

*DATA STRUCTURES AND ALGORITHM (CSE 220)*

***PROJECT REPORT***

TOPIC: "Campus networking Site"

Presented by:

Prashant Kumar Bhardwaj 14BCE1177

AMULYA 14BCE1002

Angad Sachdave 14BCE1183

**Introduction**

* Cambuzz is a web application product which may be called a campus media or campus networking site.
* All the colleges, schools and every other institute may have their own cambuzz, logged in with their own students and teachers.
* The user who signs up will receive a confirmation mail with a link on his/her VIT mail and the link will further redirect the user to his/her confirmed account.
* Once the user logs in, he will see three sections:
* Buzz
* Track your teacher
* Ask a Question

**Sections**

**1. BUZZ**

* This section contains a buzz form and news feed.
* The user can create a buzz by entering the details about any information, event, guest lecture or workshops in the campus. The users who create the buzz can also set the flags to that buzz, like if it is related to some branch or club.
* The news feed will be filled with the buzzes of all the users and any user can filter the news feed with buzzes of his/her choice.
* Every event will have a starting and ending date and time, the buzz will be deleted after the end time.

**2. Track your Teacher:**

* This section will contain a search box and a track button.
* The user can enter the name of any teacher of his/her campus he wishes to track.
* As the track button is clicked, the user can see that whether that teacher is free or not? If he is free, till what time will he be free? And if he has a class, which class he is in. And at what time will he be free again? He can also check the free slots of the teacher for Monday on any weekend.

**3. Ask a Question:**

* This section will act like a forum for all the members of the institute.
* A user may post any question he wishes to know about, the openly posted question shall be open for all to answer.
* The user may also search for a topic and may get all the question-answers related to that topic.

**How will this act as a product?**

* This web application contains two sides.
* User Side, which will be available and visible to all the users.
* Admin Side, which will be only available and visible to the admin or the head of the institution, where he will have access to all the database and tables of the website. The admin doesn’t need to be from a technical background and a basic knowledge of computers will do the work.
* The admin can change, delete and update things without getting his/her hands on the coding or backend part.

**Algorithms used**

Z algorithm (Linear time pattern searching Algorithm)

This algorithm finds all occurrences of a pattern in a text in linear time. Let length of text be n and of pattern be m, then total time taken is O(m + n) with linear space complexity. Now we can see that both time and space complexity is same as KMP algorithm but this algorithm is Simpler to understand.

In this algorithm, we construct a Z array.

**What is Z Array?**For a string str[0..n-1], Z array is of same length as string. An element Z[i] of Z array stores length of the longest substring starting from str[i] which is also a prefix of str[0..n-1]. The first entry of Z array is meaningless as complete string is always prefix of itself.

**Example:**

Index 0 1 2 3 4 5 6 7 8 9 10 11

Text a a b c a a b x a a a z

Z values X 1 0 0 3 1 0 0 2 2 1 0

**More Examples:**

str = "aaaaaa"

Z[] = {x, 5, 4, 3, 2, 1}

str = "aabaacd"

Z[] = {x, 1, 0, 2, 1, 0, 0}

str = "abababab"

Z[] = {x, 0, 6, 0, 4, 0, 2, 0}

**How is Z array helpful in Searching Pattern in Linear time?**The idea is to concatenate pattern and text, and create a string “P$T” where P is pattern, $ is a special character should not be present in pattern and text, and T is text. Build the Z array for concatenated string. In Z array, if Z value at any point is equal to pattern length, then pattern is present at that point.

Example:

Pattern P = "aab", Text T = "baabaa"

The concatenated string is = "aab$baabaa"

Z array for above concatenated string is {x, 1, 0, 0, 0,

**3**, 1, 0, 2, 1}.

Since length of pattern is 3, the value 3 in Z array

indicates presence of pattern.

**How to construct Z array?**     A Simple Solution is two run two nested loops, the outer loop goes to every index and the inner loop finds length of the longest prefix that matches substring starting at current index. The time complexity of this solution is O(n2).

      We can construct Z array in linear time.

The idea is to maintain an interval [L, R] which is the interval with max R

such that [L,R] is prefix substring (substring which is also prefix).

Steps for maintaining this interval are as follows –

1) If i > R then there is no prefix substring that starts before i and

ends after i, so we reset L and R and compute new [L,R] by comparing

str[0..] to str[i..] and get Z[i] (= R-L+1).

