Лабораторная работа № 4. Функция хеширования (4 часа)

4.1 ЦЕЛЬ РАБОТЫ

Изучение функций хеширования и получение навыков программной реализации алгоритмов хеш-функций.

4.2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Функцией хеширования (хеш-функцией) h называется преобразование данных, переводящее строку M произвольной длины в значение $m=h(M)$ (хеш-образ или дайджест сообщения) некоторой фиксированной длины.

Хорошая хеш-функция должна удовлетворять следующим условиям:

1. Хеш-функция $h(M)$ должна быть чувствительна к любым изменениям входной последовательности M.

2. Хеш-функция $h(M)$ должна применяться к блоку данных любой длины.

3. Хеш-функция $h(M)$ создает выход фиксированной длины.

4. Для данного значения $h(M)$ должно быть невозможным нахождение значения M.

5. Для данного значения $h(M)$ должно быть невозможным нахождение M', такого, что $h(M') = h(M)$.

6. Вычислительно невозможно найти произвольную пару (M_1, M_2) такую, что $h(M_1) = h(M_2)$.

7. Вероятность возникновения ситуации, называемой коллизией, когда для различных входных последовательностей M_1 и M_2 совпадают значения их хеш-образов: $h(M_1) = h(M_2)$, должна быть чрезвычайно мала.

При построении хеш-образа входная последовательность M разбивается на блоки m_i фиксированной длины и обрабатывается поблочно по формуле:

$$H_i = f(H_{i-1}, m_i). \qquad (4.1)$$

Хеш-значение, вычисленное в результате обработки последнего блока сообщения, становится хеш-образом всего сообщения.

В качестве примера рассмотрим упрощенный вариант хеш-функции следующего вида:

$$H_i = (H_{i-1} + m_i)^2 \bmod n, \qquad (4.2)$$

где $n = p \cdot q$, p и q – большие простые числа, H_0 – произвольное начальное значение, m_i – i-й блок сообщения $M = \{m_1, m_2, \ldots, m_k\}$.

Например, вычислим хеш-образ для строки "БГУИР". Для перехода от символов к числовым значениям будем использовать следующее соответствие: 'А' – 1, 'Б' – 2, 'В' – 3, …, 'Я' – 33. Тогда сообщение M примет

вид $M = \{2, 4, 21, 10, 18\}$. Выберем два простых числа $p = 17$ и $q = 19$, тогда модуль $n = 323$. Пусть H_0 будет равен 100. Тогда используя (4.2), получим:

$$H_1 = (H_0+m_1)^2 \bmod n = (100+2)^2 \bmod 323 = 10404 \bmod 323 = 68,$$
$$H_2 = (H_1+m_2)^2 \bmod n = (68+4)^2 \bmod 323 = 5184 \bmod 323 = 16,$$
$$H_3 = (H_2+m_3)^2 \bmod n = (16+21)^2 \bmod 323 = 1369 \bmod 323 = 77,$$
$$H_4 = (H_3+m_4)^2 \bmod n = (77+10)^2 \bmod 323 = 7569 \bmod 323 = 140,$$
$$H_5 = (H_4+m_5)^2 \bmod n = (140+18)^2 \bmod 323 = 24964 \bmod 323 = 93.$$

Таким образом, хеш-образ сообщения "БГУИР" будет $h(M) = H_5 = 93$.

4.2.1 FNV (Fowler–Noll–Vo)

Одна из простых хеш-функций. Разработана Гленом Фаулером, Лондоном Керт Нолом и Фогном Во для общего применения. Не является криптографической хеш-функцией. Разработаны варианты алгоритма для 32-, 64-, 128-, 256-, 512-, и 1024-битных хешей.

Приведем описание модифицированного варианта алгоритма, получившего название FNV1A:

```
public uint FNV1AHash(string input)
{
	const uint FNV_prime = 0x1000193;
	const uint FNV_offset_basic = 0x811C9DC5;

	uint hash = FNV_offset_basic;

	foreach (var item in input)
	{
		char byte_of_data = item;

		hash ^= byte_of_data;
		hash *= FNV_prime;
	}

	return hash;
}
```

Рисунок 1. Алгоритм хеширования FNV1A

Пример:

Входная последовательность: BSUIR
Результат хеширования: 0xFF7CBDF2

4.2.2 JOAAT (Jenkins One At A Time)

Одна из простых хеш-функций. Разработана Бобом Дженкинсом для общего применения. Не является криптографической хеш-функцией.

```
public uint JOAATHash(string input)
{
```

```
        uint hash = 0;

        foreach (var item in input)
        {
                byte byte_of_data = (byte)item;

                hash += byte_of_data;
                hash += hash << 10;
                hash ^= hash >> 6;
        }

        hash += hash << 3;
        hash ^= hash >> 11;
        hash += hash << 15;

        return hash;
}
```

Рисунок 2. Алгоритм хеширования JOAAT

Пример:

