Mateusz Marciniewicz
293150

[AAL] – dokumentacja projektu

1. Treść zadania

1.1. Uniwersalna część

Przedmiotem analizy jest tablica mieszająca przechowująca rekordy zawierające napisy. Długość tablicy ograniczona arbitralnie przez pewną stałą K. Dla danego napisu *s*, obliczamy *k = M(s)* i umieszczamy strukturę reprezentującą napis w tablicy mieszającej: *H[k]*. Przedmiotem implementacji powinno być dodanie i usunięcie elementów.

1.2. Wariant *W14*

W przypadku kolizji, obliczamy nową lokalizację i tak do skutku.

1.3. Wariant *W22*

Testy przeprowadzić dla sztucznie wygenerowanych słów. Generator ma posługiwać się tablicą prawdopodobieństw wystąpienia danej litery na początku słowa oraz litery po poprzedzającej literze (spacja, kropka, przecinek etc. traktowane są jako „koniec słowa”). Prawdopodobieństwa należy uzyskać z próbki tekstu polskiego.

1.4. Wariant *W32*

Zastosować dwie znacząco różne funkcje mieszające.

2. Język implementacji: C++

3. Implementacja tablicy mieszającej

3.1. Struktury danych

Tablica mieszająca przechowuje następujące dane:

- ***int*** *capacity* – maksymalna obecna pojemność tablicy mieszającej
- ***int*** *size* – aktualna liczba elementów w tablicy
- ***const float*** *MAX_LOAD_FACTOR* = 0.3 – wyjaśnione później
- tablica wskaźników do par { ***key*** , ***value*** }

Aby zwiększyć jej użytkowość, zaimplementowana została jako szablon klas. Na potrzeby prezentacji działania struktury, aplikacja klienta korzysta ze skonkretyzowanej pary { ***int*** , ***wstring*** }.

3.2. Implementowane metody tablicy mieszającej:

3.2.1. Metody „dla użytkownika”

- konstruktory domyślny, domyślny z parametrem *capacity*, destruktor
- przeciążony *operator[]* - za jego pośrednictwem możliwe jest wstawianie elementów do tablicy: *H[key] = value*
- ***void*** *remove(**Key** key)* – usuwa element o zadanym kluczu
- ***int*** *getsize()* - zwraca liczbę elementów w tablicy
- ***Value*** *valueOf(**Key** key)* – zwraca wartość elementu o zadanym kluczu
- ***void*** *show()* - pomocnicza funkcja wypisująca wszystkie info o tablicy

3.2.2. Metody prywatne

- ***int*** *findIndex(Key key)* – metoda zwraca bądź indeks do elementu o kluczu key, bądź indeks miejsca w tablicy, do którego może zostać przypisany taki element, jeśli jeszcze go nie ma. Metoda realizuje tym samym wariant **W14**
- ***void*** *reallocate()* - podwaja rozmiar tablicy mieszającej, zachowując jednocześnie wszystkie elementy. Uruchamiana w momencie, gdy liczba kolizji w trakcie wstawiania elementu będzie większa niż *capacity*MAX_LOAD_FACTOR*
- ***int*** *convertKey(Key k)* – konwertuje klucz dowolnego typu na ***int***
- ***void*** *hash1(**int** key)*, ***void*** *hash2(**int** key)* - dwie funkcje haszujące, realizujące wariant **W32**

4. Implementacja generatora słów (wariant *W22*)

4.1. Opis działania

Głównym zadaniem generatora jest produkowanie losowych słów w oparciu o prawdopodobieństwa wystąpienia liter w różnych przypadkach w języku polskim. Aby mogło to nastąpić, generator musi najpierw wykonać kilka operacji:

1. Odczytanie z pliku pewnej próbki tekstu w języku polskim, a następnie zliczenie wystąpień danej litery na początku słowa oraz po poprzedzającej literze.
2. Na podstawie zliczania, generator liczy prawdopodobieństwa wystąpienia liter w danych miejscach.
3. Na podstawie obliczonych prawdopodobieństw, generator „skleja” dystrybuanty, które posłużą do produkcji sztucznych słów.

Po wykonaniu powyższej procedury, generator jest gotów do pracy.

4.2. Struktury danych

Generator słów przechowuje następujące dane:

- ***Distribution*** *firstLetter* – dystrubuanta odnosząca się do litery stojącej na pierwszym miejscu słowa.
- ***vector<Distribution>*** *afterletter* – wektor dystrybuant. Każda dystrybuanta określa rozkład prawdopodobieństwa wystąpienia liter po literze określonej przez dystrybuantę

4.3. Implementowane metody

- ***void*** *collectSample(**string** pathName)* - realizacja punku 1. w **4.1**
- ***void*** *calculateProbabilities()* - realizacja punku 2. w **4.1**
- ***void*** *generateDistributions()* - realizacja punku 3. w **4.1**
- ***void*** *showLetterDistribution(**wchar_t** letter)* - wypisuje dystrybuantę odnoszącej się do *letter*
- ***void*** *generateRandomWord()* - generuje sztuczne słowo. Losuje liczbę z zakresu <0,1>, a następnie „zagląda” do *firstletter* i wybiera literę, której zakresowi w dystrybuancie odpowiada. Każdy następny krok odnosi się do *afterLetter*. Koniec, gdy trafi na ‘*’

4.4. Pomocnicze struktury danych

Generator słów wykorzystuje pomocniczą strukturę ***Distribution***. Zawiera pola:

- ***wchar_t*** *letter* - określa, literę, której dotyczy dystrybuanta
- ***list<pair<wchar_t, double>>*** cumulativeDistribution – lista par { litera , x ϵ <0,1> }

5. Analiza spodziewanej złożoności algorytmu

- optymistyczna złożoność czasowa: T(n) = 1 + 4 + 1 = 5
- pesymistyczna złożoność czasowa:
  T(n) = 1 + 2 + 2*0.3n + 1 + n + 4 = 1.6n + 8
- optymistyczna złożoność obliczeniowa: O(5) – brak jakichkolwiek kolizji
- pesymistyczna złożoność obliczeniowa: O(1.6n) – konieczność realokacji wskutek wystąpienia zbyt dużej liczby kolizji. n to liczba el. w tablicy