**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO ĐỒ ÁN**

Môn: tính toán đa phương tiện.

Giảng viên hướng dẫn: Đỗ Văn Tiến.

Thành viên:

Lâm Văn Loát - 16520674

Dương Chí Vinh - 16521438

Giảng Phúc Vinh - 16521548

Nguyễn Hữu Thắng - 16521102

**I. Nội dung đồ án.**

\*Tìm hiểu, phân tích và xây dựng chương trình minh hoạ, đánh giá hiệu quả các thuật toán dùng để nén và giải nén dữ liệu multimedia.

Các thuật toán nhóm chọn để thực hiện:

Huffman

LZW

JPEG lossless (Huffman)

Ảnh có chứa ảnh chụp màn hình

Mô tả được tạo tự độngDữ liệu mà nhóm sẽ áp dụng để chạy các thuật toán trên là các ảnh sau:

test 1: color.jpg (980x325) – 955500 bytes

Ảnh có chứa người, người phụ nữ, mũ, phụ kiện đội đầu

Mô tả được tạo tự động

test 2: lena.jpg (512x512) – 786432 bytes



Ảnh có chứa bầu trời, cỏ, ngoài trời, tòa nhà

Mô tả được tạo tự độngtest 3: stripe.jpg (485x710) – 1033050 bytes

test 4: tower.jpg (572x388) – 665808 bytes

**II. Chi tiết thực hiện.**

**LZW**

LZW là thuật toán nén ảnh với ý tưởng xây dựng 1 bảng tra mã để phục vụ cho việc nén và giải nén. Đặc biệt, bảng mã nãy không cần lưu trong quá trình nén. Tuy nhiên, điều kiện cần có là việc nén và giải nén phải có cùng chung bảng mã khởi tạo (initial dictionary). Đối với việc nén, lần lượt quét các kí tự, kiểm tra bảng mã và xử lí, gửi tín hiệu cho bên giải nén. Phía giải nén sẽ lần lượt nhận từng tín hiệu và xử lí, xây dựng lại bảng mã.

Tuy nhiên với việc thực hiện trên máy tính không thể code theo cách stream dữ liệu như vậy nên chúng em sẽ nén và lưu trên file và đọc file để giải nén.

Chi tiết thực hiện :

Compression:

* Get\_initial\_table: khởi tạo bảng mã ( initial code )
* Write\_keys, write\_values: lưu lại giá trị của bảng mã khởi tạo.
* Encoding: Mã hoá dữ liệu bằng thuật toán LZW
* Adapt\_encoded: xử lí dữ liệu đã mã hoá trước khi lưu.

Decompression:

Đọc các file đã lưu và áp dụng thuật toán Decode LZW.

**Huffman**

Thuật toán Huffman dựa trên ý tưởng mã hoá các giá trị có trong dữ liệu. Giá trị nào xuất hiện với tần số càng lớn sẽ có mã càng ngắn.

Chi tiết thực hiện

Compression:

* Make frequency dictionary : tạo bảng tuần số của dữ liệu.
* Make heap (use min heap): sắp xếp lại bảng tần số bằng heap.
* Merge nodes and build tree: lần lượt ghép 2 node có giá trị thấp nhất lại cho đến khi heap chỉ còn 1 node, từ đó ta có cây Huffman.
* Make codes: Từ cây Huffman, tạo thành bảng mã để encode dữ liệu.
* Encode text: Mã hoá dữ liệu.
* Pad encoded text: Xử lí dữ liệu nếu độ dài dữ liệu không chia hết cho 8. Thêm phần padding vào dữ liệu để đánh dấu.
* Make byte array: biến dữ liệu thành mảng bit để dễ dàng trong việc lưu trữ.
* Output the byte array to binary file: Lưu dữ liệu.
* Output the codes\_reverse\_mapping: Lưu bảng mã để decode.

Decompression:

* Rebuild the codes\_revers\_mapping: Đọc bảng mã từ file binary
* Decode text: Giải mã dữ liệu.
* Rebuild the channel of the original image: Tái tạo lại hình ảnh.

**JPEG Lossless ( Huffman )**

Đối với JPEG Lossless, việc encode sẽ không được thực hiện trực tiếp trên dữ liệu ảnh có sẵn, mà phải qua 1 bước tiền xử lí. Tính chất của ảnh là các pixel gần nhau thường sẽ có giá trị giống nhau hoặc giá trị của pixel sẽ biến đổi đều theo không gian.