Входная последовательность: BSUIR
Результат хеширования: 0x75D6F2C1

4.2.3 PJW-32 (hashpjw)

Одна из простых хеш-функций. Разработана Питером Вэйнбергером для общего применения. Не является криптографической хеш-функцией.

```
public uint PJW32Hash(string input)
{
        uint hash = 0;

        foreach (var item in input)
        {
                byte byte_of_data = (byte)item;

                hash = (hash << 4) + byte_of_data;

                uint h1 = hash & 0xf0000000;

                if (h1 != 0)
                {
                        hash = ((hash ^ (h1 >> 24)) & (0xfffffff));
                }
        }
        return hash;
}
```

Рисунок 3. Алгоритм хеширования PJW-32

Пример:

Входная последовательность: BSUIR
Результат хеширования: 0x004789E2

4.2.4 MurmurHash2

Простая и быстрая хеш-функция. Разработана Остином Эпплби для общего применения. Не является криптографической хеш-функцией.

```
public uint MurmurHash2(string input)
{
	const uint m = 0x5bd1e995;
	const uint seed = 21;
	const int r = 24;

	uint len = (uint)input.Length;

	uint hash = seed ^ len;
	string data = input;

	uint k;
	int currentIndex = 0;

	while (len >= 4)
	{
		k = data[currentIndex];
		k |= (uint)data[currentIndex + 1] << 8;
		k |= (uint)data[currentIndex + 2] << 16;
		k |= (uint)data[currentIndex + 3] << 24;

		k *= m;
		k ^= k >> r;
		k *= m;

		hash *= m;
		hash ^= k;

		currentIndex += 4;
		len -= 4;
	}

	switch (len)
	{
		case 3:
			hash ^= (uint)data[currentIndex + 2] << 16;
		goto case 2;
		case 2:
			hash ^= (uint)data[currentIndex + 1] << 8;
		goto case 1;
		case 1:
			hash ^= data[currentIndex];
			hash *= m;
		break;
	};

	hash ^= hash >> 13;
	hash *= m;
	hash ^= hash >> 15;

	return hash;
}
```

Рисунок 4. Алгоритм хеширования MurmurHash2 (x86, 32 бита)

Пример:

Входная последовательность: BSUIR

Результат хеширования: 0xA91046B8

4.2.5 MurmurHash3

Переработанная версия алгоритма MurmurHash2. Не является криптографической хеш-функцией.

```
public uint MurmurHash3(string input)
{
	const uint seed = 21;

	const uint c1 = 0xcc9e2d51;
	const uint c2 = 0x1b873593;

	int curLength = input.Length;
	int length = curLength;
	uint h1 = seed;
	uint k1 = 0;

	int currentIndex = 0;

	while (curLength >= 4)
	{
		k1 = (uint)(input[currentIndex++]
		| input[currentIndex++] << 8
		| input[currentIndex++] << 16
		| input[currentIndex++] << 24);

		k1 *= c1;
		k1 = rotl32(k1, 15);
		k1 *= c2;

		h1 ^= k1;
		h1 = rotl32(h1, 13);
		h1 = h1 * 5 + 0xe6546b64;

		curLength -= 4;
	}

	if ((curLength & 3) > 0)
	{
		switch (curLength)
		{
			case 3:
				k1 = (uint)(input[currentIndex++]
				| input[currentIndex++] << 8
				| input[currentIndex++] << 16);
			break;
			case 2:
				k1 = (uint)(input[currentIndex++]
				| input[currentIndex++] << 8);
			break;
			case 1:
				k1 = (input[currentIndex++]);
			break;
		};

		k1 *= c1;
		k1 = rotl32(k1, 15);
		k1 *= c2;
		h1 ^= k1;
	}

	h1 ^= (uint)length;
```

```
        h1 ^= h1 >> 16;
        h1 *= 0x85ebca6b;
        h1 ^= h1 >> 13;
        h1 *= 0xc2b2ae35;
        h1 ^= h1 >> 16;


        return h1;
}

//Вспомогательный метод
private static uint rotl32(uint x, byte r)
{
        return (x << r) | (x >> (32 - r));
}
```

Рисунок 5. Алгоритм хеширования MurmurHash3 (x86, 32 бита)

Пример (x86, 32 бита, seed = 21):

Входная последовательность: BSUIR

Результат хеширования: 0xCF31D00D

4.3 ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Изучить теоретический материал по лабораторной работе.
2. Реализовать программное средство вычисления хеш-значения:
    a. вариант №1 - по алгоритму FNV1A.
    b. вариант №2 - по алгоритму JOAAT.
    c. вариант №3 - по алгоритму PJW-32.
    d. вариант №4 - по алгоритму MurmurHash2.
    e. вариант №5 - по алгоритму MurmurHash3.
3. Выбранный алгоритм хеш-функции необходимо реализовать в виде отдельного класса.
4. Проверить работоспособность реализованного алгоритма, использовав контрольные значения (см. пример работы алгоритмов).
5. Реализованное программное средство должно вычислять хеш-значение для произвольного текстового файла (*.txt), размером более 1Kb. Для вычисления хеш-образа сообщения использовать хеш-функцию согласно выбранному варианту.

Пояснение по выбору варианта:

Ваш_Номер_Варианта=(Номер_Вашей_зачетной_книжки) mod 5+1

Например №=123456
(123456) mod 5 + 1 = 1 + 1 = 2