Politechnika Świętokrzyska

Wydział Elektrotechniki, Automatyki i Informatyki

Zespół:

**Kot Jarosław**

**Prusicki Jakub**

**Łamacz haseł**

Projekt zespołowy

na studiach stacjonarnych

o kierunku **Informatyka**

Stopień **II**

Opiekun projektu:

**dr inż. Paweł Paduch**

Kielce, 2023

**SPIS TREŚCI**

[1. Wstęp teoretyczny. 3](#_heading=h.gjdgxs)

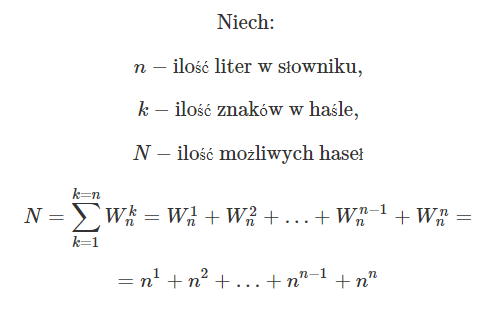
1.1. Algorytm Brute Force. 3

1.2. Metoda słownikowa. 4

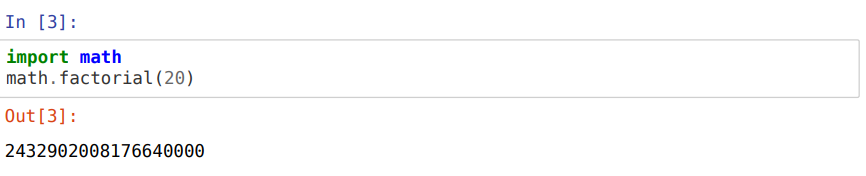
2[. Przykładowe rozwiązania.](#_heading=h.gjdgxs) 7

1. **Wstęp teoretyczny.**
   1. **Algorytm Brute Force.**

**Algorytm siłowy** (z ang. **brute force**) jest to taki algorytm, którego działanie można streścić następująco: polega on na sukcesywnym sprawdzaniu wszystkich możliwych kombinacji, w poszukiwaniu rozwiązania problemu. Dla niektórych klas problemów, jest to podejście, pod pewnymi warunkami, jedyne niezawodne. Stosując takie podejście w programowaniu, rozwiązujemy problem przez weryfikację i ocenę wszystkich wariantów postępowania. Gdybyśmy dysponowali, dowolnie długim czasem na poszukiwanie rozwiązania oraz zasobami sprzętowymi (na przykład nieskończenie dużą pamięcią operacyjną lub przestrzenią dyskową), to algorytm taki zakończy się zawsze powodzeniem. O związkach metody brute force, z kombinatoryką wspomniano nie bez przyczyny. Korzystając z kombinatoryki, można policzyć ile jest możliwych takich haseł. Po obliczeniach można stwierdzić, że wartość tego zagadnienia jest sumą wariacji z powtórzeniami. Dla przypomnienia, wariację z powtórzeniami wykorzystujemy wtedy, gdy chcemy wiedzieć ile możemy stworzyć różnych układów k-elementowych, mając do dyspozycji n-elementów, przy czym kolejność elementów w układzie jest istotna, a elementy mogą się powtarzać[1].



Rys. 1.1.1 Wzór na liczbę kombinacji

Innymi słowy metoda siłowa rozwiązania jakiegoś zadania polega na wyczerpaniu wszystkich możliwości. Dla anagramów oznacza to wygenerowanie listy wszystkich możliwych łańcuchów ze znaków łańcucha s1 i sprawdzenie, czy s2 znajduje się na tej liście. Nie jest to jednak zalecane podejście, przynajmniej w tym przypadku. Zauważmy mianowicie, że dla ciągu znaków s1 o długości n mamy n wyborów pierwszego znaku, (n-1) możliwości dla znaku na drugiej pozycji, (n-2) na trzeciej pozycji itd. Musimy zatem wygenerować n! łańcuchów znaków. Dla ustalenia uwagi przyjmijmy, że *s1* składa się z 20 znaków. Oznacza to konieczność wygenerowania**** Rys. 1.1.2 Ilość możliwych łańcuchów.

łańcuchów znaków, a następnie odszukanie wśród nich ciągu s2. Można było dostrzec takie podejście już wcześniej - szczególnie,jeśli chodzi o to ile czasu zajmuje algorytm klasy O(n!), dlatego nie jest to polecane podejście do zagadnienia anagramów[2].

* 1. **Metoda słownikowa**

Ataki słownikowe niezmiennie pozostają jedną z najskuteczniejszych [metod odzyskiwania haseł](http://sekurak.pl/lamanie-hasel-z-wykorzystaniem-cpugpu/). Tam, gdzie próby siłowe nie są w stanie w rozsądnym czasie doprowadzić do jakichkolwiek rezultatów, dobry słownik może zdziałać cuda. Do skutecznego działania niezbędne jest odpowiednie oprogramowanie oraz właśnie dobry słownik. O ile programy takie jak John The Ripper czy hashcat są dobrze znane i łatwo dostępne, o tyle znalezienie dobrego słownika nie jest już wcale takie proste. W celu odzyskania haseł z odpowiadających im skrótów (hashy) możemy skorzystać z odpowiedniego oprogramowania. Możemy też odwiedzić jeden z wyspecjalizowanych serwisów internetowych. Często są one stanie w ciągu kilku sekund odzyskać hasła ze skrótów wyznaczonych za pomocą algorytmów takich jak: LM, NTLM, md2,sha224, ripeMD160, whirlpool, MySQL 4.1+. Do wykonania powyższego zadania serwisy wykorzystują ogromne tablice skrótów i odpowiadających im haseł o wielkościach liczonych w setkach gigabajtów. Odpowiednie tablicowanie wpływa na szybkość całego procesu, zaś o skuteczności decyduje jakość oraz rozmiar bazowego słownika[3]

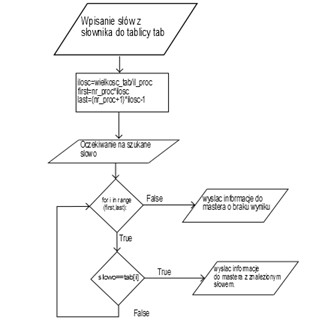
Słownikowe algorytmy kodowania polegają na kolejnym czytaniu sekwencji danych wejściowych i przeszukiwaniu słownika w dokładnie tej samej sekwencji danych. W przypadku zakończenia przeglądu sukcesem, indeks właściwej pozycji słownika staje się reprezentacją wyjściową przy czym im dłuższa jest sekwencja tym efektywniejszą jest kompresja. W kodowaniu słownikowym można także zastosować statyczny model słownika. Jest on skuteczniejszy, gdyż zna z wyprzedzeniem pojawiające się ciągi danych. Problemy związane z koniecznością dopisania do zbioru danych skompresowanych słownika użytego przy kodowaniu, są identyczne jak w metodach statycznych. Są stosowane w praktyce, ale jedynie do określonych przypadków kodowania w miarę stacjonarnych zbiorów danych o charakterystyce znanej a priori. Algorytmy przeznaczone do ogólnych zastosowań są w zdecydowanej większości adaptacyjne. W statycznym (entropijnym) kodowaniu pojedynczy symbol zastępowany jest ciągiem bitów o zmiennej długości. W przypadku kodowania słownikowa koncepcja jest odwrotna. Ciąg symboli o zmiennej długości jest zastępowany sekwencją kodową - indeksem w tablicy słownika. Ogólnie rzecz biorąc model ten polega na budowaniu słownika na podstawie znaków alfabetu, bądź to uzupełniany o słowa oparte na analizie danych pochodzących ze zbiorów kompresji. Następnie słownik ten jest zapisywany, bądź przesyłany do dekodera.[4]

**Algorytm[4]:**

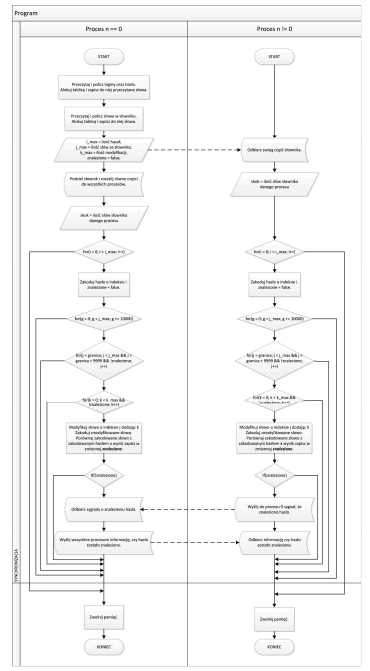
1. Inicjacja słownika - pierwsza pozycja jako NULL.
2. Przeszukaj aktualny słownik i ustal najdłuższy łańcuch z kolejno czytanych symboli zgodny ze słownikiem.
3. Utwórz słowo kodowe najdłuższego łańcucha: indeks + kolejny symbol.
4. Dopisz nową frazę do słownika (z pkt.3).
5. Powtarzaj od pkt.2 aż do zakodowania wszystkich symboli na wejściu.

Algorytm łamania haseł metodą słownikową jest techniką używaną do siłowego odgadywania haseł do systemów. Jego realizacja wymaga dużych nakładów obliczeniowych, dlatego też uzasadnione jest wykorzystanie do tego celu programowania równoległego. W artykule przedstawiono i porównano ze sobą, równoległe implementację tego algorytmu w trzech różnych środowiskach programowania równoległego, realizujące zrównoleglenie obliczeń na poziomie danych oraz środowisko procesorów kart graficznych[5].

