Лабораторна робота  
з дисципліни  
«Теорія інформації кодування»

на тему  
**«Оцінювання обсягу інформації. Теорема Шенона»**

Виконав:

студент групи ДА-51

Болобан О. А.

Пункт 1. В ручному вигляді або в Excel реалізувати алгоритм Шеннона-Фанно для власного ФІО.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| o | l | \_ | a | i | n | A | B | O | b | c | e | g | h | t | v | y |

o->000

l->001

\_->010

a->0110

i->0111

n->1000

A->1001

B->1010

O->10110

b->10111

c->1100

e->11010

g->11011

h->11100

t->11101

v->11110

y->11111

Пункт 2. Розрахувати середню кількість біт на символ в повідомленні та порівняти її з ентропією повідомлення.

=3.962

## Пункт 3. Розробити алгоритм та структури даних для програмної реалізації алгоритму Шеннона-Фано.

Алгоритм Шеннона-Фано базується на розподіленні кодів символів відповідно до їх частоти зустрічі у певному повідомленні. Тому доцільним буде попереднє визначення частоти входження символів алфавіту повідомлення у повідомлення. Для реалізації може бути створений масив, що складається з 256 комірок типу char; номер кожного елементу масиву відповідає коду символу в системі кодування asci, а значення елементу відповідає кількості входжень даного символу у повідомлення. При зчитуванні певного символу з файлу, до елементу масиву, номер якого відповідає коду зчитаного символу, додається одиниця. Після закінчення читання з файлу, усі елементи, що мають відмінне від нуля значення, переписуються у інший масив, у якому сортируються за спаданням. Номер елемента, який зберігає кількість входження символу #k до файлу зберігається у первинному масиві. Створюється двовимірний (третинний) масив, що містить a\*256 елементів типу char, де a – кількість символів у алфавіті повідомлення (розмір вторинного масиву). Даний масив зберігатиме коди символів, що будуть використані при архівації, а саме: k-й рядок масиву містить код символу, що зберігається у вторинному масиві у k-му елементі.

Після цього робиться певна кількість проходів по вторинному масиву, кожен з яких визначає номер символу, для якого сума частот зустрічі усіх символів масиву до нього, починаючи з того, з якого починався прохід, та закінчуючи даним символом, включно, найбільш близька до половини суми частот усіх елементів масиву, по яким в даний момент часу робиться прохід (кажучи іншими словами, ці елементи розбиваються на дві групи з максимально близькими сумами частот зустрічі символів з цих груп). Після цього реалізується прохід по кожній з двох новоутворених груп символів, і так до тих пір, поки у новоутвореній групі не залишиться лише один символ При потраплянні певного елементу до групи, по якій було зроблено прохід, до коду елементу, що зберігається у третинному масиві, додається 0 або 1, при входженні цього елементу до першої або другої новоутворених груп відповідно. При завершенні проходів по всім новоутвореним групам, у третинному масиві зберігається таблиця кодів для усіх символів алфавіту повідомлення. Ця таблиця записується у файл архіву. За допомогою вторинної та третинної таблиць вираховується кількість нулів, що містить код кожного символу вихідного файлу; це число записується у файл архіву.

Починається повторне зчитування вихідного файлу. При зчитуванні певного символу, за його номером, за допомогою первинної таблиці, визначається номер рядка третинної таблиці, що відповідає цьому символу.

Символи, з яких складається архів, є відображенням послідовності символів 0 та 1, якими кодується кожен символ алфавіту. В програмі реалізовано ansi-кодування символів архіву, тобто кожен символ має код від 0 до 255. При його представленні у бінарному алфавіті, необхідно 8 символів. Тому доцільно розбити отриману при кодуванні символів вихідного файлу послідовність символів 0 та й на блоки, довжиною по 8 символів, та відображати кожен блок у архіві через символ, закодований ansi-кодуванням.

Отже, при зчитуванні символу з вихідного файлу, він за третинною таблицею буде перетворений на послідовність нулів та одиниць, які будуть накопичуватися. При накопиченні восьми символів, вони перетворюються у десяткове число, яке використовується для запису в архів символу з відповідним кодом. Якщо при перетворенні останнього символу вхідного файлу на послідовність нулів та одиниць, кількість накопичених символів не досягла восьми, вона доповнюється нулями і також перетворюється на символ, що останнім записується до архіву.

