ЭЛЕМЕНТЫ ТЕОРИИ ИНФОРМАЦИИ. ИНФОРМАТИВНОСТЬ ДАННЫХ В РАЗЛИЧНЫХ КОДИРОВКАХ

Выполнил: Горощеня Владислав Сергеевич, 3 курс 4 группа 1 подгруппа

2024

**Ход работы**

Энтропия максимальна при равномерном появлении букв в любом месте сообщения, и появляется интерес её сравнения с фактической. В этом смысле введение понятие избыточности – уменьшения информационной нагрузки на символ вследствие разной вероятности его появления в сообщениях, вычисляемой по формуле:

Дополнительная избыточность при передаче одного и того же сообщения на разном языке обусловлена переносом сообщения из одной среды в другую.

В листинге 2.1 приведена реализация расчёта избыточности:

Листинг 2.1. Расчёт избыточности

|  |
| --- |
| double GetRedunancy(double x, double y) {  return x - y/x;  }  cout << "Redunancy: " << GetRedunancy(HartleyEntropy(myText),ShannonEntropy(myText)) << endl; |

Несмотря на свою стандартность, кодировка ASCII вызвала некоторые проблемы, ибо при существовании определённой 7-битной инфраструктуры происходило обрезание восьмого бита в сообщении. И одним из решений стал алгоритм **base64**, разработанный для представления произвольных последовательностей октетов в форме, использующей строчные/прописные буквы из 65-символьного подмножества US-ASCII, приведённый на рисунке 1.

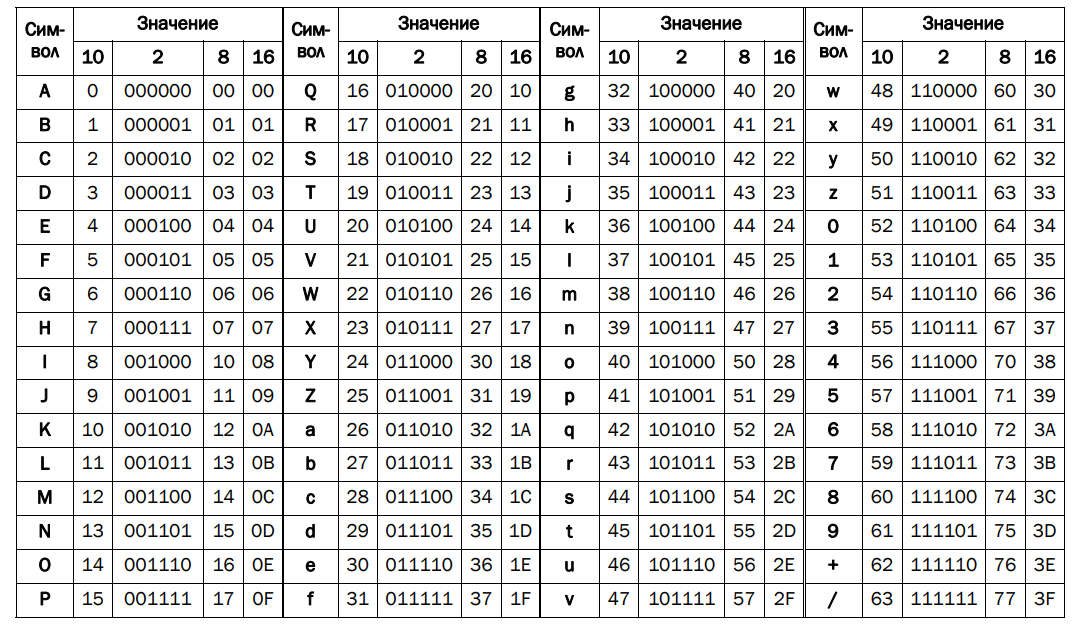


Рисунок 1 - 65-символьное подмножество US-ASCII

Данное подмножество обеспечивает представление одним печатным символом 6 битов данных. Процесс кодирования есть группа из 24 битов в форме строки из 4 символов. Обработка идёт слева направо, а 24-битная группа образуется конкатенацией трёх байтов. Данные затем трактуются как 4 сцепленные группы по 6 битов, и каждая транслируется в символ алфавита base64. При кодировании только одного/двух байтов, в результате получаются два/три символа строки, а выходная дополняется «=», что предовращает добавления дополнительных битов к восстановленным данным. Реализация алгоритма приведена в листинге 2.2.

