КОМПЬЮТЕРНАЯ ОБРАБОТКА ИЗОБРАЖЕНИЙ

Неделя 14.

Лекция «Сжатие изображений без потерь».

Д. В. Иванов, А. С. Карпов, Е. П. Кузьмин, В. С. Лемпицкий, А. А. Хропов. Алгоритмические основы растровой машинной графики: Учебное пособие// Интернет-университет информационных технологий; Бином. Лаборатория знаний, 2010. – с. 230 – 238.

1. Необходимость сжатия изображений

Типичное изображение, полученное цифровой фотокамерой, имеет разрешение порядка 3000x2000, т.е. около 6 мегапикселей; для передачи цвета обычно используется 24 бита на пиксель. Таким образом, объем исходных данных составляет порядка 17 *мегабайт*. Для профессиональных устройств ввода изображений размер получаемого растра может быть значительно больше, а глубина цвета - достигать 48*бит* на пиксель. Соответственно, размер одного изображения может быть больше 200 *мегабайт*. Поэтому весьма актуальными являются алгоритмы **сжатия** изображений, или, иными словами, алгоритмы, которые позволяют уменьшить объем данных, представляющих изображение.

2. Два основных класса алгоритмов сжатия изображений

1. А называется алгоритмом сжатия без потерь (англ. lossless compression), если существует алгоритм А-1 (обратный к А) такой, что для любого изображения Img A(Img) = Img1 и А-1(Img1) = Img. Сжатие без потерь применяется в следующих графических форматах представления изображений: GIF, PCX, PNG, TGA, TIFF1, множество собственных форматов от производителей цифровых фотокамер.
2. А называется алгоритмом сжатия с потерями (англ. lossy compression), если он не обеспечивает точного восстановления исходного изображения. Алгоритм, обеспечивающий восстановление, обозначается как А*. Алгоритмы А, А* подбираются так, чтобы обеспечить большие коэффициенты сжатия при минимальной разнице между Img и Img2. Сжатие с потерями применяется в следующих графических форматах: JPEG, JPEG2000 и т.д.

3. Алгоритм кодирования повторений RLE - битовый уровень

Пусть дана последовательность битов; например, представляющих черно-белое изображение. Подряд обычно идет несколько 0 или 1, и можно помнить лишь количество идуших подряд одинаковых цифр. Можно считать, что каждое число повторений изменяется от 0 до 7 (т.е. можно закодировать ровно тремя битами). Тогда последовательность, состоящая из 21 единицы, 21 нуля, 3 единиц и 7 нулей закодируется так: 111 000 111 000 111 111 000 111 000 111 011 111, т.е. из исходной последовательности, которая имеет длину 51 бит, получили последовательность длиной 36 бит.

4. Примеры

1) 001 001 111 101 110 011 111 111 111 111

0 1 2 3 4 5 6 7 8 9

2) 111 111 000 111 000 111 000 111 000 111

0 1 2 3 4 5 6 7 8 9

3) При каких условиях с максимальное жатие?

4) 001 001 001 001 001 001 001 001 001 001

0 1 2 3 4 5 6 7 8 9

Коэффициент сжатия?

5. Алгоритм кодирования повторений RLE – байтовый уровень

- Входной поток разбивается на байты (буквы), и повторяющиеся буквы кодируются (количество, буква).
  Т.е. AABBBCDAA кодируем (2A) (3B) (1C) (1D) (2A).
- Чему равен коэффициент сжатия?

6. Алгоритм кодирования повторений RLE – байтовый уровень (модификация)

Пусть есть фиксированное число (буква) M (от 0 до 255), обычно M=127.. Тогда одиночную букву $S \leq M$ можно закодировать ею самою, - выход = S, а если букв несколько или $S : M < S \leq 255$, то выход = CS, где C > M, а C - M - количество идущих подряд букв S.

Другой вариант: признаком счетчика служит наличие единиц в 2старших битах считываемого байта. Остальные 6 битов суть значение счетчика. Такая модификация данного алгоритма используется в формате PCX, а также как одна из стадий конвейера сжатия (например, в формате JPEG).

7. Алгоритм декодирования

```
// M - фиксированная граница
// считываем байты из входного потока
// in - вход - сжатая последовательность
// out - выход - разжатая последовательност
while( in.Read( c ) )
{     if( c > M )
    {    // случай счетчика
         n = c - M;
         in.Read( c );
         for (i = 0;  i < n;  i++)
              out.Write( c );
    }
    else // случай просто символа
         out.Write( c );
}
```

8. Словарные алгоритмы. Алгоритм LZ77

Словарь - это N последних уже закодированных элементов последовательности. Кодирование состоит в нахождении наибольшей цепочки из словаря, совпадающей с обрабатываемой последовательностью. Если же совпадений нет, то записывается нулевое смещение, единичная длина и только первый элемент незакодированной последовательности - {0, 1, e}. Такая схема кодирования представляет собой скользящее окно (англ. sliding window), состоящее из двух частей:

1. подпоследовательность уже закодированных элементов длины N - словарь - буфер поиска (англ. search buffer);
2. подпоследовательность длины n из цепочки элементов, для которой будет произведена попытка поиска совпадения - буфер предварительного просмотра (англ. look-ahead buffer).

