TIMKoD – Lab 4 – Kompresja bezstratna – Wstęp

Opis pliku z zadaniami

Wszystkie zadania na zajęciach będą przekazywane w postaci plików `.pdf`, sformatowanych podobnie do tego dokumentu. Zadania będą różnego rodzaju. Za każdym razem będą one odpowiednio oznaczone:

- Zadania do wykonania na zajęciach oznaczone są symbolem $\triangle$ – nie są one punktowane, ale należy je wykonać w czasie zajęć.
- Punktowane zadania do wykonania na zajęciach oznaczone są symbolem $\diamond$ – należy je wykonać na zajęciach i zaprezentować prowadzącemu, w wypadku nie wykonania zadania w czasie zajęć lub nieobecności, zadania staje się zadaniem do wykonania w domu.
- Zadania programistyczne można wykonywać w dowolnym języku programowania, używając jedynie biblioteki standardowej dostępnej dla tego języka.

Cel zajęć

Druga część przedmiotu będzie dotyczyć metod bezstratnej kompresji danych. W ramach tych zajęć przygotowany zostanie podstawowy program służący do kompresji i dekompresji danych tekstowych. Na kolejnych zajęciach przygotowane narzędzie będzie rozszerzane o inne metody kodowania, przykładowo przy użyciu kodu Huffmana. Stworzony dzisiaj jednolity *framework* pozwoli łatwo porównać stopień kompresji implementowanych metod.

Przygotowanie do zajęć

- Do wykonania zadań potrzebny będzie korpus tekstowy, który można pobrać z katalogu przy laboratorium w systemie e-kursy.
- Teksty są znormalizowane, zawierają jedynie 26 małych liter alfabetu łacińskiego, cyfry i spacje (czyli w sumie 37 znaków).

1 Kodowanie binarne *10pt*◇

Treść

Stwórz program służący do zapisu i odczytu danych tekstowych używając kodów krótszych niż kod ASCII (8 bitów). Program będzie przeprowadzać kompresję symbol po symbolu (ang. *symbol-by-symbol*) używając kody o stałej długości (ang. *fixed-length coding*). Jaka jest najkrótsza możliwa długość takiego kodu dla korpusu z dzisiejszych zajęć? Ile wyniesie stopień kompresji?

Implementacja powinna zawierać poniższe funkcje:

- **create** – na podstawie listy częstości poszczególnych znaków tworzy kod,
- **encode** – tworzy zakodowaną reprezentację tekstu,
- **decode** – odkodowuje zakodowany tekst,
- **save** – zapisuje kod oraz zakodowany tekst,
- **load** – wczytuje zakodowany tekst oraz kod.

Implementacja powinna umożliwiać odkodowanie pliku z tekstem i kodem bez dostarczenia informacji z wcześniejszego etapu kodowania. W programie powinna być weryfikowana poprawność kodowania i dekodowania poprzez porównanie czy tekst oryginalny i zakodowany-odkodowany są identyczne.

Jaki jest rozmiar oryginalnego pliku, a jaki pliku (lub plików) z kodem i zakodowaną reprezentacją?

Zastanów się nad poniższymi zagadnieniami:

- Co można zrobić, żeby bardziej skompresować tekst?
- Co z nieużytymi kodami?
- Jak odkodowywać kody o zmiennej długości (ang. *variable-length code*)?
- Jaka jest granica wydajności takich kodów (ang. *symbol-by-symbol*)?

Programy stworzone na kolejnych zajęciach będą różnić się implementacją **create**, **encode** oraz **decode**. Stworzenie elastycznej implementacji podstawowego programu skróci czas potrzebny na zrobienie zadań na kolejnych zajęciach.

Użytkownicy Pythona mogą skorzystać z pakietu `bitarray` (jest on dostępny domyślnie w dystrybucji Anacoda, można go zainstalować za pomocą PyPi, dla Windowsa można popbrać również instalator ze strony: https://www.lfd.uci.edu/~gohlke/pythonlibs/#bitarray). Należy zwrócić uwagę na wyrównwanie do 8 bitów dodawane przez funkcje pakietu. Najprościej jest

operować na ciągach znaków '0' i '1' i z nich tworzyć reprezentację `bitarray`. Nie należy używać fukcji encode i decode z tego pakietu.

Użytkownicy Javy mogą użyć klasy BitSet. W C++ dostępny jest std::bitset, a w C#, BitVector i BitArray.