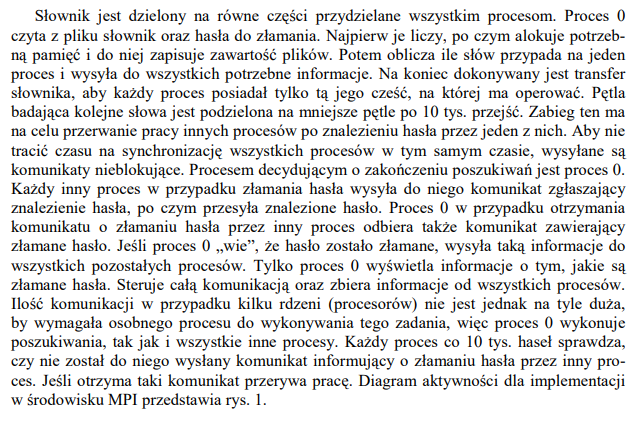
Nie zawsze odzyskiwanie czy też testowanie siły haseł sprowadza się do łamania hashy. Wiele programów lub metod słownikowego odzyskiwania haseł wymaga podania wprost słownika haseł, które chcemy po kolei sprawdzić. Przykładem mogą być tu ataki online, czyli takie, które próbują odzyskać hasło na podstawie przeprowadzania kolejnych prób logowania do docelowego systemu/usługi.[3]



Rys. 1.2.1 Schemat blokowy algorytmu słownikowego.

1. **Przykład innego rozwiązania.**

Rys. 2.1Diagram aktywności dla implementacji.



Rys. 2.2 Opis przedstawionego rozwiązania.

**Literatura**

1. Algorytm brute force

*http://mumin.pl/Skrypt\_A\_do\_Z/lekcje/l06.html*

1. Działanie algorytmu brute force

<http://prac.im.pwr.wroc.pl/~szwabin/assets/algo/lectures/5.pdf>

1. Opis ataków słownikowych.

https://sekurak.pl/crackstation-udostepnia-ogromny-slownik-hasel/

1. Kodowanie słownikowe

<https://www.ire.pw.edu.pl/~rois/dydaktyka/skrypt/5.pdf>

1. Działanie metody siłowej

<http://neo.dmcs.p.lodz.pl/ptda/wyklady/wyklad_5.pdf>

1. Równoległe łamanie haseł

<http://yadda.icm.edu.pl/baztech/element/bwmeta1.element.baztech-article-BGPK-3546-3483>

1. Schemat blokowy algorytmu słownikowego

https://slideplayer.pl/slide/11339059/

1. Równoległe łamanie haseł - przykład

*https://repozytorium.biblos.pk.edu.pl/redo/resources/32576/file/suwFiles/PlazekJ\_RownolegleLamanie.pdf*

1. Brute force/metoda słownikowa

https://docplayer.pl/8991975-Akd-metody-slownikowe.html

Politechnika Świętokrzyska

Wydział Elektrotechniki, Automatyki i Informatyki

Zespół:

**Kot Jarosław**

**Prusicki Jakub**

**Łamacz haseł**

Projekt zespołowy

na studiach stacjonarnych

o kierunku **Informatyka**

Stopień **II**

Opiekun projektu:

**dr inż. Paweł Paduch**

Kielce, 2023

**SPIS TREŚCI**

1[. Architektura i technologie.](#_heading=h.gjdgxs)3

2[. Algorytm Brute Force.](#_heading=h.gjdgxs) 4

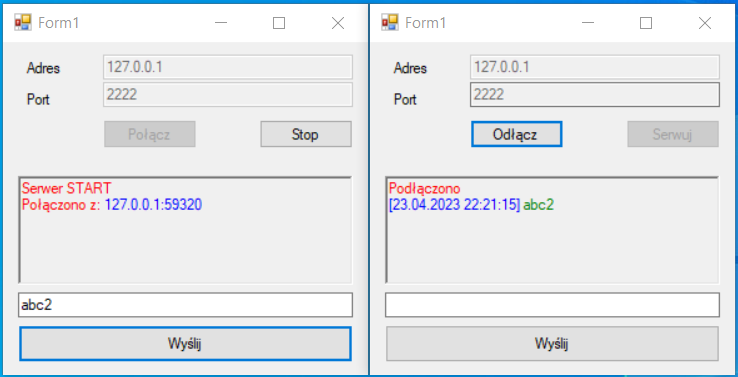
2.1. Implementacja. 4

2.2. Wady i zalety. 6

2.3. Przykładowe rozwiązania. 7

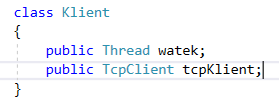
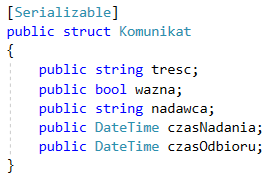
3. Wydajność 7

1. **Architektura i technologie.**

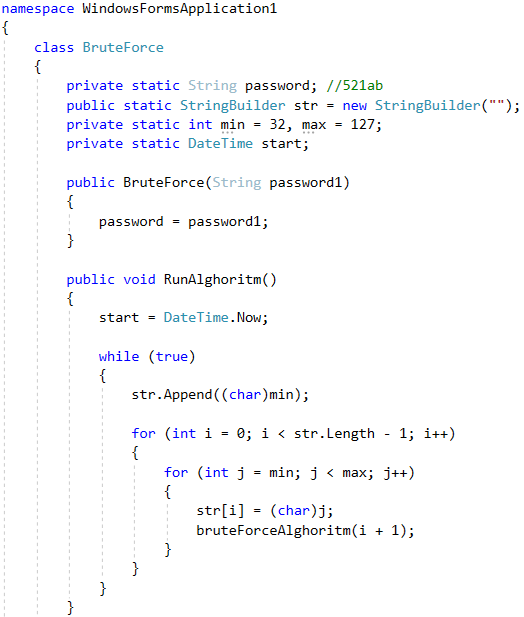
Do wykonania projektu zdecydowano się wykorzystać kod źródłowy udostępnionego już rozwiązania na platformie *achilles.tu.kielce.pl* przez opiekuna projektu. Jest to rozproszony komunikator tekstowy wykorzystujący połączeniowy protokół TCP. Zaletą tego konkretnego podejścia jest uniwersalność zaprojektowanego rozwiązania - odpowiednie przyciski *serwuj* oraz *połącz* pozwalają na opóźnienie decyzji, który z programów ma być serwerem. GUI aplikacji zostało wykonane z użyciem formatek okienkowych WinForms, będących z resztą częścią .Net Framework.

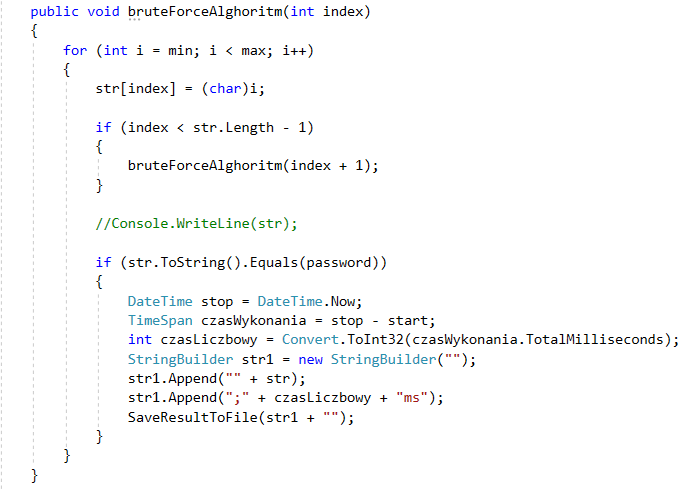
Rys. 1.1 Okienka aplikacji.

Podstawowymi elementami projektu są: struktura *Komunikat* oraz klasa *Klient*. Pozwalają one na prawidłowe przesyłanie treści w prezentowanym powyżej komunikatorze. Podczas wykonywania prac nad rozpraszaniem to właśnie ich modyfikacje będą miały największe znaczenie.

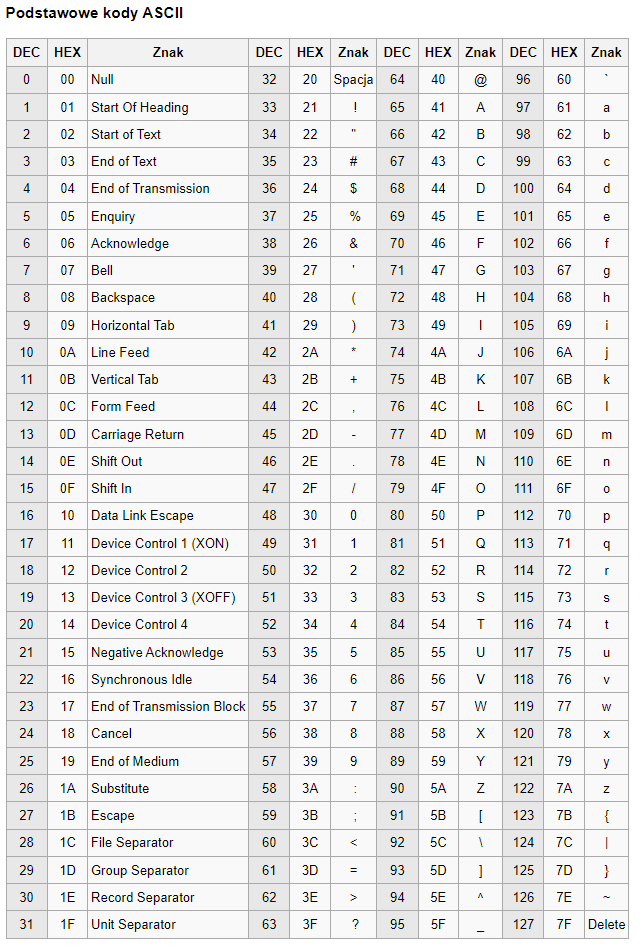


Rys. 1.2. Struktura komunikat.Rys. 1.3. Klasa Klient.

