwe jest również badanie innych metod kompresji tekstów. W szczególności zastosowanie metod adaptywnych, z wykorzystaniem korpusu jako prefiksu każdej kompresowanej wiadomości (podobna idea przedstawiona jest w pracy [RGF06]).

Podstawowym wyzwaniem podczas realizacji projektu była mała liczba gotowych rozwiązań możliwych do wykorzystania. Konieczne więc było stworzenie własnego korpusu językowego z danych o mocno zróżnicowanej jakości. Ze względu na brak bibliotek do kompresji krótkich tekstów (w szczególności darmowych i napisanych w Javie) potrzebna była samodzielna implementacja algorytmu kompresji w środowisku o mocno ograniczonych zasobach. Dodatkowo konieczne było użycie języka programowania wysokiego poziomu, nieprzystosowanego do takich zadań (przykładowo bez obsługi liczb zmiennopozycyjnych). Z pozoru więc niewielki projekt wymagał sporego wkładu pracy.

Niniejsza praca umożliwiła mi poznanie wielu fascynujących dziedzin informatyki. Mam nadzieję, że powstała aplikacja spotka się z uznaniem użytkowników i będzie nadal rozwijana.

Dodatek A

Instrukcja instalacji

A.1. Wymagania

Program WiseSMS testowany był na telefonie Nokia 6151. Z przeprowadzonych symulacji i praktyki korzystania wynikają następujące wymagania dotyczące telefonu:

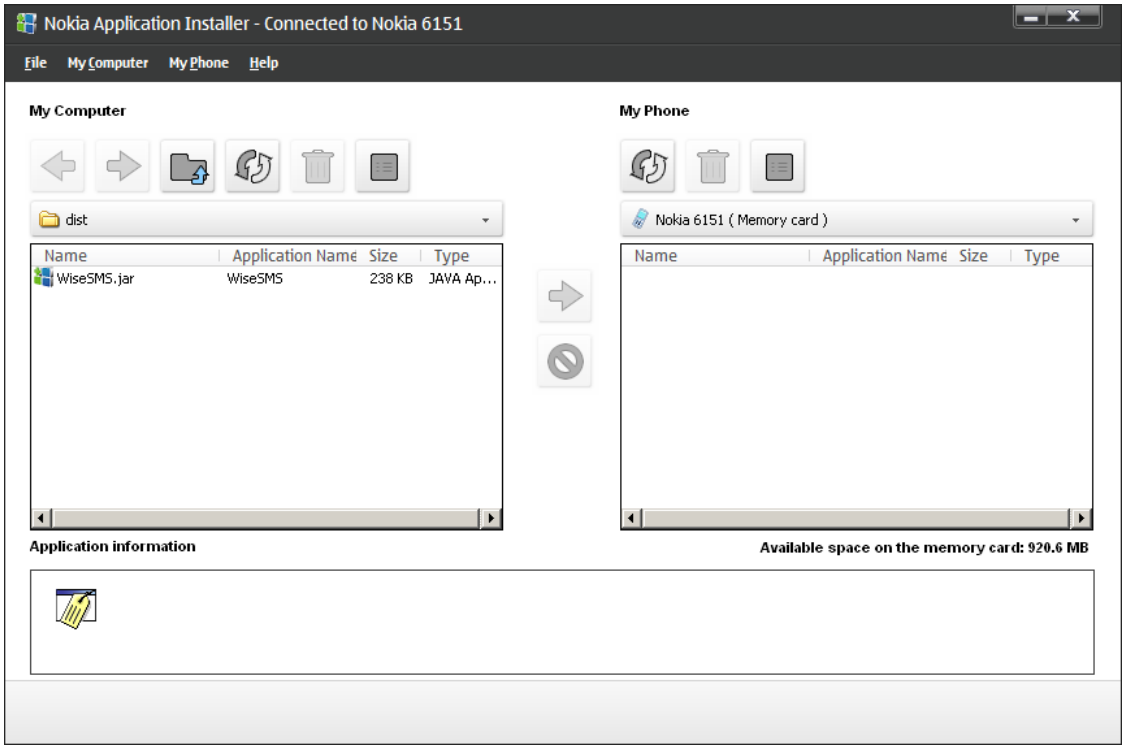
- co najmniej 2MB pamięci,
- wydajny procesor (ma wpływ na szybkość kompresji i czas uruchamiania programu),
- CLDC 1.0,
- MIDP 2.0,
- Wireless Messaging API 1.1,
- obsługa File Connection oraz PIM Optional Packages 1.0.

