Sprawozdanie z testowania ./progtest

(generatora tekstu na podstawie łańcuchów Markova)

Projekt wykonany na zajęcia „Języki i metodyka Programowania”
wydz. Elektryczny Politechniki Warszawskiej, rok akademicki 2014/2015
wykonał: Paweł Drapiewski

1. Wstęp

Wykonywane prze zemnie testowanie w głównej mierze polegało na pisaniu własnych mini programów testujących, których głównym zadaniem było przede wszystkim uruchomienie danej funkcji z zadanymi parametrami. A także późniejsze wypisanie wyników działania tejże funkcji, dlatego ciężar oceny poprawności działania leżał na osobie uruchamiającej tejże testy, czyli na twórcy :D. Chociaż nie obyło się tu bez użycia programów zewnętrznych jak valgrind, pozwalający ocenić umiejętność uwalniania pamięci, czy też GNU debbugera. Jeżeli chodzi o valgrida, to jego działanie jest niezastąpione, ponieważ w umożliwiał on metodą prób i błędów (poprawianie kodu poprzez edycję funkcji free) osiągnąć całkowite uwalnianie pamięci. Za to GNU debugger okazał się wielkim pomocnikiem przy przeglądaniu zaimplementowanych strukutr. Czyli tego czy elementy zostały odpowiednio wczytane do struktury oraz czy malloc'y pamięci rzeczywiście wykonały swoją pracę. Debbuger był także niezbędny przy wszelkich błędach z naruszeniem ochrony pamięci, ponieważ z jego użyciem można było zlokalizować w której linie kodu ferelny błąd miał miejsce.

2. Testowanie modułów

Każdy moduł był testowany niezależnie od innych, i każdy z nich posiada swój własny test. Testy wszystkich modułów (tj ich kody źródłowe) są przechowywane w repozytorium pod adresem: https://github.com/drapek/jimp2/tree/master/src/tests. Co do samych efektów testów to niestety nie tworzyłem na bieżąco żadnej dokumentacji opisującej zaistniałe problemy, dlatego też postaram się choć szczątkowo odtworzyć niektóre z nich.

2.1 Obsługa flag

Był to dość długotrwały proces ponieważ używana biblioteka do rozpoznawania flag nie obsługiwała flag kilkuelementowych. Dlatego dla tych flag musiałem zrobić swoje funkcje interpretujące,, i oto w tym miejscu narodził się problem ponieważ nie było sposobu na odróżnienie końca flag kilku argumentowej oraz frazy początkowej generatora (który musiał znajdować się na końcu) wprowadziłem pewne ograniczenie, albowiem od tej pory wszystkie flagi muszą być zakończone znakiem „@”, co bardzo ułatwiło pracę. Po tym fakcie testowanie szło już gładko, i wszelkie kolejne testy jak np. podawanie tekstu zamiast liczby nie powodowały przewrotu interpretatora.

2.2 Struktury z ngramstruct

Każda ze struktur w ngramstruct ma swój szereg funkcji obsługujących, i każda z nich była oddzielnie testowana, niezastąpiony był tu DEBUGGER, przy użyciu którego na bieżąco podglądałem czy np. funkcja structadd odpowiednio dobrze alokuje pamięć. Oczywiście nie zapominałem o testowaniu na dużą ilość danych, tak by dać możliwość popisania się reallocowi... Ale i tutaj nie było większych problemów. Jedynym polem pod którym poległem to błędy przy dodawaniu 41 elementu do struktury ngramów. I aby ten błąd miał miejsce to użytkownik będzie musiał podać 41 plików wejściowych, szanse na to są niewielkie dlatego pominąłem ten błąd, oraz sporządziłem odpowiednią notatkę by o tym pamiętać w przyszłości.