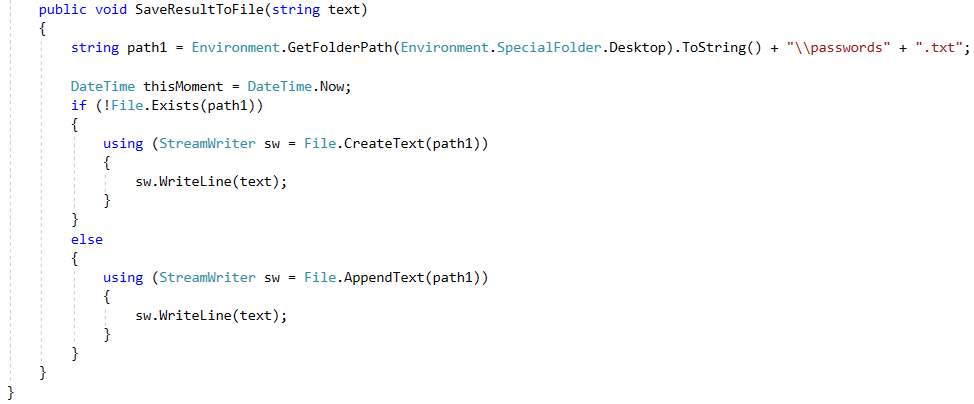
1. **Algorytm Brute Force.**
   1. **Implementacja.**



Rys. 2.1 Algorytm Brute Force.

Cały algorytm mieści się w jednej klasie **BruteForce**i jest uruchamiany za pomocą metody ***RunAlghoritm()***. Program jest początkową wersją implementacji, która będzie rozbudowywana w celu poprawy jej obsługi (konfiguracja parametrów) co może mieć znaczący wpływ na zmianę wydajności działania samego algorytmu. Najważniejszymi atrybutami tej klasy są pola: **password, min** oraz **max**. Pierwszy z nich służy oczywiście do obsługi samego hasła, natomiast kolejne wyznaczają odpowiednio początek i koniec alfabetu. Są to liczby, ponieważ w tym programie odpowiadają one znakom z tablicy ASCII. Najczęściej używane zostały przedstawione w tabeli poniżej. 

Rys. 2.2 Tablica znaków ASCII.

Ponadto możemy dostrzec mierzenie czasu, które rozpoczyna się tuż po wywołaniu samej metody, a kończy w ostatniej instrukcji warunkowej - stwierdzającej czy dane hasło zostało złamane. Czas wykonania zapisywany jest w **milisekundach** w postaci liczbowej. Następnie hasło wraz z wartością czasu operacji jest zapisywane do pliku (możliwe, że format tekstowy będzie podlegał jeszcze modyfikacjom, bądź całkowitym zamianie np. na format typu excel). Sam plik tworzony jest w sposób uniwersalny na pulpicie wykonywanego sprzętu komputerowego. Co pozwala uelastycznić rozwiązanie i zaoszczędzić czasu na zmiany samych ścieżek, a skupić się na poważniejszych kwestiach obsługi samego programu. Oczywiście jeżeli taki plik już istnieje jest on już tylko nadpisywany. Kod zapisu wyników programu zaprezentowano na poniższym listingu.

Rys. 2.3 Zapis haseł do pliku.

* 1. **Wady i zalety.**

Zalety:

* łatwość zrozumienia i zaimplementowania
* uniwersalność - algorytm może być stosowany do rozwiązywania różnego rodzaju problemów, w tym problemów wielowymiarowych
* nie wymaga dodatkowych założeń
* możliwość stosowania nawet w przypadku problemów nieliniowych

Wady:

* mniejsza dokładność w porównaniu do innych metod numerycznych
* wymaga dużego nakładu obliczeniowego
* wymaga wyznaczenia dokładnej funkcji sił, która może być trudna do znalezienia w niektórych przypadkach
  1. **Inne rozwiązania.**

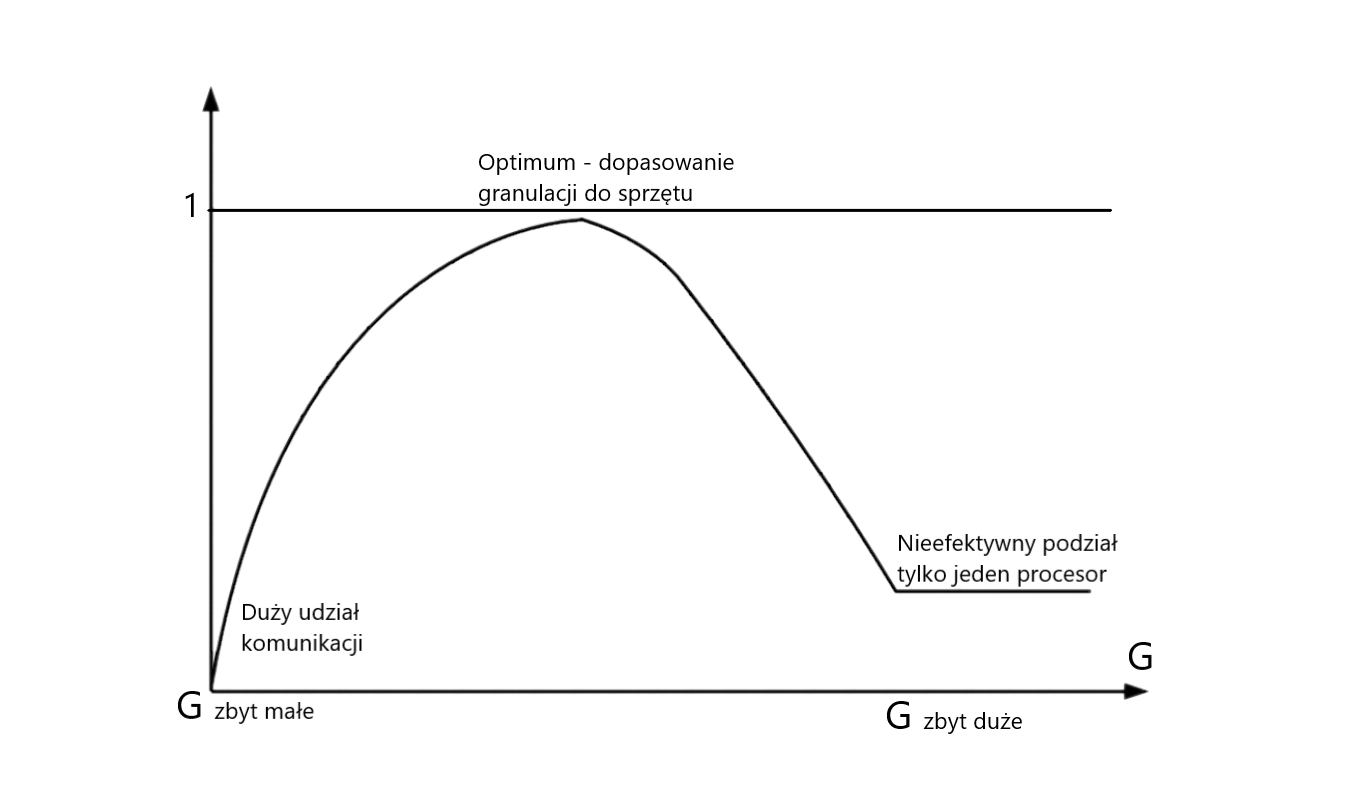
Aby wybrać odpowiednią wersję algorytmu, sprawdzono dotychczasowe rozwiązania opublikowane już w zasobach internetu. Najważniejsze z nich, które pomogły w zbudowaniu algorytmu pochodzą z następujących źródeł:

* Dot Net Office[1]
* Użytkownik serwisu github.com *jwoschitz*[2]
* C # Corner[3]

Oczywistym jest fakt, że nasz kod będzie nadal modyfikowany, będą implementowane dodatkowe funkcje, a te istniejące rozwijane, jednak na ten moment efekty prostego i poprawnie działającego algorytmu wystarczyły aby przetestować program przesyłający pojedyncze hasło od klienta do serwera aby ten mógł je złamać i zapisać do opisywanego już wcześniej pliku.

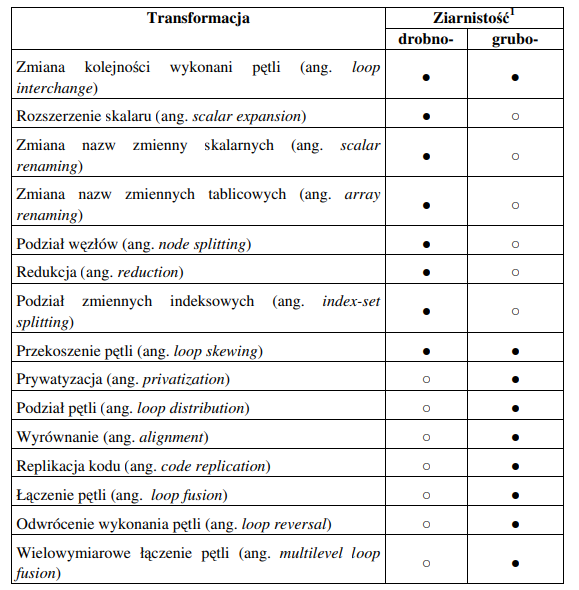
**3. Wydajność**

Ziarnistość projektu to poziom podziału pracy w projekcie, który określa, na jakie mniejsze części jest podzielony projekt i jakie zależności występują między nimi. W kontekście programowania współbieżnego, ziarnistość projektu odnosi się do sposobu podziału pracy pomiędzy wątki lub procesy. Wydajność ziarnistości projektu zależy przede wszystkim od rodzaju zadania oraz architektury systemu, w którym jest realizowane. W ogólnym przypadku, mniejsza ziarnistość (czyli mniejsze podziały pracy) może przyspieszyć wykonanie zadania, ale może też wprowadzić koszty synchronizacji i komunikacji między wątkami lub procesami. Z kolei większa ziarnistość (większe podziały pracy) może zmniejszyć koszty synchronizacji i komunikacji, ale może też wprowadzić koszty narzutu, związanego z tworzeniem i zarządzaniem dodatkowymi wątkami lub procesami. W przypadku projektów o dużym obciążeniu procesora, mniejsza ziarnistość jest zwykle bardziej wydajna, ponieważ zapewnia lepsze wykorzystanie zasobów sprzętowych. W przypadku projektów o dużym obciążeniu sieciowym lub wejścia-wyjścia, większa ziarnistość może być bardziej wydajna, ponieważ zmniejsza ilość operacji synchronizacji i komunikacji. Ostatecznie, wybór odpowiedniej ziarnistości projektu zależy od indywidualnych wymagań projektu oraz dostępnych zasobów sprzętowych. Warto pamiętać, że optymalna ziarnistość projektu może być osiągnięta tylko poprzez testowanie i analizę wyników działania systemu w różnych konfiguracjach.



