Specyfikacja implementacyjna „gentex”

Maciej Skarbek
Nr albumu: 271088

23 marca 2015

Wprowadzenie

Celem projektu jest stworzenie aplikacji w języku C, która będzie generować teksty wyjściowe na podstawie analizy innych tekstów wykorzystując przy tym łańcuchy Markova. Przy tworzeniu programu korzystać będziemy z systemu kontroli wersji git.

Opis modułów

managment

Rozpoznaje argumenty wywołań i steruje innymi modułami. Zawiera funkcje "main".

store

Przechowuje macierz przejść z podziałem na prefiksy i sufiksy. Implementacja drzewa.

generation

Generuje tekst wynikowy i zapisuje go do pliku.

reading

Odczytuje teksty podane przez użytkownika. Analizuje je dzieląc na prefiksy i sufiksy a następnie przekazuje do modułu "store".

backup

Tworzy i umożliwia wczytanie plików pośrednich z których w przyszłości można generować teksty wynikowe.

stats

Tworzy statystyki łączone dla tekstów bazowych i oddzielną statystykę dla tekstu wynikowego (prawdopodobieństwo wystąpienia pojedyńczego słowa, n-gramu z jakich został wygenerowany tekst i wskaźnik PMI).

error

Obsługuje błędy zaistniałe podczas działania.

Opis najważniejszych funkcji i struktur

managment

- *void add(char **prefix, char * suffix)*
  prefix - tablica wyrazów z których składa sie prefiks
  suffix - sufiks
  Rozpoznanie argumentów wywołania i przekazanie sterowania do odpowiednich modułów.

store

- Struktury

```
typedef struct{
  char **prefix;
  char **sufix;
  int size_s;
  int n_s;
}ngram;
```

```
typedef structnode{
  ngram *g;
  structnode *left, *right;
}node_t, *tree_t;
```

```
typedef struct{
  tree_t t;
  int number_gram;
  int size;
  ngram **tab;
  int n_s_max;
}store;
```

  Tworzone jest drzewo do przechowywania prefiksów i sufiksów jak również tablica w której będą wskaźniki do komurek drzewa. Tablica będzie używana przy losowaniu prefiksów jak również przy tworzeniu pliku pośredniego (gdybyśmy chcieli stworzyć plik pośredni z drzewa był by on posortowany i przy wczytywaniu zamiast drzewo powstała by nam lista co znacząco wydłużyło by czas pracy programu).

- *tree_t insert(tree_t t, char **prefix, char * suffix)*
  return - funkcja zwraca drzewo z dodanym elementem
  t - drzewo do którego ma zostac dodany element
  prefix - tablica wyrazów z których składa sie prefiks
  suffix - sufiks
  Wstawia do drzewa prefiksy i sufiksy, jeśli prefiks juz istnieje to dopisuje tylko sufiks. Dodaje również wzkaźnik na "ngram" do "tab".

- *void add_from_backup(char **prefix, char **suffix, int n_s)*
  prefix - tablica wyrazów z których składa sie prefiks
  suffix - tablica wszystkich sufiksów dla danego prefiksu
  Dodaje prefiks i całą listę sufiksów do drzewa.

- *ngram* rand_prefix()*
  return - zwraca strukturę pokazaną wyżej
  Losuje prefiks.

- *char* rand_suffix(char** prefix)*
  return - zwraca sufiks

prefix - tablica wyrazów z których składa sie prefiks
Losuje sufiks dla podanego prefiksu.

generation

- *void generation()*
  Generuje tekst wynikowy i zapisuje go do pliku.

reading

- *void reading(char * name_file)*
  name_file - plik który ma zostać zanalizowany i wczytany do "store"
  Czyta podany plik, dzieli na prefiksy i sufiksy a następnie przekazuje do "store" i "stat".

backup

- *void backup()*
  Tworzy plik wczytuje i tworzy plik pośredni.

stats

- *void stat_add_word(char * word)*
  word - wyraz przeczytany z pliku poddanego analizie
  Dodaje pojedynczy wyraz do drzewa ze statystykami (zlicza ilośc wystąpień).

- *void stat_add_ngram(char** prefix,char * suffix)*
  prefix - tablica wyrazów z których składa sie prefiks
  suffix - sufiks
  Dodaje n-gram do drzewa ze statystykami i zlicza ilość wystąpień.