2) If i <= R then let K = i-L, now Z[i] >= min(Z[K], R-i+1) because

str[i..] matches with str[K..] for atleast R-i+1 characters (they are in

[L,R] interval which we know is a prefix substring).

Now two sub cases arise –

a) If Z[K] < R-i+1 then there is no prefix substring starting at

str[i] (otherwise Z[K] would be larger) so Z[i] = Z[K] and

interval [L,R] remains same.

b) If Z[K] >= R-i+1 then it is possible to extend the [L,R] interval

thus we will set L as i and start matching from str[R] onwards and

get new R then we will update interval [L,R] and calculate Z[i] (=R-L+1).

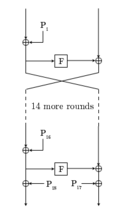
Blowfish encryption

Blowfish is a [symmetric-key](https://en.wikipedia.org/wiki/Symmetric-key_algorithm) [block cipher](https://en.wikipedia.org/wiki/Block_cipher), designed in 1993 by [Bruce Schneier](https://en.wikipedia.org/wiki/Bruce_Schneier) and included in a large number of cipher suites and encryption products. Blowfish provides a good encryption rate in software and no effective [cryptanalysis](https://en.wikipedia.org/wiki/Cryptanalysis) of it has been found to date. However, the [Advanced Encryption Standard](https://en.wikipedia.org/wiki/Advanced_Encryption_Standard) (AES) now receives more attention.

Schneier designed Blowfish as a general-purpose algorithm, intended as an alternative to the aging [DES](https://en.wikipedia.org/wiki/Data_Encryption_Standard) and free of the problems and constraints associated with other algorithms. At the time Blowfish was released, many other designs were proprietary, encumbered by [patents](https://en.wikipedia.org/wiki/Patent) or were commercial or government secrets. Schneier has stated that, "Blowfish is unpatented, and will remain so in all countries. The algorithm is hereby placed in the [public domain](https://en.wikipedia.org/wiki/Public_domain), and can be freely used by anyone."

Notable features of the design include key-dependent [S-boxes](https://en.wikipedia.org/wiki/S-box) and a highly complex [key schedule](https://en.wikipedia.org/wiki/Key_schedule).

Blowfish has a 64-bit [block size](https://en.wikipedia.org/wiki/Block_size_(cryptography)) and a variable [key length](https://en.wikipedia.org/wiki/Key_length) from 32 bits up to 448 bits.[[2]](https://en.wikipedia.org/wiki/Blowfish_(cipher)#cite_note-blowfish-paper-2) It is a 16-round [Feistel cipher](https://en.wikipedia.org/wiki/Feistel_cipher" \o "Feistel cipher) and uses large key-dependent [S-boxes](https://en.wikipedia.org/wiki/Substitution_box). In structure it resembles [CAST-128](https://en.wikipedia.org/wiki/CAST-128), which uses fixed S-boxes.

[](https://en.wikipedia.org/wiki/File:BlowfishDiagram.png)

The Feistel structure of Blowfish

The diagram to the left shows the action of Blowfish. Each line represents 32 bits. The algorithm keeps two subkey arrays: the 18-entry P-array and four 256-entry S-boxes. The S-boxes accept 8-bit input and produce 32-bit output. One entry of the P-array is used every round, and after the final round, each half of the data block is [XORed](https://en.wikipedia.org/wiki/Exclusive_or" \o "Exclusive or) with one of the two remaining unused P-entries.

The diagram to the upper right shows Blowfish's F-function. The function splits the 32-bit input into four eight-bit quarters, and uses the quarters as input to the S-boxes. The outputs are added [modulo](https://en.wikipedia.org/wiki/Modular_arithmetic) 232 and XORed to produce the final 32-bit output.

Decryption is exactly the same as encryption, except that P1, P2,..., P18 are used in the reverse order. This is not so obvious because xor is commutative and associative. A common misconception is to use inverse order of encryption as decryption algorithm (i.e. first XORing P17 and P18 to the ciphertext block, then using the P-entries in reverse order).

Blowfish's [key schedule](https://en.wikipedia.org/wiki/Key_schedule) starts by initializing the P-array and S-boxes with values derived from the [hexadecimal](https://en.wikipedia.org/wiki/Hexadecimal) digits of [pi](https://en.wikipedia.org/wiki/Pi), which contain no obvious pattern (see [nothing up my sleeve number](https://en.wikipedia.org/wiki/Nothing_up_my_sleeve_number)). The secret key is then, byte by byte, cycling the key if necessary, XORed with all the P-entries in order. A 64-bit all-zero block is then encrypted with the algorithm as it stands. The resultant ciphertext replaces P1 and P2. The same ciphertext is then encrypted again with the new subkeys, and the new ciphertext replaces P3 and P4. This continues, replacing the entire P-array and all the S-box entries. In all, the Blowfish encryption algorithm will run 521 times to generate all the subkeys - about 4KB of data is processed.