Rys. 2.4.1 Wykres doboru ziarnistości[4].

W ogólnym przypadku ziarnistość kodu jest pojęciem względnym, ściśle związanym z określona architekturą systemu wieloprocesorowego. Kod wykonany na systemie z pamięcią dzieloną sklasyfikowany jako gruboziarnisty, może zostać uznany za drobnoziarnisty w systemie z pamięci rozproszoną. Zgodnie z tym, co zostało przedstawione, w zależności od architektury koszt zarządzania wielozadaniowością (np. zarządzanie wątkami) może znacznie różnić się od siebie. W zależności od typu użytej transformacji pętli moliwe jest uzyskanie podanej ziarnistości kodu dla wybranej architektury. Tabela 2.4.2 przedstawia podział transformacji pętli programowych w zależności od możliwej do uzyskania ziarnistości.

Tabela 2.4.2 Tabela transformacji do uzyskania określonej ziarnistości[5]

**Literatura**

1. Kod algorytmu brute force

*https://www.dotnetoffice.com/2022/10/brute-force-algorithm-in-c.html*

1. Kod algorytmu brute force - 2.

*https://gist.github.com/jwoschitz/1129249*

1. Kod algorytmu brute force - 3.

<https://www.c-sharpcorner.com/article/how-using-brutal-force-could-improve-code-quality>

1. Wykres doboru ziarnistości.

*https://achilles.tu.kielce.pl/portal/Members/332bd2d158a24964b2b98a7fc2879842/programowanie-wspolbiezne/wyklady/1-wstep/wyk1\_wprowadzenie.pdf*

1. Tabela transformacji do uzyskania określonej ziarnistości.

*http://zbc.ksiaznica.szczecin.pl/Content/4621/PHD\_Krzysztof\_Siedlecki.pdf*

Politechnika Świętokrzyska

Wydział Elektrotechniki, Automatyki i Informatyki

Zespół:

**Kot Jarosław**

**Prusicki Jakub**

**Łamacz haseł**

Projekt zespołowy

na studiach stacjonarnych

o kierunku **Informatyka**

Stopień **II**

Opiekun projektu:

**dr inż. Paweł Paduch**

Kielce, 2023

**SPIS TREŚCI**

[1. Metoda słownikowa - opis algorytmu. 3](#_heading=h.gjdgxs)

2[. Haszowanie.](#_heading=h.gjdgxs) 4

2.1. Co to jest hash ? 4

2.2. MD5. 4

2.3. SHA-1,SHA-256. 5

2.4. Haszowanie “Solone”. 8

3. Rozwój projektu. 8

3.1. Wydajność Brute Force. 8

3.2. Generowanie możliwych haseł na podstawie słownika. 9

3.3. Wyliczanie zakresów. 10

3.4. Popularne hasła. 11

1. **Metoda słownikowa- opis algorytmu.**

Metoda słownikowa to jedna z najprostszych i najskuteczniejszych metod łamania haseł. Polega ona na próbie odgadnięcia hasła za pomocą słownika zawierającego różne kombinacje słów, haseł i znaków specjalnych. Istnieją różne wersje metody słownikowej, ale podstawowa koncepcja jest taka sama. Atakujący używa listy haseł, która zawiera popularne hasła, słowa i kombinacje znaków. Jeśli hasło znajduje się na liście, atakujący może uzyskać dostęp do chronionej informacji. Jedną z popularnych wersji metody słownikowej jest tzw. "attack by list". W tej metodzie atakujący wykorzystuje gotową listę haseł, która jest zazwyczaj tworzona na podstawie często używanych haseł, słów kluczowych, nazw użytkowników itp. Lista ta może być pobrana z internetu lub stworzona przez samodzielne zbieranie informacji o ofierze. Następnie atakujący przeprowadza atak, używając tych haseł na kolejnych stronach internetowych lub w systemach informatycznych, które próbuje złamać. Inną wersją metody słownikowej jest tzw. "brute force attack". Ta metoda polega na wypróbowaniu wszystkich możliwych kombinacji haseł, aż zostanie znalezione poprawne hasło. To jednak wymaga ogromnej ilości czasu i mocy obliczeniowej, więc jest mało efektywna w przypadku dłuższych i bardziej złożonych haseł. Istnieją również metody hybrydowe, które łączą w sobie metody słownikowe i brute force. Atakujący używa listy haseł, ale dodaje do niej różne kombinacje liter i cyfr, co zwiększa szansę na odgadnięcie hasła. Jednym ze sposobów zabezpieczenia się przed atakami metodą słownikową jest stosowanie silnych haseł. Silne hasło to takie, które składa się z co najmniej ośmiu znaków, zawiera duże i małe litery, cyfry oraz znaki specjalne. Należy również unikać używania popularnych haseł i słów, takich jak "123456" czy "password". Innym sposobem jest stosowanie technik szyfrowania haseł, takich jak funkcje skrótu (hashing) lub algorytmy szyfrowania. Szyfrowanie uniemożliwia odczytanie hasła przez atakującego nawet w przypadku jego przechwycenia. Co zostanie opisane szerzej w punkcie nr **2** (rodzaje haszowań).

1. **Rodzaje haszowań.**
   1. **Co to jest hash ?**

Hash’em nazywamy wynik działania operacji matematycznej (nazywanej funkcją skrótu) na określonym ciągu znaków (np. na haśle lub pliku). Funkcja ta przekształca podane przez użytkownika dane wejściowe (np. hasło) na krótką, posiadającą stały rozmiar wartość znakową. Ważną własnością funkcji skrótu jest to, że jest ona nieodwracalna . Osoba mając hash nie może zastosować funkcji odwrotnej do funkcji skrótu by poznać ciągu znaków (np. hasła), dla którego dany hash został wygenerowany. Haszowanie jest to pewna technika rozwiązywania ogólnego problemu słownika. Przez problem słownika rozumiemy tutaj takie zorganizowanie struktur danych i algorytmów, aby można było w miarę efektywnie przechowywać i wyszukiwać elementy należące do pewnego dużego zbioru danych (uniwersum). Przykładem takiego uniwersum mogą być liczby lub napisy (wyrazy) zbudowane z liter jakiegoś alfabetu[4]. Ogólnie rzecz biorąc hash to wynik zastosowania funkcji skrótu na pewnym ciągu znaków (haśle lub pliku). Hash pozostaje stały dla danego pliku, a każda choćby najmniejsza modyfikacja jednego znaku powoduje niezgodność nowo zmodyfikowanego pliku ze starym hashem.[1]

Przykład:

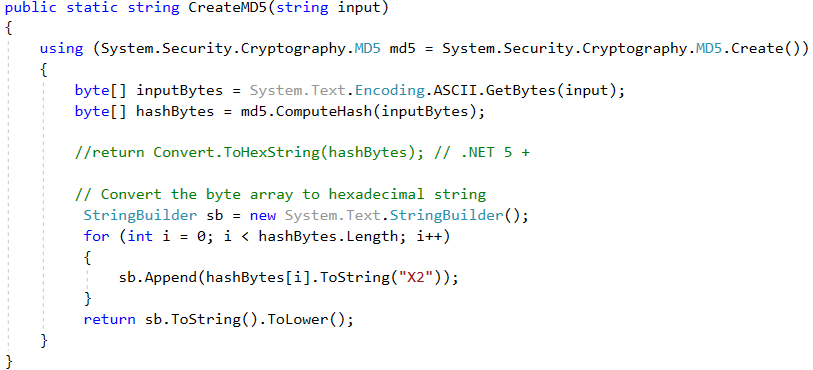
Po zastosowaniu funkcji skrótu MD5 na haśle **AlaMaKota1234!** otrzymamy wartość wyjściową w postaci ciągu znaków ***e3f52f31c9bd800b9724ced17ba8be96****.*

* 1. **MD5**

Mimo popularności MD5 nie jest już uważane za funkcję w pełni bezpieczną – opracowano wiele technik znajdujących kolizję tej funkcji. Jedną z nich jest metoda *MD5 Tunneling* znajdująca kolizję w czasie poniżej 10 sekund, wykorzystując do tego moc laptopa przeciętnej klasy.

Odkrycia tego rodzaju oczywiście są ciekawe, ale w praktyce nie wpływają znacznie na bezpieczeństwo hashy MD5. Większość technik znajdywania kolizji podaje na wejściu MD5 dane binarne, a więc dane ze zdecydowanie szerszego zakresu niż znaki ASCII.

Chociaż dzisiaj nie istnieją efektywne metody znajdujące kolizje w hashach MD5 haseł, odradza się stosowanie tej funkcji. Istnieją realne przesłanki wskazujące, że sytuacja ta zmieni się w najbliższych latach.