2.3 Database

Moduł ten był dla najmniej problemowy, gdyż w przeszłości robiłem już projekt z zapisem binarnym do pliku. Dlatego proces tworzenia tego kodu polegał jedynie na jego przystosowaniu do mojego problemu. Sam test polegał na zapisywaniu różnych sekwencji do pliku binarnego, oraz późniejsze jego odczytanie oraz porównując czy dane się zgadzają. W tym przypadku nie było żadnych specjalnych błędów.

2.4 Errortest

Także prosty moduł, wypisujący ostrzeżenia do pliku oraz na ekran. Nic skomplikowanego, testy obejmowały to czy na errory krytyczne zakończy swoje działanie oraz czy dopisuje do pliku nowe dane (a nie je nadpisuje).

2.5 Textfile_analize

Moduł stanowiący pierwsze większe wyzwanie, miał on wczytywać słowa do struktury. Jego testowanie polegało na podawaniu ścieżek do plików tekstowych utworów literackich, a następnie wypisywaniu zawartości struktury by sprawdzić czy zostało to odpowiednio wykonane. Wszystkie testy, (nawet całe Quo Vadis) odbyły się pomyślnie. Jedynym problemem okazało się wczytywanie ponad 10 plików, wtedy następuje dość nietypowy błąd... Bo ostrzeżenie o naruszeniu pamięci wykonywało się po tym jak program kończył swe działanie. I z tego powodu, iż po pierwsze rzadko będzie wczytywanie 10 plików i więcej oraz po drugie cały program się wykonuje (co jest dziwne …) to doszedłem do wniosku, iż można pominąć ten błąd.

2.6 Generator

Najbardziej skomplikowany algorytmicznie moduł, który podlegał największemu testowaniu. Testowanie polegało na wpierw testowaniu każdej z funkcji modułu generatora, czyli patrzeniu jakie daje wyniki i ewentualnemu poprawianiu. I dopiero po testach każdej funkcji na tworzeniu bazy ngramów na podstawie argumentów wywołania, i uruchamianiu generatora w różnych wariancjach np. zmieniając typ ngramu. I tu miłym zaskoczeniem był fakt iż bardzo niewiele błędów wystąpiło podczas testowania i wszystkie one zostały natychmiastowo naprawione.

2.7 Statistic

Wpierw był testowany jako samodzielny byt, gdzie na podstawie argumentów wywołania sprawdzono funkcję dodającą ngramy różnych typów na własne struktury stats. Potem funkcję PMI, gdzie jego wyniki były porównywalne z rachunkiem ręcznym, oraz funkcję top (która listowała najczęściej występujące ngramy) patrząc na wyniki przez nią produkujące. Dopiero po pozytywnych testach dołączono ją do modułu generator (gdzie trzeba było dodać troszkę kodu tworzącego struktury do przechowywania anagramów i ngramów), i tu nastąpił błąd, który pogrążył cały moduł statistic. Okazało się że w połączeniu z generatorem funkcja dodająca anagramy i ngramy do odpowiednich struktur po prostu przestała odpowiednio działać. Sam błąd (prawdopodobnie) polega na tym, iż błędnie są porównywane elementy nowy (który jest dodawany do strukutry) oraz te już istniejące. Skutkiem czego inkrementowane są liczniki ngramów nie te co trzeba, a także zamazywane są istniejące już ngramy. Niestety ale bezpośrednia przyczyna nie jest mi jeszcze dobrze znana, jedynie co to podejrzewam iż winowajcą są operacje na adresach, bo funkcja add nie mallocuje pamięci i nie kopiuje tekstu, lecz adresy ngramów. Dlatego statystyki musiały zostać wyłączone chwilowo z programu.

