Politechnika Śląska
Wydział Automatyki, Elektroniki i Informatyki

Programowanie Komputerów

Szyfry z NN

autor	Mateusz Kucharczyk
prowadzący	mgr inż. Grzegorz Kwiatkowski
rok akademicki	2020/2021
kierunek	informatyka
rodzaj studiów	SSI
semestr	2
termin laboratorium	wtorek, 10:15 - 11:45
	czwartek, 12:30 - 14:00
sekcja	62
termin oddania sprawozdania	2021-06-23

1 Zadanie

Napisać obiektowo program do deszyfrowania, szyfrowania i łamania szyfru vigenere, cezar i bacon. Program jest uruchamiany z linii poleceń z wykorzystaniem nastepujących przełączników:

- `-t` nazwa szyfru (vigenere/cezar/bacon)
- `-a` działanie (szyfrowanie/deszyfrowanie/crackowanie)
- `-d` tekst lub ścieżka do pliku z tekstem
- `-k` klucz
- `-s` numer początkowej kombinacji klucza
- `-e` numer ostaniej kombinacji klucza (niewłącznie)
- `-v` ilosc najlepszych propozycji (domyslnie 1)
- `-u` spacje mają byc interpretowane jako znaki do szyfrowania
- `-o` ścieżka do pliku wyjściowego

2 Specyfikacja zewnętrzna

Program jest uruchamiany z linii poleceń. Należy przekazać do programu `-t` nazwę szyfru (vigenere/cezar/bacon), `-a` działanie, które chcemy wykonać (szyfrowanie/deszyfrowanie/crackowanie) oraz `-d` tekst na, którym chcemy wykonywać operacje. Dla szyfrowania i deszyfrowania musimy podać `-k` klucz. W szyfrze bacon nie możemy użyć parametru `-u`. Do łamania szyfrogramu musimy podać `-s` początek i `-e` koniec, można również podać `-v` ilość najlepszych propozycji do wyświetlenia oraz `-u` (spacje mają być iterpretowane jako znaki do szyfrowania zamiast jako separatory). Można również podać `-o` ścieżkę do pliku wyjściowego. Przykładowe wywołania:

```
program -t vigenere -a deszyfrowanie -d tajny tekst -k xd -u
program -t cezar -a szyfrowanie -d szyfrogram -k x
program -t bacon -a crackowanie -d szyfrogram
```

3 Specyfikacja wewnętrzna

3.1 Ogólna struktura programu

W funkcji głownej tworzony jest obiekt klasy Parametry który wczytuje i analizuje parametry podane w wierszu poleceń. W przypadku nieprawidłowego podania praramterów, wyświetla się pomoc i program się kończy. Następnie program tworzy obiekt klasy Szyfry wykorzystując polimorfizm i metody wirtualne, który obsługuje szyfr podany w parametrze −t. Później wykonuje działanie podane w parametrze −a. W przypadku szyfrowania i deszyfrowania uruchamia odpowiednią metodę. Natomiast dla łamania bez klucza tworzy obiekt typu Crack. Obiekt ten wykorzystuje obiekt klasy NeuralNet który ocenia zdeszyfrowany tekst oraz obiekt klasy MyMap ktory przechowuje najlepsze klucze, teksty zdeszyfrowane oraz oceny. Na koniec wyświetalana jest odpowiedź i program się kończy.

3.2 Szczegółowy opis klas i strukur danych

Szczegółowy opis klas i strukur danych znajduje się w załączniku.

4 Testowanie

Program został przetestowany na różnego rodzaju wartościach parametrów, kolejności parametrów i ilości parametrów.

Program został sprawdzony pod kątem wycieków pamięci.

5 Algorytmy

5.1 Szyfr Cezar

Każda litera tekstu jawnego jest zastępowana inną, oddaloną od niej o stałą liczbę w alfabecie.
Szyfrowanie wyrażane jest wzorem

$f(a) = (a + k) \mod n$

Deszyfrowanie wyrażane jest wzorem

$f(a) = (a - k) \mod n$
`a` - szyfrowana litera
`k` - klucz
`n` - liczba liter w alfabecie

5.2 Szyfr Vigenere

Każda litera tekstu jawnego szyfrowana jest szyfrem Cezara w którym kluczem jest odpowidająca litera kodu użytego w szyfrze Vigenere. W celu zdeszyfrowania każdą literę szyfrogramu deszyfruje się szyfrem Cezara w którym kluczem jest odpowidająca litera kodu użytego w szyfrze Vigenere

5.3 Szyfr Bacon

Szyfrowanie polega na zamianie znaków tekstu jawnego na pięcioliterowe ciągi złożone z liter klucza. W celu zdeszyfrowania pięcioliterowe ciągi zamienia się na odpowiadające im znaki.