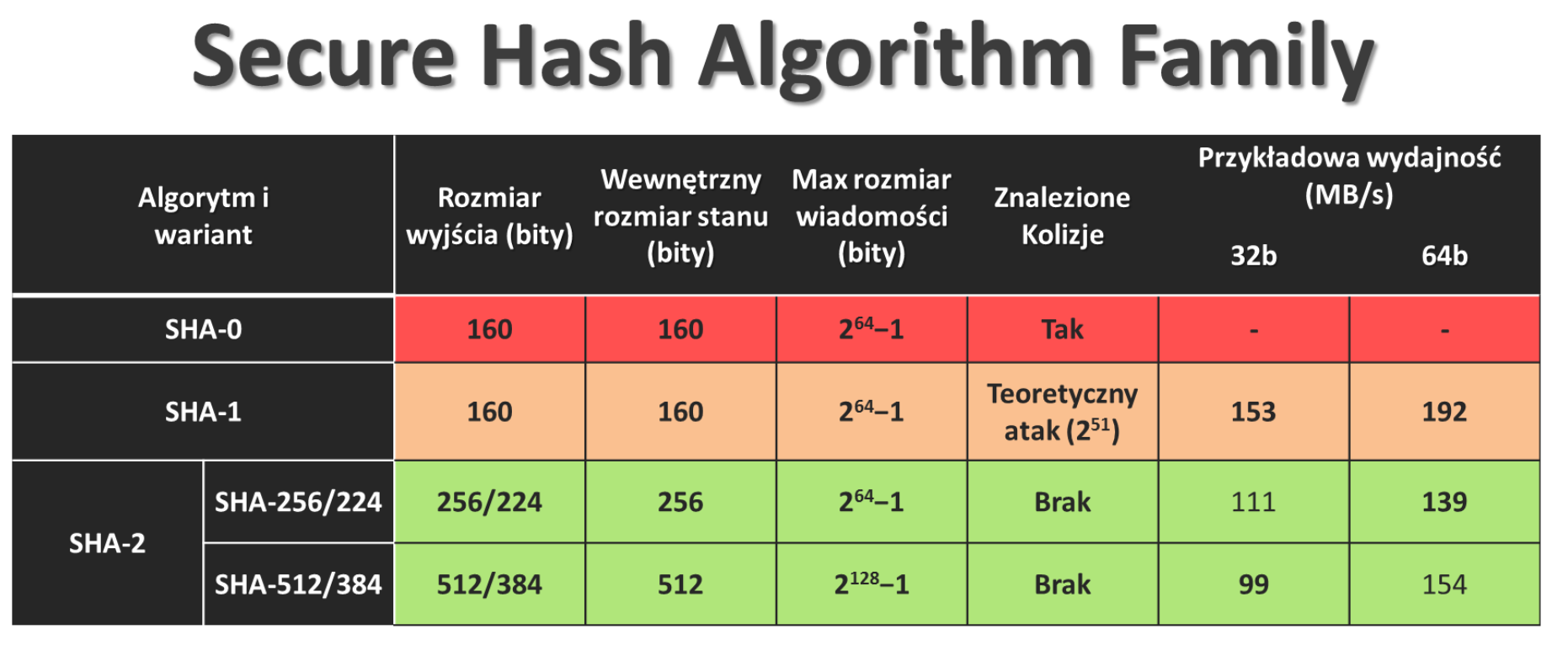
Rys. 2.2.1 MD5.

* 1. **SHA.**

SHA1 jest obecnie największym konkurentem MD5. Istnieje kilka ataków teoretycznych na tę funkcję, jednak w praktyce nie zagrażają one jeszcze bezpieczeństwu hashy. Jednak tego rodzaju sytuacje sugerują, że warto zainteresować się nowszą wersją algorytmu.

SHA2 jest w tej chwili jedną z najbezpieczniejszych wersji algorytmów rodziny SHA. SHA2 istnieje w czterech odmianach: SHA-224, SHA-256, SHA-384 oraz SHA-512. Wszystkie warianty działają w podobny sposób, jednak używają struktur danych o różnej wielkości oraz zwracają hash różnej długości.Niestety funkcje z tej grupy nie są często stosowane, ponieważ ich użycie często wymaga kompilacji dodatkowych modułów (np. do baz danych). Dodatkowym powodem było też oczekiwanie na finalną wersję algorytmu SHA3, który używa nowego podejścia – więc teoretycznie ataki na wcześniejsze wersje SHA nie powinny wpływać na jego bezpieczeństwo. Niestety – dalej jest to algorytm bardzo szybki.

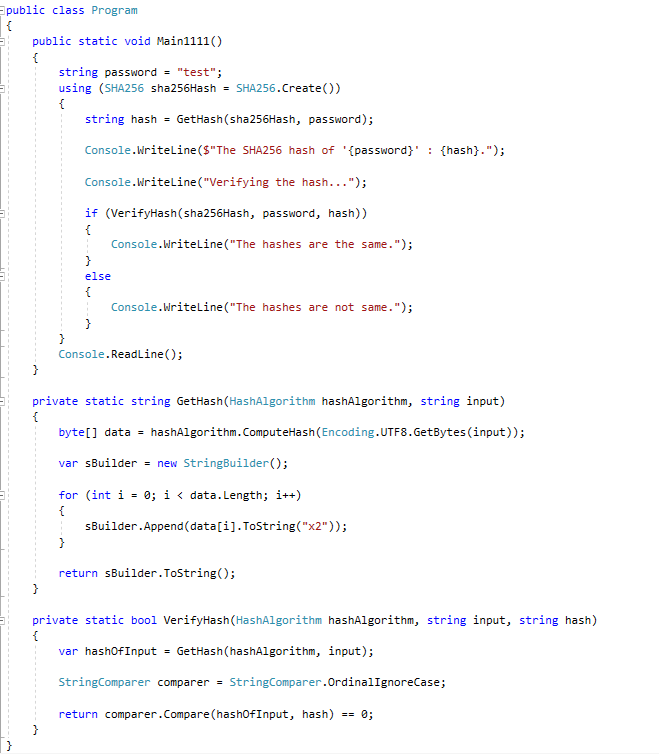
Różnice między algorytmami z rodziny SHA przedstawia poniższa tabela:



Rys. 2.3.1 SHA.

Poniżej przedstawiono obrazowy przykład działania programu ze strony *https://sha256algorithm.com* z wykorzystaniem tego sposobu haszowania (w wersji SHA-256) dla wygenerowania unikalnego hasza do hasła **AlaMaKota1234!** .

Rys. 2.3.2 SHA.[3]



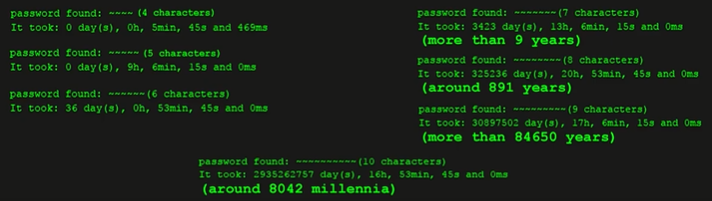
Rys. 2.3.3 Przykładowa implementacja SHA-256.

* 1. **Hashowanie solone.**

Hash "solony" to technika używana w kryptografii i informatyce, która polega na dodaniu losowej wartości (tzw. "soli") do hasła przed jego haszowaniem. W procesie solenia, losowa wartość jest dodawana do hasła, a następnie całość jest poddawana funkcji skrótu (hash), takiej jak np. SHA-256. Wynikowy skrót jest zapisywany w bazie danych razem z wartością soli, która była użyta do solenia hasła. Dzięki temu, nawet jeśli dwa użytkownicy używają tego samego hasła, to ich solone hasła w bazie danych będą się od siebie różnić, ponieważ soli użyte do ich solenia będą różne. Dzięki temu, złamanie jednego hasła nie oznacza złamania wszystkich solonych haseł w bazie danych. Haszowanie solone jest jednym z podstawowych sposobów przechowywania haseł i jest stosowane w wielu systemach logowania i uwierzytelniania, w celu zwiększenia bezpieczeństwa przechowywanych haseł.

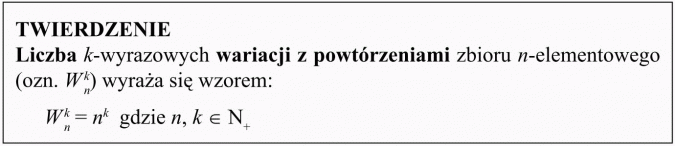
1. **Postępy w projekcie.**
   1. **Wydajność Brute Force ?**

Tempo łamania hasła zależy od wielu czynników, takich jak długość hasła, użyte znaki, algorytm szyfrowania, moc obliczeniowa używanego sprzętu, itp. Oczywiście, im dłuższe i bardziej złożone hasło, tym dłużej będzie trwać proces łamania za pomocą brute force. Dla przykładu, hasło składające się z 6 małych liter (26 możliwości na każdej pozycji) może zostać złamane w ciągu kilku minut na zwykłym komputerze. Natomiast, hasło składające się z 12 znaków (liczby, litery, znaki specjalne) może wymagać lat pracy dla najpotężniejszych superkomputerów.