Dựa vào điều trên, bước tiền xử lí này sẽ chuyển đổi ma trận ảnh thành ma trận giá trị biến đổi của ảnh.

Chi tiết thực hiện:

Compression :

Đầu tiên, ta sẽ đọc ảnh, chia ảnh thành 3 kênh màu, mà biến đổi tất cả thành ma trận 1 chiều (gọi là ma trận A). Như đã nói phía trên, t sẽ xử lí dữ liệu bằng cách với mỗi phần tử trong A, giá trị mới sẽ bằng giá trị của của chính nó trừ cho giá trị phần từ đứng trước. Phần tử đầu tiên sẽ giữ nguyên. Việc này sẽ do hàm make\_data(r, g, b) đảm nhận. Tiếp theo, ta áp dụng thuật toán Huffman để lưu dữ liệu trên

Decompression:

Áp dụng thuật toán giải nén Huffman. Tuy nhiên, sau đó ta phải xử lí lại dữ liệu ngược lại như cách compress để tạo lại ma trận ảnh ban đầu.

**III. Thực nghiệm.**

Compress: Với mỗi thuật toán, ta sẽ có thời gian nén và độ lớn dữ liệu sau khi nén như sau:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test | Original Size  (KB) | LZW  (Time(s)|Size(KB)) | | Huffman  (Time(s)|Size(KB)) | | JPEG  (Time(s)|Size(KB)) | |
| 1 | 933 | 3.4 | 39 | 1.82 | 311 | 14.6 | 246 |
| 2 | 768 | 3.3 | 1022 | 2.55 | 695 | 3.53 | 512 |
| 3 | 1009 | 4.9 | 1510 | 3.72 | 915 | 6.38 | 831 |
| 4 | 650 | 2.9 | 815 | 2.13 | 614 | 3 | 424 |

Decompress: Với mỗi thuật toán, ta sẽ có thời gian giải nén như sau:

|  |  |  |  |
| --- | --- | --- | --- |
| Test | LZW | Huffman | JPEG |
| 1 | 1 s | 180 s | 2.5 s |
| 2 | 2.75 s | 55 s | 4.8 s |
| 3 | 4 s | 258 s | 8.8 s |
| 4 | 2.1 s | 38 s | 3.76 s |

**Nhận xét:**

* Thời gian decompress của thuật toán Huffman rất lớn, nguyên nhân vẫn chưa xác định được. Như đã báo cáo với thầy, khi lập trình thuật toán Huffman nếu tụi em để chung code compress và decompress trong 1 file thì thời gian chạy rất nhanh, tuy nhiên khi tách thành 2 file thì thời gian của decompress là rất lâu.
* Với thuật toán LZW, dữ liệu encode ra lớn hơn dữ liệu gốc. Nhóm xác định nguyên nhân là do thuật toán này thực hiện bằng cách encode các cụm được lặp nhiều lần trong dữ liệu nhưng đó không phải là tính chất của ảnh. Với test 1, vì tính chất của ảnh nên LZW mới có thể nén thành công.
* Đối với thuật toán Huffman truyền thống và JPEG thì JPEG hoàn toàn vượt trội về dung lượng nén vì JPEG tận dụng được tính chất của ảnh (giá trị biến đổi đều về không gian) nhưng lại khá chậm hơn vì phải tốn thêm công đoạn tiền xử lí dữ liệu.

**IV. Phân công công việc.**

|  |  |
| --- | --- |
| Thành viên | Công việc |
| Lâm Văn Loát | Code thuật toán Huffman, thực nghiệm. |
| Dương Chí Vinh | Code thuật toán LZW, thực nghiệm. |
| Giảng Phúc Vinh | Xây dựng slide, debug, viết báo cáo. |
| Nguyễn Hữu Thắng | Debug, code thuật toán JPEG lossless, viết báo cáo. |

**V. Tham khảo**

* **Fundamentals of Multimedia**, Second Edition, Ze-Nian Li, Mark S. Drew, Jiangchuan Liu
* <http://bhrigu.me/blog/2017/01/17/huffman-coding-python-implementation/>