5.4 Łamanie

W celu łamania szyfrów generowane są kolejo możliwe klucze. Szyfrogram jest deszyfrowany wygenerowanymi kluczami. Zdeszyfrowany szyfrogram jest dzielony na wyrazy (spacjami) i liczona jest średnia ocen wyrazów nie przekraczajacych 14 znaków. Ocenę wyrazów przprowadza sieć neuronowa składająca się z trzech warstw neuronów (390 wejściowych, 30 ukrytych i jednego wyjściowego). Funkcją aktywacji ukrytej warsty jest funkcja ReLu, a neuronu wyjściowego funkcja sigmoid. Następnie ocena, klucz i zdeszyfrowany tekst są umieszczane w automatycznie sortującej dynamicznej liście MyMap.

6 Kod źródłowy

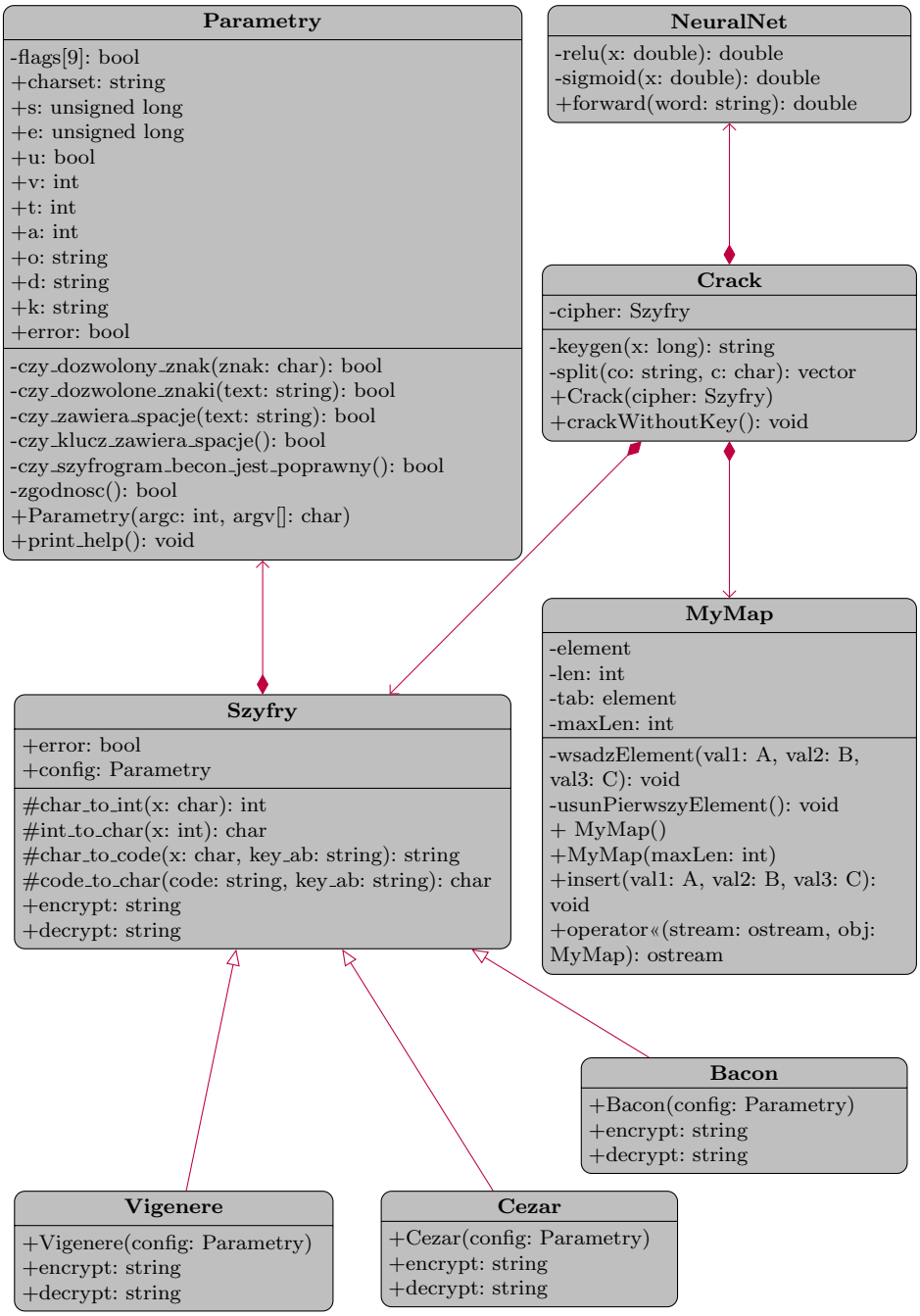
Kod źródłowy zamieszczony jest pod linkiem:
https://github.com/polsl-aei-pk2/f359c17d-gr62-repo/tree/main/Projekt

7 Wnioski

Podczas tworzenia programu nauczyłem się wczytywać przełączniki z lini poleceń.

Dodatek
Szczegółowy opis klas i strukur danych

1 Graf klas



2 Opis klas

Opis klas oraz metod, pól i struktur

2.1 MyMap

Dynamiczna lista automatycznie sortująca i uwuwająca nadmiar (typ A musi być sortowalny). Wykorzystana do przechowania oceny, klucza i tekstu jawnego.