Because the P-array is 576 bits long, and the key bytes are XORed through all these 576 bits during the initialization, many implementations support key sizes up to 576 bits. While this is certainly possible, the 448 bits limit is here to ensure that every bit of every subkey depends on every bit of the key, as the last four values of the P-array don't affect every bit of the ciphertext. This point should be taken in consideration for implementations with a different number of rounds, as even though it increases security against an exhaustive attack, it weakens the security guaranteed by the algorithm. And given the slow initialization of the cipher with each change of key, it is granted a natural protection against brute-force attacks, which doesn't really justify key sizes longer than 448 bits.

uint32\_t P[18];

uint32\_t S[4][256];

uint32\_t f (uint32\_t x) {

uint32\_t h = S[0][x >> 24] + S[1][x >> 16 & 0xff];

**return** ( h ^ S[2][x >> 8 & 0xff] ) + S[3][x & 0xff];

}

void encrypt (uint32\_t & L, uint32\_t & R) {

**for** (int i=0 ; i<16 ; i += 2) {

L ^= P[i];

R ^= f(L);

R ^= P[i+1];

L ^= f(R);

}

L ^= P[16];

R ^= P[17];

swap (L, R);

}

void decrypt (uint32\_t & L, uint32\_t & R) {

**for** (int i=16 ; i > 0 ; i -= 2) {

L ^= P[i+1];

R ^= f(L);

R ^= P[i];

L ^= f(R);

}

L ^= P[1];

R ^= P[0];

swap (L, R);

}

{

*// ...*

*// initializing the P-array and S-boxes with values derived from pi; omitted in the example*

*// ...*

**for** (int i=0 ; i<18 ; ++i)

P[i] ^= key[i % keylen];

uint32\_t L = 0, R = 0;

**for** (int i=0 ; i<18 ; i+=2) {

encrypt (L, R);

P[i] = L; P[i+1] = R;

}

**for** (int i=0 ; i<4 ; ++i)

**for** (int j=0 ; j<256; j+=2) {

encrypt (L, R);

S[i][j] = L; S[i][j+1] = R;

}

}

## Blowfish in practice

Blowfish is a fast [block cipher](https://en.wikipedia.org/wiki/Block_cipher), except when changing keys. Each new [key](https://en.wikipedia.org/wiki/Key_(cryptography)) requires pre-processing equivalent to encrypting about 4 kilobytes of text, which is very slow compared to other block ciphers. This prevents its use in certain applications, but is not a problem in others.

In one application Blowfish's slow key changing is actually a benefit: the [password](https://en.wikipedia.org/wiki/Password)-hashing method used in [OpenBSD](https://en.wikipedia.org/wiki/OpenBSD" \o "OpenBSD) uses an algorithm derived from Blowfish that makes use of the slow key schedule; the idea is that the extra computational effort required gives protection against [dictionary attacks](https://en.wikipedia.org/wiki/Dictionary_attack). See [key stretching](https://en.wikipedia.org/wiki/Key_stretching).

Blowfish has a memory footprint of just over 4 kilobytes of [RAM](https://en.wikipedia.org/wiki/Random_Access_Memory). This constraint is not a problem even for older desktop and [laptop computers](https://en.wikipedia.org/wiki/Laptop_computers), though it does prevent use in the smallest [embedded systems](https://en.wikipedia.org/wiki/Embedded_systems) such as early [smartcards](https://en.wikipedia.org/wiki/Smartcard).

Blowfish was one of the first secure block ciphers not subject to any patents and therefore freely available for anyone to use. This benefit has contributed to its popularity in cryptographic software.

bcrypt is a cross-platform file encryption utility implementing Blowfish developed in 2002.

**Languages Used**

* HTML5
* CSS
* JavaScript
* PHP
* Mysql
* Ajax
* Jason

**CONCLUSION**

Summing up, this project made us use different functionalities of algorithms and data structures.

Doing this project, we learnt many new languages, ways and solutions for tackling some real life programming problems.

Making a campus networking website was fun and interesting.

At the end we would like to give our sincere thanks to our guide in project, Prof. Ramesh Ragala, and all the other professors, friends who laid their extensive support in the completion of this project.