|  |  |  |  |
| --- | --- | --- | --- |
| ***KOLEGIUM KARKONOWSKIE***  ***WYDZIAŁ TECHNICZNY*** | | | |
| ***KODOWANIE I KRYPTOGRAFIA W TELEKOMUNIKACJI LABORATORIUM*** | | | |
| *Numer grupy: 4* | *Prowadzący: mgr inż. Andrzej Rehlis* | | |
| *Wykonał:* | *Temat ćwiczenia:*  *Kod Hamminga (7,4)* | | |
| *Data wykonania:*  *25.03.2009 r.*  *01.04.2009 r.* | *Ocena:* | *Podpis:* |

1. **Wstęp**

W [telekomunikacji](http://pl.wikipedia.org/wiki/Telekomunikacja) **kod Hamminga** jest [liniowym kodem korekcyjnym](http://pl.wikipedia.org/w/index.php?title=Liniowy_kod_korekcyjny&action=edit&redlink=1), wynalezionym przez [Richarda Hamminga](http://pl.wikipedia.org/wiki/Richard_Hamming). Kody Hamminga wykrywają i korygują błędy polegające na przekłamaniu jednego bitu - dla niezawodnej transmisji wymagane jest, aby [odległość Hamminga](http://pl.wikipedia.org/wiki/Odleg%C5%82o%C5%9B%C4%87_Hamminga) między słowami transmitowanymi i odbieranymi wynosiła zero lub jeden. Kody te mogą też wykryć (ale już nie korygować) błędy podwójne (dwa jednocześnie przekłamane bity).

Dla porównania prosty kod z kontrolą parzystości nie może korygować żadnych błędów ani też nie może być używany do detekcji błędu na więcej niż jednym bicie.

W sensie matematycznym kody Hamminga są klasą [liniowych kodów binarnych](http://pl.wikipedia.org/w/index.php?title=Liniowy_kod_binarny&action=edit&redlink=1). Dla każdej [liczby całkowitej](http://pl.wikipedia.org/wiki/Liczba_ca%C5%82kowita) m>1 istnieje kod o parametrach [2*m* − 1,2*m* − *m* − 1,3]. Macierz kontroli parzystości dla kodu Hamminga tworzy się wypisując wszystkie kolumny o długości m, które są parami niezależne.

Z uwagi na swoją prostotę kody Hamminga są szeroko używane w pamięciach komputerowych ([RAM](http://pl.wikipedia.org/wiki/RAM)).

Jeśli w wiadomości jest więcej bitów korygujących i jeśli te bity dla różnej kombinacji bitów przekłamanych dają różne rezultaty, wtedy możemy zidentyfikować nieprawidłowe bity. W 7-bitowej wiadomości jest 7 prawdopodobnych błędów pojedynczych, więc trzy bity korygujące wystarczą by potencjalnie wskazać nie tylko że błąd wystąpił lecz także na której pozycji.

Hamming zauważył problem przy przekłamaniu dwóch lub więcej bitów i wprowadził pojęcie odległości (teraz nazywanej odległością Hamminga). Kod z kontrolą parzystości ma odległość równą 2 (przekłamanie dwóch bitów jest niewidoczne - kod nie zgłasza błędów, ale słowo jest inne niż przesłane).

Hamming skupił się na dwóch problemach: zwiększeniu odległości między słowami jak to tylko możliwe, i jednocześnie jak największym stosunku liczby bitów informacyjnych do długości słowa. Główną ideą zastosowanych przez niego schematów kodowania jest nakładanie się bitów parzystości, tak, aby mogły sprawdzać siebie nawzajem.

1. **Koder kodu Hamminga**

Program Koder\_Hamminga;

uses Crt;

type TT=array[1..20] of integer; {deklaracja tablicy o nazwie TT składającej się z 20 elementów typu integer}

var {deklaracja zmiennych}

i, a1, a2, a3, a4: integer; {zmienne i, a1, a2, a3, a4 typu całkowitego}

Twej, Twyj: TT; {deklaracja tablic Twej i Twyj}

function FXOR(x,y: integer): integer; {początek funkcji FXOR; deklaracja parametrów x i y typu integer; wynik typu integer}

begin

FXOR:=abs(x-1)\*y+abs(y-1)\*x; {instrukcja obliczająca wartość funkcji}

end;

procedure Oblicz; {początek procedury Oblicz}

var blad: boolean; {deklaracja obiektów lokalnych}

begin

blad:=false; {podstawienie błąd = fałsz}

for i:=1 to 4 do {pętla for od 1 do 4}

if (Twej[i]<0) or (Twej[i]>1) then blad:=true; {jeśli i jest mniejsze od 0 lub większe od 1 wtedy podstawiamy za błąd = prawda}