3. Test całego programu

Końcowym etapem testów jest testowanie już złożonego programu. Jak się okazało dzięki dokładnemu testowaniu każdego z modułów, złożony program nie wykazywał specjalnie żadnych problemów. Jedynie co jakiś czas (można rzec że losowy) pozostałe funkcję po statystykach w generatorze generowały naruszenie ochrony pamięci. Dlatego z faktu iż jedynie służyły modułowi od statystyk, a został on wyłączony z projektu, to funkcje te zostały chwilowo za komentowane, aż do czasu naprawy statystyk. Test odbywał się dla różnych danych wejściowych, które prezentuje poniższa tabela:

Co jest testowane?	**Polecenie przy pomocy którego został uruchomiony program**	**Czas wykonania programu (przy użyciu unixowego programu *time)***	**Uwagi**
Analizator tekstowy, oraz generator tekstu. Wejście: jeden plik tekstowy.	./prodtekst -p tests/text_files/do_malbarki_Baudelaire.txt @	003s	BRAK
Analizator tekstowy, i generator tekstu. Wejście 10 plików tekstowych.	./prodtekst -p tests/text_files/* @	1.812s	BRAK
Analizator tekstowy, tworzenie bazy danych, generacja tekstu. Wejście: 2 pliki tekstowe Wyjście: Baza danych, plik z wygenerowanym tekstem	./prodtekst -p tests/text_files/do_malbarki_Baudelaire.txt tests/text_files/bak-i-pilka_Andersen.txt -x bazadanych1 -z wynik_produkcji @	0.022s	BRAK
Wczytywanie bazy danych, generowanie tekstu Wejście: plik bazy danych, utworzony w poprzednim przykładzie	./prodtekst -b bazadanych1 @	019s	BRAK
Wczytywanie bazy danych, analizator pliku, zapis bazy danych Wejście: plik tekstowy, plik bazy danych Wyjście: utworzony plik bazy danych	./prodtekst -b bazadanych1 -p tests/text_files/chlop_i_zmija_Mickiewicz.txt -x Bazadanych2 @	028s	BRAK
Wczytywanie dwóch baz danych	./prodtekst -b bazadanych1 Bazadanych2 @	020s	BRAK
Wczytywanie bazydanych, i zmiana	./prodtekst -b Bazadanych2 -n 4 @	023s	BRAK

ngramu			
Wczytywanie bazy danych, zmiana ngramu, oraz długości wygenerowanego tekstu	./prodtekst -b Bazadanych2 -n 4 -d 10000 @021s	021s	BRAK
Wczytywanie bazy danych, generacja tekstu od zadanej frazy	./prodtekst -b Bazadanych2 @ Mała sierota	003s	Nie wygenerowało tekstu, ale to poprawne działanie, bo nie odnalazło w bazie danych odpowiednika dla sierota
-||- ,	./prodtekst -b Bazadanych2 @ Mała mahoniowy	021s	Wygenerowało poprawnie tekst, bo w bazie odnaleziono „mahoniowy”
-||-, zmiana ngramu	./prodtekst -b Bazadanych2 -n 3 @ Takich miałam rodziców!	021s	Znalazło i wygenerowało tekst
A teraz dane niepoprawne:			
Nieporawne nazyw plików bazy danych oraz tekstowego	./prodtekst -b Bazadanychs -p lol23 @	002s	Rozpoznało iż pliki nie istnieją i zgłosiło to na ekran
Nieporawna długość ngramu oraz długość generowanego tekstu	./prodtekst -b bazadanych1 -n -32 -d -234 @	001s	Rozpoznało i poinformowało o tym iż podano złe wartości. A także podało jaki jest zakres poprawnych
Zbyt krótka fraza początkowa w stosunku do wybranego typu ngramu	./prodtekst -b bazadanych1 -n 3 @ Ala nie	002s	Rozpoznało i zgłosiło
Próba zapisu bazy w folderze do którego nie ma się uprawnień	./prodtekst -b bazadanych1 -x /root/baza1 @	002s	Rozpoznało i zgłosiło
To samo, ale dla pliku z generatora	./prodtekst -b bazadanych1 -z /root/baza1 @	002s	Rozpoznało i zgłosiło
Nie podanie znaku kończącego flagi	./prodtekst -b bazadanych1	001s	Rozpoznało i zgłosiło