- **Metody**
  - **MyMap()** - konstruktor, pobiera maksymalny rozmiar listy (-v)
  - **~MyMap()** - destruktor
  - **void insert()** - wczytuje dane do listy i usuwa nadmiar
  - **void usunPierwszyElement()** - usuwa pierwszy element
  - **void wsadzElement(A val1, B val2, C val3)** - wczytuje argumenty do listy
  - **friend std::ostream& operator«(std::ostream& stream, const MyMap& obj)** - wypisuje liste

- **Pola**
  - **int len** - przechowuje aktualną długość listy
  - **int maxLen** - maksymalna długość listy
  - **element* tab** - wskaźnik na element tablicy

- **Struktury**
  - **element** - zawiera trzy wartości oraz wskaźnik na następny element

2.2 NeuralNet

Wykorzystywana do oceny tekstu zdeszyfrowanego

- **Metody**
  - **double relu(double x)** - funkcja ReLu
  - **double sigmoid(double x)** - funkcja sigmoid
  - **double forward(std::string word)** - zwraca ocene słowa word

2.3 Parametry

Obsługuje wiersz poleceń.

- **Metody**
  - **bool zgodnosc()** - sprawdza zgodność parametrów wiersza poleceń

- **void print_help()** - wyświetla pomoc
- **bool czy_klucz_zawiera_spacje()** - sprawdza czy klucz zawiera spacje
- **bool czy_szyfrogram_becon_jest_poprawny()** - sprawdza czy szyfrogram becon jest poprawny
- **bool czy_dozwolony_znak(const char& znak)** - sprawdza czy znak jest dozwolony
- **Parametry(const int& argc, const char* argv[])** - konstruktor, przyjmuje argumenty wiersza poleceń
- **bool czy_zawiera_spacje(const std::string& text)** - sprawdza czy zawiera spacje
- **bool czy_dozwolone_znaki(const std::string& text)** - sprawdza czy znaki są dozwolone

- **Pola**
  - **int t** - warość parametru t
  - **int v** - warość parametru v
  - **int a** - warość parametru a
  - **bool u** - warość parametru u
  - **bool error** - czy błędne parametry
  - **bool flags[8]** - czy argument został podany
  - **std::string o** - warość parametru o
  - **std::string k** - warość parametru k
  - **std::string d** - warość parametru d
  - **unsigned long s** - warość parametru s
  - **unsigned long e** - warość parametru e
  - **const std::string charset** - dozwolone znaki

2.4 Szyfry

Klasa bazowa wszystkich szyfrów.

- **Metody**
  - **virtual std::string encrypt()** - wirtaulna metoda encrypt

- **virtual std::string decrypt()** - wirtaulna metoda decrypt
- **int char_to_int(const char& x)** - zamienia znak na liczbę
- **char int_to_char(const int& x)** - zamienia liczbę na znak
- **std::string char_to_code(const char& x, const std::string& key_ab)** - zamienia znak na sekwencję szyfru bacona
- **char code_to_char(const std::string& code, const std::string& key_ab)** - zamienia sekwencję szyfru bacona na znak

- **Pola**
  - **bool error** - czy błąd operacji
  - **Parametry* config** - wskaźnik na obiekt z parametrami

2.5 Vigenere

Odpowiada za szyfrowanie i deszyfrowanie szyfru Vigenerea z użyciem klucza.

- **Metody**
  - **std::string encrypt()** - szyfrowanie
  - **std::string decrypt()** - deszyfrowanie
  - **Vigenere(Parametry* config)** - konstruktor, pobiera obiekt zawierający parametry

2.6 Cezar

Odpowiada za szyfrowanie i deszyfrowanie szyfru Cezara z użyciem klucza.

- **Metody**
  - **std::string encrypt()** - szyfrowanie
  - **std::string decrypt()** - deszyfrowanie
  - **Cezar(Parametry* config)** - konstruktor, pobiera obiekt zawierający parametry

2.7 Bacon

Odpowiada za szyfrowanie i deszyfrowanie szyfru Bacona z użyciem klucza.

- **Metody**
  - **std::string encrypt()** - szyfrowanie

- **std::string decrypt()** - deszyfrowanie
- **Bacon(Parametry* config)** - konstruktor, pobiera obiekt zawierający parametry

2.8 Crack

Odpowiada za łamanie szyfru Vigenere, Cazar i Bacon (bez klucza).

- **Metody**
  - **std::string keygen(const long& x)** - generuje klucz
  - **Crack(Szyfry* cipher)** - konstruktor, pobiera obiekt zawierający szyfr
  - **void crackWithoutKey()** - metoda realizująca łamnie bez klucza
  - **std::vector<std::string>split(std::string& co, char& c)** - rozdziela tekst podanym znakiem
- **Pola**
  - **Szyfry* cipher** - wskaźnik na obiekt z szyfrem