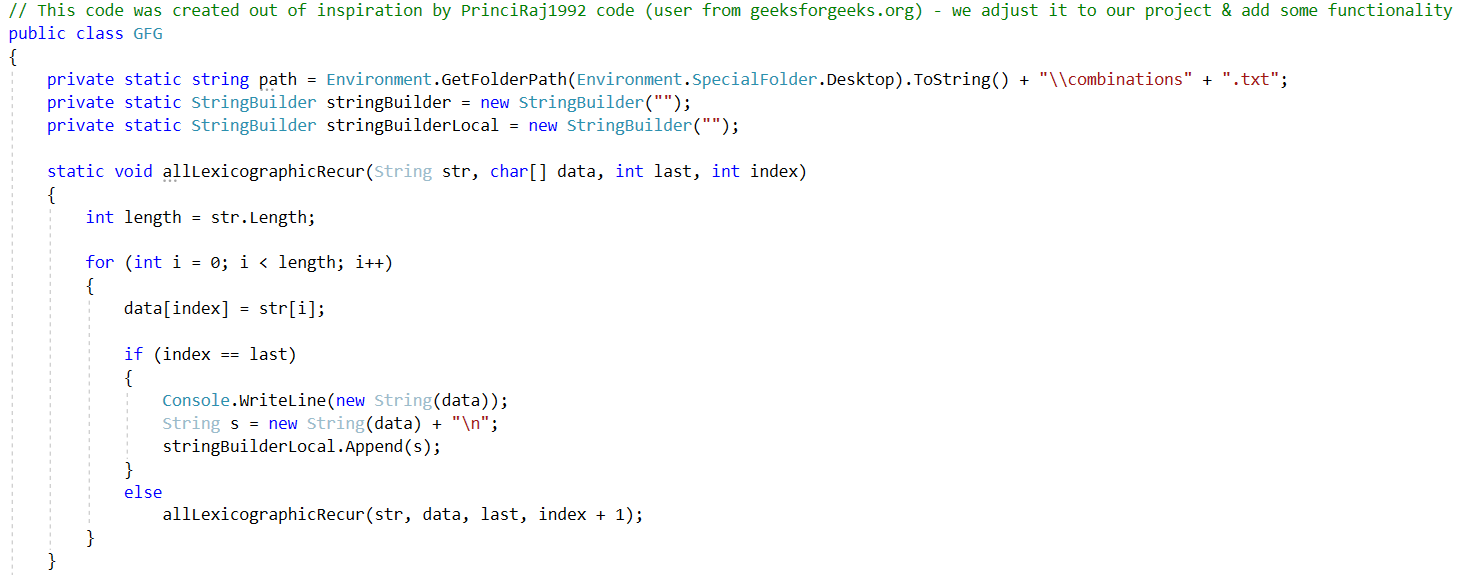


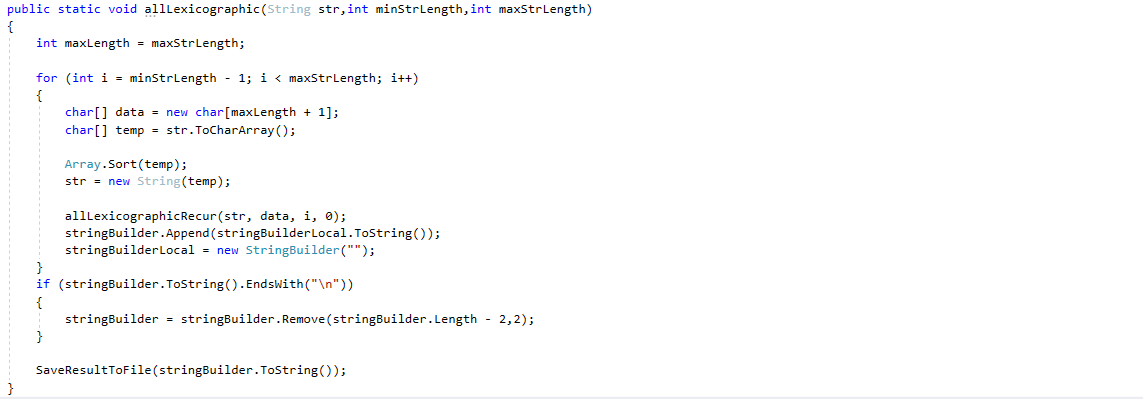
Rys. przykładowe tempo łamania haseł

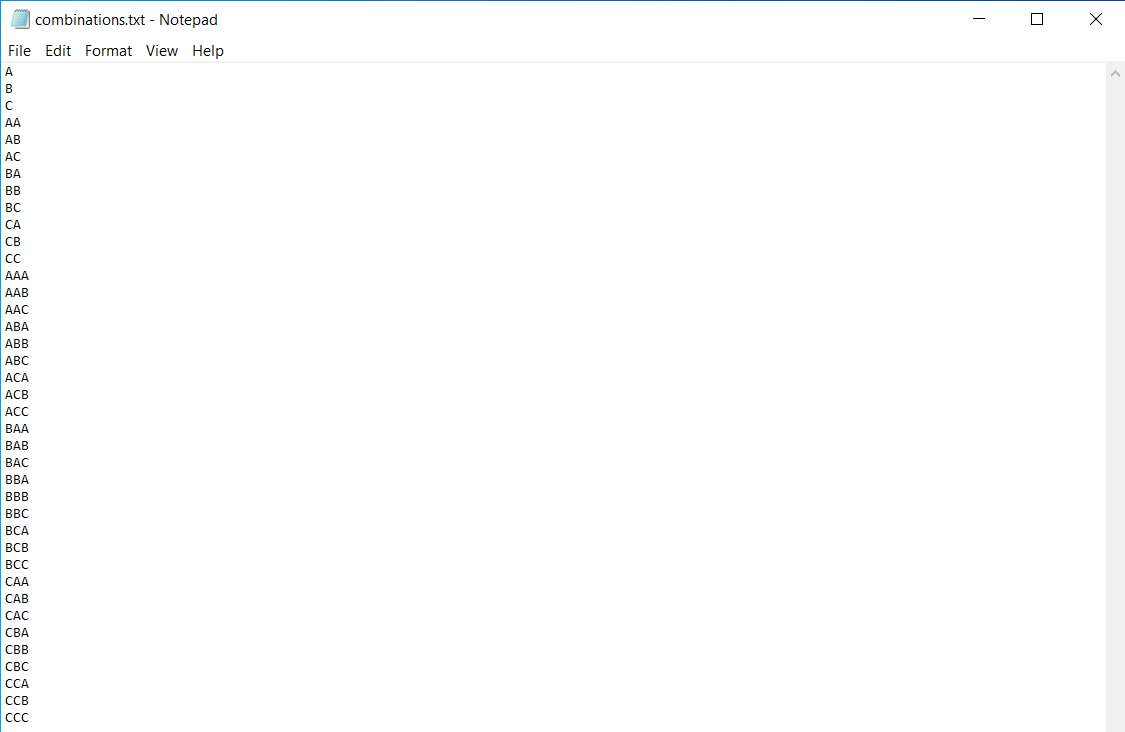
**3**[**.2. Generowanie możliwych haseł na podstawie słownika.**](#_heading=h.gjdgxs)

Wariacje z powtórzeniami to matematyczna technika, która polega na wyznaczaniu różnych możliwych permutacji elementów, gdzie elementy te mogą się powtarzać. Oznacza to, że wariacje z powtórzeniami umożliwiają nam wyboru tego samego elementu więcej niż raz, co odróżnia je od wariacji bez powtórzeń, gdzie każdy element musi zostać wykorzystany tylko raz.

Rys. Wzór na wariacje z powtórzeniami wraz z twierdzeniem[5]

Implementacja generowania wszystkich możliwych haseł na podstawie danego słownika przedstawia się następująco:



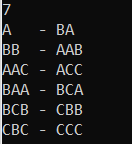


\

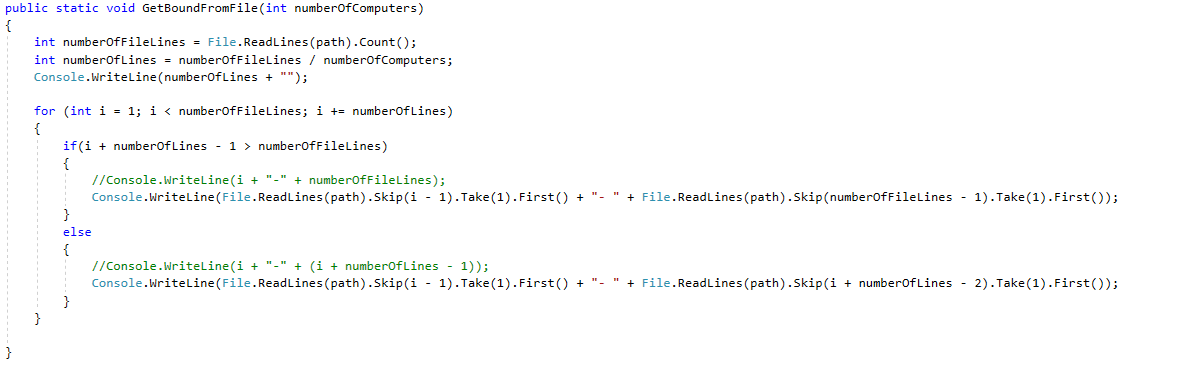
Rys. Wyniki generowania możliwych kombinacji.

Funkcja została zaimplementowana pod kątem używania różnych długości generowanych ciągu znaków - można je ustawiać za pomocą odpowiednio zdefiniowanych parametrów (*minStrLength,maxStrLength*). Po wygenerowaniu wszystkich możliwych kombinacji, wynik zostaje zapisany do pliku tekstowego.

**3**[**.3. Wyznaczanie granic.**](#_heading=h.gjdgxs)

Aby obciążenie wszystkich komputerów w rozproszonej wersji projektu było równomierne potrzebowaliśmy odpowiedniej funkcji obliczającej oraz wyznaczającej granice wygenerowanych możliwych kombinacji, aby móc je rozsyłać do pozostałych komputerów(potencjalnych klientów).

Rys. Wyniki sugerowanych granic.



Rys. Wyniki generowania możliwych kombinacji.

**3**[**.4. Popularne hasła.**](#_heading=h.gjdgxs)

RockYou.txt to nazwa pliku tekstowego zawierającego ponad 32 miliony haseł, które zostały skradzione z serwisu społecznościowego RockYou w 2009 roku. RockYou był popularnym serwisem oferującym narzędzia do gier społecznościowych i aplikacji na Facebooku, MySpace i innych platformach. Jednak w grudniu 2009 roku serwis został zhakowany, a hakerzy ukradli dane logowania i hasła do ponad 32 milionów kont użytkowników. Hakerzy wykorzystali technikę ataku słownikowego, korzystając z dużej bazy haseł, w tym właśnie z pliku RockYou.txt. Plik ten zawierał ogromną liczbę haseł, które były używane przez użytkowników RockYou, a także przez innych użytkowników internetu na całym świecie. Hakerzy wykorzystali te hasła, aby uzyskać nieuprawniony dostęp do kont użytkowników RockYou oraz innych serwisów, na których ci użytkownicy mieli konta. RockYou.txt stał się jednym z największych i najbardziej popularnych plików z hasłami, wykorzystywanych przez hakerów na całym świecie. Plik ten stał się również jednym z najważniejszych narzędzi dla specjalistów ds. bezpieczeństwa w celu testowania zabezpieczeń systemów informatycznych i sieciowych. Dzięki RockYou.txt specjaliści mogą testować skuteczność swoich systemów zabezpieczeń i zapobiegać atakom hakerskim. Można go pobrać m.in z Githuba lub innych ogólnodostępnych stron.[6]

**Literatura**

1. Haszowanie

*https://bezpieczny.blog/co-to-jest-hash/*

1. SHA

[*https://sekurak.pl/kompendium-bezpieczenstwa-hasel-atak-i-obrona/#:~:text=SHA2%20istnieje%20w%20czterech%20odmianach,-384%20oraz%20SHA-512*](https://sekurak.pl/kompendium-bezpieczenstwa-hasel-atak-i-obrona/#:~:text=SHA2%20istnieje%20w%20czterech%20odmianach,-384%20oraz%20SHA-512)*.*

