Алимагадов Курбан, ПМ-11М

Отчётное домашнее задание №3

Задание 1

Выполнить сжатие изображений Im1 и Im2 адаптивным арифметическим кодером, в котором следует задать параметр NO_OF_CHARS=16. Сравнить полученные битовые затраты (бит/пиксель) с величиной двоичной энтропии первого порядка.

Рассмотрим два проквантованных изображения “Lena” (с шумом и без).

Изображение “Lena”, проквантованное с добавлением шума.

Изображение “Lena”, проквантованное без добавления шума.

Так как оба изображения сохранены в формате .bmp их файлы содержат метаданные в кодировке ASCII (8 бит на пиксель), поэтому при попытке сократить алфавит кодера с 256 символов до 16 могут возникнуть проблемы с кодированием/декодированием, а также потери данных. Чтобы избежать этого сохраним информацию о яркостях пикселей изображений в .bin файлах “quantized_with_noise.bin” и “quantized_without_noise.bin” соответственно (код в файле task1.m) и будем работать с ними.

Полученные результаты представлены в таблице.

Название файла	Исходные битовые затраты	Энтропия (с учётом EOF)	Энтропия (без учёта EOF)	Затраты после сжатия	Затраты восстановленного файла
quantized_with_noise.bin	8 бит/пиксель	3.716760 бит/пиксель	3.525497 бит/пиксель	3.646637 бит/пиксель	8 бит/пиксель
quantized_without_noise.bin	8 бит/пиксель	3.649869 бит/пиксель	3.481060 бит/пиксель	3.619385 бит/пиксель	8 бит/пиксель

Следует отметить, что если при подсчёте энтропии учитывать вероятность появления EOF-символа, то величина энтропии может получиться выше битовых затрат. Однако нужно помнить, что в исходном файле EOF-символа нет, и используется он только при кодировании. Из теоретических сведений известно, что при увеличении длины кодируемой последовательности битовые затраты на хранение EOF-символа стремятся к нулю. При подсчёте энтропии без учёта EOF-символа её значение получается меньше битовых затрат.

В дальнейшем при подсчёте энтропии будем вычислять её без учёта EOF-символа. Код на языке C++ в файле adaptive1.cpp.

Задание 2

Модифицировать программу адаптивного арифметического кодера, введя вместо одной 16 моделей (гистограмм), которые выбираются кодером по значению предыдущего закодированного (декодированного) символа $s \in \{0,\ldots,15\}$. Сжать два тестовых изображения и сравнить полученные битовые затраты с их двоичными энтропиями второго порядка. (Как оценить энтропию первого и второго порядка?)

Введём вместо одной 16 гистограмм:

```
#define LEVELS_OF_BRIGHTNESS NO_OF_CHARS
unsigned int    cum_freq[LEVELS_OF_BRIGHTNESS][NO_OF_SYMBOLS + 1];
```

Для того, чтобы контролировать выбор очередной гистограммы для кодирования следующего символа воспользуемся указателем, который будет обращаться к одной из гистограмм `cum_freq[][]` в зависимости от значения последнего закодированного/декодированного символа.

```
unsigned int                *temp_cum_freq;


inline void update_model2(int symbol)
{
      if (temp_cum_freq[NO_OF_SYMBOLS] == MAX_FREQUENCY)
      {
            int cum = 0, freq = 0;
            for (int i = 0; i < NO_OF_SYMBOLS; i++)
            {
              freq = (temp_cum_freq[i + 1] - temp_cum_freq[i] + 1) >> 1;
              temp_cum_freq[i] = cum;
              cum += freq;
            }
            temp_cum_freq[NO_OF_SYMBOLS] = cum;
      }
      for (int i = symbol + 1; i <= NO_OF_SYMBOLS; i++)
            temp_cum_freq[i]++;
}


void encode(void)
{
      int q = 256 / LEVELS_OF_BRIGHTNESS;

      int symbol;
      start_model1();
      start_model2();
      start_encoding();
      while ((symbol = getc(in)) != EOF)
      {
            symbol = (symbol - q / 2) / q;
            encode_symbol(symbol);
            update_model1(symbol);
            update_model2(symbol);
            temp_cum_freq = cum_freq[symbol];
      }
      encode_symbol(EOF_SYMBOL);
      done_encoding();
}
```

Для оценки энтропии 1-го порядка можно использовать ещё одну гистограмму `cum_freq1[]`, которая будет обновляться как в первом задании – после каждого закодированного/декодированного символа. Вычислив по ней оценки вероятностей появления символов алфавита, можно подсчитать энтропию 1-го порядка.

Для оценки энтропии 2-го порядка будем использовать массив гистограмм `cum_freq[][]`. Каждая i-ая гистограмма в этом массиве соответствует закону распределения вероятностей появления символов алфавита при условии, что на предыдущем шаге был закодирован/декодирован i-й символ. Поэтому сначала необходимо рассчитать 16 условных энтропий $H(X|x_i)$, а затем, воспользовавшись информацией о вероятностях появления символов $p(x_i)$ (полученной из `cum_freq1[]`), подсчитать энтропию 2-го порядка по формуле:

$$H_2(X) = \sum_{k=0}^{15} p(x_k) H(X|x_k).$$

Полученные результаты представлены в таблице.