## Пункт 4. Скласти програму, яка забезпечує стиснення довільних даних за алгоритмом Шеннона-Фано. Лістинг додається

Пункт 5. Порівняти ефективність роботи з архіваторами RAR прикладах файлів: текстовому, мультимедійному, exe. Результати занести в таблицю:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Файл | Розмір | Ентропія | Обсяг інформації | Розмір .rar | Розмір .paa |
| input.txt | 699 423 | 4,762 | 416 332 | 246 757 | 421 777 |
| TLINK.exe | 5 901 737 | 5,385 | 3 972 607 | 1 273 623 | 4 064 123 |
| Music.mp3 | 5 572 766 | 7,863 | 5 477 332 | 5 300 469 | 5 519 478 |
| У середньому | 4 057 975 | 6,003 | 3 288 757 | 2 273 616 | 3 335 126 |

Music.mp3.a:

input.txt.a

TLINK.exe.a

Висновок

Отримані в результаті виконання роботи результати показують, що алгоритм Шеннона-Фано дозволяє перекодувати повідомлення досить стисло: ентропія, яка характеризує мінімальну кількість біт/символ, для розглянутого повідомлення рівна 3.835, а кількість біт/символ в результаті стиснення за Шеннона-Фано рівна 3.92, що на 2,2% перевищує ентропію. Даний результат є добрим для статистичного стиснення інформації.

Було розроблено програму, яка за алгоритмом Шеннона-Фано виконує архівацію та розархівацію файлів, вказаних користувачем. Результати роботи програмі:

**.txt –файл.** 39,7% стиснення. Художня література, як текст, має дуже неоднорідне розподілення символів: велика кількість пробілів, знаків пунктуації, певних голосних та приголосних літер робить можливим закодувати ці символи більш коротким кодом, тим самий значно заощаджуючи пам'ять, що витрачається на них. З іншого боку, деякі символи зовсім не використовуються в подібних текстах (наприклад керуючі символи, чи спеціальні знаки типу →,§ і т.п.), що також значно зменшує розмір архіву. Гістограма частот символів у архіві .txt- файлу, має також пилкоподібний вигляд, так як різні комбінації символі у вихідному файлу призводять до досить нерівномірного розподілення символів кодування. Методом словарного стиснення досягається 64,7% стиснення, що пов’язано з повторенням певних частин слова чи окремих слів, які автор використовує за змістом.

**.exe-файл.** 31,1% стиснення. Виконуючий файл, як результат компіляції програмного коду, містить у собі велику кількість керуючих символів, а також символів латиниці, що призводить до нерівномірного розподілення символів за частотами. Де дозволяє виконуючі файли також ефективно стискати статистичними методами. Гістограма частот символів у архіві виконуючого файлу також має пилкоподібний характер, але менш виражений через те, що використовується ширший діапазон символів, ніж у текстових файлах, а також більш рівномірне розподілення більшої кількості символів з цього діапазону. Це і призводить до того, що розподілення кодуючи символів за частотами є більш рівномірним, ніж у текстовому файлі, і гістограма виглядає згладженою. Методом словарного стиснення досягається 78,4% стиснення, що пов’язано з постійним використанням однакових лексем, якими є оператори.

**.mp3-файл.** 0,96% стиснення. Мультимедійний файл mp3 вже використовує власну систему кодування, тому окремі символи не несуть у собі ніякого додаткового змістового значення, окрім того, що кодують певну послідовність біт мультимедійного файлу. Через це розподіл символів по частотам є рівномірним, а статистичне кодування не дає майже ніяких результатів, так як немає сенсу перекодовувати символи з майже однаковою частотою зустрічі. Через це і гістограма частот мультимедійного файлу має такий вигляд – перерозподілення пам’яті між символами майже не відбувалося і вид гістограми мало змінився, порівняно з вихідним файлом. Методом словарного стиснення досягається 4,9% стиснення. Це не значно перевищує статистичний метод, так як символи у мультимедійному файлі використовуються для кодування інформації за певним алгоритмом і не мають власного змісту. Але деякі їх комбінації повторюються, так як використовуються для кодування звукових сигналів; ці комбінації кодуються як лексеми, тому словарний метод є більш ефективним.