A.2. Instalacja

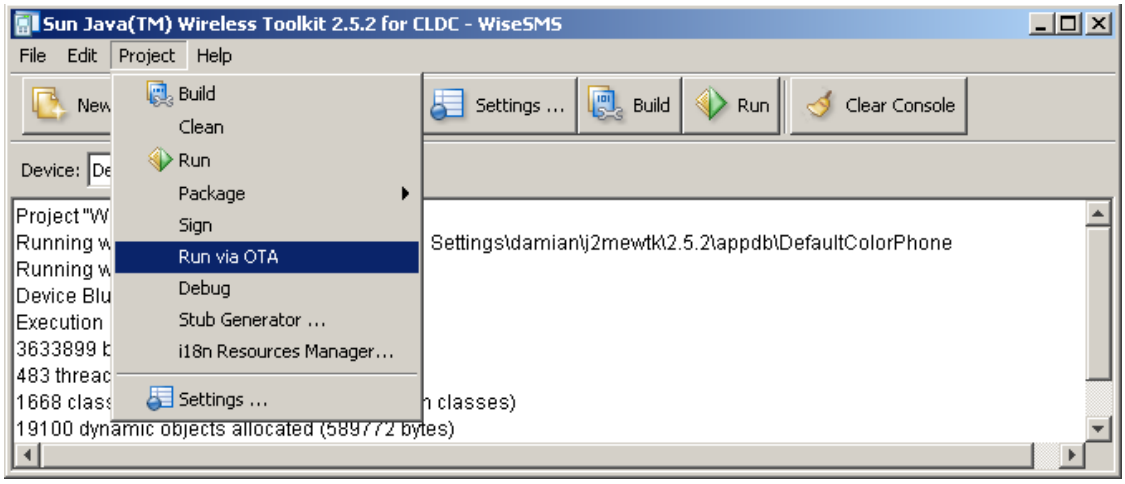
Instalacja programu jest standardową instalacją aplikacji Java na telefonie komórkowym. Przykładowo dla telefonów Nokia możemy skorzystać z aplikacji `Nokia PC Suite`, w której po podłączeniu telefonu z komputerem i włączeniu programu wybieramy `Nokia Application Installer`, a następnie kopiujemy plik `WiseSMS.jar` do pamięci telefonu (rysunek A.1).

A.3. Emulacja

Aplikacja wykorzystuje technologię `Push Registry`, która pozwala na automatyczne uruchamianie aplikacji po otrzymaniu wiadomości SMS przez urządzenie. W standardowym trybie emulacji nie jest możliwe testowanie tej funkcjonalności. Do jej testowania konieczna jest symulacja instalacji aplikacji. Możemy do tego wykorzystać opcję `Run via OTA` w emulatorze. Przykład dla emulatora `Sun Java Wireless Toolkit` przedstawia rysunek A.2.



Rysunek A.1: Instalacja na telefonie Nokia przy pomocy Nokia PC Suite



Rysunek A.2: Run via OTA

Dodatek B

Zawartość płyty CD

Na dołączonej do pracy płycie CD znajdują się:

- kopia pracy magisterskiej wraz ze źródłami,
- aplikacja `WiseSMS` wraz ze źródłami w wersji testowej oraz w wersji finalnej,
- zmodyfikowany robot `Heritrix` (na licencji GPL),
- korpus zebrany przez program,
- skrypty służące do tworzenia statystyk z zebranego korpusu wraz z instrukcją obsługi.

Bibliografia

[ACT04] Bashir Ahmed, Sung-Hyuk Cha, Charles Tappert, *Language Identification from Text Using N-gram Based Cumulative Frequency Addition*, Pace University, 2004.

[BCW90] Timothy C. Bell, John G. Cleary, Ian H. Written *Text Compression*, Prentice Hall, 1990.

[Can94] William B. Cavnar, John M. Trenkle, *N-Gram-Based Text Categorization* Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval, 1994

[Ciw84] John G. Cleary, Ian, H. Witten, *Data compression using adaptive coding and partial string matching*, IEEE Transactions on Communications, 1984

[Droz07] Adam Drozdek, *Wprowadzenie do kompresji danych*, WNT, 2007.

[Hab05] Ján Habdák *N-gram based Text Categorization*, Comenius University, Bratislava, 2005

[Heritirix] `http://crawler.archive.org/`, *Strona projektu Heritrix*

[Kran05] Simon Kranig, *Evaluation of Language Identification Methods*, University of Tübingen, 2005

[PIM] `http://developers.sun.com/mobility/apis/ttips/pim/index.html`, *An Overview of the PIM Optional Package Article*

[PushRegistry] `http://developers.sun.com/mobility/midp/articles/pushreg/`, *The MIDP 2.0 Push Registry Article*

[RGF06] Stephan Rein, Clemens Gühmann, Frank Fitzek, *Compression of Short Text on Embedded Systems*, Journal of Computers, vol.1, wrzesień 2006.

[RMS] `http://developers.sun.com/mobility/midp/articles/databaserms/`, *Databases and MIDP Article*

[Shan48] Claude Shannon, *A Mathematical Theory of Communication*, Bell System Technical Journal, 1948.

[SMSZipper] `http://smszipper.com/`, *Strona projektu SMSZipper*

[WMA] `http://jcp.org/aboutJava/communityprocess/final/jsr120/`, *Wireless Messaging API (WMA) for Java^{TM} 2 Micro Edition Version 1.0*