

MMN13

Programming assignment

Bloom filters & password strength

version 1.0

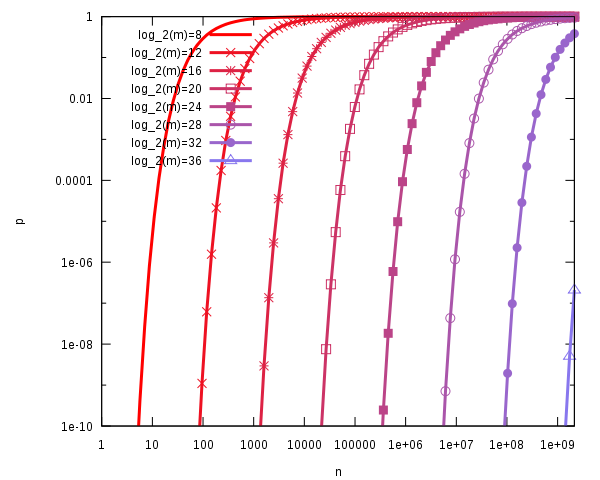


Figure1 - The false positive probability P   
as a function of number of elements N in the filter and the filter size M.   
An optimal number of hash functions k=(m/n)ln 2 has been assum

Table of contents

[Scope 2](#_Toc515903255)

[Background 2](#_Toc515903256)

[bloom filters 2](#_Toc515903257)

[Password strength 2](#_Toc515903258)

[Definition & Requirements 3](#_Toc515903259)

[Application 3](#_Toc515903260)

[password strength rules 4](#_Toc515903261)

[GUI design requirments 5](#_Toc515903262)

[Technologies 6](#_Toc515903263)

[Application & Code Structure 6](#_Toc515903264)

[Password Checker Class diagram 6](#_Toc515903265)

[Bloom Filter and Helper Hahser – Class diagram 7](#_Toc515903266)

[GUI design requirments 7](#_Toc515903267)

# Scope

This document defines the details of a proposed programming assignment for MMN 13 in course 22923, semester 2018B.

The document will explain about Bloom filters, and password strength check, and will propose a programming assignment that shall demonstrate the understanding of the learnt material as well as the implementation of that knowledge in a full scale application.

**NOTE:** *Bloom filters are used extensively in the field of* ***Big Data*** *to improve performanc. They are found in Google BigTable, Apache HBase, Cassandra, PostgreSQL and many more.*

# Background

## bloom filters

A Bloom filter is a space-efficient probabilistic data structure, conceived by Burton Howard Bloom in 1970, that is used to test whether an element is a member of a set. False positive matches are possible, but false negatives are not – in other words, a query returns either "possibly in set" or "definitely not in set". Elements can be added to the set, but not removed (though this can be addressed with a "counting" filter); the more elements that are added to the set, the larger the probability of false positives.

Bloom proposed the technique for applications where the amount of source data would require an impractically large amount of memory if "conventional" error-free hashing techniques were applied. He gave the example of a hyphenation algorithm for a dictionary of 500,000 words, out of which 90% follow simple hyphenation rules, but the remaining 10% require expensive disk accesses to retrieve specific hyphenation patterns. With sufficient core memory, an error-free hash could be used to eliminate all unnecessary disk accesses; on the other hand, with limited core memory, Bloom's technique uses a smaller hash area but still eliminates most unnecessary accesses. For example, a hash area only 15% of the size needed by an ideal error-free hash still eliminates 85% of the disk accesses.

More generally, fewer than 10 bits per element are required for a 1% false positive probability, independent of the size or number of elements in the set.

## Password strength

Password strength is a measure of the effectiveness of a password against guessing or brute-force attacks. In its usual form, it estimates how many trials an attacker who does not have direct access to the password would need, on average, to guess it correctly. The strength of a password is a function of length, complexity, and unpredictability.

