МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Дальневосточный федеральный университет»
(ДВФУ)

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

Департамент математического и компьютерного моделирования

РЕФЕРАТ
о практическом задание по дисциплине АИСД
«Алгоритм сжатия информации арифметическое кодирование»

направление подготовки 09.03.03 «Прикладная информатика»
профиль «Прикладная информатика в компьютерном дизайне»

Выполнил студент
гр. Б9121-09.03.03пикд
Безрукова Анастасия Леонидовна
_____
(подпись)
Руководитель практики
Доцент ИМКТ А.С Кленин
(должность, уч. звание)
_____
(подпись)
«_____» _____ 2022г.

Реферат защищен:
С оценкой _____

Рег. № _____

«____» _____ 2022 г.

г. Владивосток
2022

Аннотация

Сейчас существует множество алгоритмов сжатия информации. Большинство из них широко известны, но есть и некоторые весьма эффективные, но, тем не менее, малоизвестные алгоритмы.

Прежде чем рассказывать об арифметическом кодировании, надо сказать пару слов об алгоритме Хаффмана. Этот метод эффективен, когда частоты появления символов пропорциональны $1/2^n$ (где n – натуральное положительное число). Это утверждение становится очевидным, если вспомнить, что коды Хаффмана для каждого символа всегда состоят из целого числа бит. Рассмотрим ситуацию, когда частота появление символа равна 0,2, тогда оптимальный код для кодирования это символа должен иметь длину $-\log_2(0,2)=2,3$ бита. Понятно, что префиксный код Хаффмана не может иметь такую длину, т.е. в конечном итоге это приводит к ухудшению сжатия данных.
Арифметическое кодирование предназначено для того, чтобы решить эту проблему. Основная идея заключается в том, чтобы присваивать коды не отдельным символам, а их последовательностям.

Постановка задачи

Основная задача состоит в том, чтобы изучить данный нам алгоритм арифметического кодиравания, а затем на основе изученного материала его реализовать.

Авторы и история

Базовые алгоритмы арифметического кодирования были разработаны независимо Йорма Дж. Риссаненом из IBM Research и Ричардом К. Паско, аспирантом Стэнфордского университета; оба были опубликованы в мае 1976 года.

Описание алгоритма

Арифметическое кодирование — один из алгоритмов энтропийного сжатия.

В отличие от алгоритма Хаффмана, не имеет жесткого постоянного соответствия входных символов группам бит выходного потока. Это даёт алгоритму большую гибкость в представлении дробных частот встречаемости символов.

Как правило, превосходит алгоритм Хаффмана по эффективности сжатия, позволяет сжимать данные с энтропией, меньшей 1 бита на кодируемый символ, но некоторые версии имеют патентные ограничения от компании IBM.

При арифметическом кодировании каждый символ кодируется нецелым числом бит, что эффективнее кода Хаффмана (теоретически, символу ***a*** с вероятностью появления ***p(a)*** допустимо ставить в соответствие код длины $-$ ***log2p(a)***, следовательно, при кодировании алгоритмом Хаффмана это достигается только с вероятностями, равными обратным степеням двойки).

При арифметическом кодировании текст представляется вещественными числами в интервале от 0 до 1. По мере кодирования текста, отображающий его интервал уменьшается, а количество битов для его представления возрастает. Очередные символы текста сокращают величину интервала исходя из значений их вероятностей, определяемых моделью. Более вероятные символы делают это в меньшей степени, чем менее вероятные, и, следовательно, добавляют меньше битов к результату.

Заданное множество символов — это, как правило, ASCII+. Для того, чтобы обеспечить остановку алгоритма распаковки вначале сжимаемого сообщения надо поставить его длину или ввести дополнительный символ-маркер конца сообщения.

Пример работы

Перед началом работы кодера соответствующий кодируемому тексту исходный интервал составляет **[0; 1).**

Алфавит кодируемого сообщения содержит следующие символы (буквы): { **Р, А, Д, И, О, В, З** }.

Определим количество (встречаемость, вероятность) каждого из символов алфавита в сообщении и назначим каждому из них интервал, пропорциональный его вероятности. С учетом того, что в кодируемом слове всего 10 букв, получим (см. Таблица 1).