Лістинг

#include <fstream>

#include <iostream>

#include <iomanip>

#include <ctime>

#include <conio.h>

#include <cmath>

#include <string.h>

#include <stdlib.h>

using namespace std;

char \*\*code,z;

int \*chNum;

long \*chAmo;

FILE \*Fin,\*Fout,\*FList;

int i=0,j,a=0,alp;

unsigned char c;

double ent=0,avectore=0;//entropi & avectorerage character size

long ASCII[256],t,sum=0;

char FName[256],Comm[9],ListName[256];

bool unexp=false;

struct path

{ path \*p0;

path \*p1;

char Let;

bool fnl;

};

path \*root;

int Tdel(path \*d)//cleaning tree

{ path \*p;

p=d;

if (p->p0!=NULL)

Tdel(p->p0);

if (p->p1!=NULL)

Tdel(p->p1);

delete p;

return 0;

}

int bin(int b, int e) //coding to binary

{ long sum0=0,sum1=chAmo[b];

for(i=b;i<e+1;i++)

{ sum0+=chAmo[i];

}

for(i=b;i<e;)

{ if (abs(sum0/2.0-sum1)>abs(abs(sum0/2.0-sum1-chAmo[i+1])))

{ sum1+=chAmo[i+1];

i++;

}

else

{ break;

}

}

if (b==e) return 0;

else

{ for(j=b;j<i+1;j++)

{ strcat(code[j],"0");

}

bin(b,i);

for(j=i+1;j<e+1;j++)

{ strcat(code[j],"1");

}

bin(i+1,e);

}

}

int archivectore(){

for (i=0;i<256;i++)

ASCII[i]=0;

while (!feof(Fin))

{ c=getc(Fin);

ASCII[c]++;

sum++;

}

if (ASCII[255])

{ ASCII[255]--;

sum--;

}

fclose(Fin);

double d;

for (i=0;i<256;i++)

if (ASCII[i])

{ a++;

d=double(ASCII[i])/double(sum);

ent+=d\*log(d)/log(2);

}

chAmo=new long[a];

chNum=new int[a];

code=new char \*[a];

for(i=0;i<a;i++)

{ code[i]=new char[256];

for(j=0;j<256;j++)

code[i][j]=\*"";

}

j=0;

for (i=0;i<256;i++)

{ if (ASCII[i])

{ chNum[j]=i;

chAmo[j]=ASCII[i];

ASCII[i]=j;

j++;

}

}

for(i=0;i<a-1;i++)

for(j=0;j<a-1;j++)

if (chAmo[j]<chAmo[j+1])

{ t=chAmo[j];

chAmo[j]=chAmo[j+1];

chAmo[j+1]=t;

t=chNum[j];

chNum[j]=chNum[j+1];

chNum[j+1]=t;

ASCII[chNum[j]]=j;

ASCII[chNum[j+1]]=j+1;

}

if (a)

bin(0,a-1);

else

return 0;

if (a-1==0)

code[0][0]='0';

Fin=fopen(FName,"rb");

strcat(FName,".a");

Fout=fopen(FName,"wb");

t=0;

for (i=0;i<a;i++)

{ putc(chNum[i],Fout);

putc(chNum[i],FList);

putc(' ',Fout);

alp=0;

for (j=0;j<256;j++)

if (code[i][j])

{ alp++;

putc(code[i][j],Fout);

putc(code[i][j],FList);

if (code[i][j]=='0')

t+=chAmo[i];

}

else if (j<20)

putc(' ',FList) ;

putc('.',Fout);

putc(' ',FList);

fprintf(FList,"%d\n",chAmo[i]);

avectore+=double(chAmo[i])\*double(alp);

}

fprintf(Fout,":");

avectore=avectore/double(sum);

fclose(FList);

alp=a;

a=0;

i=7;

unsigned char o=char(0);

while (!feof(Fin))