Using strong passwords lowers overall risk of a security breach, but strong passwords do not replace the need for other effective security controls. The effectiveness of a password of a given strength is strongly determined by the design and implementation of the factors (knowledge, ownership, inherence). The first factor is the main focus in this article.

The rate at which an attacker can submit guessed passwords to the system is a key factor in determining system security. Some systems impose a time-out of several seconds after a small number (e.g. three) of failed password entry attempts. In the absence of other vulnerabilities, such systems can be effectively secured with relatively simple passwords. However the system must store information about the user passwords in some form and if that information is stolen, say by breaching system security, the user passwords can be at risk.

# Definition & Requirements

## Application

The application shall include include 3 projects:

* WPF GUI (XAML based)
* Bloom filter Utils
* Password Strength Utils

The application shall:

* Allow selection of bloom filter parameters
  + N – size of the bit array
  + K – the number of the hash functions to be used by the filter
* Allow reset of bloom filter with the inserted parameters
* Show current status of the bloom filter
  + Show number of passwords inserted
  + Show characteristics of the filter (i.e. N, K)
* Visualize the filter of reasonably small filters (several thousands of values in the bit array)
* Allow insertion of a new password to the filter,
  + Indicate the number of collisions, in other words, how many hash functions resulted in values that were already in the filter.  
    **NOTE:** *if all hash values were in the filter, the result is a* ***false positive!***
* Allow a check if an arbitrary password is present in the filter
  + Prove that if a password was previously set, it will indicate positive
  + Prove that a negative answer is necessarily an indication that this specific password was never set before
  + Allow simple illustration of getting a "false-positive" by allowing a visual display of a filter with small bit array, such that collision occur quickly
* For relatively small filter (several thousands) show the progress of the bit array in runtime, by a display that will show the "lit" bits"
* Provide sanity check on values of N, K (i.e. that N will not be less than K, that both are positive integers etc)

## password strength rules

Compute a "score" for a given password using the following rules:

**Top Requirement** (will also be used in the calculation of the score, see below table for details):

* Password Length >= 8
* Contains Uppercase Letters (A-Z)
* Contains Lowercase Letters (a-z)
* Contains Digits (0-9)
* Contains Symbols (in C# - Char.IsSymbol(ch) or Char.IsPunctuation(ch))

**Penalties** (that will cause reductions in score):

|  |  |
| --- | --- |
| Condition | Action |
| IF Password is all letters | **Reduce** Password length |
| IF Password is all digits | **Reduce** Password length |
| IF Password has repeated characters | **Reduce** (Number of repeated characters \* (Number of repeated characters -1)) |
| IF Password has consecutive uppercase letters | **Reduce** (Number of consecutive uppercase characters \* 2) |
| IF Password has consecutive lowercase letters | **Reduce** (Number of consecutive lowercase characters \* 2) |
| IF Password has consecutive digits | **Reduce** (Number of consecutive digits \* 2) |
| IF Password has sequential letters | **Reduce** (Number of sequential letters \* 3) E.g.: ABCD or DCBA. |
| IF Password has sequential digits | **Reduce** (Number of sequential digits \* 3) E.g.: 1234 or 4321. |

**Strenghts** (what will add to the score):

|  |  |
| --- | --- |
| Condition | Action |
| Length | **Add** length \* 4 |
| Upper Case chars | **Add** Length - Number of Upper Case Letters)\*2 |
| Lower Case chars | **Add** Length - Number of Lower Case Letters)\*2 |
| Digits | **Add** Number of Digits \* 4 |
| Symbols | **Add** Number of Symbols \* 6 |
| Digits & Symbols in the middle | (Number of Digits or Symbols in the Middle of the Password) \* 2 |
| If (Number of Requirements Met > 3) | **Add** Number of Requirements Met \* 2 |

Score value legend :

**Score** == 0 "Very Weak"

20 =< **score** < 40 "Weak"