1. SHA

[*https://sha256algorithm.com*](https://sha256algorithm.com)

1. Hashowanie

*http://wojciech.bozejko.staff.iiar.pwr.wroc.pl/elearning/Wyk5\_tabl\_hash.pdf*

1. Twierdzenie wariacji z powtórzeniami

[*https://opracowania.pl/opracowania/matematyka/wariacje-z-powtorzeniami,oid,2046*](https://opracowania.pl/opracowania/matematyka/wariacje-z-powtorzeniami,oid,2046)

1. Opis rockyou.txt

*https://en.wikipedia.org/wiki/RockYou*

Politechnika Świętokrzyska

Wydział Elektrotechniki, Automatyki i Informatyki

Zespół:

**Kot Jarosław**

**Prusicki Jakub**

**Łamacz haseł**

Projekt zespołowy

na studiach stacjonarnych

o kierunku **Informatyka**

Stopień **II**

Opiekun projektu:

**dr inż. Paweł Paduch**

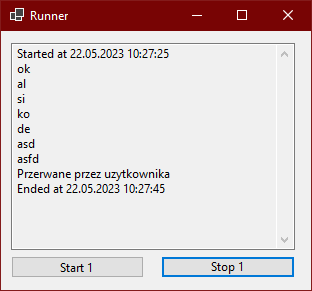
Kielce, 2023

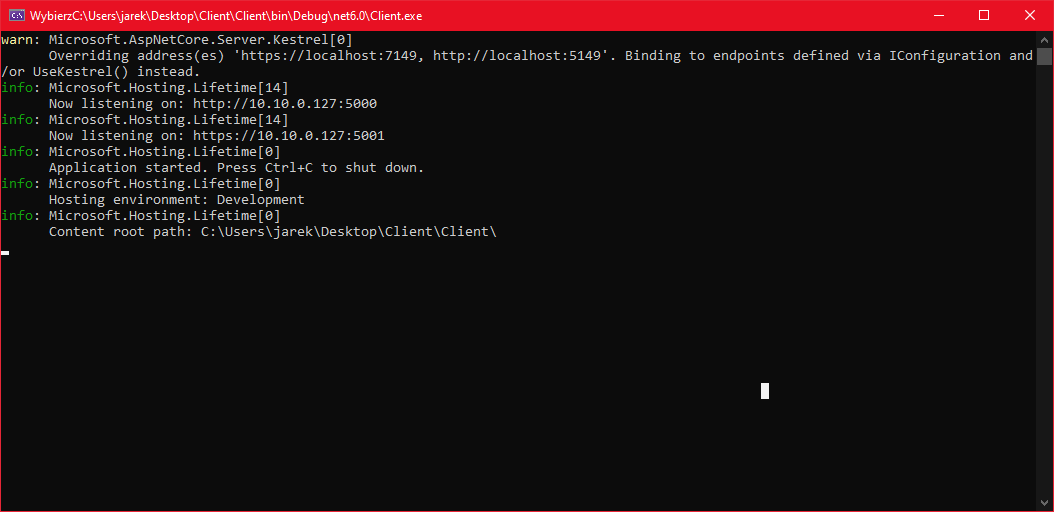
**SPIS TREŚCI**

[1. Aplikacja - Łamacz Haseł. 3](#_heading=h.gjdgxs)

.

1. **Aplikacja - Łamacz Haseł**

Aplikacja “Łamacz Haseł” stworzona w języku C# ma za zadanie łamania haseł za pomocą serwera wysyłającego hasła oraz klientów, którzy łamią te hasła. Program dzieli się na dwie części “Serwer” oraz “Klient”. Pierwsza część służy do obsługiwania klientów oraz haseł. Metoda BtnStart1\_Clickjest wywoływana po kliknięciu przycisku **"Start"**. Czyści pole tekstowe textBox1, a następnie dodaje wiadomość o rozpoczęciu operacji oraz odczytuje zawartość pliku "hasla.txt".

Następnie zawartość pliku jest podzielona na bloki i przekazana do metody CallToClients w celu wywołania żądań HTTP na serwerze. Wykorzystywana jest klasa Task do tworzenia zadań asynchronicznych, a metoda Parallel.ForEach jest używana do uruchomienia zadań równolegle. Na końcu oczekuje się na zakończenie wszystkich zadań asynchronicznych i wyświetla wiadomość o zakończeniu operacji. Metoda CallToClients wykonuje żądanie HTTP na podany adres URL. Tworzony jest klient HttpClient z zdefiniowanym obsługiwaniem certyfikatów klienta i serwera. Następnie wywoływane jest żądanie GET na podany adres URL, a odpowiedź jest odczytywana asynchronicznie. Jeśli odpowiedź jest udana (kod statusu 200), odczytywana jest treść odpowiedzi. Następnie odpowiedź i URL są dodawane do słownika Catalog. Kolejnym krokiem jest przygotowanie danych do żądania POST, które są zserializowane do formatu JSON. Nagłówki żądania są ustawiane na wartość akceptującą dane w formacie JSON. Następnie wywoływane jest żądanie POST na podany adres URL, a odpowiedź jest odczytywana asynchronicznie. Jeśli odpowiedź jest udana, deserializowany jest obiekt JSON do klasy Model, a wartość Message jest dodawana do pola tekstowego textBox1. Metoda BtnStop1\_Click jest wywoływana po kliknięciu przycisku "Stop". Wykonuje żądanie POST na wszystkich adresach URL zapisanych w słowniku Catalog w celu zakończenia działania klientów.

Obraz.6.5 przykładowy wygląd po uruchomieniu klienta.

Druga część aplikacji to klient w którym łamane są hasła za pomocą “BruteForce“. Dzieli się on na “Service” oraz “Controller”. Klasa BruteForceService posiada pole m\_TokensCatalog, które jest słownikiem przechowującym identyfikatory żetonów (tokenów) generowane za pomocą klasy Guid oraz odpowiadające im obiekty CancellationTokenSource. Metoda GetToken() generuje nowy żeton (token) za pomocą klasy Guid, tworzy nowy obiekt CancellationTokenSource, dodaje żeton i odpowiadający mu obiekt CancellationTokenSource do słownika m\_TokensCatalog i zwraca wygenerowany żeton jako wartość typu string. Metoda BruteForce(string[] passwords, string guid) przyjmuje tablicę haseł do przeprowadzenia ataku brute force oraz identyfikator żetonu (tokenu) jako parametry. Na podstawie identyfikatora żetonu, metoda pobiera odpowiadający mu obiekt CancellationToken z m\_TokensCatalog i wywołuje metodę BruteForce(string[] passwords, CancellationToken cancellationToken). Metoda BruteForce(string[] passwords, CancellationToken cancellationToken) wykonuje atak brute force na podane hasła. Iteruje przez tablicę haseł, sprawdzając jednocześnie, czy żądane jest przerwanie ataku poprzez sprawdzenie wartości IsCancellationRequested obiektu CancellationToken. Jeśli żądane jest przerwanie, metoda zwraca informację o przerwaniu. W przeciwnym razie, metoda kontynuuje atak i zwraca znalezione hasła w postaci tekstu. Metoda Cancel(string guid) przyjmuje identyfikator żetonu (tokenu) jako parametr. Na podstawie tego identyfikatora, metoda pobiera odpowiadający mu obiekt CancellationTokenSource z m\_TokensCatalog i wywołuje metodę Cancel() dla tego obiektu, co powoduje przerwanie ataku brute force. GetEnumerator() ta metoda implementuje interfejs IEnumerable i zwraca enumerator, który umożliwia iterację po obiektach klasy BruteforceIter. W tej metodzie następuje generowanie permutacji ciągu alphabet dla określonego zakresu wartości od \_min do \_max. Oblicza ciąg znaków a na podstawie wartości l i power. Wartość l jest dzielona przez długość alphabet, a reszta wskazuje na kolejne znaki w alphabet. Metoda zwraca wynik jako ciąg znaków. SaveResultToFile(string text) metoda zapisuje podany tekst do pliku "passwords2.txt" na pulpicie. Jeśli plik nie istnieje, zostaje utworzony, a tekst jest zapisywany jako nowa linia. Jeśli plik już istnieje, tekst jest dołączany jako nowa linia do istniejącej zawartości pliku. W kontrolerze natomiast mamy udostępnianie trzech akcje, które obsługują żądania HTTP związane z operacjami bruteforce. Wykorzystuje serwis IBruteForceService do wykonywania tych operacji i zwraca odpowiednie wyniki jako odpowiedzi HTTP. Dodatkowo w projekcie “Client” dość istotny jest plik appsettings.json w którym to wpisujemy nasze ip w celu nasłuchiwania serwera.