Символ	*Вероятность*	*Интервал*
А	*0,1*	*0 – 0,1*
Д	*0,1*	*0,1 – 0,2*
В	*0,1*	*0,2 – 0,3*
И	*0,3*	*0,3 – 0,6*
З	*0,1*	*0,6 – 0,7*
О	*0,1*	*0,7 – 0,8*
Р	*0.2*	*0,8 – 1*

Таблица 1

Располагать символы в таблице можно в любом порядке: по мере их появления в тексте, в алфавитном или по возрастанию вероятностей – это совершенно не принципиально. Результат кодирования при этом будет разным, но эффект – одинаковым.

Итак, перед началом кодирования исходный интервал составляет [0 – 1).

После просмотра первого символа сообщения **Р** кодер сужает исходный интервал до нового - [0.8; 1), который модель выделяет этому символу. Таким образом, после кодирования первой буквы результат кодирования будет находиться в интервале чисел [0.8 - 1).

Следующим символом сообщения, поступающим в кодер, будет буква **А**. Если бы эта буква была первой в кодируемом сообщении, ей был бы отведен интервал [0 - 0.1), но она следует за **Р** и поэтому кодируется новым подынтервалом внутри уже выделенного для первой буквы, сужая его до величины [0.80 - 0.82). Другими словами, интервал [0 - 0.1), выделенный для буквы **А**, располагается теперь внутри интервала, занимаемого предыдущим символом (начало и конец нового интервала определяются путем прибавления к началу предыдущего интервала произведения ширины предыдущего интервала на значения интервала, отведенные текущему символу). В результате получим новый рабочий интервал [0.80 - 0.82), т. к. предыдущий интервал имел ширину в 0.2 единицы, и одна десятая от него есть 0.02.

Следующему символу **Д** соответствует выделенный интервал [0.1 - 0.2), что применительно к уже имеющемуся рабочему интервалу [0.80 - 0.82) сужает его до величины [0.802 - 0.804).

Следующим символом, поступающим на вход кодера, будет буква **И** с выделенным для нее фиксированным интервалом [0,3 – 0,6). Применительно к уже имеющемуся рабочему интервалу получим [0,8026 - 0,8032).

Продолжая в том же духе, имеем:

вначале [0,0 - 1,0)

после просмотра **Р** [0,8 - 1,0)

А [0,80 - 0,82)

Д [0,802 - 0,804)

И [0,8026 - 0,8032)

О [0,80302 - 0,80308)

В [0,803032 - 0,803038)

И [0,8030338 - 0,8030356)

З [0,80303488 - 0,80303506)

И [0,803034934 - 0,803034988)

Р [0,8030349772 - 0,8030349880)

Результат кодирования: интервал [0,8030349772 – 0,8030349880]. На самом деле, для однозначного декодирования теперь достаточно знать только одну границу интервала – нижнюю или верхнюю, то есть результатом кодирования может служить начало конечного интервала - 0,8030349772. Если быть еще более точным, то любое число, заключенное внутри этого интервала, однозначно декодируется в исходное сообщение. К примеру, это можно проверить с числом 0,80303498, удовлетворяющим этим условиям. При этом последнее число имеет меньшее число десятичных разрядов, чем числа, соответствующие нижней и верхней границам интервала, и, следовательно может быть представлено меньшим числом двоичных разрядов.

Описание реализации

Реализация алгоритма состоит из 4 функций (***get_number (), get_code (), coding (), decoding ()***), а также главной функции ***main (),*** где, собственно, и вызываются функции.

Для начала через директиву ***#define*** объявим две глобальные переменные ***M*** и ***N***, где переменная ***M*** отвечает за длину сообщения (в нашем случае длина 100), и ***N*** отвечает за количество символов в используемом нами словаре (в нашем случае 4).

```
1  #include <iostream>
2  #define M 100
3  #define N 4
4  using namespace std;
```

Далее определяем приватные переменные, присваиваем их классу ***suanshu*** и определяем публичные функции для данного класса. Публичные функции нужны для того, чтобы мы могли иметь доступ к ним из любого места кода.

```
1  class suanshu
2  {
3    int count, length;
4    char number[N], n;
5    long double chance[N], c;
6    char code[M];
7    long double High, Low, high, low, d;
8  public:
9    suanshu()
10   {
11      High = 0; Low = 0;
12   }
13   void get_number();
14   void get_code();
15   void coding();
16   void decoding();
17   ~suanshu() {}
18 };
```

~suanshu() – Это функция деструктора, для очистки памяти.