- *double get_probability(tree_stat t, char * word)*
  return - zwraca prawdopodobieństwo
  t - nazwa drzewa w którym mamy szukać wyrazu(może to być kilka wyrazów) i informacji o ilości wystąpień
  word - wyraz dla którego liczone jest prawdopodobieństwo
  Liczy prawdopodobieństwo dla n-gramu.

- *long double get_pmi(char* wngram)*
  return - zwraca wartość wskaźnika PMI
  wngram - napis dal którego ma być liczony wskaźnik
  Liczy wskaźnik PMI dla n-gramu.

- *void write_stat(char * name_file_stat)*
  name_file_stat - nazwa pliku do którego mają być zapisywane statystyki Zapisuje statystyki do podanego pliku.

error

- *void fatal(int err, const char *msg)*
  err - instrukcja logiczna która warunkuje wystąpienie błędu np. uchwyt_do_pliku == NULL
  msg - Wiadomość która ma zostać wyświetlona w przypadku spełnienia się warunku Dostaje wyrażenie logiczne "err" i w przypadku potwierdzenia wystąpienia błędu pokazuje wiadomość "msg" i np. zamyka program.

Testowanie

Użyte narzędzia

time

Polecenie time do pomiaru czasu działania programu aby wybrać optymalny algorytm jaki zaimplementujemy.

valgrind

Narzędzie do debugowania pamięci i wykrywania wycieków pamięci.

Sposób testów

Testy będą wykonywane dla pojedynczych funkcjonalności programu, dla całych modułów a następnie po przyłączeniu kolejnego modułu zostaną wykonane kompleksowe testy całego programu. Szczególną uwagę należy zwrócić na punkty krytyczne w których możemy spodziewać sie błędów.