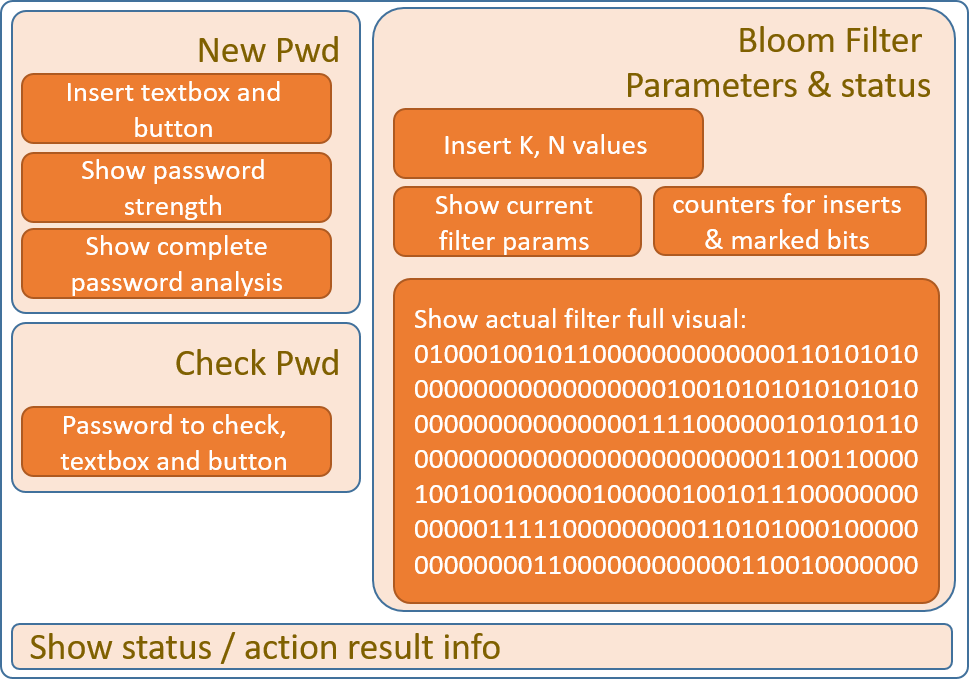
40 =< **score** < 60 "Good"

40 =< **score** < 80 "Strong"

80 =< **score** <= 100 "Very Strong"

## GUI design requirments

The following illustration is the required screen that will provide comfortable user interface that will accommodate for all the above requirements



# Technologies

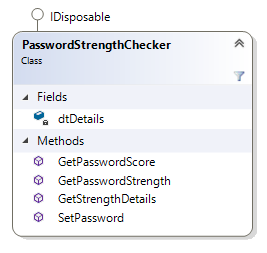
The development shall be accomplished using:

* IDE - Visual Studio (2017)
* Language – C#
* .NET Framework version 4.6.1 (on all projects)
* Graphics – WPF
* Project Management – TFS online, (GIT format)

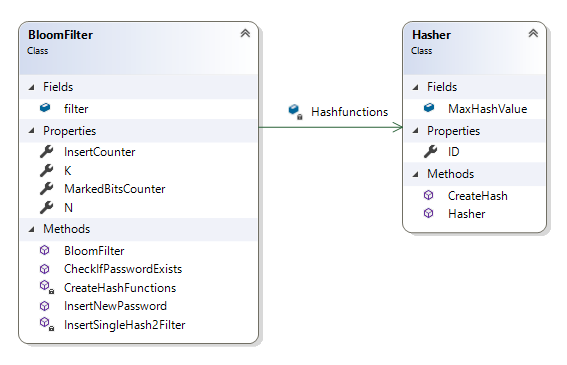
# Application & Code Structure

* Exe containing the WPF project
* DLL of Filter utils
* DLL of Password Strength checker utils

## Password Checker Class diagram



## Bloom Filter and Helper Hahser – Class diagram



## GUI design requirments