if blad=false then {porównanie, jeśli błąd = fałsz wykonujemy poniższe instrukcje}

begin

a1:=FXOR(Twej[1], Twej[2]); {za a1 podstawiamy wynik działania funkcji FXOR dla danych z tablicy z miejsc 1 i 2}

a1:=FXOR(a1, Twej[4]); {za a1 podstawiamy wynik działania funkcji FXOR dla a1 i danej z tablicy z miejsca 4}

a2:=FXOR(Twej[1], Twej[3]); {za a2 podstawiamy wynik działania funkcji FXOR dla danych z tablicy z miejsc 1 i 3}

a2:=FXOR(a2, Twej[4]); {za a2 podstawiamy wynik działania funkcji FXOR dla a2 i danej z tablicy z miejsca 4}

a4:=FXOR(Twej[2], Twej[3]); {za a4 podstawiamy wynik działania funkcji FXOR dla danych z tablicy z miejsc 2 i 3}

a4:=FXOR(a4, Twej[4]); {za a4 podstawiamy wynik działanie funkcji FXOR dla a4 i danej z tablicy z miejsca 4}

Twyj[1]:=a1; Twyj[2]:=a2; Twyj[3]:=Twej[1]; Twyj[4]:=a4; {wpisywanie do tablicy wyjściowe wartości a1, a2, a4 na odpowiednich miejscach oraz wartości z tablicy wejściowej}

Twyj[5]:=Twej[2]; Twyj[6]:=Twej[3]; Twyj[7]:=Twej[4];

end

else

writeln('Dane wejsciowe moga byc rowne "0" lub "1"'); {jeśli błąd = prawda wyświetla się komunikat}

end;

BEGIN {początek programu głównego}

ClrScr;

Writeln('Wprowadz informacje 4 bitowa, a po kazdym 0 lub 1 daj spacje'); {Wyświetlanie komunikatu}

Writeln;

Read(Twej[1], Twej[2], Twej[3], Twej[4]); writeln; {odczytanie wpisanych wartości oraz wpisanie ich do tablicy wejściowej}

Oblicz; {uruchomienie procedury Oblicz}

Writeln('Wektor kodowy ma postac : '); writeln; {Wyświetlanie komunikatu}

Writeln(Twyj[1], Twyj[2], Twyj[3], Twyj[4], Twyj[5], Twyj[6], Twyj[7]); {wyświetlanie tablicy wyjściowej, która jest jednocześnie wektorem kodowym}

repeat until keypressed;

END.

1. **Wynik działania programu**

Do programu wprowadzamy 4 bitową informację binarną, a w wyniku działania tego programu otrzymujemy 7 bitowy wektor kodowy. Przedstawia to Tabela 3.1.

Tabela 3.1. Informacja 4 bitowa i jej wektor kodowy

|  |  |
| --- | --- |
| 4 bitowa informacja | Wektor kodowy |
| 0000 | 0000000 |
| 0001 | 1101001 |
| 0010 | 0101010 |
| 0011 | 1000011 |
| 0100 | 1001100 |
| 0101 | 0100101 |
| 0110 | 1100110 |
| 0111 | 0001111 |
| 1000 | 1110000 |
| 1001 | 0011001 |
| 1010 | 1011010 |
| 1011 | 0110011 |
| 1100 | 0111100 |
| 1101 | 1010101 |
| 1110 | 0010110 |
| 1111 | 1111111 |

Optymalizacją tego programu może być deklaracja tablic 7 elementowych, a nie 20 elementowych, ponieważ wynik działanie programu zapisywany jest w tablicy Twyj, a wektor kodowy jest właśnie 7 bitowy. Ponadto tablica 20 elementowa zajmuje więcej miejsca w pamięci.

1. **Dekoder kodu Hamminga**

Program Dekoder\_Hamminga;

uses Crt;

type TT=array[1..20] of integer; { deklaracja tablicy o nazwie TT składającej się z 20 elementów typu integer}

var {deklaracja zmiennych}

Twej, Twyj: TT; {deklaracja tablic}

s1, s2, s3, syndrom, i: integer; {deklaracja zmiannych typu całkowitaego}

odp: char; {deklaracja zmiannej typu char – znak}

function FXOR(x,y: integer) : integer; {deklaracja funacji FXOR; parametry x i y typu integer; wynik typu integer}

begin

FXOR:=abs(x-1)\*y+abs(y-1)\*x; {instrukcja obliczająca wartość funkcji}

end;