Функция ***get_number ()*** отвечает за получение символов (***number [i]***) и их вероятностей (***chance[i]***), которые мы вводим.

```
1  void suanshu::get_number()
2  {
3      cout << "введите букву и ее вероятность:" << endl;
4      int i = 0;
5      for (; i < N; i++)
6     {
7          cin >> n >> c;
8          number[i] = n;
9          chance[i] = c;
10     }
```

```
11 if (i == 20) {
12     cout << "the number is full." << endl;
13 }
14 count = i;
15 }
```

Функция ***get_code ()*** выполняет ввод длины кодируемого сообщения, чтобы программа знала, когда ей остановиться. Так же эта функция получает на ввод сам код сообщения, которое нам нужно закодировать.

```
1  void suanshu::get_code()
2  {
3      cout << "укажите длину:";
4      cin >> length;
5      while (length >= M)
6      {
7        cout << "длина слишком большая, введите меньше:";
8        cin >> length;
9      }
10     cout << "Код вывода:";
11     for (int i = 0; i < length; i++)
12     {
13       cin >> code[i];
14     }
15 }
```

Функция ***coding ()*** выполняет само кодирование сообщения, которое мы ввели. Сначала она обрабатывает первый введённый нами символ, оценивает его подстрочный индекс (строка 5) и находит его верхнюю и нижнюю границы (строки 7,9).

```
1  void suanshu::coding()
2  {
3    int i, j = 0;
4    for (i = 0; i < count; i++)
5      if (code[0] == number[i]) break;
6    while (j < i)
7      Low += chance[j++];
8    d = chance[j];
9    High = Low + d;
```

Затем рассматривается второй символ. Если он равен предыдущему, то нижнюю границу символа остаётся прежней и меняется только верхняя (строки 4–12). Так же пересчитывается вероятность. Иначе если второй символ другой, то меняется и нижняя и верхняя границы, так же меняется и вероятность (строки 14 – 23). В этой же функции выводится результат кодирования, в нашем случае это нижняя граница (строка 27).

```
 1  for (i = 1; i < length; i++)
 2      for (j = 0; j < count; j++)
 3     {
 4           if (code[i] == number[j])
 5          {
 6                 if (j == 0)
 7                {
```

```
 8                      low = Low;
 9                      high = Low + chance[j] * d;
 10                     High = high;
 11                     d *= chance[j];
 12                 }
 13                 else
 14                 {
 15                     long double chance_l = 0.0;
 16                     for (int k = 0; k <= j - 1; k++)
 17                     chance_l += chance[k];
 18                     low = Low + d * chance_l;
 19                     high = Low + d * (chance_l + chance[j]);
 20                     Low = low;
 21                     High = high;
 22                     d *= chance[j];
 23                  }
 24             }
 25             else continue;
 26         }
 27   cout << "the result is:" << Low << endl;
 28 }
```

Функция ***decoding ()*** отвечает за декодирование нашего сообщения. В начале объявляем переменную типа char, которая отвечает за количество декодируемых символов (в нашем случае не больше 100).

```
1  void suanshu::decoding()
2  {
3    int i, j;
4    char out[100];
```

Далее расшифровываем символы по одному, постоянно меняя интервал кодирования, исходя из полученного результата в предыдущей функции (нижней границы ***Low***).

```
1  for (i = 0; i < length; i++)
2  {
3      long double m0 = 0.0;
4      long double m1 = 0.0;
5      for (j = 0; j < count; j++)
6      {
7          m0 = m1;
8          m1 = m0 + chance[j];
9          if ((Low >= m0) && (Low < m1))
10         {
11            out[i] = number[j];
12            Low -= m0;
13            Low = Low / (chance[j]);
14            break;
15         }
16         continue;
17     }
18    cout << out[i];
```

```
19   }
20  cout << endl;
21 }
```

Последняя главная функция ***main ()*** отвечает за вызов всех функций.

```
1  int main()
2  {
3      setlocale(LC_ALL, "Russian");
4      suanshu a;
5      a.get_number();
6      a.get_code();
7      a.coding();
8      a.decoding();
9      return 0;
10 }
```

Список литературы

[1] Arithmetic coding. URL: https://translated.turbopages.org/proxy_u/en-ru.ru.fbaf1e08-63ce7c75-d7985506-74722d776562/https/en.wikipedia.org/wiki/Arithmetic_encoder#History_and_patents

[2] Арифметическое кодирование. URL: Арифметическое кодирование — Википедия (wikipedia.org)

[3] Алгоритмы сжатия. URL: https://mf.grsu.by/UchProc/livak/po/comprsite/theory_arithmetic.html

[4] Арифметическое кодирование. URL: Арифметическое кодирование (helpiks.org)