Листинг 2.2. Реализация перевода текста в base64

|  |
| --- |
| string TextToBase64(string str) {  string base64chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";  string a, b;  int mod = str.size() % 3;  //making a str multiple to 3 for encoding to base64  if (mod > 0) {  for (; mod < 3; mod++) {  a += '=';  str += '\0';  }  }  mod = 0;  while(mod<str.size()) {  if (mod > 0 && (mod / 3 \* 4) % 76 == 0) {  b += '\r\n';  }  unsigned long n = (CodeAt(str[mod]) << 16) + (CodeAt(str[mod+1]) << 8) + CodeAt(str[mod+2]);  b += string() + base64chars[(n >> 18) & 63] + base64chars[(n >> 12) & 63] + base64chars[(n >> 6) & 63] + base64chars[n & 63];  mod += 3;  }  return b.substr(0,b.size() - a.size()) + a;  } |

Результат преобразования представлен на рисунке 2.

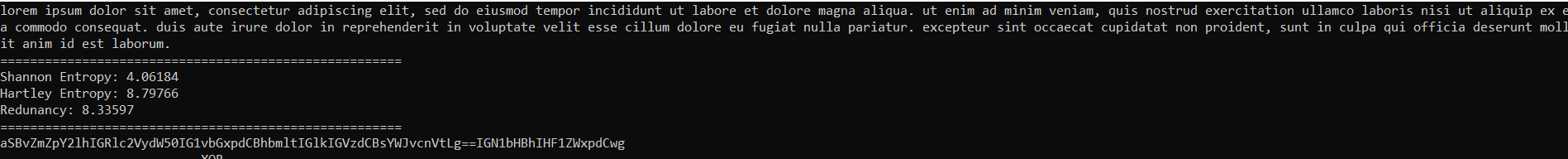


Рисунок 2 – результат перевода в base64

Для дальнейшего преобразования данных этого формата, самым простым способом является использование оператора XOR (сложения по модулю 2). Мы побитово складываем биты строки и находим остаток от деления получившегося бита на 2. Реализация представлена в листинге 2.3.

Листинг 2.3. Реализация XOR

|  |
| --- |
| string Xor(string a, string b) {  string resultBit, result;  cout << "First string:" << a << endl;  cout << "Second string:" << b << endl;  if (a.size() > b.size()) {  while (a.size() != b.size()) {  b += '\0';  }  }  else if (a.size() < b.size()) {  while(a.size()!=b.size()){  a += '\0';  }  }  for (int i = 0; i < b.size(); i++) {  bitset<8>c1(a[i]);  bitset<8>c2(b[i]);  resultBit += (c1 ^ c2).to\_string();  result += char((c1 ^ c2).to\_ulong());  }  cout << "Binary Result: " << resultBit << endl;  cout << "Result: ";  return result;  } |

Результат представлен на рисунке 3.

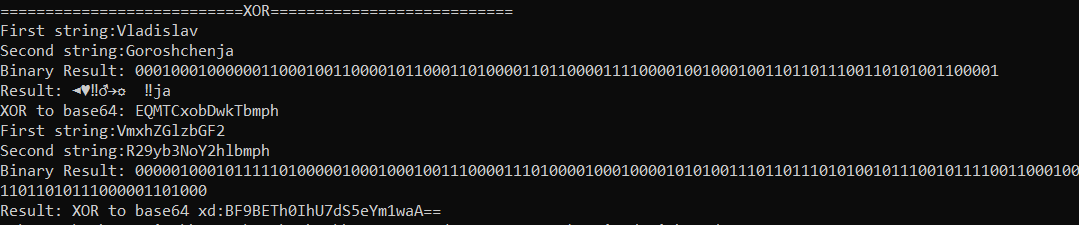


Рисунок 3 – результат XOR

При расчёте aXORbXORb будет возвращена строка a.