Название файла	Исходные битовые затраты	Энтропия 1-го порядка	Энтропия 2-го порядка	Затраты после сжатия	Затраты восстановленного файла
quantized_with_noise.bin	8 бит/пиксель	3.525497 бит/пиксель	1.532649 бит/пиксель	1.678833 бит/пиксель	8 бит/пиксель
quantized_without_noise.bin	8 бит/пиксель	3.481060 бит/пиксель	1.159489 бит/пиксель	1.253052 бит/пиксель	8 бит/пиксель

Можно видеть, что использование информации о предыдущем закодированном/декодированном символе позволяет добиться более эффективного сжатия. Значение энтропии 2-го порядка для обоих изображений меньше значения энтропии 1-го порядка.

Код на языке C++ в файле adaptive2.cpp.

Задание 3

Модифицировать программу адаптивного арифметического кодера, введя вместо одной 256 моделей (гистограмм), номер N которой выбирается кодером по значению предыдущего закодированного (декодированного) вертикального $v \in \{0,\ldots,15\}$ и горизонтального $h \in \{0,\ldots,15\}$ соседних пикселей как $N=16v+h$. Сжать два тестовых изображения и сравнить результаты между собой и с полученными в п.2 задания. (Что делать, если соседних пикселей нет?)

Введём вместо одной 256 гистограмм:

```
unsigned int cum_freq[LEVELS_OF_BRIGHTNESS * LEVELS_OF_BRIGHTNESS][NO_OF_SYMBOLS + 1];
```

Чтобы использовать информацию о соседних вертикальных пикселях при кодировании/декодировании изображений размером 512x512 пикселей, придётся хранить в памяти последние обработанные 512 обработанных пикселей. Для решения этой задачи удобно использовать структуру двухсторонняя очередь (deque), так как для неё операции добавления/удаления элемента в начало (конец) имеют константную сложность $O(1)$:

```
std::deque<int> mas_sym;
```

Рассмотрим случаи, когда кодируемого пикселя нет соседей. Когда у пикселя нет горизонтального соседа, можно считать, что соседним является любой. Когда у пикселя нет вертикального соседа, можно считать, что вертикальный сосед равен горизонтальному.

Полученные результаты представлены в таблице.

Название файла	Исходные битовые затраты	Результаты 2-го задания	Затраты после сжатия	Затраты восстановленного файла
quantized_with_noise.bin	8 бит/пиксель	1.678833 бит/пиксель	1.510162 бит/пиксель	8 бит/пиксель
quantized_without_noise.bin	8 бит/пиксель	1.253052 бит/пиксель	1.072601 бит/пиксель	8 бит/пиксель

Можно видеть, что использование информации о горизонтальных и вертикальных соседях кодируемого/декодируемого пикселя позволяет

добиться ещё большей эффективности сжатия, по сравнению с результатами задания 2.

Код на языке C++ в файле adaptive3.cpp.

Задание 4

Используя программу, полученную при выполнении п.3, повторить сжатие двух тестовых изображений, выбирая модель кодирования как N=16M+m, где M=max(v,h), m=min(v,h). Сколько из 256 моделей будет при этом использоваться? Сравнить результаты сжатия с полученными в п.3.

Выбирая модель кодирования как N=16M+m, где M=max(v,h), m=min(v,h), из всех 256 моделей, содержащихся в таблице `cum_freq[]`, мы будем использовать только те, для которых справедливо условие: номер строки ячейки в таблице больше равен номера её столбца, т. е. ячейки, лежащие на побочной диагонали таблицы и ниже (их 136).

Полученные результаты представлены в таблице.

Название файла	Исходные битовые затраты	Результаты 3-го задания	Затраты после сжатия	Затраты восстановленного файла
quantized_with_noise.bin	8 бит/пиксель	1.510162 бит/пиксель	1.512207 бит/пиксель	8 бит/пиксель
quantized_without_noise.bin	8 бит/пиксель	1.072601 бит/пиксель	1.086731 бит/пиксель	8 бит/пиксель

Можно видеть, что полученные результаты уступают по эффективности сжатия результатам 3-го задания, однако очень незначительно, причём данная реализация кодера использует почти в 2 раза меньше гистограмм, по сравнению с реализацией кодера из задания 3.

Код на языке C++ в файле adaptive4.cpp.

Выводы:

В работе был рассмотрен алгоритм арифметического кодирования полутоновых изображений с учётом информации о соседних ранее закодированных/декодированных пикселях и без её учёта. Результаты кодирования показали, что наиболее эффективного сжатия можно добиться,

используя информацию о вертикальных и горизонтальных соседях кодируемых пикселей. Несильно потеряв в эффективности сжатия, можно ускорить работу кодера, используя не 256 гистограмм, а 136 выбирая модель кодирования как N=16M+m, где M=max(v,h), m=min(v,h).

Код программ представлен в файлах adaptive1.cpp, adaptive2.cpp, adaptive3.cpp, adaptive4.cpp. Скрипт для преобразования BMP файлов в формат .bin и просмотра декодированных изображений приведён в файле task1.m.