9. Алгоритм LZ77 (продолжение)

В терминах скользящего окна алгоритм сжатия описывается так: если e1, . . . , ei - уже закодированная подпоследовательность, то ei-N+1, . . . , ei - суть словарь или буфер поиска, а ei+1, . . . , ei+n - буфер предварительного просмотра. Пусть в скользящем окне найдена максимальная по длине совпавшая цепочка элементов ei-p, . . . , ei+q, тогда она будет закодирована следующим образом: {p+1, q+p+1, ei+p+q+2} - смещение относительно начала буфера предварительного просмотра (ei+1), длина совпавшей цепочки, элемент, следующий за совпавшей цепочкой из буфера предварительного просмотра.

10. Алгоритм LZ77 (продолжение)

Если в результате поиска найдено два совпадения с одинаковой длиной, то выбирается ближайшее к началу буфера предварительного просмотра. После этого скользящее окно смещается на p + q + 2 элементов вперед, и процедура поиска повторяется. Выбор чисел N и n является отдельной важной проблемой, т.к. чем больше N и n, тем больше места требуется для хранения значений смещения и длины. Естественно, что время работы алгоритма также возрастает с ростом N и n. Отметим, что обычно N и n различаются на порядок.

11. пример сжатия строки "TOBEORNOTTOBE" с параметрами N = 10 и n = 3

шаг	*скользящее окно*	макс. совпавшая цепочка	выход
1	""+"TOB"	T	0,1,T
2	"T"+"OBE"	O	0,1,O
3	"TO"+"*BEO*"	B	0,1,B
4	"TOB"+"*EOR*"	E	0,1,E
5	"TOBE"+"ORN"	O	3,1,R
6	"TOBEOR"+"NOT"	N	0,1,N
7	"TOBEORN"+"OTT"	O	3,1,T
8	"TOBEORNOT"+"TOBE"	TOB	9,3,E

12. Анализ алгоритма LZ77

Если выделить для хранения смещения 4 бита, длины - 2 бита, а элементов - 8 бит, то длина закодированной последовательности (без учета обозначения конца последовательности) составит 4 x 8 + 2 x 8 + 8 x 8 = 112 бит, а исходной - 102 бита. В данном случае мы не сжали последовательность, а, наоборот, увеличили избыточность представления. Это связано с тем, что длина последовательности слишком мала для такого алгоритма. Но, например, рисунок кодового дерева Хаффмена, занимающий 420 килобайт дискового пространства, после сжатия имеет размер около 310 килобайт.

13. Псевдокод алгоритма сжатия.

```
// M - фиксированная граница
// считываем символы из входного потока
// in - вход - сжатая последовательность
// n - максимальная длина цепочки
// pos - положение в словаре, len - длина цепочки
// nelem - элемент за цепочкой, str - найденная цепочка
// in - вход, out - выход
// SlideWindow - буфер поиска
while( !in.EOF() ) //пока есть данные
{ // ищем максимальное совпадение и его параметры
    SlideWindow.FindBestMatch( in, n, pos, len, nelem );
    // пишем выход: смещение, длина, элемент
    out.Write( pos ); out.Write( len ); out.Write( nelem );
    // сдвинем скользящее окно на len + 1 элементов
    SlideWindow.Move( in, len + 1);
```

14. Алгоритм LZ77 (декодирование)

Декодирование сжатой последовательности является прямой расшифровкой записанных кодов: каждой записи сопоставляется цепочка из словаря и явно записанного элемента, после чего производится сдвиг словаря. Очевидно, что словарь воссоздается по мере работы алгоритма декодирования. Процесс декодирования значительно проще с вычислительной точки зрения.

15. Псевдокод алгоритма LZ77 (декодирование)

```
// pos - положение в словаре, len - длина цепочки
// nelem - элемент за цепочкой, str - найденная цепочка
// in - вход, out – выход,  Dict - словарь
while(!in.EOF()) //пока есть данные
{ in.Read( pos );  in.Read( len ); in.Read( nelem );
    if(pos == 0) //новый отдельный символ
    { //удаляем из словаря первый (один) элемент
        Dict.Remove(1);
        //добавляем в словарь элемент
        Dict.Add(nelem); out.Write(nelem); }
    else { //скопируем строку из словаря
        str = Dict.Get( pos, len );
        //удалим из словаря len + 1 элементов
        Dict.Remove(len + 1);
        //Добавляем в словарь цепочку
        Dict.Add(str + nelem); out.Write(str + nelem );}
}
```