Politechnika Świętokrzyska

Wydział Elektrotechniki, Automatyki i Informatyki

Zespół:

**Kot Jarosław**

**Prusicki Jakub**

**Łamacz haseł**

Projekt zespołowy

na studiach stacjonarnych

o kierunku **Informatyka**

Stopień **II**

Opiekun projektu:

**dr inż. Paweł Paduch**

Kielce, 2023

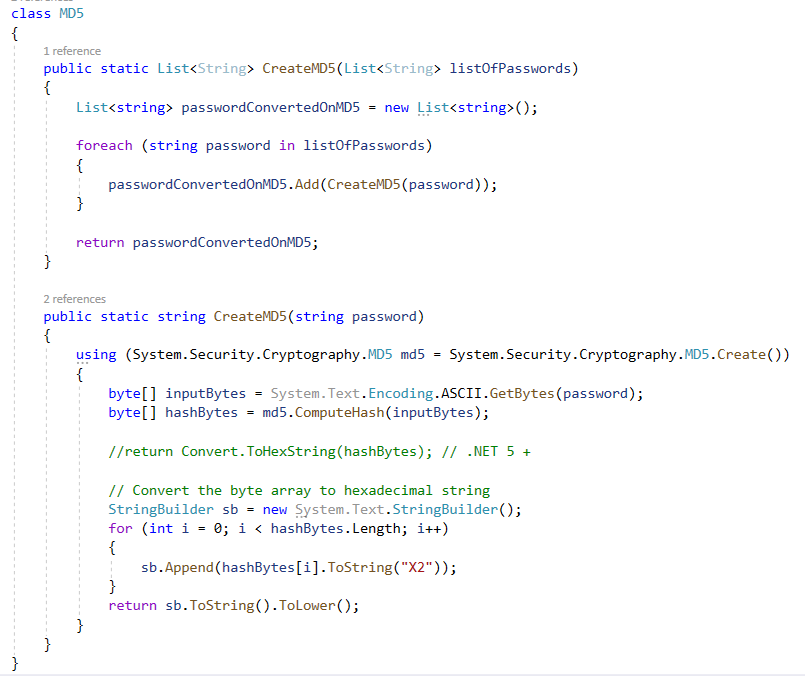
**SPIS TREŚCI**

[1. Haszowanie listy haseł. 3](#_heading=h.gjdgxs)

2[. Generowanie permutacji na podstawie podanego zakresu.](#_heading=h.gjdgxs) 5

1. **Haszowanie listy haseł.**

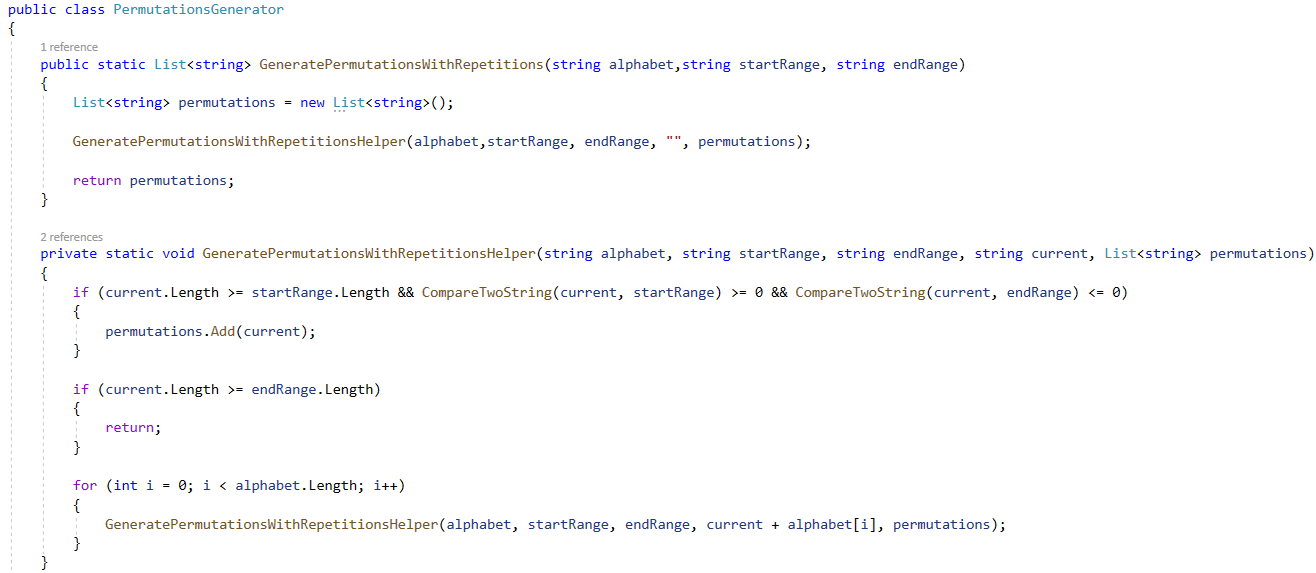
Do zrealizowania projektu zdecydowano się na skorzystanie z algorytmu haszującego MD5. Pomimo mniejszego bezpieczeństwa w porównaniu do innych algorytmów, które nie jest kluczowe w tym projekcie zyskał przewagę pod względem szybkości obliczeniowej oraz prostoty - zarówno w zrozumieniu jak i samej implementacji. Ma niewielką liczbę kroków i prostą strukturę, co czyni go łatwym do użycia.



Rys. 1. Klasa MD5.

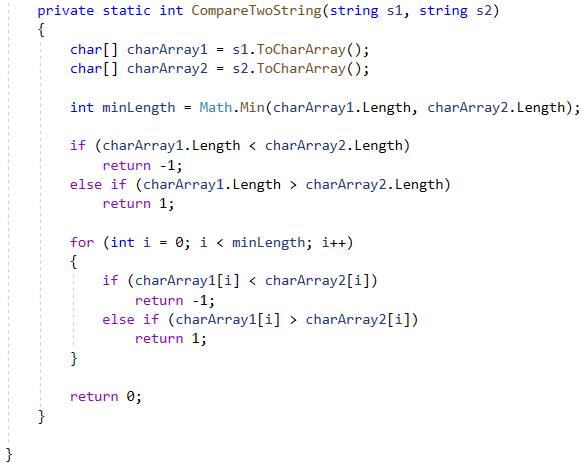
Na podstawie działającego już kodu (sprawozdanie dot. tematu nr 4) został rozbudowany kod, który na tą chwilę umożliwia hashowanie już hashowanie listy haseł.

**2. Generowanie permutacji na podstawie zakresu.**

Kiedy algorytm do generowania wszystkich możliwości (permutacji z powtórzeniami) haseł na podstawie zadanego alfabetu był już gotowy, pojawiło się pytanie czy można to bardziej zoptymalizować aby nie wykonywać niepotrzebnych obliczeń. W związku z rozproszeniem obliczeń zdecydowano się generować tylko odpowiednią część możliwości. Aby to wykonać postanowiono wprowadzić ograniczenia poprzez podawanie zakresów w postaci początkowej i końcowej możliwości wygenerowanego hasła na podstawie alfabetu.

Rys. 2.1 Generowanie permutacji z zakresu cz1.

Podczas wstępnych założeń nie przewidywano większych trudności, jendka te pojawiły się dopiero w fazie testów. Okazało się że program działa dla zakresów, jednak nie pokazuje wszystkich wyników. Problemem była sama funkcja porównująca (**string.Compare**) , która zapewne była napisana z myślą o innych porównaniach. Jej działanie polega na kolejnym porównywaniu kolejnych odpowiadających liter w tych napisach. Problem jednak pojawia się kiedy podamy zakres, który łamie tę logikę choćby poprzez długość napisów np. BB - AAB. W takim przypadku napis AAB zostanie uznany za ‘mniejszy’, ponieważ już po porównaniu pierwszych liter program uznał, że skoro B > A to BB musi być większe od AAB. Poniżej przedstawiono funkcję porównującą napisy z uwzględnieniem długości ciągu znaków, wykorzystaną w tym projekcie (w funkcji generującej możliwe hasła).



Rys. 2.1 Generowanie permutacji z zakresu cz.2 - funkcja porównująca.

Politechnika Świętokrzyska

Wydział Elektrotechniki, Automatyki i Informatyki

Zespół:

**Kot Jarosław**

**Prusicki Jakub**

**Łamacz haseł**

Projekt zespołowy

na studiach stacjonarnych

o kierunku **Informatyka**

Stopień **II**

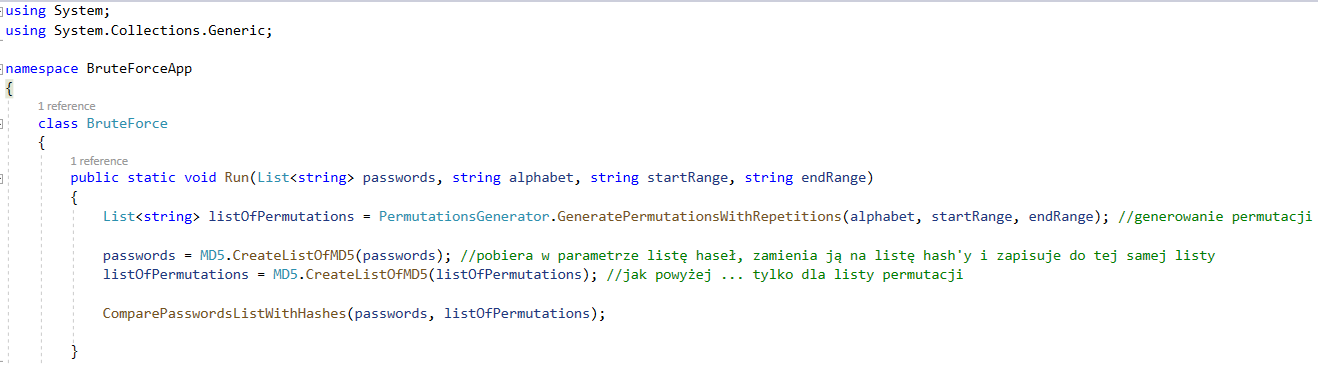
Opiekun projektu:

**dr inż. Paweł Paduch**

Kielce, 2023

1. **Nowy algorytm Brute Force.**

Aby spełnić założenia projektowe niezbędnym było odpowiednie zaprojektowanie algorytmu BruteForce, który potrafiłby łamać kolejne hasła. Program powinien działać na podstawie porównywania kolejnych haseł (otrzymanych np. w postaci listy) od serwera, a następnie porównywać je z wszystkimi możliwymi permutacjami wygenerowanymi na podstawie zadanego alfabetu oraz zakresów.





Rys. 1. Nowa klasa BruteForce.

Warto podkreślić, że poprzednie wersje generowały możliwe permutacje na podstawie całego alfabetu, a nie tak jak teraz tylko dzięki odpowiednim zakresom. Pozwala to na odciążenie komputerów z wykonywania zbędnej pracy oraz rozłożenie jej na pozostałe komputery. Dzięki temu program staje się bardziej elastyczny i działa na lepszym poziomie optymalizacji. Jeśli chodzi o postępy w tworzonym oprogramowaniu, to nie można pominąć faktu iż program został wdrożony do wersji z rozproszeniem sieciowym (Web-API).