procedure Dekoduj; {początek procedury dekoduj}

var blad: boolean; {deklaracja obietków lokalnych}

i: integer;

begin

writeln;

blad:=false; {podstawienie błąd = fałsz}

for i:=1 to 7 do {pętla for od 1 do 7}

if (Twej[i]<0) or (Twej[i]>1) then blad:=true; {jeśli i jest mniejsze od 0 lub większe od 1 wtedy podstawiamy za błąd = prawda}

if blad=false then {porównanie, jeśli błąd = fałsz wykonujemy poniższe instrukcje}

begin

s1:=FXOR(Twej[1], Twej[3]); {za s1 podstawiamy wynik działania funkcji FXOR dla danych z tablicy wejściowej z miejsc 1 i 3}

s1:=FXOR(s1, Twej[5]); {za s1 podstawiamy wynik działania funkcji FXOR dla s1 i danej z tablicy wejściowej z miejsca 5}

s1:=FXOR(s1, Twej[7]); {za s1 podstawiamy wynik działania funkcji FXOR dla s1 i danej z tablicy wejściowej z miejsca 7}

s2:=FXOR(Twej[2], Twej[3]); {za s2 podstawiamy wynik działania funkcji FXOR dla danych z tablicy wejściowej z miejsc 2 i 3}

s2:=FXOR(s2, Twej[6]); {za s2 podstawiamy wynik działania funkcji FXOR dla s2 i danej z tablicy wejściowej z miejsca 6}

s2:=FXOR(s2, Twej[7]); {za s2 podstawiamy wynik działania funkcji FXOR dla s2 i danej z tablicy wejściowej z miejsca 7}

s3:=FXOR(Twej[4], Twej[5]); {za s3 podstawiamy wynik działania funkcji FXOR dla danych z tablicy wejściowej z miejsc 4 i 5}

s3:=FXOR(s3, Twej[6]); {za s3 podstawiamy wynik działania funkcji FXOR dla s3 i danej z tablicy wejściowej z miejsca 7}

s3:=FXOR(s3, Twej[7]); {za s3 podstawiamy wynik działania funkcji FXOR dla s3 i danej z tablicy wejściowej z miejsca 7}

syndrom:=s3\*4+s2\*2+s1; {obliczenie wartości syndromu}

for i:=1 to 7 do {pętla for od 1 do 7}

if i=syndrom then {jeżeli i równe jest syndromowi wykonujemy poniższe instrukcje}

begin

Twyj[i]:=FXOR(1, Twej[i]); {do tablicy wyjściowej wpisujemy wartości funkcji FXOR dla danych z tablicy wejściowej oraz liczby 1}

writeln('Blad wystapil na ',i,' pozycji'); {wypisanie, na którym miejscu wystąpił błąd}

writeln;

end

else Twyj[i]:=Twej[i]; {jeśli i jest różne od syndromu do tablicy wyjściowej wpisujemy wartości z tablicy wejściowej}

end

else writeln('Dane wejsciowe moga byc rowne 0 lub 1'); {jeśli błąd równy jest prawda wyświetla się komunikat}

end;

BEGIN {początek programu głównego}

repeat; {pętla repeat, która będzie się wykonywać dopóki będzie wprowadzana litera „t”}

ClrScr;

writeln('Wprodadz informacje 7 bitowa, a po kazdym 0 lub 1 daj spacje:'); {wyświetlenie komunikati}

read(Twej[1], Twej[2], Twej[3], Twej[4], Twej[5], Twej[6], Twej[7]); {odczytanie wpisanych wartości oraz wpisanie ich do tablicy wejściowej}

Dekoduj; {uruchomienie procedury Dekoduj}

writeln('Poprawny wektor zakodowany ma postac: ');

writeln(Twyj[1], Twyj[2], Twyj[3], Twyj[4], Twyj[5], Twyj[6], Twyj[7]); {wyświetlanie tablicy wyjściowej}

writeln;

writeln('Zdekodowana inforamcja ma postac: ');

writeln(Twyj[3], Twyj[5], Twyj[6], Twyj[7]); {wyświetlanie zakodowanej informacji}

writeln;

writeln('Powtorzyc? t/n');

readln(odp); {odczytanie wpisaniej litery}

until odp='t';

END.

1. **Wynik działania programu**

Program Dekoder Hamminga ma za zadanie sprawdzić, czy wprowadzony wektor kodowy ma poprawną postać. Jeśli wprowadzony wektor nie jest poprawny program wyświetli informację, na którym miejscu wystąpił błąd. Drugim zadaniem programu jest wyświetlanie zdekodowanej 4 bitowej informacji.