{ c=getc(Fin);

for(j=0;j<256;j++)

if (code[ASCII[c]][j])

{ z=code[ASCII[c]][j];

o+=atoi(&z);

if (i==0)

{

o=int(0);

i=7;

}

else

{ o<<=1;

i--;

}

}

}

while(i)

{ o <<1;

i--;

}

fclose(Fin);

fclose(Fout);

for (i=0;i<alp;i++)

delete code[i];

delete code;

delete chAmo;

delete chNum;

return 0;

}

bool fill(char s)//binary read archive file

{

path \*p, \*head;

char c;

p=root;

c=getc(Fin);

while(c!='.')

{ if (c==':')

return true;

if (c=='1')

{ if (p->p1==NULL)

{ head=new path;

head->p1=NULL;

head->p0=NULL;

head->Let=\*"";

head->fnl=false;

p->p1=head;

}

p=p->p1;

}

if (c=='0')

{ if (p->p0==NULL)

{ head=new path;

head->p1=NULL;

head->p0=NULL;

head->Let=\*"";

head->fnl=false;

p->p0=head;

}

p=p->p0;

}

if (c!='1'&&c!='0')

{ unexp=true;

return false;

}

c=getc(Fin);

}

p->Let=s;

p->fnl=true;

return false;

}

int dearchivectore()//dearchivectorate

{ path \*p;

int seq[8];

long sum0=0, summ0=0;

int numb;

bool text=false;

root=new path;

root->p1=NULL;

root->p0=NULL;

root->Let=\*"";

root->fnl=false;

while (!text)

{

c=getc(Fin);

if (feof(Fin))

{ Tdel(root);

return 2;

}

text=fill(c);

if (unexp)

return 2;

}

strcat(FName,".da");

Fout=fopen(FName,"wb");

c=getc(Fin);

while (c!=':')

{ z=c;

sum0=10\*sum0+atoi(&z);

if (feof(Fin))

{ Tdel(root);

return 2;

}

c=getc(Fin);

}

p=root;

while(!feof(Fin))

{ c=getc(Fin);

numb=c;

for(i=0;i<8;i++)

{ seq[7-i]=numb%2;

numb=numb/2;

}

for(i=0;i<8;i++)

{ if (seq[i]==0)

{ summ0++;

p=p->p0;

}

else

p=p->p1;

if (summ0==sum0+1)

{ Tdel(root);

return 1;

}

if (p->fnl)

{ putc(p->Let,Fout);

p=root;

}

}

}

Tdel(root);

return 2;

}

int main()

{

cout<<"Type 'archivectore' or 'dearchivectore': ";

Comm[7]=Comm[8]=' ';

cin>>Comm;

cout<<"Enter the file name, which will "<<Comm<<" {filename.type}: ";

cin>>FName;

while (strstr(Comm,"archive")!=Comm&&strstr(Comm,"dearchive")!=Comm)

{ cout<<"Invectoralid command "<<Comm<<endl;

cin>>Comm>>FName;

}

while (!(Fin=fopen(FName,"rb")))

{ cout<<"File '"<<FName<<"' does not exist. Input the filename: ";

cin>>FName;

}

clock\_t time;

if (strstr(Comm,"archive")==Comm)

{ time=clock();

archivectore();

time=clock()-time;

if(ent) ent=-ent;

cout<<"Entropy = "<<setprecision(4)<<ent<<endl;

cout<<"Avectorerage character size = "<<avectore<<endl;

cout<<"Archivectoration time is "<<(double)time/CLOCKS\_PER\_SEC<<" seconds;"<<endl<<"Name archivectore file is "<<FName<<endl;

}

if (strstr(Comm,"dearchive")==Comm)

{ int res;

time=clock();

res=dearchivectore();

time=clock()-time;

if (res==1)

{

cout<<"Dearchivectoration complete!"<<endl<<"Name dearchivectore file is "<<FName<<endl;

}

else

{ cout<<"Unexpected end of file";

}

}

fclose(Fout);

fclose(Fin);

\_getch();

return 0;

}