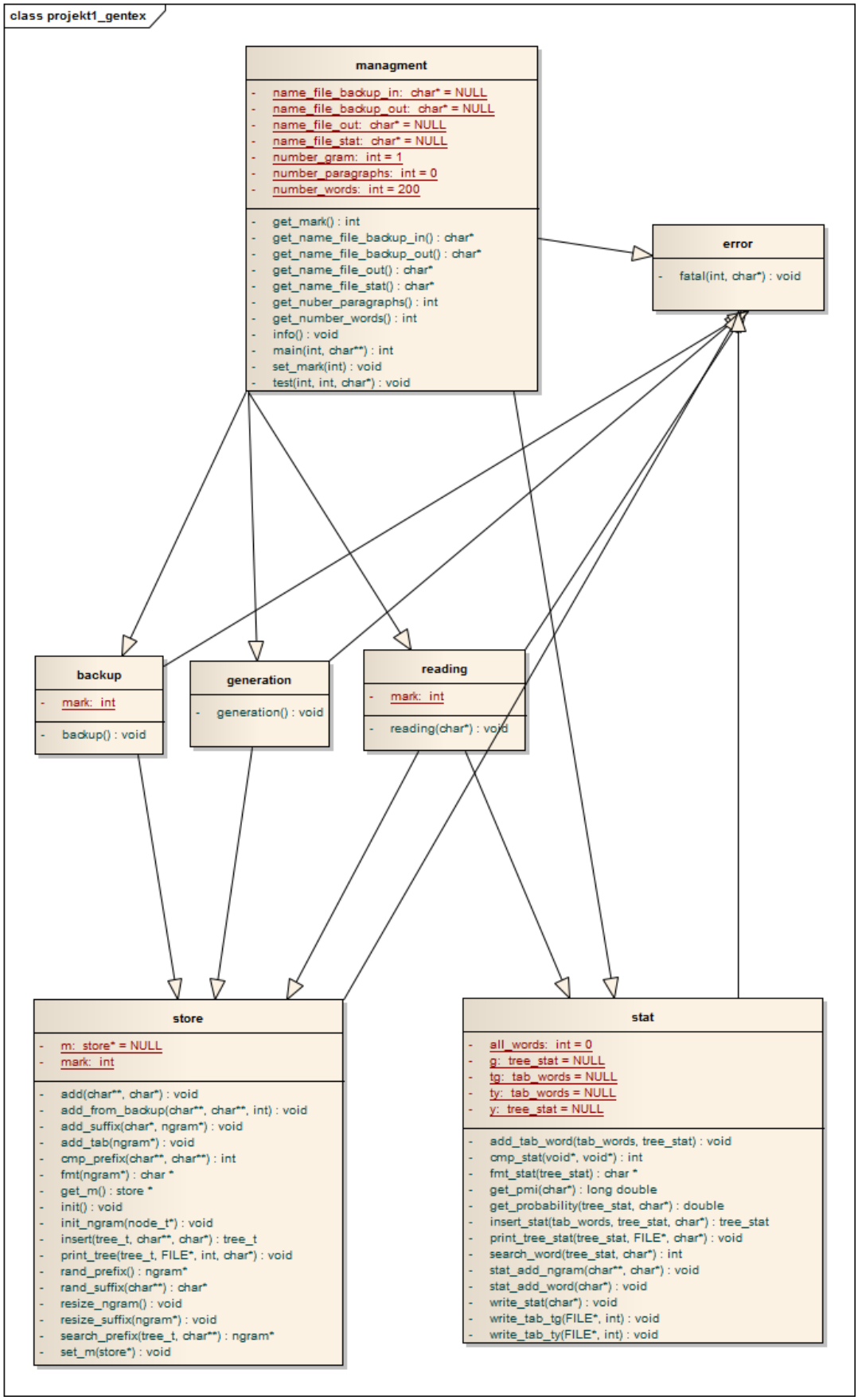
Punkty krytyczne

- Bardzo duże pliki wejściowe
- Puste pliki wejściowe
- Bardzo małe pliki wejściowe (np 2 słowa)
- Błędnę wywołanie programu

- Podanie pliku do zapisu który już istnieje
- Mała ilość pamięci urządzenia na którym zostanie uruchomiony program
- Wycieki pamięci

Diagram modułów

Znaczenie strzałki (moduł1 $\longrightarrow$ moduł2) moduł1 korzysta z jakiejś funkcji w moduł2.


class projekt1_gentex
managment
- name_file_backup_in: char* = NULL
- name_file_backup_out: char* = NULL
- name_file_out: char* = NULL
- name_file_stat: char* = NULL
- number_gram: int = 1
- number_paragraphs: int = 0
- number_words: int = 200
- get_mark() : int
- get_name_file_backup_in() : char*
- get_name_file_backup_out() : char*
- get_name_file_out() : char*
- get_name_file_stat() : char*
- get_nuber_paragraphs() : int
- get_number_words() : int
- info() : void
- main(int, char**) : int
- set_mark(int) : void
- test(int, int, char*) : void
error
- fatal(int, char*) : void
backup
- mark: int
- backup() : void
generation
- generation() : void
reading
- mark: int
- reading(char*) : void
store
- m: store* = NULL
- mark: int
- add(char**, char*) : void
- add_from_backup(char**, char**, int) : void
- add_suffix(char*, ngram*) : void
- add_tab(ngram*) : void
- cmp_prefix(char**, char**) : int
- fmt(ngram*) : char *
- get_m() : store *
- init() : void
- init_ngram(node_t*) : void
- insert(tree_t, char**, char*) : tree_t
- print_tree(tree_t, FILE*, int, char*) : void
- rand_prefix() : ngram*
- rand_suffix(char**) : char*
- resize_ngram() : void
- resize_suffix(ngram*) : void
- search_prefix(tree_t, char**) : ngram*
- set_m(store*) : void
stat
- all_words: int = 0
- g: tree_stat = NULL
- tg: tab_words = NULL
- ty: tab_words = NULL
- y: tree_stat = NULL
- add_tab_word(tab_words, tree_stat) : void
- cmp_stat(void*, void*) : int
- fmt_stat(tree_stat) : char *
- get_pmi(char*) : long double
- get_probability(tree_stat, char*) : double
- insert_stat(tab_words, tree_stat, char*) : tree_stat
- print_tree_stat(tree_stat, FILE*, char*) : void
- search_word(tree_stat, char*) : int
- stat_add_ngram(char**, char*) : void
- stat_add_word(char*) : void
- write_stat(char*) : void
- write_tab_tg(FILE*, int) : void
- write_tab_ty(FILE*, int) : void

Rysunek 1: