

PERCEPCIJSKO HEŠIRANJE

Studenti:

Jurica Migač

David Lazar

Kolegij: Sigurnost informacijskih sustava

Sadržaj prezentacije

Percepcijsko hashiranje

Kratak uvod u percepcijsko hashiranje

Algoritmi percepcijskog hashiranja

Algoritmi percepcijskog hashiranja

Usporedba slika pomoću algoritama

Usporedba slika kroz 4 algoritma percepcijskog hashiranja unutar programskog rješenja

Zaključak

Zašto koristiti percepcijsko hashiranje?
Moguće implementacije?

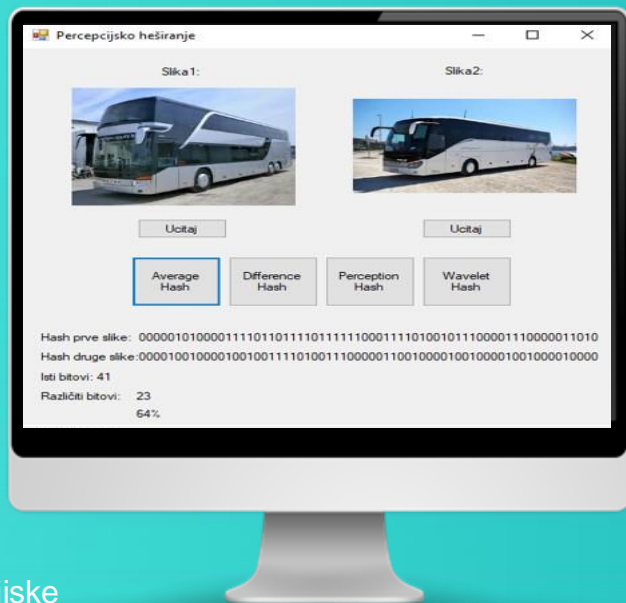
01

02

03

04

Aplikacija percepcijskog hashiranja



Aplikacija za izračunavanje 4 algoritma percepcijskog hashiranja dvije multimedijske datoteke te izvršavanje poredbe hash vrijednosti tih multimedijских datoteka.

Što je percepcijsko hashiranje?

Percepcijsko hashiranje

≠

Kriptografsko hashiranje

Razlika u hashevima između
percepcijskog hashiranja i
kriptografskog hashiranja

Digitalna forenzika

Pronalaženje sličnih
multimedijskih „otisaka
prstiju”

Slučajevi plagiranja

Velika implementacija u
sustavima za provjeru
plagijata

**Mogućnost izrade
korelacije između
hasheve**

Mogućnost poredbe
bitova, heksadekatskih
vrijednosti, samih
hasheva za izradu
korelacije



Average Hash

1. Promjena dimenzije slike
 - smanjenje na dimenziju 8x8 piksela iz bilo koje dimenzije slike bez obzira na omjer slike.
 - uklanjanje visoke frekvencije slike i detalje. Rezultat ovog koraka je slika u 64 piksela koja predstavlja sastav boja i raspodjelu izvorne slike. Smanjenjem dimenzije slike olakšava se rad sa hashiranjem.
2. Smanjenje boja slike ili bojanje u sivo
 - smanjiti vrijednost crvene, zelene, plave (RGB) boje u jednu sivu vrijednost boje piksela
 - svaki piksel ima vrijednost između 0 i 255
3. Izračunavanje prosječne vrijednosti boje
 - Prosječna vrijednost boje se izračunava tako da se zbroje vrijednosti svih piksela te se taj iznos podijeli sa 64.
4. Kreiranje hasha
 - Uspoređivanje vrijednosti svakog piksela slike iz 2. koraka s prosječnom vrijednošću iz 3. koraka.
 - Vrijednost veća ili jednaka $\rightarrow 1$
 - Vrijednost manja $\rightarrow 0$

Difference Hash

1. Promjena dimenzija slike
 - smanjenje na dimenziju 8x8 piksela iz bilo koje dimenzije slike bez obzira na omjer slike
 - uklanjanje visokih frekvencija slike i detalja
2. Smanjenje boja slike ili bojanje u sivo
 - smanjiti vrijednost crvene, zelene, plave (RGB) boje u jednu sivu vrijednost boje piksela
 - svaki piksel ima vrijednost između 0 i 255
3. Kreiranje hasha
 - uspoređuju se pikseli po redcima slijeva na desno sa svojim desnim susjedom
 - manji ili jednaki od svojeg desnog susjeda → 1
 - U suprotnom → 0
 - Kreira se još jedan hash koji se kreira tako da se uspoređuju pikseli po stupcima odozgo prema dolje sa svojim donjim susjedom
 - gornji susjed manji ili jednaki donjem susjedu → 1
 - U suprotnom → 0

Perception Hash

1. Smanjiti veličinu slike kao i kod Average Hash funkcije na 32x32 piksela
 - Zbog pojednostavljenja izračunavanja DCT funkcije
2. Smanjiti boju slike, odnosno bojanje slike u sive nijanse kao i kod Average Hash funkcije
3. Provođenjem DCT-a, dobiva se matrica 32x32, gdje je energija slike sadržana u nekoliko koeficijenata
4. Smanjiti DCT tako da zadržimo samo 8x8 gornje lijeve komponente kod matrice koeficijenata kako bi preostale najniže frekvencije slike
5. Izračunavanje prosječne DCT vrijednosti koeficijenata
6. Kreira se hash tako da se DCT koeficijenti uspoređuju s prosječnom DCT vrijednošću. Po principu ako je koeficijent u matrici veći ili jednaki od prosjeka u hash se zapisuje 1, a ako je manji zapisuje se 0.

Wavelet Hash

1. Smanjiti veličinu slike na 32x32 piksel te izračunavanje DWT-a zadržavajući visoku frekvenciju
2. Smanjiti boje slike, odnosno bojanje slike u sive nijanse. Provođenjem DWT-a, dobije se matrica DWT koeficijenata veličine 32x32, gdje je energija slike sadržana u nekoliko koeficijenata.
3. Smanjiti tako da zadržimo 8x8 gornje lijeve komponente kod matrice koeficijenata → predstavljanje najniže frekvencije
4. Izračunava se prosječna DWT vrijednost koeficijenata.
5. Kreira se hash tako da se DWT koeficijenti uspoređuju s prosječnom DWT vrijednošću te ako je DWT koeficijent u matrici veći ili jednaki od prosjeka u hash se zapisuje 1, a ako je manji zapisuje se 0.

Usporedba slika pomoću Average Hash algoritma

```
Bitmap slika = new Bitmap(bmpSlika, new Size(8, 8));
```

```
Bitmap grayscale;  
int x, y, avg, a, r, g, b;  
for (x = 0; x < slika.Width; x++)  
{  
    for (y = 0; y < slika.Height; y++)  
    {  
        Color bojaPiksela = slika.GetPixel(x, y);  
        a = bojaPiksela.A;  
        r = bojaPiksela.R;  
        g = bojaPiksela.G;  
        b = bojaPiksela.B;  
        avg = (r + g + b) / 3;  
        slika.SetPixel(x, y, Color.FromArgb(a, avg, avg, avg));  
    }  
}  
grayscale = slika;
```



Usporedba slika pomoću Average Hash algoritma

```
int avgCrvena = 0, avgZelena = 0, avgPlava = 0;
for (x = 0; x < grayscale.Width; x++)
{
    for (y = 0; y < grayscale.Height; y++)
    {
        r = grayscale.GetPixel(x, y).R;
        g = grayscale.GetPixel(x, y).G;
        b = grayscale.GetPixel(x, y).B;
        avgCrvena += r;
        avgZelena += g;
        avgPlava += b;
    }
}
Color prosjekBoja = Color.FromArgb(avgCrvena / 64, avgZelena / 64, avgPlava / 64);
```

```
for (x = 0; x < grayscale.Width; x++)
{
    for (y = 0; y < grayscale.Height; y++)
    {
        if (grayscale.GetPixel(x, y).GetBrightness() < prosjekBoja.GetBrightness())
        {
            hashVrijednost += "1";
        }
        else
        {
            hashVrijednost += "0";
        }
    }
}
```



Usporedba slika pomoću Difference Hash algoritma

```
Bitmap pomocnaSlika = new Bitmap(slika, new Size(9, 9));
```

```
Bitmap grayscale;  
int x, y, avg, a, r, g, b;  
for (x = 0; x < pomocnaSlika.Width; x++)  
{  
    for (y = 0; y < pomocnaSlika.Height; y++)  
    {  
        Color bojaPiksela = pomocnaSlika.GetPixel(x, y);  
        a = bojaPiksela.A;  
        r = bojaPiksela.R;  
        g = bojaPiksela.G;  
        b = bojaPiksela.B;  
        avg = (r + g + b) / 3;  
        pomocnaSlika.SetPixel(x, y, Color.FromArgb(a, avg, avg, avg));  
    }  
}  
grayscale = pomocnaSlika;
```

Percepcijsko heširanje

Slika1:  Slika2: 

Učitaj Učitaj

Average Hash Difference Hash Perception Hash Wavelet Hash

Hash prve slike: 11000111111010010001111100100001110011000010001111011110001001011111
Hash druge slike: 11111111111000111110110000011011110011011110110110011000100001110111
Isti bitovi: 74
Različiti bitovi: 54
57%

Usporedba slika pomoću Difference Hash algoritma

```
for(int redak = 0; redak < 8; redak++)
{
    for(int stupac = 0; stupac < 8; stupac++)
    {
        if (grayscale.GetPixel(stupac, redak + 1).R >= grayscale.GetPixel(stupac, redak).R)
        {
            rHashVrijednost += "1";
        }
        else
        {
            rHashVrijednost += "0";
        }
    }
}

for (int stupac = 0; stupac < 8; stupac++)
{
    for (int redak = 0; redak < 8; redak++)
    {
        if (grayscale.GetPixel(stupac+1, redak).R >= grayscale.GetPixel(stupac, redak).R)
        {
            cHashVrijednost += "1";
        }
        else
        {
            cHashVrijednost += "0";
        }
    }
}

PostaviHashVrijednost();
```

Percepcijsko heširanje

Slika1: 

Slika2: 

Učitaj Učitaj

Average Hash Difference Hash Perception Hash Wavelet Hash

Hash prve slike: 110001111101001000111100100001110011000010001111011110001001011111

Hash druge slike: 111111111100011111011000001101111001101111011011001100010000111011

Isti bitovi: 74

Različiti bitovi: 54

57%



Usporedba slika pomoću Perception Hash algoritma

```
public PerceptionHash(Bitmap bitmap)
{
    this.prvobitniHashSlike = ImagePhash.ComputeDigest(bitmap.ToLuminanceImage());
    byte[] hashSlikeBytes = Encoding.ASCII.GetBytes(this.prvobitniHashSlike.ToString());
    this.hashVrijednost = GetBitsFromBytes(new BitArray(hashSlikeBytes));
}
```

```
public Digest GetDigest()
{
    return this.prvobitniHashSlike;
}
```

```
public static double IzracunajSlicnostPercepcijskogHasha(PerceptionHash pHashPrveSlike, PerceptionHash pHashDrugeSlike)
{
    return ImagePhash.GetCrossCorrelation(pHashPrveSlike.GetDigest(), pHashDrugeSlike.GetDigest()) * 100;
}
```

Percepcijsko heširanje

Slika1:  Slika2: 

Učitaj Učitaj

Average Hash Difference Hash **Perception Hash** Wavelet Hash

Hash prve slike: 000011000001111010000010000111000110110001001100100000100010001010101
Hash druge slike: 0000110000011110010000101010001001101100100111001110110010101100000001
Isti bitovi: 373
Različiti bitovi: 283
60%

Usporedba slika pomoću Wavelet Hash algoritma

```
Bitmap slika = new Bitmap(bitmap, new Size(32, 32));
```

```
Bitmap pomocnaBitmapa;  
int x, y, avg, a, r, g, b;  
for (x = 0; x < slika.Width; x++)  
{  
    for (y = 0; y < slika.Height; y++)  
    {  
        Color bojaPiksela = slika.GetPixel(x, y);  
        a = bojaPiksela.A;  
        r = bojaPiksela.R;  
        g = bojaPiksela.G;  
        b = bojaPiksela.B;  
        avg = (r + g + b) / 3;  
        slika.SetPixel(x, y, Color.FromArgb(a, avg, avg, avg));  
    }  
}  
pomocnaBitmapa = slika;  
double[,] Crvena = new double[pomocnaBitmapa.Width, pomocnaBitmapa.Height];  
double[,] Zelena = new double[pomocnaBitmapa.Width, pomocnaBitmapa.Height];  
double[,] Plava = new double[pomocnaBitmapa.Width, pomocnaBitmapa.Height];  
  
Color boja;  
for (int j = 0; j < pomocnaBitmapa.Height; j++)  
{  
    for (int i = 0; i < pomocnaBitmapa.Width; i++)  
    {  
        boja = pomocnaBitmapa.GetPixel(i, j);  
        Crvena[i, j] = (double)Scale(0, 255, -1, 1, boja.R);  
        Zelena[i, j] = (double)Scale(0, 255, -1, 1, boja.G);  
        Plava[i, j] = (double)Scale(0, 255, -1, 1, boja.B);  
    }  
}  
FWT(Crvena, 4);  
FWT(Zelena, 4);  
FWT(Plava, 4);
```

Percepcijsko heširanje

Slika1:  Slika2: 

Učitaj Učitaj

Average Hash Difference Hash Perception Hash **Wavelet Hash**

Hash prve slike: 0001000001101111011110101101101001101110101011001001111111000

Hash druge slike: 0001010100011110000111101101000000101110000110100001111011110000

Isti bitovi: 40

Različiti bitovi: 24

62%

Usporedba slika pomoću Wavelet Hash algoritma

```
private void FWT(double[,] data, int iterations)
{
    int rows = data.GetLength(0);
    int cols = data.GetLength(1);

    double[] row = new double[cols];
    double[] col = new double[rows];

    for (int k = 0; k < iterations; k++)
    {
        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < row.Length; j++)
                row[j] = data[i, j];

            FWT(row);

            for (int j = 0; j < row.Length; j++)
                data[i, j] = row[j];
        }

        for (int j = 0; j < cols; j++)
        {
            for (int i = 0; i < col.Length; i++)
                col[i] = data[i, j];

            FWT(col);

            for (int i = 0; i < col.Length; i++)
                data[i, j] = col[i];
        }
    }
}
```

Percepcijsko heširanje

Slika1:  Slika2: 

Učitaj Učitaj

Average Hash Difference Hash Perception Hash **Wavelet Hash**

Hash prve slike: 0001000001101111011110101110100110111010110010011111111000
Hash druge slike: 0001010100011110000111101101000000101110000110100001111011110000
Isti bitovi: 40
Različiti bitovi: 24
62%

Usporedba slika pomoću Wavelet Hash algoritma

```
for (int j = 0; j < pomocnaBitmapa.Height; j++)  
{  
    for (int i = 0; i < pomocnaBitmapa.Width; i++)  
    {  
        pomocnaBitmapa.SetPixel(i, j, Color.FromArgb((int)Scale(-1, 1, 0, 255, Crvena[i, j]), (int)Scale(-1, 1, 0, 255, Zelena[i, j]), (int)Scale(-1, 1, 0, 255, Plava[i, j])));  
    }  
}
```

```
int avgr = 0, avgg = 0, avgb = 0;  
for (x = 0; x < bitmapPromjena.Width; x++)  
{  
    for (y = 0; y < bitmapPromjena.Height; y++)  
    {  
        r = bitmapPromjena.GetPixel(x, y).R;  
        g = bitmapPromjena.GetPixel(x, y).G;  
        b = bitmapPromjena.GetPixel(x, y).B;  
        avgr += r;  
        avgg += g;  
        avgb += b;  
    }  
}  
Color prosjekBoja = Color.FromArgb(avgr / 64, avgg / 64, avgb / 64);
```

Percepcijsko heširanje

Slika1:  Slika2: 

Učitaj Učitaj

Average Hash Difference Hash Perception Hash **Wavelet Hash**

Hash prve slike: 0001000001101111011110101110100110111011010110010011111111000
Hash druge slike: 0001010100011110000111101101000000101110000110100001111011110000
Isti bitovi: 40
Različiti bitovi: 24
62%

Usporedba slika pomoću Wavelet Hash algoritma

```
for (x = 0; x < bitmapPromjena.Width; x++)  
{  
    for (y = 0; y < bitmapPromjena.Height; y++)  
    {  
        if (bitmapPromjena.GetPixel(x, y).GetBrightness() < prosjekBoja.GetBrightness())  
            this.hashVrijednost += "1";  
        else  
            this.hashVrijednost += "0";  
    }  
}
```

Percepcijsko heširanje

Slika1:  Slika2: 

Učitaj Učitaj

Average Hash Difference Hash Perception Hash **Wavelet Hash**

Hash prve slike: 000100000110111101111010111010011011101011001001111111000
Hash druge slike: 0001010100011110000111101101000000101110000110100001111011110000
Isti bitovi: 40
Različiti bitovi: 24
62%



Zaključak

- algoritam stvaranja "otiska prsta" nad multimedijским sadržajem
- korištenje percepcijskog hashiranja kao baze podataka koja bi pratila sadržaj zaštićen autorskim pravima te spremala hash vrijednosti proizvedenih postova nekim od algoritama percepcijskog hashiranja
- provjera hash vrijednosti novoobjavljene objave s podacima iz baze podataka te dobivanje podataka o mogućem plagijatu dobivenim u postotcima ili sličnim korelacijskim vrijednostima
- razvojem online medija i objavljivanja sadržaja na internetu dolazimo do značajnog interesa i velike implementacijske zone za korištenje algoritama percepcijskog hashiranja koji su djelotvorni i efikasniji od standardnih algoritama provjera multimedijskog sadržaja s povećim sadržajem zapisanog u bazi podataka